VLSI Products
1990

Application Book

INTRODUCTION

This Application Book provides a collection of application notes and design hints for advanced LSIs from NEC. It is based upon practical experience made by NEC application engineers. Its intention is to provide users of NEC components with valuable design information for their own developments. The authors welcome any kind of feedback for their work and future editions of this Application Book.

NOTE:

All missing numbers of APPLICATION NOTES
are intentionally not released.

Part A
μCOM Application Notes

Part B
High Complex Peripherals Application Notes

Part C
Technical Letters

Part D
Application News

Глава 4
Моя любимая МООЦ

Глава 5
Автобиография Кайлиса Пейн
Моя любимая МООЦ

Глава 6
Заключительный

Глава 7
Моя любимая

Table of Contents

Part A μCOM Application Notes

Part B High Complex Peripherals Application Notes

Part C Technical Letters

Part D Application News

Part A
μCOM Application Notes

A CMOS Single Board Computer using the 16-Bit CMOS V-Series Standard Microprocessor μPD70116 (V30) and CMOS Peripheral Chips

Contents:

Authors: Shyam Gupta, Rolf Elter
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD70108/70116	User's Manual
μPD70116	Product Description
μPD71011	Product Description
μPD71051	Product Description
μPD71054	Product Description
μPD71055	Product Description
μPD71059	Product Description
μPD71088	Product Description

Related Products

μPD70116	16-bit microprocessor	CMOS
μPD71011	Clock generator/driver	CMOS
μPD71051	Serial interface unit	CMOS
μPD71054	Timer controller unit	CMOS
μPD71055	Parallel interface unit	CMOS
μPD71059	Interrupt controller unit	CMOS
μPD71088	Bus interface unit	CMOS

1. Introduction

The SBC-V30 is a single board computer built around the 16-bit CMOS standard microprocessor μPD70116 (also called V30 microprocessor) of NEC's V-series family. The supporting devices such as, latches, line drivers, bus controller, serial and parallel interface units, timer and the interrupt controller unit used in this application are all members of NEC's CMOS peripheral devices series.

The aim of this application note is to demonstrate the ease of designing a complete CMOS stand alone microcomputer board and in particular, to show the interface between the CPU and the various CMOS peripheral devices offered by NEC. It is beyond the scope of this application note to go into details of the peripheral devices and therefore only the application specific features will be described in brief. Figure 1.1 shows the block diagram of SBC-V30.

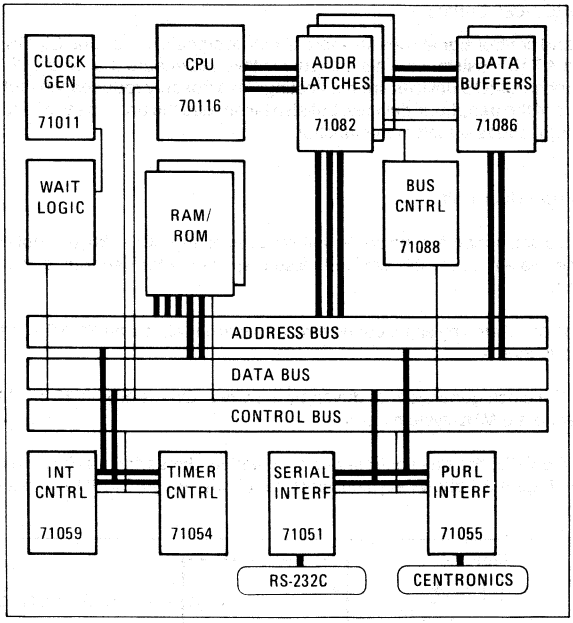


Figure 1.1 SBC-V30 block diagram

The main features of the SBC-V30 have been chosen so that the basic functions of a typical microcomputer system are realized, and yet simplicity of design is maintained for a understanding of various features of the microprocessor and peripheral devices (the parallel interface unit, the serial interface unit, the interrupt controller etc...).

2. Circuit description

In this section a brief description of the major CMOS devices used and their function on this single board computer is given.

2.1 Central processing unit

Although the design described there is a single master stand alone system, the CPU operates in the so-called "large scale system". Under the large scale system configuration, the processor control signals for accessing the memory and I/O devices are given by the bus status signals BS0, BS1 and BS2. The μPD71088 bus controller unit monitors these signals and generates the memory and I/O read/write signals together with the address latch, buffer control signal etc.

The μPD70116 microprocesor address/data signals are output in time multiplex mode. The address outputs are latched using the 8-bit CMOS μPD71082 latches with the help of the address strobe signal. The data lines are buffered through the μPD71086 buffers.

All address/data lines and control signal lines (in short all output signals) to various CMOS peripheral devices have been pulled up using a 4.7 KΩ resistance. This measure is taken to ensure that when CPU enters the hold mode, input stages of the attaches CMOS devices are not floating. This not only reduces the higher current consumption through the input stages of the peripheral devices, but also eliminates the probability of undesired oscillations at the device output which might cause a system malfunction.

2.2 Clock generator and driver unit

The μPD71011, a CMOS clock generator and driver unit are used on the SBC-V30 to supply the CPU with the system clock. In addition to the system clock, this device is also responsible for generating the CPU "Ready"- and the system "RESET" signal.

The system clock can be generated by either connecting a crystal directly to the X1, X2 inputs of μPD71011 or by connecting an external clock source.

The μPD71011 divides the clock generated, or driven by it, by two. On the SBC-V30, a 16 MHz parallel resonance crystal is used to generate a 8 MHz system clock.

This clock is used for the CPU, the bus controller unit μPD71088, the serial interface unit μPD71051 and the wait state generation circuit. Figure 2.1 shows a typical clock circuit with the μPD71011 using a crystal for clock generation.

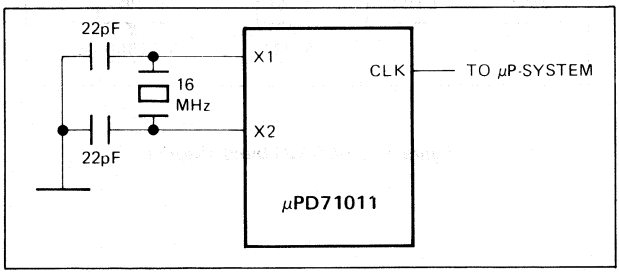


Figure 2.1 Typical clock generation circuit

2.3 The serial interface unit

The μPD71051 is a serial interface unit capable of data communication in asynchronous/synchronous mode. On SBC-V30, the μPD71051 has been configured to function in asynchronous mode to build a RS-232 serial interface. The RS-232c data communication format used is:

— 9600 baud rate transfer rate
— X16 clock rate
— 8 bits/character
— 2 stop bits
— no parity

Prior to the operation, a set of two registers must be programmed. The first one is the mode control register which defines the communication data format as shown above and determines whether the communication is to take place in synchronous or asynchronous mode. The second is the status word register which controls the communication protocol and is also used to read the status of interface control lines during operation, if required. The mode control instruction must be the first instruction after the device has been reset.

Using the μPD71051, serial communication can be handeled in polling or interrupt mode. In interrupt mode the CPU is busy attending to other jobs and the serial interface is served only is an interrupt occurs. For this purpose one can use the 'TXRDY' (transmitter ready) and the 'RXRDY' (receiver ready) lines. These lines are connected to the interrupt controller unit which generates an interrupt to the CPU.

On SBC-V30 the μPD71051 asynchronous data communication is done in the so called polling mode. The baud rate clock is derived from the timer O of the μPD71054 timer controller unit.

Programming examples of the same are given in appendix A.

2.4 The timer controller unit

The μPD71054 timer controller unit contains three 16-bit timers which may be programmed independently. All in all, six different programming modes for these timers are offered to suit a wide range of application requirements.

After power-on, the state of the counters is undefined and must be initialized. Counters are programmed by writing first to the control word and then loading an initial count for the timer. The control word selects the counter to be programmed, it's operation and count mode (binary count or BCD), etc. Once the counter has been loaded, it starts it's operation and continues the next initialization.

On SBC-V30, the timer O is used to realize a real time clock. The timer is fed with 1.23 MHz clock. This clock is divided by the timer and a pulse is output every 50 milli-seconds to the interrupt controller device μPD71059, which generates an interrupt to the CPU. The interrupt service routine updates the clock in the memory and the time is displayed on a terminal through the RS-232c serial interface controlled by the μPD71051.

Timer 2 is used to generate the baud rate clock for the serial interface unit. For this purpose, again the incoming 1.23 MHz clock is divided by a factor to generate a 9600X 16 Hz clock. This clock is fed to the "RXC" and "TXC" input of the serial interface unit for 9600 baud data communication. Different baud rates can be realized easily by altering the dividing factor for timer 2.

The timer 1 of the μPD71054 is used as an interval counter. For this purpose the contents of the timer are latched at the start of an event, the contents read and saved in memory.

At the end of the event, the latch command is used again. The difference between the two latched readings provides the duration of the event.

Programming examples for μPD71054 are given in appendix A.

2.5 The parallel interface unit

The μPD71055 parallel interface unit offers three 8-bit wide parallel ports which may be programmed in three different modes and two groups. Each of the ports can be used as input, output or control port, depending upon the mode and group combination selected by the user.

On the SBC-V30, the μPD71055 has been programmed in group 0 and mode 0. In this configuration the port 0 has been defined as 8-bit output port and the lower nibble of the port 2 is used for control signal inputs.

The parallel output port and control function of port 2 are used to configure a "Centronics" interface for connection to a printer. The 8-bit parallel data for the printer is sent over the port 0, and printer handshake (lines like 'BUSY', etc.) are controlled through the port 2.

Port 1 of the μPD71055 can be very easily configured as another 8-bit parallel input or output port. This port is free on the SBC-V30.

2.6 The interrupt controller unit

The μPD71059 interrupt controller unit can handle up to eight programmable interrupts for die CPU. The device is capable of operation in cascade mode for up to a maximum of 64 interrupt inputs. Each interrupt can be individually masked and prioritized. When one or more interrupts occur at the same time, the interrupt controller generates an interrupt request for the highest priority interrupt to the CPU.

The interrupt controller can be initialized by programming two sets of registers. The first one is the set of IIW's (Interrupt initialization word) and the second one is the set of 0CW's (Operation command words). Which one and how many of these words are to be programmed depends upon the application and the IIW1.

The interrupt initialization words determine the type and mode of interrupt operation (like master/slave mode, single/cascade mode, edge/level triggered interrupt etc.). The IIWs must be defined immediately after the device reset, whereas the operation command words may be defined/changed at any time during the system operation.

On SBC-V30, the μPD71059 is used in a single master mode with interrupts zero to seven programmed in descending order of priority. Out of these eight interrupts only two are used:

— Interrupt level 0 for μPD71054 timer 0 interrupt.

— Interrupt level 1 for time out interrupt generated by the wait state logic.
All other interrupts are not used on this board.

2.8 WAIT state logic

On the SBC-V30, the user can select between zero, one, or two wait states for each memory or I/O access by the CPU. The wait states are generated by the 74HC175 D-type flip-flops.

The 74HC175 is cleared by the address strobe signal at the start of every bus cycle and counts the system clocks. The latched output is used as wait state together with the chip select signal for memory or I/O access. This signal fed to the μPD71011 generates the ready signal for the CPU.

If for any reason, the ready signal is not generated by the wait state logic, an on board watch dog timer terminates the access cycle of the CPU after 16 clock cycles have elapsed. At the same time, an interrupt is generated to inform the CPU about the abnormal cycle termination.

As on the SBC-V30 ROMs and only static RAMs are used and no other bus master is present to complete for the system memory or I/O devices, (the wait state logic is only of academic interest).

On SBC-V30, all memory and I/O cycles are executed without wait states.

3. Memory and I/O map

The μPD70116 offers to the user IM-bytes of memory addressing area and 64K-bytes of I/O area. Out of this memory area the SBC-V30 has 32K-bytes of RAM and 16K-bytes of ROM on board. The various I/O devices are located in the I/O area. The complete memory and I/O area map are shown below:

A) Memory map

Address	Device
0H - 7FFFH	RAM 32K-bytes
FC000H - FFFFFH	ROM 16K-bytes

B) I/O map

Address	Register	Device
0000H	Data port	μPD71051
0002H	Control/status port	
0010H	Counter 0	μPD71054
0012H	Counter 1	
0014H	Counter 2	
0016H	Mode register	
0020H	Port A	μPD71055
0022H	Port B	
0024H	Port C	
0026H	Mode register	
0030H	Int. control word	μPD71059
0032H	Int. command word	

APPENDIX A: SBC-V30 PROGRAMMING EXAMPLES

NEC ELECTRONICS (EUROPE) GMBH, APPLICATION DEPT.
OVERVIEW OF START ADDRESSES FOR V30 DESIGN KIT DEMONSTRATIONS

ADDRESSES OF ROUTINES

FC00:0000	MAIN PROGRAM
:001C	KEYBOARD INPUT FOR MENUE
:0060	CLEAR LOWER PAGE PART
:0073	REPAIR MENUE AFTER DEMO
:0080	READ ONE CHARACTER FROM TERMINAL
:00B0	READ SEVERAL CHARACTERS FROM TERMINAL AND SAVE
:0100	SEND CHARACTERS TO TERMINAL IN LOOP UNTIL '00'
:0120	START BENCHMARK COUNTER
:0130	HEX TO ASCII CONVERSION FOR FIVE CHARACTERS
:0180	COPY INTERRUPT VECTORS TO SEGMENT '0000'
:0200	RESET BUFFER MEMORY AREA
:0220	INITIALIZE TIMER/COUNTER 71054
:0250	INITIALIZE SERIAL INTERFACE 71051 FOR TRANSMIT
:0280	INITIALIZE SERIAL INTERFACE 71051 FOR TRANSCEIVE
:02A0	STOP BENCHMARK COUNTER
:0300	INITIALIZE PARALLEL INTERFACE 71055 FOR CENTRONICS INTERFACE AND PRINT IN LOOP
:0360	INITIALIZE INTERRUPT CONTROLLER 71059 FOR WATCH
:03FF	INTERRUPT SERVICE ROUTINE FOR WATCH
:0900	SERIAL INTERFACE DEMONSTRATION
:0A00	PRINTER DEMONSTRATION
:1000	DEMONSTRATION PROGRAM #3
:1100	DEMONSTRATION PROGRAM #4
:1300	DEMONSTRATION PROGRAM #5
:1500	DEMONSTRATION PROGRAM #6
:1700	DEMONSTRATION PROGRAM #7

ADDRESSES OF TABLES

0000:0402	DATA BUFFER FOR WATCH
:0409	BUFFER FOR SELECTED DEMO NUMBER
:040A	BUFFER FOR ASCII CHARACTERS OF STOPWATCH
:0500	DATA BUFFER FOR PRINTER AND SIO DEMO

FC00:01C0	INTERRUPT VECTORS
:04C0	CURSOR MOVE TO WATCH / START POSITION
:04E0	CURSOR MOVE FOR STOP WATCH / START POSITION
:0580	MAIN MENUE
:0700	RESET SCREEN IN LOWER PART FOR DEMO DISPLAY
:0B00	ASCII TEXT FILE FOR DEMO #1
:0C00	ASCII TEXT FILE FOR DEMO #2
:0D00	NEC V-SERIES LOGO FOR DOT MATRIX PRINTER

```
;*************************************************
;                                                 :
;          MAINPROGRAM V30 DESIGN KIT DEMO        :
;                                                 :
;*************************************************
;
;
;WRITTEN BY
;NEC ELECTRONICS (EUROPE), APPLICATION DEPT.
;
;
;THE MAIN PROGRAM INITIALIZES THE PERIPHERALS
;AND HANDLES THE DEMONSTRATION ROUTINES IN A
;MENUE DRIVEN WAY
;
;FC00:0580 TABLE MENUE MASK
;

      ORG 0000

      DI

      CALL BUFRES      ;RESET DATA BUFFER AREA IN RAM
      CALL INIVECT     ;INITIALIZE INTERRUPT VECTORS
      CALL INICOUN     ;INITIALIZE TIMER/COUNTER
      CALL INIISOT     ;INITIALIZE SIO FOR OUTPUT
      CALL ININT       ;INITIALIZE INTERRUPT CONTROLLER

      DI               ;DISABLE TIMER INTERRUPT

      MOV AW,0580      ;SET OFFSET ADDRESS FOR MENUE PATTERN
      MOV BW,FC00
      CALL DISPLAY     ;WRITE MENUE PATTERN TO TERMINAL

      EI               ;ENABLE TIMER INTERRUPT FOR WATCH

MAINRET:

      CALL INISIOR     ;INITIALIZE INPUT ROUTINE FOR ONE CHAR^CTER
      MOV CL,31        ;LOAD CHARACTER TO START COMPARISON WITH
      MOV BW,OFFSET BREAKTAB    ;LOAD BASE OFFSET FOR BREAK TABLE

CHARCMP:

      CMP CL,DL        ;COMPARE READ CHARACTER WITH REFERENCE
      BE BREVEC        ;IF FOUND JUMP TO BREAK VEKTOR
      CMP CL,37        ;TEST FOR MAX. ASCII VALUE
      BE BREND         ;JUMP OUT OF LOOP AT MAX. VALUE

      ADD BW,05        ;CALCULATE NEXT BREAK VEKTOR ADDRESS
      INC CL           ;INCREMENT REFERENCE VALUE
      BR CHARCMP       ;LOOP

BREVEC:

      BR BW            ;JUMP TO CALCULATED BRAEK ADDRESS
```

```
BREAKL:

      BRK 20            ;GOSUB DEMO #1

BREND:

      BR MASKREP        ;REWRITE USED SCREEN AREA AND CONTINUE
                            MAIN PROGRAM
      BRK 21            ;GOSUB DEMO #2
      BR MASKREP        ;RETURN TO MAIN PROGRAM

      BRK 22            ;GOSUB DEMO #3
      BR MAINRET        ;RETURN TO MAIN PROGRAM

      BRK 23            ;GOSUB DEMO #4
      BR MAINRET        ;RETURN TO MAIN PROGRAM

      BRK 24            ;GOSUB DEMO #5
      BR MAINRET        ;RETURN TO MAIN PROGRAM

      BRK 25            ;GOSUB DEMO #6
      BR MAINRET        ;RETURN TO MAIN PROGRAM

      BRK 26            ;GOSUB DEMO #7
      BR MAINRET        ;RETURN TO MAIN PROGRAM


                ;
                ;END OF MAIN PROGRAM
                ;
```

```
                ;************************************************
                ;                                               :
                ;     SUBROUTINE TO READ THE STOPWATCH AND      :
                ;          DISPLAY THE CALCULATED VALUE         :
                ;                                               :
                ;************************************************
                ;
                ;THIS SUBROUTINE FIRST READ OUT THE CLOCK COUNTS
                ; AND THEN CALCULATE THE REAL TIME.
                ;AFTER CALCULATION THE VALUE IS CONVERTED TO AN
                ;ASCII CHARACTER AND DISPLAYED ON SCREEN.
                ;
                ;FC00:500 TABLE FOR CURSOR MOVE
                ;0000:409 STACK FOR ASCII CONVERTED CHARACTERS

        ORG 02A0

STOPCNT:

        IN AL,10        ;READ FIRST COUNT LOW BYTE
        MOV BL,AL       ;BUFFER VALUE
        IN AL,10        ;READ FIRST COUNT HIGH BYTE
        MOV BH,AL

        CALL BUFCOUN    ;LATCH SECOND COUNT VALUE

        IN AL,10        ;READ SECOND COUNT VALUE AND SAVE
        MOV DL,AL
        IN AL,10
        MOV DH,AL

        EI              ;ALLOW NORMAL TIMER INTERRUPTS AGAIN


                        ;*** CALCULATION OF TIME ***

        SUBC DW,BW      ;CALCULATE TIME DIFFERENCE
        SUBC DW,001C    ;SUBTRACT TIME FALILURE OF ROUTINE

                        ;*** STORE AND DISPLAY VALUE ***

        CALL HEXCON5    ;CONVERT HEX TO ASCII

        MOV AW,0500     ;SET TABLE ADDRESS FOR CURSOR MOVE
        MOV BW,FC00
        MOV DS0,BW

        CALL DISPLAY    ;MOVE CURSOR TO DISPLAY ADDRESS

        MOV AW,0409     ;SET MEMORY ADDRESS FOR DISPLAY VALUE
        MOV BW,0000
        MOV DS0,BW

        CALL DISPLAY    ;DISPLAY COUNTER VALUE

        RET             ;END OF SUBROUTINE
```

```
;***********************************************
;                                              :
;     SUBROUTINE FOR HEX TO ASCII CONVERSION   :
;     OF A WORD AND STORAGE IN MAIN MEMORY     :
;                                              :
;***********************************************
;
;THIS SUBROUTINE CONVERTS A HEXADECIMAL WORD INTO
;A NUMBER OF ASCII CHARACTERS.
;THE INPUT VALUE MUST BE STORED IN REGISTER DW
;THE MEMORY AREA FOR BUFFERING THE CHARCTERS IS
;0000.040A .... 040E
;THE LOOP DIVIDES THE NUMBER BY TEN. THE RESULT
;IS STORED IN MEMORY AND THE REMAINDER IN THE
;REGISTER DW. THE REMAINDER IS DIVIDED AGAIN FOR
;THE NEXT CHARACTER.
;
;0000:40A TO 40E  STACK FOR ASCII CHARACTERS
;
      ORG 0130

HEXCON5:

      MOV BW,0000     ;LOAD MEMORY AREA FOR BUFFER
      MOV DS0,BW
      MOV IX,040E

      MOV CW,000A     ;LOAD DIVISOR

HEXASC:

      MOV AW,DW       ;SHIFT CONTENT
      MOV DW,0000     ;CLEAR ACCU FOR RESULT

      DIVU CW         ;DIVIDE BY TEN
      ADD DL,30       ;ADD 30HEX FOR CONVERSION TO ASCII
      MOV [IX],DL     ;SAVE CHARACTER IN MEMORY

      DEC IX          ;SET POINTER FOR NEXT MEMORY CELL
      INC BL          ;INCREMENT LOOP COUNTER
      CMP BL,05       ;TEST FOR END OF LOOP

      BNE HEXASC      ;LOOP UNTIL 5 TIMES

      RET             ;END OF SUBROUTINE
```

```
                ;***********************************************
                ;                                              :
                ;     SUBROUTINE TO ERASE LOWER SCREEN         :
                ;                                              :
                ;***********************************************
                ;
                ;THIS SUBROUTINE ERASES THE SCREEN FROM THE
                ;FOURTH LINE TO BOTTOM LINE
                ;
                ;THE PROCEDURE IS STORED WITH ASCII-CHARACTERS
                ;IN A TABLE (VT102-CODE)
                ;
                ;FC00:0700  TABLE FOR ASCII CHARACTERS
                ;

      ORG 0060

CLSCRN:

      MOV AW,0700     ;SET BASE ADDRESS OF TABLE
      MOV BW,FC00     ;
      CALL DISPLAY    ;SEND CHARACTERS TO TERMINAL

      RET

              ;END OF SUBROUTINE



                ;***********************************************
                ;                                              :
                ;     SUBROUTINE TO REWRITE SCREEN             :
                ;                                              :
                ;***********************************************
                ;
                ;THIS SUBROUTINE ERASES THE LOWER SCREEN AND
                ;REWRITES THE MENUE FOR THIS PART
                ;
                ;AT END OF SUBROUTINE THE PROGRAM JUMPS TO
                ;THE WAIT LOOP FOR INPUT OF SELECTION KEY
                ;

      ORG 0073

MASKREP:

      CALL CLSCRN     ;GOSUB SCREEN ERASE ROUTINE

      MOV AW,05A0     ;LOAD BASE ADDRESS FOR REWRITE PATTERN
      CALL DISPLAY    ;WRITE PATTERN

      BR MAINRET      ;JUMP BACK INTO MAIN PROGRAM

              ;
              ;END OF SUBROUTINE
              ;
```

```
                ;************************************************
                ;                                               :
                ;    SUBROUTINE TO START STOPWATCH              :
                ;                                               :
                ;************************************************
                ;
                ;THIS SUBROUTINE STARTS THE STOPWATCH FUNCTION
                ;FOR BENCHMARK TESTS. IT USES THE COUNT LATCH
                ;FUNCTION OF THE UPD 71054.
                ;

     ORG 0120

STRTCNT:

     DI              ;DISABLE INTERRUPT

     MOV AL,00       ;LOAD LATCH COMMAND FOR COUNTER #0
     OUT 16,AL

     RET             ;END OF SUBROUTINE
```

```
                ;*********************************************
                ;                                            :
                ;     SUBROUTINE TO READ ONE CHARACTER       :
                ;                                            :
                ;*********************************************
                ;
                ;THIS SUBROUTINE WAITS FOR ONE CHARACTER FROM
                ;THE SIO AND ECHOS IT TO THE TERMINAL
                ;
     ORG 0080

INISIOT:

     CALL INISIO        ;INITIALIZE THE SIO FOR TRANSCEIVE ASYNC

TLOOP11:

     IN AL,02           ;TEST STATUS IF INPUT BUFFER FULL
     AND AL,02
     BE TLOOP11         ;WAIT LOOP FOR CHARACTER INPUT

     IN AL,00           ;READ CHARACTER FROM SIO
     MOV DL,AL          ;BUFFER DATA IN REGISTER DL

TLOOP12:

     IN AL,02           ;TEST STATUS IF READY TO SEND
     AND AL,01
     BE TLOOP12

     MOV AL,DL          ;RELOAD CHARACTER
     OUT 00,AL          ;SEND CHARACTER TO TERMINAL

TLOOP13:

     IN AL,02           ;TEST FOR END OF OUTPUT
     AND AL,01
     BE TLOOP13

     EI                 ;ENABLE TIMER INTERRUPT FOR WATCH

     RET                ;END OF SUBROUTINE
```

```
;***********************************************
;                                              :
;    SUBROUTINE TO STORE SEVERAL INPUT         :
;         CHARACTERS FROM TERMINAL             :
;                                              :
;***********************************************
;
;THIS SUBROUTINE READS THE CHARACTERS COMING FROM
;THE TERMINAL VIA SERIAL TRANSFER TO A DATA
;BUFFER IN SEGMENT '0000' AND ECHOS EACH CHARACTER
;TO THE TERMINAL
;
;REGISTERS USED FOR COMMUNICATION TO MAIN
;PROGRAM :
;AW   CARRY POINTER IX
;BW   CARRY SEGMENT DSO
;CL   CARRY THE NUMBER OF CHARACTERS SPACE
;
    ORG 00B0

INISIOT:

    CALL INISIO     ;INITIALIZE SIO FOR TRANSCEIVE

    MOV DSO,BW      ;LOAD START ADDRESS IN MEMORY AREA
    MOV IX,AW
    EI

CHARINP:

    IN AL,02        ;TEST STATUS IF INPUT BUFFER FULL
    AND AL,02
    BE CHARINP      ;WAIT FOR CHARACTER

    IN AL,00        ;READ CHARACTER FROM SIO
    CMP AL,0D       ;TEST FOR CARRIAGE RETURN
    BE ZERLOAD      ;JUMP OUT OF THE LOOP IF RETURN WAS SENT

    DEC CL          ;COUNT DOWN FOR REST SPACE
    CMP CL,00
    BE ZERLOAD      ;JUMP IF NO SPACE LEFT

CHASAVE:

    MOV [IX],AL     ;SAVE CHARACTER IN MEMORY
    MOV DL,AL       ;BUFFER CHARACTER FOR ECHO
    DI

TLOOP22:

    IN AL,02        ;TEST STATUS IF READY TO ECHO CHARACTER
    AND AL,01
    BE TLOOP22

    MOV AL,DL       ;RELOAD CHARACTER
    OUT 00,AL       ;ECHO CHARACTER
```

```
TREC:

     IN AL,02          ;TEST IF SIGN WAS RECEIVED
     AND AL,01
     BE TREC

     EI                ;ENABLE INTERRUPT

     INC IX            ;SELECT NEXT MEMORY LOCATION
     BR CHARINP        ;LOOP

ZERLOAD:

     MOV AL,00         ;LOAD A ZERO FOR DETECTION OF LAST MEMORY
                       ;LOCATION AT READ OPERATION
     MOV [IX],AL


     RET               ;END OF SUBROUTINE
```

```
                ;***********************************************
                ;                                              :
                ;     SUBROUTINE FOR PRINTING CHARACTERS       :
                ;                                              :
                ;***********************************************
                ;
                ;THIS SUBROUTINE SENDS CHARACTERS TO THE
                ;TERMINAL FOR PRINTING ON SCREEN AND CURSOR
                ;CONTROL
                ;
                ;AW IS INPUT PARAMETER FOR IX
                ;BW IS INPUT PARAMETER FOR DS0
                ;

     ORG 0100

DISPLAY:

     MOV IX,AW          ;LOAD OFFSET ADDRESS FOR TABLE
     MOV DS0,BW

TLOOP31:

     IN AL,02           ;TEST STATUS FOR READY TO SEND
     AND AL,01
     BE TLOOP31

     MOV AL,[IX]        ;READ CHARACTER OUT OF THE TABLE
     OUT 00,AL

     INC IX             ;SET NEXT TABLE ADDRESS
     CMP AL,00          ;TEST FOR END
     BNE TLOOP31        ;IF NOT END THEN LOOP

TLOOP32:

     IN AL,02           ;TEST IF LAST TRANSFER FINISHED
     AND AL,01
     BE TLOOP32


     RET                ;END OF SUBROUTINE
```

```
                ;***********************************************
                ;                                              :
                ;    SUBROUTINE FOR SETTING INTERRUPT VEKTOR    :
                ;       ADDRESSES IN SPECIFIED LOCATIONS       :
                ;                                              :
                ;***********************************************
                ;
                ;THIS SUBROUTINE SETS THE VECTOR ADDRESSES FOR
                ;THE USED INTERRUPTS BY COPYING THE ADDRESS DATA
                ;FROM THE PROGRAM SEGMENT BY A BLOCK MOVE
                ;
                ;THE TRANSFER IS DONE IN TWO LOOPS:
                ;FIRST FOR SOFTWARE INTERRUPT VECTORS #20 ... 26
                ;THEN FOR HARDWARE INTERRUPT VECTOR #8
                ;
                ;0000:0080  VECTOR FIELD
                ;FC00:01C0  TABLE INTERRUPT VECTORS
                ;
        ORG 0180

INIVECT:

        MOV AL,00
        MOV CL,00
        MOV BW,0000     ;LOAD DESTINATION SEGMENT
        MOV DS1,BW
        MOV BW,FC00     ;LOAD SOURCE SEGMENT
        MOV DS0,BW
        MOV BW,0080     ;LOAD DESTINATION OFFSET
        MOV IY,BW
        MOV BW,01C0     ;LOAD SOURCE OFFSET
        MOV IX,BW

LTRANS:

        MOVBKW          ;MOVE DATA WORDWISE

        INC CL          ;COUNT
        CMP CL,0E       ;TEST FOR END OF LOOP
        BNE LTRANS

        CMP AL,01       ;TEST FOR SECOND BASE ADDRESS
        BE LTREND       ;ALL DATA COPIED

        MOV AL,01       ;SET SOFT FLAG FOR SECOND BASE
        MOV CL,0C       ;PRESET LOOP COUNTER FOR SECOND BASE
        MOV BW,0020     ;LOAD NEW DESTINATION BASE
        MOV IY,BW
        BR LTRANS       ;JUMP BACK FOR SECOND LOOP

LTREND:

        RET             ;END OF SUBROUTINE
```

```
                ;***********************************************
                ;                                              :
                ;     SUBROUTINE TO RESET BUFFER AREA          :
                ;                                              :
                ;***********************************************
                ;
                ;THIS SUBROUTINE CLEARS ALL MEMORY CELLS USED FOR
                ;BUFFERING THE CHARACTERS IN INPUT/OUTPUT
                ;OPERATIONS
                ;
                ;0000:402 TO 0700  BUFFER AREA IN MEMORY
                ;
     ORG 0200

BUFRES:

     MOV AW,0000     ;SET START ADDRESS
     MOV DS0,AW
     MOV IX,0402

REPLOOP:

     MOV [IX],AW     ;WRITE A ZERO TO THE LOCATION SELECTED
     INC IX          ;SELECT NEXT ADDRESS
     CMP IX,0700     ;TEST FOR END
     BNE REPLOOP     ;LOOP IF NOT END

     RET             ;END OF SUBROUTINE
```

```
                ;*********************************************
                ;                                            :
                ;    SUBROUTINE TO INITIALIZE TIMER/COUNTER  :
                ;                                            :
                ;*********************************************
                ;
                ;THIS SUBROUTINE INITIALIZES THE TIMER/COUNTER
                ;UNIT 71054.
                ;COUNTER 0 COUNTS AN EXTERNAL CLOCK
                ;TO GENERATE AN INTERRUPT SIGNAL EVERY 50MSEC
                ;(WATCH CLOCK)
                ;COUNTER 2 GENERATES THE BAUD RATE FOR THE
                ;SERIAL INTERFACE
                ;

        ORG 0220

INICOUN:

        MOV AL,B6       ;INITIALIZE BAUD RATE 9600
        OUT 16,AL
        MOV AL,08       ;SET LOW BYTE
        OUT 14,AL
        MOV AL,00       ;SET HIGH BYTE

        MOV AL,36       ;INITIALIZE 50MSEC CLOCK
        OUT 16,AL
        MOV AL,3C       ;SET LOW BYTE
        OUT 10,AL
        MOV AL,F0       ;SET HIGH BYTE
        OUT 10,AL

        RET             ;END OF SUBROUTINE
```

```
;************************************************
;                                                :
;     SUBROUTINE TO INITIALIZE THE SIO FOR       :
;          TRANSMIT ONLY                         :
;                                                :
;************************************************
;
;THIS SUBROUTINE INITIALIZES THE SERIAL CONTROL
;UNIT FOR A TRANSMIT IN ASYNCHRONOUS MODE WITH
;8BIT, 2 STOPBITS AND NO PARITY
;
     ORG 0250

INISIOT:

     MOV AL,00       ;START SOFTWARE RESET ROUTINE
     OUT 02,AL
     OUT 02,AL
     OUT 02,AL
     MOV AL,40
     OUT 02,AL

     MOV AL,CE       ;SET WORD FOR ASYNCH MODE
     OUT 02,AL

     MOV AL,33       ;SET TRANSMIT COMMAND
     OUT 02,AL

     RET             ;END OF SUBROUTINE
```

```
;*************************************************
;                                                 :
;     SUBROUTINE TO INITIALIZE THE SIO FOR        :
;          TRANSMIT AND RECEIVE                   :
;                                                 :
;*************************************************
;
;THIS SUBROUTINE INITIALIZES THE SIO FOR TRANSMIT
;AND RECEIVE TO ENABLE KEYBOARD INPUTS WITH ECHO
;
      ORG 0280

INISIO:

      MOV AL,00       ;START SOFTWARE RESET
      OUT 02,AL
      OUT 02,AL
      OUT 02,AL
      MOV AL,40
      OUT 02,AL

      MOV AL,CE       ;SET ASYNCH MODE
      OUT 02,AL

      MOV AL,37       ;SET TRANSMIT AND RECEIVE
      OUT 02,AL


      RET             ;END OF SUBROUTINE
```

```
;************************************************
;                                               :
;     SUBROUTINE FOR CENTRONICS INTERFACING     :
;                                               :
;************************************************
;
;THIS SUBROUTINE INITIALIZES THE PARALLEL INTER-
;FACE 71055 AS CENTRONICS INTERFACE AND PRINTS
;CHARACTERS STORED IN A MEMORY AREA DEFINED BY
;THE REGISTERS AW AND BW . THE END OF TRANSFER
;IS DETECTED BY READING A '00' AS MEMORY CONTENT.
;
      ORG 0300

INICENT:

      MOV IX,AW       ;LOAD BASE ADDRESS OF MEMORY AREA
      MOV DS0,BW

      MOV AL,81       ;INITIALIZE THE DEVICE
      OUT 26,AL
      MOV AL,FF
      OUT 24,AL

CENTOUT:

      IN AL,24        ;TEST STATUS IF READY TO SEND
      TEST1 AL,01
      BNE CENTOUT

      MOV AL,[IX]     ;READ CHARACTER FROM BUFFER
      CMP AL,00       ;TEST IF LAST CHARACTER
      BE CENTEND      ;JUMP TO RETURN OPERATION IF END

      OUT 20,AL       ;SEND CHARACTER TO THE PRINTER
      CALL WLCENT     ;WAIT

      MOV AL,00       ;SET HANDSHAKE
      OUT 24,AL
      CALL WLCENT     ;WAIT

      MOV AL,FF       ;RESET HANDSHAKE
      OUT 24,AL

      INC IX          ;INCREMENT ADDRESS FOR NEXT CHARACTER

      BR CENTLOOP     ;LOOP

CENTEND:

      RET             ;END OF SUBROUTINE

                      ;SUBMODUL FOR WAITLOOP/PRINTER
WLCENT:

      MOV CL,02
```

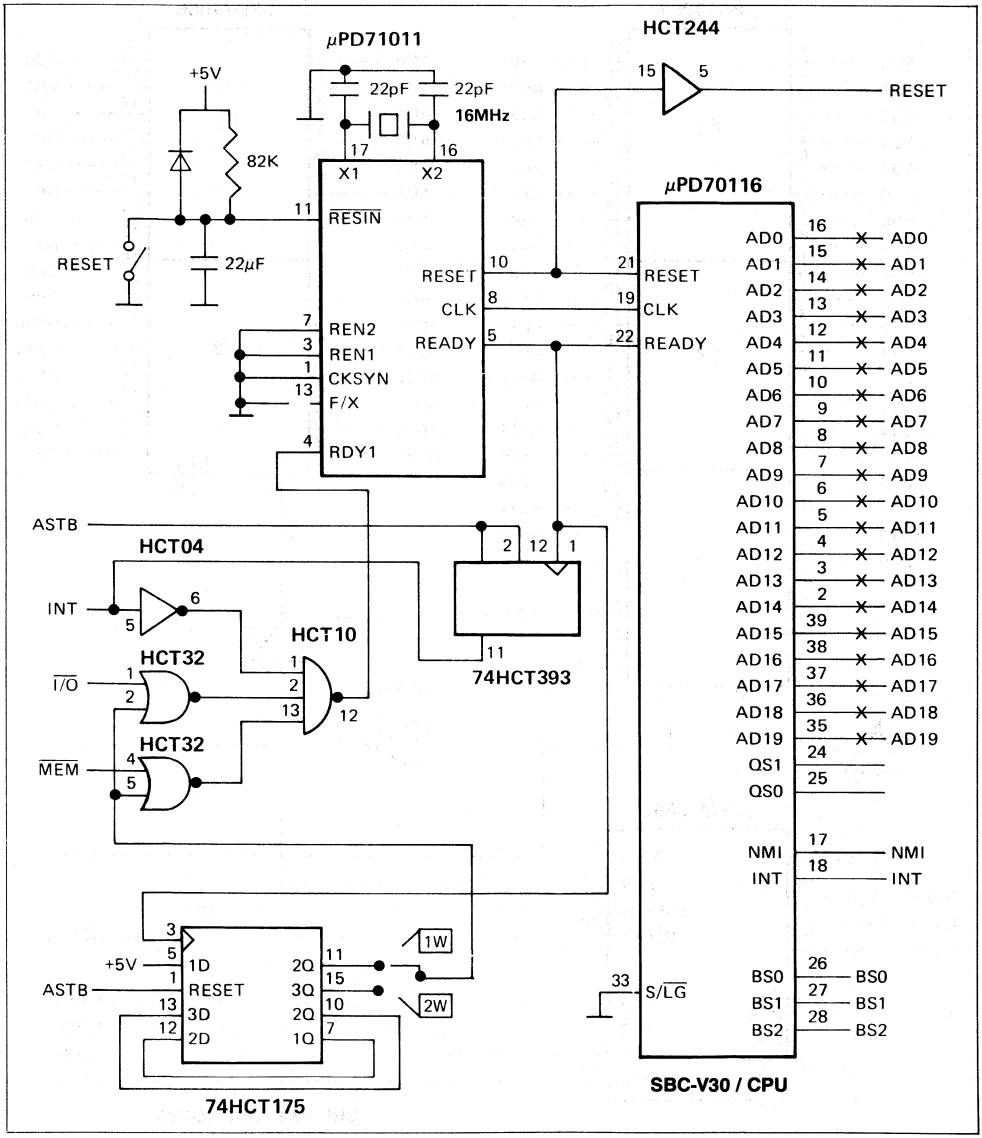
```
TLOOPC:

        DEC C
        BNE TLOOPC
        RET


                ;***********************************************
                ;                                              :
                ;     INITIALIZATION OF INTERRUPT              :
                ;                                              :
                ;***********************************************
                ;
                ;THIS SUBROUTINE INITIALIZES THE INTERRUPT
                ;CONTROL UNIT FOR A SINGLE VECTOR INTERRUPT FROM
                ;THE TIMER/COUNTER
                ;

        ORG 0360

ININT:
        MOV AL,13       ;SET IW1
        OUT 30,AL

        MOV AL,08       ;SET IW3
        OUT 32,AL

        MOV AL,03       ;SET IW4
        OUT 32,AL

        MOV AL,FE       ;SET INTERRUPT MASK WORD
        OUT 32,AL

        MOV AL,20       ;SET PRIORITY AND FINISH CONTROL WORD
        OUT 30,AL

        RET             ;END OF SUBROUTINE
```

```
;***********************************************
;                                              :
;     INTERRUPT SERVICE ROUTINE 'WATCH'        :
;                                              :
;***********************************************
;
;THIS ROUTINE GENERATES AN INTERRUPT DRIVEN WATCH
;
;AN INTERRUPT TRIGGERS A SOFTWARE COUNTER.
;THE CONTENT IS CONVERTED TO A NORMAL COUNTING OF
;A WATCH AND STORED IN MEMORY AT EACH SERVICE.
;IF THERE IS A CARRY FOR THE SECONDS, THE WATCH
;IS DISPLAYED ON THE TERMINAL.
;
;0000:0410  MEMORY WATCH COUNTER
;

     ORG 0400

WATCH:

     DI              ;DISABLE A SECOND INTERRUPT
     PUSH DS0        ;SAVE DS0 CONTENT
     PUSH R          ;SAVE ALL MAIN REGISTERS

     MOV BW,0000     ;LOAD COUNTER BASE ADDRESS OF WATCH
     MOV DS0,BW
     MOV IX,0410

     MOV AW,[IX]     ;LOAD PREVIOUS VALUES FOR COUNTS AND SECONDS
     INC IX
     INC IX
     MOV CW,[IX]     ;LOAD MINUTES AND HOURS

     INC AL          ;ADD ONE TO PREVIOUS COUNT

     CMP AL,14       ;TEST FOR CARRY 'ONE SECOND'
     BNE NOCARRY     ;NO CARRY

     MOV DW,0001     ;SET SOFTWARE FLAG FOR PRINTOUT

     MOV AL,00       ;RESET COUNTS
     INC AH          ;ADD ONE SECOND
     CMP AH,3C       ;TEST FOR 'ONE MINUTE'
     BNE NOCARRY     ;NO CARRY

     MOV AH,00       ;RESET SECONDS
     INC CL          ;ADD ONE MINUTE
     CMP CL,3C       ;TEST FOR 'ONE HOUR'
     BNE NOCARRY     ;NO CARRY

     MOV CL,00       ;RESET MINUTES
     INC CH          ;ADD ONE HOUR
     CMP CH,18       ;TEST FOR 1 DAY
     BNE NOCARRY     ;NO CARRY

     MOV AW,0000     ;RESET ALL FOR 1 DAY
     MOV CW,0000
```

```
NOCARRY:                ;**** SAVE COUNT VALUES IN MEMORY *****

        MOV BW,0410     ;RELOAD MEMORY ADDRESS
        MOV IX,BW
        MOV [IX],AW     ;SAVE COUNTER VALUES
        INC IX
        INC IX
        MOV [IX],CW

        TEST1 DL,01     ;TEST FOR PRINTOUT ROUTINE
        BE NODISP       ;JUMP TO END WITHOUT DISPLAY

                        ;**** PRINTOUT ROUTINE *****

        MOV IX,004C     ;LOAD BASE ADDRESS FOR CURSOR MOVE TABLE
        MOV AW,FC00
        MOV DS0,AW

                        ;**** MOVE THE CURSOR TO START POSITION ****

CURSOR:

        IN AL,02        ;TEST STATUS LINES IF READY FOR OUTPUT
        AND AL,01
        BE CURSOR

        MOV AL,[IX]     ;LOAD CHARACTER
        OUT 00,AL

        INC IX          ;LOAD NEXT ADDRESS
        CMP AL,00       ;TEST FOR LAST CHARACTER
        BNE CURSOR      ;LOOP


                        ;**** DISPLAY TIME ****

        MOV AW,0000     ;RELOAD BASE ADDRESS OF WATCH MEMORY
        MOV DS0,AW
        MOV IX,0413     ;LOAD HIGHEST BYTE FIRST
        MOV CL,0A       ;LOAD DIVISION FACTOR

                        ;**** HEX TO ASCII CONVERSION ****

HEXASC:

        MOV AW,0000     ;RESET ACCU FOR DIVISION
        MOV AL,[IX]     ;LOAD HEX VALUE BYTE
        DIV CL          ;DIVIDE BY 10 TO GET TWO CHARACTERS
        ADD AL,30       ;ADD 30H TO GET AN ASCII CHARACTER
        ADD AH,30       ;DO THE SAME FOR SECOND CHARACTER

        MOV BW,AW       ;BUFFER ASCII CHARACTERS BEFORE PRINT

        CALL WLOOP0     ;TEST STATUS LINES IF READY FOR OUTPUT
        MOV AL,BL       ;LOAD HIGH CHARACTER
        OUT 00,AL       ;SEND
```

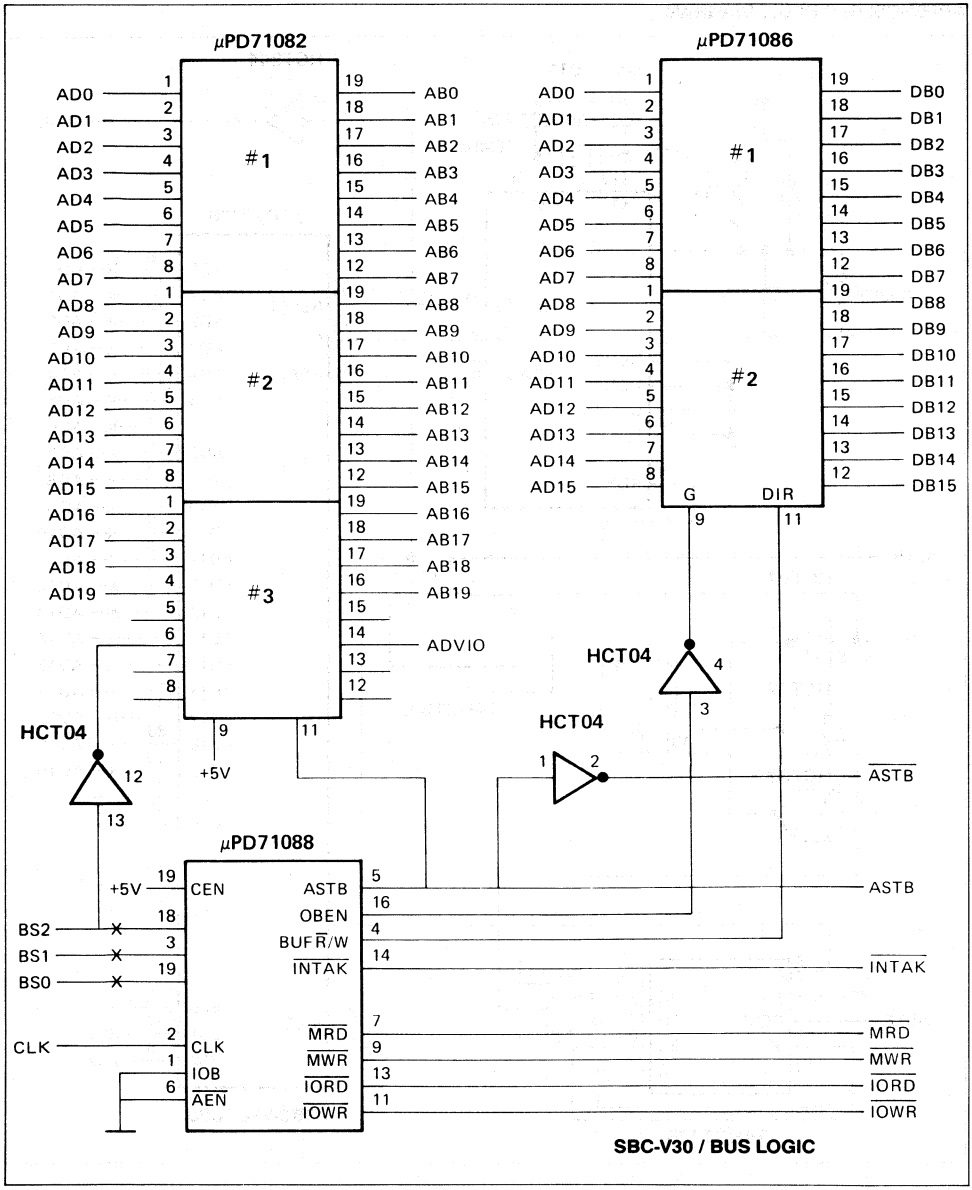
```
        CALL WLOOP0     ;TEST STATUS FOR NEXT CHARACTER
        MOV AL,BH       ;LOAD NEXT CHARACTER
        OUT 00,AL       ;SEND
        CALL WLOOP0     ;TEST IF FINISHED
        DEC IX          ;SET NEXT BYTE ADDRESS
        CMP IX,0410     ;LOOK IF END OF LOOP
        BE NODISP

        MOV AL,3A       ;SET LIMITING CHARACTER (DOUBLE POINT)
        OUT 00,AL
        CALL WLOOP0

        BR HEXASC       ;LOOP FOR NEXT CHARACTERS

NODISP:

        POP R           ;RELOAD REGISTERS
        POP DS0
        EI              ;ALLOW INTERRUPT AGAIN

        RETI            ;END OF SUBROUTINE


WLOOP0:

        IN AL,02        ;WAIT LOOP IF LINE IS NOT READY
        AND AL,01
        BE WLOOP0

        RET
```

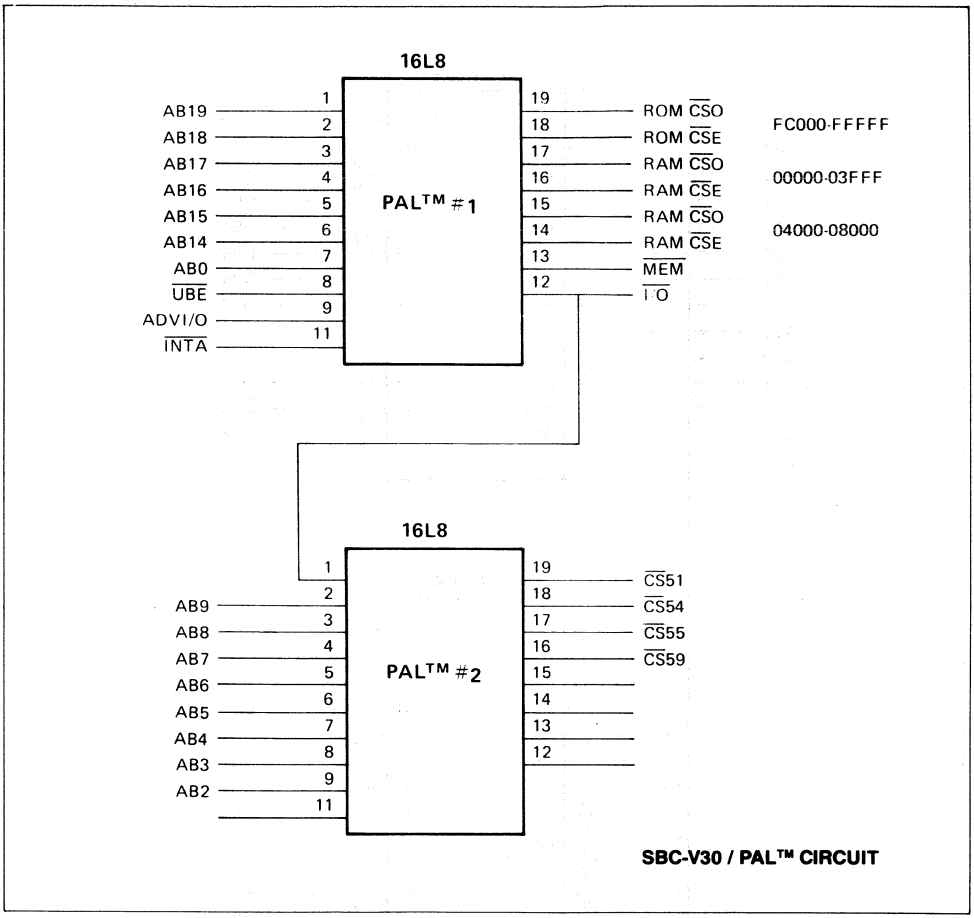
APPENDIX B: CIRCUIT DIAGRAM



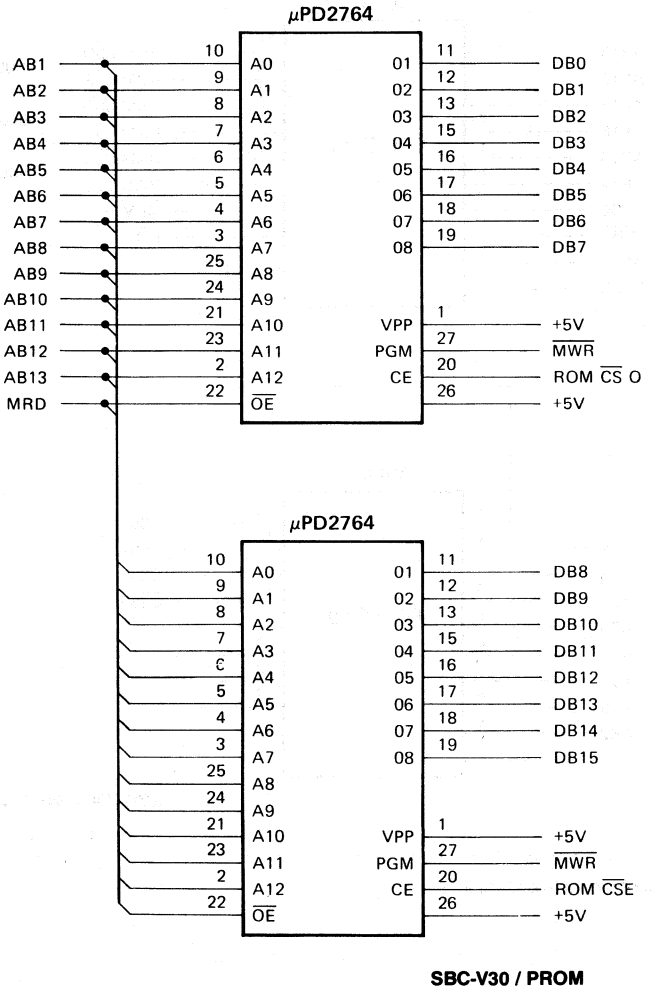
Note 1: X = 4.7 kOhm pull up resistance

Note 2: 0.1 .F disc capacitor between +5 V & GND on all devices

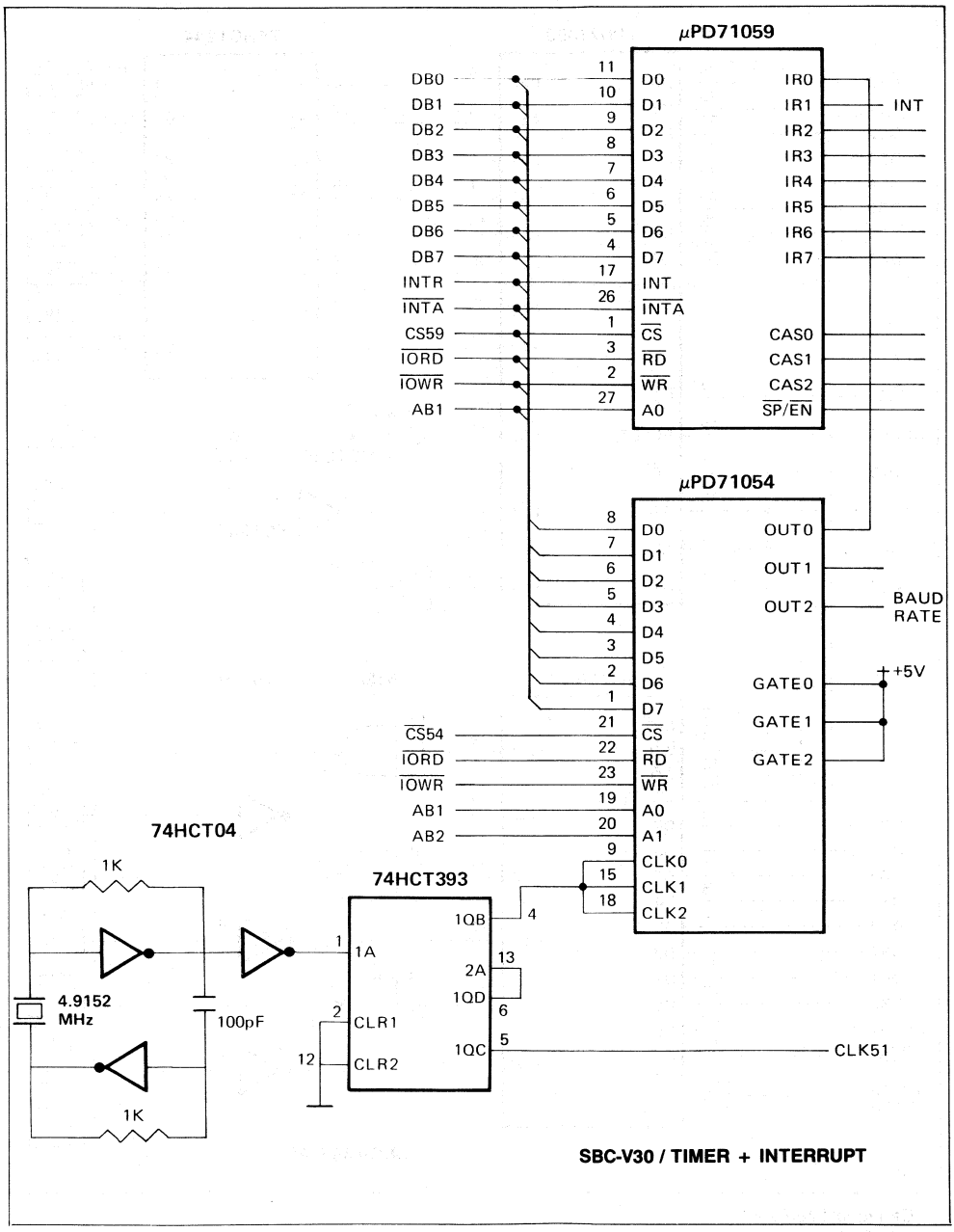

μPD71082
#1
#2
#3
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
AD8
AD9
AD10
AD11
AD12
AD13
AD14
AD15
AD16
AD17
AD18
AD19
AB0
AB1
AB2
AB3
AB4
AB5
AB6
AB7
AB8
AB9
AB10
AB11
AB12
AB13
AB14
AB15
AB16
AB17
AB18
AB19
ADVIO
μPD71086
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
DB8
DB9
DB10
DB11
DB12
DB13
DB14
DB15
G
DIR
HCT04
+5V
ASTB
μPD71088
CEN
OBEN
BUF R/W
INTAK
BS2
BS1
BS0
CLK
IOB
AEN
MRD
MWR
IORD
IOWR
SBC-V30 / BUS LOGIC



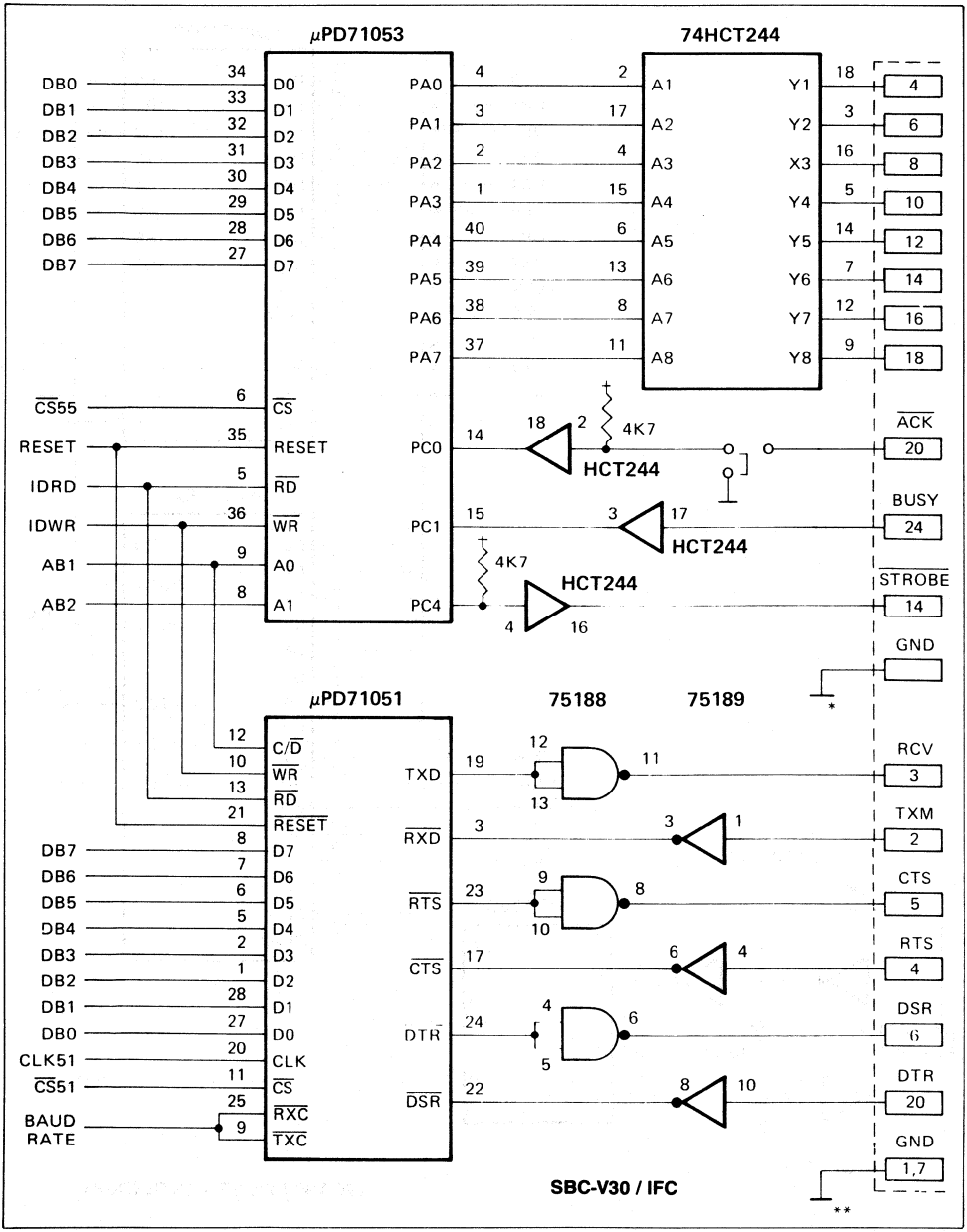
PAL 198 trademark of Monolithic Memories Inc.


μPD2764
AB1
AB2
AB3
AB4
AB5
AB6
AB7
AB8
AB9
AB10
AB11
AB12
AB13
MRD
10
9
8
7
6
5
4
3
25
24
21
23
2
22
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
OE
01
02
03
04
05
06
07
08
VPP
PGM
CE
11
12
13
15
16
17
18
19
1
27
20
26
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
+5V
MWR
ROM CS O
+5V
μPD2764
DB8
DB9
DB10
DB11
DB12
DB13
DB14
DB15
+5V
MWR
ROM CSE
+5V

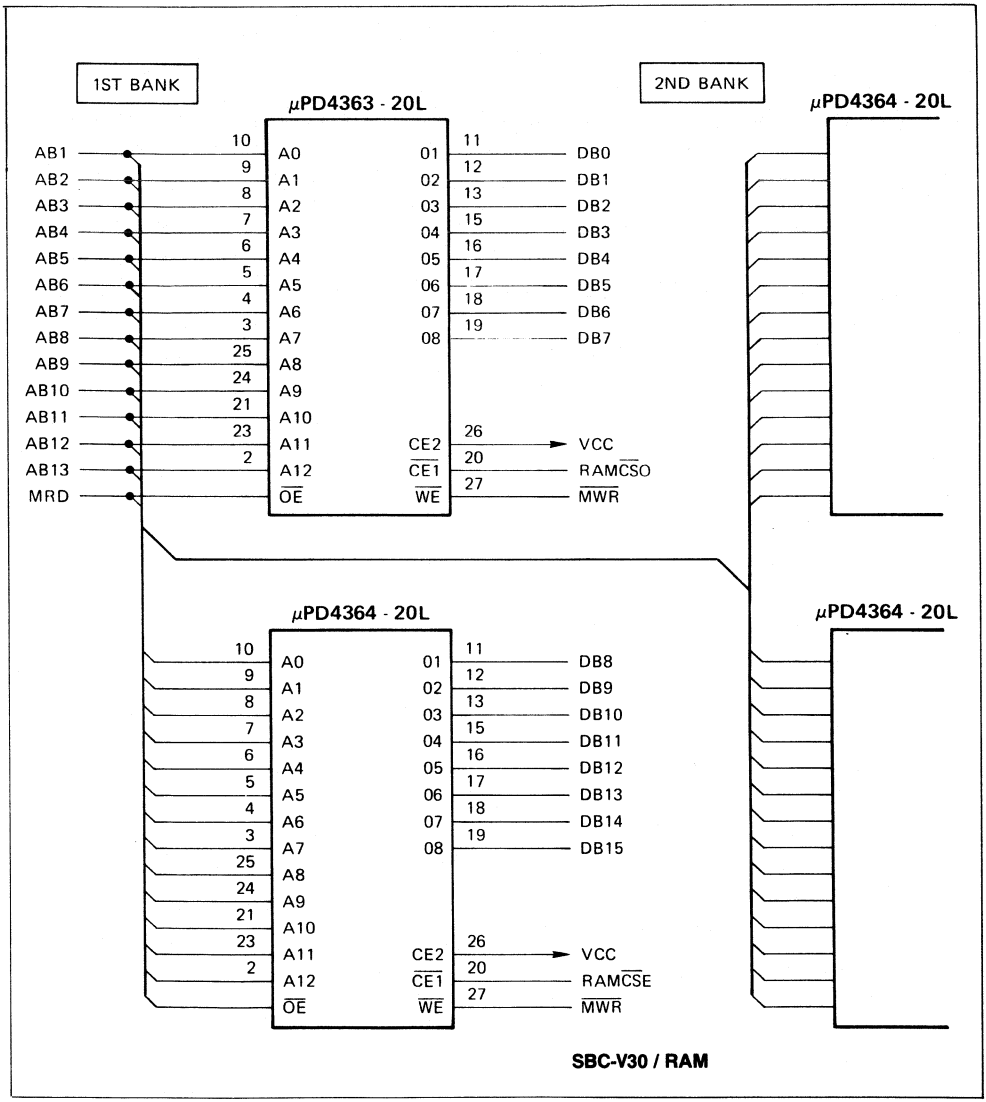
SBC-V30 / PROM

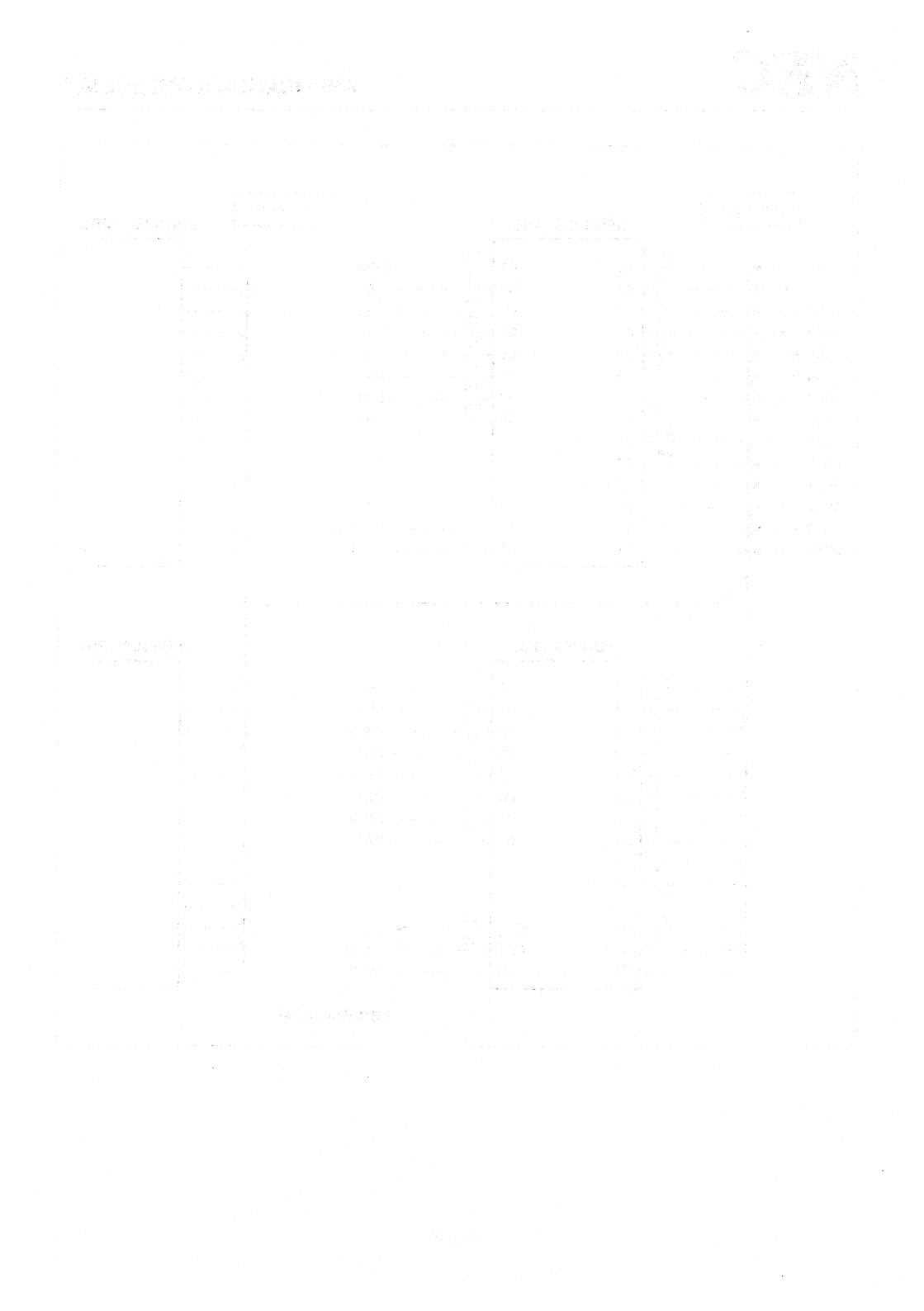


SBC-V30 / TIMER + INTERRUPT



* Centronix Interface
** RS-232C Interface


1ST BANK
μPD4363 - 20L
2ND BANK
μPD4364 - 20L
AB1
AB2
AB3
AB4
AB5
AB6
AB7
AB8
AB9
AB10
AB11
AB12
AB13
MRD
10 A0
9 A1
8 A2
7 A3
6 A4
5 A5
4 A6
3 A7
25 A8
24 A9
21 A10
23 A11
2 A12
OE
01 DB0
02 DB1
03 DB2
04 DB3
05 DB4
06 DB5
07 DB6
08 DB7
CE2 26 VCC
CE1 20 RAMCSO
WE 27 MWR
μPD4364 - 20L
μPD4364 - 20L
01 DB8
02 DB9
03 DB10
04 DB11
05 DB12
06 DB13
07 DB14
08 DB15
CE2 26 VCC
CE1 20 RAMCSE
WE 27 MWR
SBC-V30 / RAM



Interrupt Handling in the μPD8080 Emulation Mode of NEC's 16-Bit CMOS V-Series Standard Microprocessors μPD70108, μPD70116, μPD70208, and μPD70216

Contents:

Authors: T. Wadhawan
Product Marketing - μCOM
NEC Electronics (Europe) GmbH

Related Documentation

μPD70108/70116	User's Manual
μPD70208/70216	User's Manual
μPD70108	Product Description
μPD70116	Product Description
μPD70208	Product Description
μPD70216	Product Description

Related Products

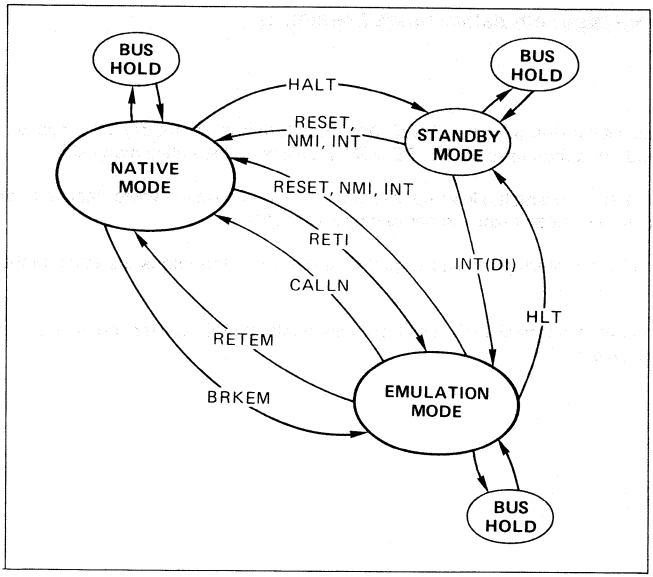
μPD70108	16-bit microprocessor	CMOS
μPD70116	16-bit microprocessor	CMOS
μPD70208	16-bit microprocessor	CMOS
μPD70216	16-bit microprocessor	CMOS

1. Introduction

The V20/V30 microprocessors operate in two modes; the native mode and the emulation mode.

The emulation mode allows programs written for μPD8080 to run on V20/V30 (16-bit environment).

2. V20 - V50 emulation mode



The microprocessors are found in the native mode after power on.

The V20/V30 can enter the emulation mode by a break into the emulation (BRKEM) instruction. It can return to the native Mode by (RETEM) instruction.

At any time the MD bit of the PSW (see figure) identifies whether the microprocessor is in antive mode or emulation mode (MD = 1 — native mode, MD = 0 — emulation mode).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MD	1	1	1	V	DIR	IE	BRK	S	Z	0	AC	0	P	1	CP

PSW

The microprocessor will always find itself in the native mode after an interrupt has been accepted (and will be primed for execution in the native mode) regardless of the mode before the interrupt; (i.e.) MD flag in PSW is always set after an interrupt, (MD = 1).

Interrupts which occur during emulation mode can be handled in two ways:

1. Interrupt routine executed in native mode with a return back to emulation mode after the routine has been finished (version 1).

2. Interrupt routine executed in emulation mode (version 2).

This application note will explain the procedure of handling interrupts when the interrupt occurs in the emulation mode.

3. Interrupt handling in emulation mode (version 1)

Summary

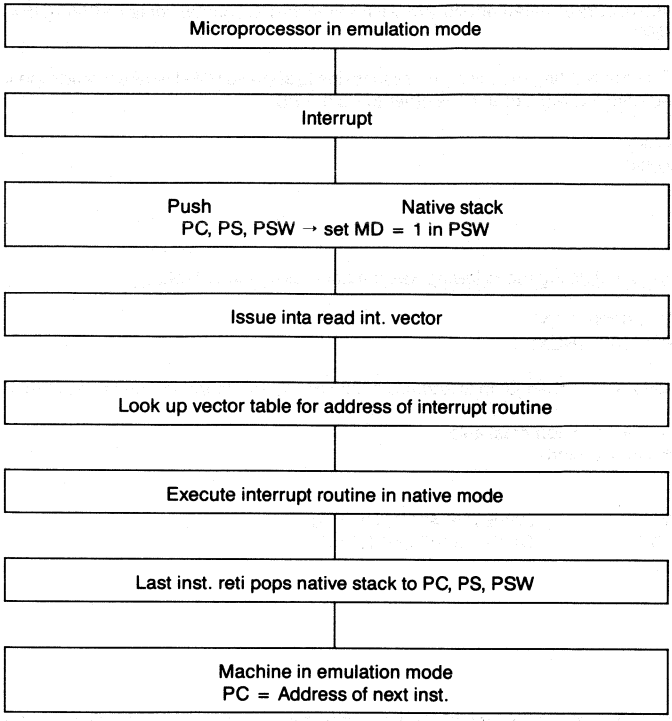
The microprocessor, as it is executing the μPD8080 microcode (emulation mode) will, on occurance of an interrupt, go to the native mode and then save the PC, PS, PSW registers on the native mode stack.

After issuing the inta and reading the interrupt vector it will then access the vector table and execute the interrupt routine (which should be written using the native mode instructions).

The last instruction in the interrupt routine should be a reti which then pops the stack to the PC, PS and PSW registers.

This then brings the processor back to the emulation mode with the PC register containing the address of the next instruction to be executed.

3A Flow chart



3B Source code example

```
                    268 ; COMMAND RECOGNIZER ROUTINE
                    269 ;
                    270 ;
87D  CD1F06         271 GETCMD      CALL    GETCH       ;GET CHR
880  CDF805         272             CALL    ECHO        ;ECHO IT
883  79             273             MOV     A, C        ;SETUP FOR COM
-------------------------------------------------------------------------
884  FE52           274             CPI     'R'         ;R?
886  CAAF08         275             JZ      RDHN        ;GET MORE
889  FE53           276             CPI     'S'         ;S?
88B  CAD708         277             JZ      SDHN        ;GET MORE
88E  FE47           278             CPI     'G'         ;G?
-------------------------------------------------------------------------
890  CAFF08         279             JZ      GDWN        ;GET MORE
893  FE54           280             CPI     'T'         ;T?
895  CADE09         281             JZ      TDWN        ;GET MORE
898  FE41           282             CPI     'A'         ;A?
89A  CA2209         283             JZ      ADHN        ;GET MORE
89D  FE5A           284             CPI     'Z'         ;Z?
89F  CA3109         285             JZ      CMODE       ;YES, GO CHANGE MODE
8A2  FE03           290             CPI     CNTLO       ;CNTL-C?
8A4  CA0800         291             JZ      MONTR       ;EXIT TO MONITOR
```

Assume that a NMI occurs at statement No. 275. At this time the PC counter contains 0889 (HEX) and the PS counter contain 1000 (HEX).

INTA signal, INT vector reading the microprocessor skips to address 008H where it reads the 4 bytes. Two of these bytes are popped onto PC register and the other to wo the PS.

PC — (008H, 009H)
PS — (00AH, 00BH)

Note

Assume that during initializing the following was the addressed vector table

ADDR.008, 009 = 0900 (HEX)
ADDR.00A, 00B = 1000 (HEX)

The microprocessor would then jump to address 10900 (HEX) and there find an interrupt routine.

```
; This routine handles the non maskable
; Interrupts in the native mode

NMI:
    PUSH AW          ; SAVE THE AW REGISTER
    MOV AW, DW       ; GET A 16-BIT OPERAND
    .... ....
    .... ....
      .
      .
      .
    RETI
```

When RETI is executed, the native mode stack is popped to the PC, PS, PSW registers. The program control is transferred to the emulation mode.

This brings the microprocessor back to 10889 (HEX) and the processor starts execution from 1889.

4. Interrupt handling in emulation mode (version II)

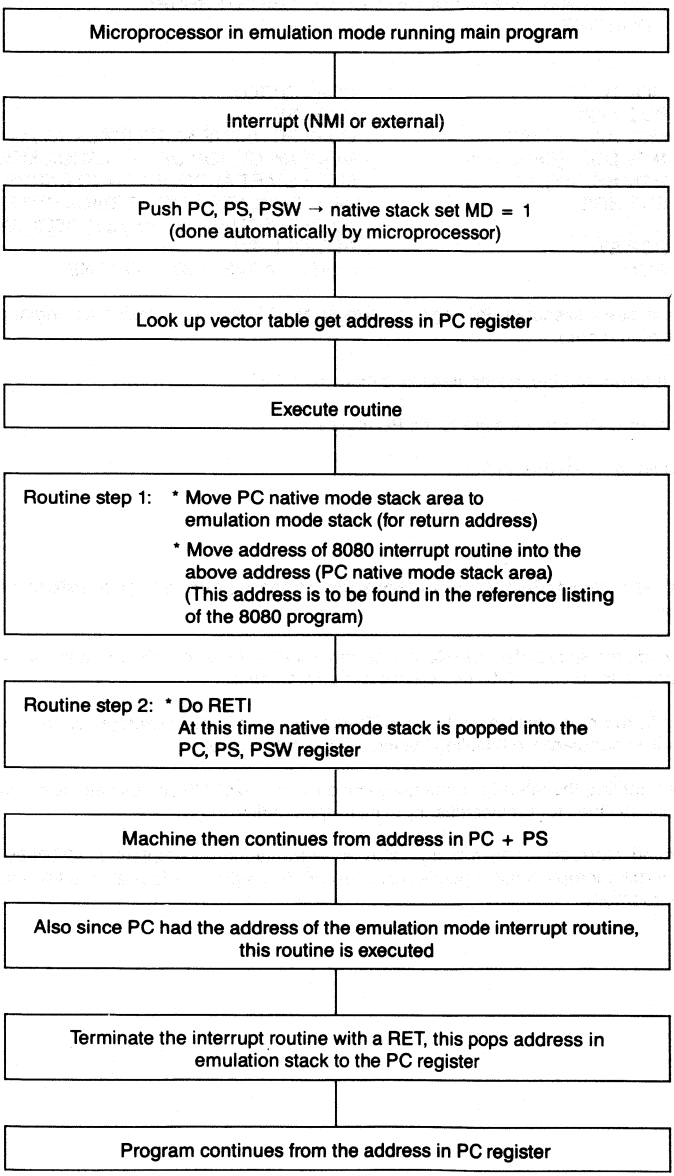
Summary

The microprocessor will go to native mode when an interrupt occurs. In the native mode a small routine needs to be written which copies the next executable address saved in the emulation mode stack (which is addressed by the BP register) and the microprocessor is brought back to the address in the emulation mode where the interrupt handling routine is located.

The routine is terminated with a ret. Instruction which brings the emulation stack contents back to the PC register.

The main program then continues its execution from there on.

4A Flow chart



4B Source code example

```
        ; IX CONTAINS START ADDRESS (M) OF 8080 INTERRUPT
        ; ROUTINE

INT. NAT:
        MOV IY, SP              ; COPY SP TO IY
        PUSH AW                 ; SAVE AW
        MOV AW, SS : [IY]       ; COPY PC NATIVE MODE STACK TO AW
        MOV DS0 : [BP]-2, AW    ; MOVE AW ON TOP OF EMULATION MODE STACK
        MOV SS : [IY], IX       ; MOVE START ADDRESS ON TO NATIVE MODE STACK
        SUB BP,2                ; BP = EMULATION MODE STACK PTR HAS TO POINT
                                  TO THE RETURN ADDRESS IN 8080 MAIN PROGRAM
        POP SW                  ; RESTORE AW
        RETI                    ; FINISH NATIVE MODE ROUTINE
```

After this RETI has been executed, PC register contains M, PS register contains its original value, PSW has MD-Bit = 0 (emulation mode).

In the emulation interrupt routine the termination is made by a RET.

This then pops the emulation mode stack to the PC register.

Main program continues from this address.

5. Comparison and advantages/disadvantages of the above two procedures of handling the interrupt

In the emulation mode the speed of the μPD8080 program would be faster on V20/V30 and it allows an easy way to operate 8-bit programs to run in a 16-bit microprocessor environment.

Interrupt handling in the emulation mode for an μPD8080 program is of advantage, as the programming effort required is minimal as compared to handling the interrupt in native mode.

The advantage of handling the interrupt in the native mode is that V20/V30 can use all the serial instructions and enhanced hardware facilities for greater effiency in interrupt handling.

The ability of handling interrupts in native mode is also very advantageous to implement μPD8080 programs so that they can execute 16-bit microprocessor operating system calls (example CP/M86 calls) so that such a program can run in a 16-bit environment.

Easy Arithmetic Software for μPD7809 / μPD7811 / C11 / C14

Contents:

- Binary Addition	32 Bit	+ 32 Bit	-> 32 Bit
- Binary Subtraction	32 Bit	— 32 Bit	-> 32 Bit
- Binary Multiply	16 Bit	x 16 Bit	-> 32 Bit
- Binary Multiply	16 Bit	x 8 Bit	-> 24 Bit
- Binary Division	32 Bit	/ 32 Bit	-> 32 Bit
- Decimal Addition	8 Place	+ 8 Place	-> 8 Place
- Decimal Subtraction	8 Place	— 8 Place	-> 8 Place
- Decimal Multiply	8 Place	x 8 Place	-> 16 Place
- Decimal Division	8 Place	/ 8 Place	-> 8 Place
- Right Shift	N-Digit-Data		
- Left Shift	N-Digit-Data		
- Right Shift	N-Byte-Data		
- Left Shift	N-Byte-Data		
- Transfer	Hex	-> BCD	
- Transfer	BCD	-> Hex	
- Transfer	ASCII	->Hex	
- Transfer	Hex	-> ASCII	
- Data Searching			
- Polynomial Interpolation			

Person to contact: R. Cherrington
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Product Descriptions
μPD7811 / 7809 / 78C11 / C14

Related Products

μPD7809	8-Bit Single chip microcomputer	NMOS
μPD7811 / C11 / C14	8-Bit Single chip microcomputer	CMOS

Introduction:

The 8 bit microcomputers of the μCOM87 Series have many arithmetic instructions. This Application Software makes use of them.

Attached you will find a summary of small, but useful arithmetic application routines. It includes some 8-32 bit software for binary and BCD calculations.

The software is written for the μPD7811 and the μPD7809 CPU. The assembling is done by ASM87 V1.0 absolute cross software. The source can be adapted easily to RA87 (relocatable μCOM87 cross software) standards.

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE    1

                (                                        )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

     1                    ;       ;       ;       ;       ;       ;       ;
     2                    ;*************************************************
     3                    ;*                                               *
     4                    ;*       UCOM87AD  ARITHMETIC  APPLICATION       *
                          *
     5                    ;*                                               *
     6                    ;*************************************************
     7                    ;
     8                    ;
     9                    ;*************************************************
    10                    ;*       POWER ON INITIALIZE ROUTINE             *
    11                    ;*************************************************
    12                    ;
    13 0000 6909                  MVI     A,09H           ;
    14 0002 4DD0                  MOV     MM,A            ; MEMORY MAPPING REG.
    15                                                    ; INITIALIZATION
    16 0004 68FF                  MVI     V,0FFH          ; V-REG
    17 0006 040000                LXI     SP,STACK        ; STACK POINTER
    18                    ;
    19                            EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                          )
E STNO ADRS OBJECT    M    SOURCE STATEMENT

    18                     ;
    19                     ;       ;       ;       ;       ;       ;       ;
    20                     ;
    21                     ; AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    22                     ; A                                                   A
    23                     ; A     **  APPLICATION  PROGRAM  START  **          A
    24                     ; A                                                   A
    25                     ; AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    26                     ;
    27           0020              START:  ORG     20H
    28                     ;
    29                     ;  ++++++++++++++++++++++++++++++++++++++++++++++++++++
    30                     ;  +                                                  +
    31                     ;  +           BINARY  ADDITION                       +
    32                     ;  +             32BIT <- 32BIT + 32BIT               +
    33                     ;  +                                                  +
    34                     ;  +         INPUT : HL <= AUGEND TOP ADDRS           +
    35                     ;  +                 DE <= ADDIT TOP ADDRS            +
    36                     ;  +                                                  +
    37                     ;  +         OUTPUT : RET .. OVERFLOW                 +
    38                     ;  +                  RETS .. NORMAL                  +
    39                     ;  +                                                  +
    40                     ;  ++++++++++++++++++++++++++++++++++++++++++++++++++++
    41                     ;
    42                     BFADD:
    43 0020 482A                    CLC
    44 0022 6B03                    MVI     C,4-1
    45                              ;
    46 0024 2B             BFAD1:   LDAX    H
    47 0025 70D4                    ADCX    D+
    48 0027 3D                      STAX    H+
    49                              ;
    50 0028 53                      DCR     C
    51 0029 FA                      GJMP    BFAD1
    52                              ;
    53 002A 481A                    SKN     CY      ; CHECK / OVERFLOW ?
    54 002C B8                      RET             ; OVERFLOW
    55 002D B9                      RETS            ; NORMAL
    56                              EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

    57                    ;
    58                    ;
    59                    ;  ++++++++++++++++++++++++++++++++++++++++++++++++
    60                    ;  +
    61                    ;  +            BINARY  SUBTRACTION
    62                    ;  +              32BIT <- 32BIT - 32BIT
    63                    ;  +
    64                    ;  +          INPUT : HL <= MINUEND TOP ADDRS
    65                    ;  +                  DE <= SUBTRC TOP ADDRS
    66                    ;  +
    67                    ;  +          OUTPUT : RET .. OVERFLOW
    68                    ;  +                   RETS .. NORMAL
    69                    ;  +
    70                    ;  ++++++++++++++++++++++++++++++++++++++++++++++++
    71                    ;
    72                    BFSUB:
    73 002E 482A                    CLC
    74 0030 6B03                    MVI     C,4-1
    75                              ;
    76 0032 2B            BFSB1:    LDAX    H
    77 0033 70F4                    SBBX    D+
    78 0035 3D                      STAX    H+
    79                              ;
    80 0036 53                      DCR     C
    81 0037 FA                      GJMP    BFSB1
    82                              ;
    83 0038 481A                    SKN     CY          ; CHECK / OVERFLOW ?
    84 003A B8                      RET                 ; BORROW
    85 003B B9                      RETS                ; NORMAL
    86                              EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                    )
E STNO ADRS OBJECT    M   SOURCE STATEMENT

     87                   ;
     88                   ;  ++++++++++++++++++++++++++++++++++++++++++++++
     89                   ;  +                                            +
     90                   ;  +        BINARY  MULTIPLY                    +
     91                   ;  +              32BIT <- 16BIT * 16BIT        +
     92                   ;  +                                            +
     93                   ;  +   MULTIPLICAND .. ( BMLCND+1,BMLCND )      +
     94                   ;  +   MULTIPLIER   .. ( BMLIER+1,BMLIER )      +
     95                   ;  +                                            +
     96                   ;  +   RESULT ..( BRSLT+3,BRSLT+2,..,BRSLT )    +
     97                   ;  +                                            +
     98                   ;  ++++++++++++++++++++++++++++++++++++++++++++++
     99                   ;
    100                   BFMUL:
    101                   ;
    102                   ;  **  RESULT  OCLEAR  **
    103                   ;
    104 003C 404303                CALL    RSTCLR
    105                   ;
    106                   ;  **  MULTIPLICAND  COUNTER SET  **
    107                   ;
    108 003F 6A01                  MVI     B,2-1
    109 0041 3404FF                LXI     H,BRSLT  ; RESULT TOP ADDRS
    110 0044 2402FF                LXI     D,BMLIER ; MULTIPLIER TOP ADDR
    111                   ;
    112                   BFMUL1:
    113 0047 B1                    PUSH    B
    114 0048 B3                    PUSH    H
    115 0049 2C                    LDAX    D+
    116 004A 1B                    MOV     C,A      ;  MULTIPLIER SET
    117 004B B2                    PUSH    D
    118                   ;
    119                   ;  --  16BIT * 8BIT -> 24BIT --
    120                   ;
    121 004C 405600                CALL    BFMULS
    122                   ;
    123 004F A2                    POP     D
    124 0050 A3                    POP     H
    125 0051 32                    INX     H
    126                   ;
    127                   ;  **  CHECK /  MULTIPLY END
    128                   ;
    129 0052 A1                    POP     B
    130 0053 52                    DCR     B
    131 0054 F2                    GJMP    BFMUL1
    132 0055 B8                    RET
    133                            EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

   134                    ;
   135                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
   136                    ; S                                  S
   137                    ; S --  24BIT <- 16BIT * 8BIT --     S
   138                    ; S                                  S
   139                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
   140                    ;
   141                    BFMULS:
   142 0056 2400FF                LXI     D,BMLCND
   143 0059 6A00                  MVI     B,0
   144                                    ;  CY 0CLEAR
   145 005B 6901                  MVI     A,2-1
   146                    ;
   147                    ;  --  8BIT * 8BIT + CY + RESULT
   148                    ;
   149                    MULSS:
   150 005D B0                    PUSH    V
   151 005E 2C                    LDAX    D+
   152 005F 482F                  MUL     C
   153 0061 7042                  EADD    EA,B
   154 0063 2B                    LDAX    H
   155 0064 7041                  EADD    EA,A
   156 0066 09                    MOV     A,EAL
   157 0067 3D                    STAX    H+
   158                            ;
   159 0068 08                    MOV     A,EAH
   160 0069 1A                    MOV     B,A     ; CY STORE !
   161                    ;
   162                    ;  **  CHECK / 16BIT*8BIT  END ? '  **
   163                    ;
   164 006A A0                    POP     V
   165 006B 51                    DCR     A
   166 006C F0                    GJMP    MULSS
   167                            ;
   168 006D 0A                    MOV     A,B
   169 006E 3B                    STAX    H
   170 006F B8                    RET
   171                            EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE     6

                (                                        )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

    172                    ;
    173                    ;  ++++++++++++++++++++++++++++++++++++++++++++++++
    174                    ;  +                                              +
    175                    ;  +           BINARY  DIVISION                   +
    176                    ;  +             32BIT <- 32BIT / 32BIT           +
    177                    ;  +                                              +
    178                    ;  +                                              +
    179                    ;  +  DIVIDEND .. ( DEND+3,DEND+2,...,DEND )      +
    180                    ;  +  SOR      .. ( SOR+3,SOR+2,...,SOR )         +
    181                    ;  +  QUOTIENT .. ( DEND+3,DEND+2,...,DEND )      +
    182                    ;  +  REMIND   .. ( BRMND+3,BRMND+2,...,BRMND )   +
    183                    ;  +                                              +
    184                    ;  ++++++++++++++++++++++++++++++++++++++++++++++++
    185                    ;
    186                     BFDIV:
    187                    ;
    188                    ;  **  CHECK FOR  / DIVISOR=0
    189                    ;
    190 0070 3410FF                LXI     H,SOR
    191 0073 4885                  LDEAX   H++
    192 0075 B6                    DMOV    D,EA
    193 0076 4883                  LDEAX   H
    194 0078 749E                  DOR     EA,D
    195 007A 481C                  SKN     Z
    196 007C B8                    RET                 ; IF  OVERFLOW
    197                    ;
    198                    ;  **  QUOTIENT  0CLEAR  **
    199                    ;
    200 007D 404603                CALL    DIVCLR
    201                    ;
    202                    ;  **  BYTE-COUNTER  SET  **
    203                    ;
    204 0080 6A07                  MVI     B,8-1
    205                    ;
    206                    ;  **  DIVIDEND,RMIND  1BYTE LEFT-SHIFT  **
    207                    ;
    208                    BFDIV2:
    209 0082 3408FF                LXI     H,DEND
    210 0085 6B07                  MVI     C,8-1   ; SHIFT, BYTE-COUNTER SET
    211 0087 403901                CALL    BCDLS   ; SHIFT !
    212                    ;
    213                    ;  **  SUBTRACT DIVISOR  FROM DIVIDEND  **
    214                    ;
    215                    BFDIV3:
    216 008A 340CFF                LXI     H,BRMND
    217 008D 2410FF                LXI     D,SOR
    218                            ;
    219 0090 402E00                CALL    BFSUB
    220 0093 C3                    GJMP    BFDIV4              ; IF BORROW
    221                                                        ; IF NO BORROW
    222 0094 2008                  INRW    DEND MOD 100H       ; 1-DIGIT QUOT (+
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE

                (                                        )

E STNO ADRS OBJECT    M    SOURCE STATEMENT


    223 0096 F3                      GJMP     BFDIV3
    224                         ;
    225                         ;  **  IF BORROW , DIVISOR + DIVIDEND  **
    226                         ;
    227                         BFDIV4:
    228 0097 340CFF                  LXI      H,BRMND
    229 009A 2410FF                  LXI      D,SOR
    230 009D 402000                  CALL     BFADD
    231 00A0 00                      NOP
    232                         ;
    233                         ;  **  BIT COUNTER DECREMENT  **
    234                         ;
    235 00A1 52                      DCR      B          ; SKIP, IF END
    236 00A2 4FDE                    GJMP     BFDIV2
    237                              ;
    238 00A4 B9                      RETS
    239                              EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE


                (                                        )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

    240                   ;
    241                   ;  ++++++++++++++++++++++++++++++++++++++++++++
    242                   ;  +                                          +
    243                   ;  +           DECIMAL  ADDITION              +
    244                   ;  +       8-PLACE <- 8-PLACE + 8-PLACE       +
    245                   ;  +                                          +
    246                   ;  +    AUGEND .. ( HL+3,...,HL )             +
    247                   ;  +    ADDEND .. ( DE+3,...,DE )             +
    248                   ;  +    RESULT .. ( HL+3,...,HL )             +
    249                   ;  +                                          +
    250                   ;  +     OUTPUT : RET .. OVERFLOW             +
    251                   ;  +              RETS .. NORMAL              +
    252                   ;  +                                          +
    253                   ;  ++++++++++++++++++++++++++++++++++++++++++++
    254                   ;
    255                   BCDADD:
    256 00A5 6B03                 MVI     C,4-1
    257                           ;
    258 00A7 482A         BCDAD1: CLC
    259 00A9 2B           BCDAD2: LDAX    H
    260 00AA 70D4                 ADCX    D+
    261 00AC 61                   DAA               ; DECIMAL ADJUST
    262 00AD 3D                   STAX    H+
    263                           ;
    264 00AE 53                   DCR     C
    265 00AF F9                   GJMP    BCDAD2
    266                           ;
    267 00B0 481A                 SKN     CY
    268 00B2 B8                   RET               ; OVERFLOW !
    269 00B3 B9                   RETS              ; NORMAL !
    270                           EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986                )  PAGE


                (                                        )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

    271                   ;
    272                   ;  +++++++++++++++++++++++++++++++++++++++++++++++
    273                   ;  +                                             +
    274                   ;  +           DECIMAL  SUBTRACTION              +
    275                   ;  +       8-PLACE <- 8-PLACE - 8-PLACE          +
    276                   ;  +                                             +
    277                   ;  +     MINUEND .. ( HL+3,..,HL )               +
    278                   ;  +     SUBTRACTER .. ( DE+3,...,DE )           +
    279                   ;  +     RESULT  .. ( HL+3,..,HL )               +
    280                   ;  +                                             +
    281                   ;  +      OUTPUT : RET .. OVERFLOW               +
    282                   ;  +               RETS .. NORMAL                +
    283                   ;  +                                             +
    284                   ;  +++++++++++++++++++++++++++++++++++++++++++++++
    285                   ;
    286                   BCDSUB:
    287 00B4 6B03                 MVI     C,4-1
    288                           ;
    289 00B6 482B         BCDSB1: STC
    290 00B8 6999         BCDSB2: MVI     A,99H     ; FOR ADJUST
    291 00BA 5600                 ACI     A,0
    292 00BC 70F4                 SBBX    D+
    293 00BE 70C3                 ADDX    H
    294 00C0 61                   DAA               ; DECIMAL ADJUST
    295 00C1 3D                   STAX    H+
    296                           ;
    297 00C2 53                   DCR     C
    298 00C3 F4                   GJMP    BCDSB2
    299                           ;
    300 00C4 480A                 SK      CY    ; CHECK / OVERFLOW ?
    301 00C6 B8                   RET           ; IF BORROW
    302 00C7 B9                   RETS          ; IF NO BORROW
    303                           EJECT
```

```
** uPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   304                     ;
   305                     ;  ++++++++++++++++++++++++++++++++++++++++++++
   306                     ;  +                                          +
   307                     ;  +          DECIMAL  MULTIPLY               +
   308                     ;  +       16 PLACE <- 8 PLACE * 8PLACE       +
   309                     ;  +                                          +
   310                     ;  +    MULTIPLICAND .. (DMLCND+3,..,DMLCND)  +
   311                     ;  +    MULTIPLIER   .. (DMLIER+3,..,DMLIER)  +
   312                     ;  +    RESULT       .. (DRSLT+7,...,DRSLT )  +
   313                     ;  +                                          +
   314                     ;  ++++++++++++++++++++++++++++++++++++++++++++
   315                     ;
   316                     ;  **  RESULT,MULTIPLICAND  OCLEAR  **
   317                     ;
   318                     BCDMLT:
   319 00C8 341CFF                 LXI     H,DRSLT
   320 00CB 6B07                   MVI     C,8-1
   321 00CD 4048B3                 CALL    RCLR0
   322                     ;
   323                     ;  **  PLACE COUNTER SET  **
   324                     ;
   325 00D0 6B07                   MVI     C,16/2-1
   326                     ;
   327                     ;  **  MULTIPLIER  RIGHT-SHIFT  **
   328                     ;
   329                     BCDML1:
   330 00D2 B1                     PUSH    B       ; PLACE-COUNTER STORE
   331                             ;
   332 00D3 341BFF                 LXI     H,DMLIER+3
   333 00D6 6B03                   MVI     C,8/2-1
   334 00D8 403101                 CALL    BCDRS
   335 00DB B0                     PUSH    V
   336                     ;
   337                     ;  **  CHECK / MULTIPLIER=0'  **
   338                     ;
   339 00DC 6A00                   MVI     B,0
   340                     BCDML2:
   341 00DE A0                     POP     V
   342 00DF 3601                   SUINB   A,1
   343 00E1 CC                     GJMP    BCDML4  ; IF  =0
   344                     ;
   345                     ;  **  RESULT + MULCND -> RESULT  **
   346                     ;
   347 00E2 B0                     PUSH    V
   348 00E3 3420FF                 LXI     H,DRSLT+4
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986                )  PAGE
                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

349 00E6 2414FF             LXI     D,DMLCND
350 00E9 40A500             CALL    BCDADD ; SKIP ,IF NORMAL
351 00EC 42                 INR     B      ; IF OVERFLOW
352                                        ; DIGIT ADDITION / NOT END
353 00ED F0                 GJMP    BCDML2
354                 ;
355                 ;  **  RESULT  RIGHT-SHIFT  WITH CARRY  **
356                 ;
357                 BCDML4:
358 00EE 0A                 MOV     A,B          ; LOAD  RESULT-MSD
359 00EF 3423FF             LXI     H,DRSLT+7
360 00F2 6B07               MVI     C,16/2-1     ; SHIFT-COUNTER SET
361 00F4 403301             CALL    BCDRS1
362
363                 ;
364                 ;  **  CHECK  / MULTIPLY END ?'  **
365                 ;
366 00F7 A1                 POP     B
367 00F8 53                 DCR     C
368 00F9 4FD7               GJMP    BCDML1
369 00FB B8                 RET             ; END !
370                         EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                        )
E STNO ADRS OBJECT    M    SOURCE STATEMENT


  371                     ;
  372                     ;  +++++++++++++++++++++++++++++++++++++++++++++
  373                     ;  +                                           +
  374                     ;  +          DECIMAL  DIVISION                +
  375                     ;  +       8-PLACE <- 8-PLACE / 8-PLACE        +
  376                     ;  +                                           +
  377                     ;  +     DIVIDEND  ..  (DIVIND+3,...,DIVIND )  +
  378                     ;  +     DIVISOR   ..  (DIVSOR+3,...,DIVSOR )  +
  379                     ;  +     RSLT      ..  (DIVIND+3,...,DIVIND )  +
  380                     ;  +     REMIND    ..  (RMIND+3,...,RMIND )    +
  381                     ;  +                                           +
  382                     ;  +++++++++++++++++++++++++++++++++++++++++++++
  383                     ;
  384                     BCDDIV:
  385                     ;
  386                     ;  **  CHECK / DIVISOR=0'   **
  387                     ;
  388 00FC 3428FF             LXI     H,DIVSOR
  389 00FF 4885               LDEAX   H++
  390 0101 B6                 DMOV    D,EA
  391 0102 4883               LDEAX   H
  392 0104 749E               DOR     EA,D
  393 0106 481C               SKN     Z
  394 0108 B8                 RET             ; OVERFLOW
  395                     ;
  396                     ;  **  RESULT,REMIND  OCLEAR  **
  397                     ;
  398 0109 404003             CALL    RMNDCL
  399                     ;
  400                     ;  **  DIGIT-COUNTER SET  **
  401                     ;
  402 010C 6A07               MVI     B,8-1
  403                     ;
  404                     ;  **  QUOTIENT,REMIND  LEFT-SHIFT  **
  405                     ;
  406                     BCDDV1:
  407 010E 3424FF             LXI     H,DIVIND
  408 0111 6B07               MVI     C,16/2-1
  409 0113 403901             CALL    BCDLS
  410                     ;
  411                     ;  **  SUBTRACT DIVISOR  FROM  DIVIDEND  **
  412                     ;
  413                     BCDDV2:
  414 0116 342CFF             LXI     H,RMIND
  415 0119 2428FF             LXI     D,DIVSOR
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986                )  PAGE

                (                                      )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  416 011C 40B400              CALL     BCDSUB
  417 011F C3                  GJMP     BCDDV3
  418                          ;
  419 0120 2024                INRW     DIVIND MOD 100H ; 1-DIGIT.QUOT (+

  420 0122 F3                  GJMP     BCDDV2
  421                  ;
  422                  ;  **  IF BORROW , DIVISOR + DIVIDEND
  423                  ;
  424                  BCDDV3:
  425 0123 342CFF              LXI      H,RMIND
  426 0126 2428FF              LXI      D,DIVSOR
  427 0129 40A500              CALL     BCDADD
  428 012C 00                  NOP
  429                  ;
  430                  ;  **  CHECK / DIVISION END ?  **
  431                  ;
  432 012D 52                  DCR      B          ; SKIP , IF DIVISION END
  433 012E 4FDE                GJMP     BCDDV1     ; NOT END !
  434 0130 B9                  RETS                ; END !
  435                          ;
  436                          EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                    )
E STNO ADRS OBJECT    M    SOURCE STATEMENT

  437                    ;
  438                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  439                    ; S                                          S
  440                    ; S      N-DIGIT DATA  RIGHT-SHIFT           S
  441                    ; S             HL <- MSD                    S
  442                    ; S             C-REG <- DIGIT COUNTER       S
  443                    ; S                                          S
  444                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  445                    ;
  446 0131 6900          BCDRS:  MVI     A,0
  447                    ;
  448 0133 4839          BCDRS1: RRD
  449 0135 33                    DCX     H
  450 0136 53                    DCR     C
  451 0137 FB                    GJMP    BCDRS1
  452 0138 B8                    RET
  453                    ;
  454                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  455                    ; S                                          S
  456                    ; S      N-DIGIT DATA  LEFT-SHIFT            S
  457                    ; S             HL <- LSD                    S
  458                    ; S             C-REG <- DIGIT COUNTER       S
  459                    ; S                                          S
  460                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  461                    ;
  462 0139 6900          BCDLS:  MVI     A,0
  463                    ;
  464 013B 4838          BCDLS1: RLD
  465 013D 32                    INX     H
  466 013E 53                    DCR     C
  467 013F FB                    GJMP    BCDLS1
  468 0140 B8                    RET
  469                    ;
  470                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  471                    ; S                                          S
  472                    ; S      N-BYTE DATA  RIGHT SHIFT            S
  473                    ; S            HL - LSB                      S
  474                    ; S            C-REG  <-  BYTE COUNTER       S
  475                    ; S                                          S
  476                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  477                    ;
  478                    BYTRST:
  479 0141 53                    DCR     C
  480 0142 53                    DCR     C
  481                    ;
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                          )

E STNO ADRS OBJECT    M    SOURCE STATEMENT
  482 0143 2B                      LDAX     H
  483 0144 B0                      PUSH     V
  484                              ;
  485 0145 32            BYTRS1:   INX      H
  486 0146 2F                      LDAX     H-
  487 0147 3D                      STAX     H+
  488                              ;
  489 0148 53                      DCR      C
  490 0149 FB                      GJMP     BYTRS1
  491                              ;
  492 014A 6900                    MVI      A,0
  493 014C 3B                      STAX     H
  494 014D A0                      POP      V
  495 014E B8                      RET
  496                    ;
  497                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  498                    ; S                                              S
  499                    ; S      N-BYTE DATA  LEFT SHIFT                 S
  500                    ; S                   HL <-  MSB                 S
  501                    ; S                   C-REG <-  BYTE COUNTER     S
  502                    ; S                                              S
  503                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  504                    ;
  505                    BYTLST:
  506 014F 53                      DCR      C
  507 0150 53                      DCR      C
  508                              ;
  509 0151 2B                      LDAX     H
  510 0152 B0                      PUSH     V
  511                    BYTLS1:
  512 0153 33                      DCX      H
  513 0154 2D                      LDAX     H+
  514 0155 3F                      STAX     H-
  515                              ;
  516 0156 53                      DCR      C
  517 0157 FB                      GJMP     BYTLS1
  518                              ;
  519 0158 6900                    MVI      A,0
  520 015A 3B                      STAX     H
  521 015B A0                      POP      V
  522 015C B9                      RETS
  523                              EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                        )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

    524                   ;
    525                   ;
    526                   ;  +++++++++++++++++++++++++++++++++++++++++++++++++
    527                   ;  +                                               +
    528                   ;  +     TRANSFER * HEX -> BCD     *               +
    529                   ;  +             (HL,HL+1)   (HL,HL+1)             +
    530                   ;  +                                               +
    531                   ;  +++++++++++++++++++++++++++++++++++++++++++++++++
    532                   ;
    533                   TBCDHX:
    534 015D 440000               LXI     EA,0     ; RESULT  0CLEAR
    535 0160 6903                 MVI     A,2*2-1  ; DIGIT COUNTER SET
    536                   TBDHX1:
    537 0162 B0                   PUSH    V
    538 0163 B3                   PUSH    H
    539                   ;
    540                   ;  **  OUTPUT  1-DIGIT FROM MSD  **
    541                   ;
    542 0164 6B01                 MVI     C,4/2-1
    543 0166 403901               CALL    BCDLS
    544                           ;
    545                   ;
    546                   ;  **  CHECK / BCD DISPLAY ?
    547                   ;
    548 0169 370A                 LTI     A,9+1
    549 016B B8                   RET             ; NOT BCD
    550                   ;
    551                   ;  *S*  SUBROUTINE / BCD -> HEX  *S*
    552                   ;
    553 016C 407601               CALL    SBCDHX
    554                   ;
    555                   ;  *S*  SUBROUTINE  END  *S*
    556                   ;
    557 016F A3                   POP     H
    558 0170 A0                   POP     V
    559 0171 51                   DCR     A        ; SKIP  IF TRANSFER END
    560 0172 EF                   GJMP    TBDHX1   ; NOT END
    561                           ;
    562 0173 4893                 STEAX   H        ; STORE RESULT
    563 0175 B9                   RETS
    564                           EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

    565                   ;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
    566                   ;  T                                            S
    567                   ;  T     TRANSFER  SUBROUTINE                   S




    568                   ;  T                                            S
    569                   ;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
    570                   ;
    571                   SBCDHX:
    572                   ;
    573                   ;  **  RESULT * 10  **
    574                   ;
    575 0176 B5                     DMOV    B,EA
    576 0177 B0                     PUSH    V
    577 0178 690A                   MVI     A,10
    578 017A 482F                   MUL     C
    579 017C B4                     PUSH    EA
    580                             ;
    581 017D 482E                   MUL     B
    582 017F A1                     POP     B
    583 0180 7042                   EADD    EA,B
    584 0182 09                     MOV     A,EAL
    585 0183 1A                     MOV     B,A
    586                   ;
    587                   ;  **  RESULT *10 + 1-DIGIT  **
    588                   ;
    589 0184 A0                     POP     V
    590 0185 A5                     DMOV    EA,B
    591 0186 7041                   EADD    EA,A
    592 0188 B8                     RET
    593                             EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE

                (                                        )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   594                          ;
   595                    ;
   596                    ;  ++++++++++++++++++++++++++++++++++++++++++++++
   597                    ;  +                                            +
   598                    ;  +     TRANSFER / BCD  ->  HEX                +
   599                    ;  +              (HL,HL+1)    (HL,HL+1)        +
   600                    ;  +                                            +
   601                    ;  ++++++++++++++++++++++++++++++++++++++++++++++
   602                    ;
   603                    THXBCD:
   604 0189 441027                 LXI     EA,10000
   605 018C B6                     DMOV    D,EA
   606                             ;
   607 018D 4883                   LDEAX   H
   608 018F 74BE                   DLT     EA,D    ; CHECK / HEX > 10000
   609 0191 B8                     RET             ; HEX > 10000 !
   610                             ;
   611 0192 6A04                   MVI     B,4     ; DIGIT-COUNTER  SET !
   612                    ;
   613                    ;  **  CHECK / TRANS END  ?  **
   614                    ;
   615 0194 743201        THXBD1: SUINB   B,1     ; COUNTER DECREMENT
   616 0197 B9                     RETS            ; END !
   617                    ;
   618                    ;  **  SUBROUTINE / HEX -> BCD  **
   619                    ;
   620 0198 40A301                 CALL    SDIGIT  ; 1-DIGIT BCD -> ACC
   621                    ;
   622                    ;  **  RESULT * 10 + 1-HEX DATA  **
   623                    ;
   624 019B B3                     PUSH    H
   625 019C 6B01                   MVI     C,4/2-1
   626 019E 403B01                 CALL    BCDLS1  ; ACC -> LSD
   627 01A1 A3                     POP     H
   628 01A2 F1                     GJMP    THXBD1
   629                             EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                          )
E STNO ADRS OBJECT    M   SOURCE STATEMENT

    630                   ;
    631                   ;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTTSTSTSTSTST
    632                   ;  T                                            T
    633                   ;  T     TRANSFORM    SUBROUTINE                 T
    634                   ;  T                                            T



    635                   ;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
    636                   ;
    637                   SDIGIT:
    638 01A3 B4                     PUSH     EA
    639 01A4 A6                     DMOV     EA,D
    640 01A5 690A                   MVI      A,10
    641 01A7 483D                   DIV      A
    642 01A9 B6                     DMOV     D,EA
    643 01AA A4                     POP      EA
    644                             ;
    645 01AB 6900                   MVI      A,0       ; RESULT 1-DIGIT 0 INITIA

    646 01AD 74B6         SDGT0:    DSUBNB   EA,D
    647 01AF C2                     GJMP     SDGT1
    648                             ;
    649 01B0 41                     INR      A
    650 01B1 FB                     GJMP     SDGT0
    651                             ;
    652 01B2 74C6         SDGT1:    DADD     EA,D      ; FOR ADJUST
    653 01B4 B8                     RET
    654                             ;
    655                             EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

    656                    ;
    657                    ; ++++++++++++++++++++++++++++++++++++++++++++++++
    658                    ; +                                              +
    659                    ; +      TRANSFER : HEX  <=  ASCII               +
    660                    ; +                 (2CODE)   (2CODE)            +
    661                    ; +                                              +
    662                    ; +           INPUT : BC REG <= ASCII            +
    663                    ; +           OUTPUT:  (HL)   <= HEX             +
    664                    ; +                                              +
    665                    ; ++++++++++++++++++++++++++++++++++++++++++++++++
    666                    ;
    667                    GETHEX:
    668 01B5 0A                    MOV     A,B       ; ASCII UPPER-CODE LOAD
    669 01B6 40C201                CALL    SGTHEX    ; GET ! HEX 1CODE
    670 01B9 B8                    RET               ; ERREGULAR ASCII !
    671                            ;
    672 01BA 4838                  RLD
    673 01BC 0B                    MOV     A,C       ; ASCII LOWER-CODE LOAD
    674 01BD 40C201                CALL    SGTHEX    ; GET ! HEX  2TH CODE
    675 01C0 B8                    RET               ; ERREGULAR ASCII !
    676 01C1 B9                    RETS              ; POSSIBLE
    677                    ;
    678                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
    679                    ; S                                          S
    680                    ; S      SUBROUTINE / GET  HEX 1-CODE(ACC)   S
    681                    ; S                                          S
    682                    ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
    683                    ;
    684                    SGTHEX:
    685 01C2 3630                  SUINB   A,'0'     ; CHECK / ASCII > 30H
    686 01C4 B8                    RET               ; ERROR , SO < 30H
    687                            ;
    688 01C5 2709                  GTI     A,'9'-'0'
    689                                              ; CHECK / 30H < ASCII <
9H
    690 01C7 B9                    RETS
    691 01C8 3611                  SUINB   A,'A'-'0'
    692                                              ; CHECK / ASCII > 41H
    693 01CA B8                    RET
    694 01CB 260A                  ADINC   A,0AH     ; CHECK / ASCII < 46H
    695 01CD B8                    RET               ;
    696 01CE B9                    RETS
    697                    ;
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  698                     ; ++++++++++++++++++++++++++++++++++++++++++++
  699                     ; +                                          +
  700                     ; +      TRANSFER / ASCII <=  HEX            +
  701                     ; +                                          +
  702                     ; +         INPUT : (HL) <= HEX 2-CODE       +
  703                     ; +         OUTPUT : BC REG <= ASCII 2-CODE  +
  704                     ; +                                          +
  705                     ; ++++++++++++++++++++++++++++++++++++++++++++
  706                     ;
  707                     GETASC:
  708 01CF 6900                   MVI     A,0
  709 01D1 4838                   RLD             ; HEX UPPER CODE  LOAD
  710 01D3 40E001                 CALL    SGTASC
  711 01D6 1A                     MOV     B,A     ; STORE RESULT
  712                             ;




  713 01D7 6900                   MVI     A,0
  714 01D9 4838                   RLD             ; HEX LOWER CODE  LOAD
  715 01DB 40E001                 CALL    SGTASC
  716 01DE 1B                     MOV     C,A     ; STORE RESULT
  717 01DF B8                     RET
  718                     ;
  719                     ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  720                     ; S                                          S
  721                     ; S      GET /  ASCII 1-CODE ( BC REG )      S
  722                     ; S                                          S
  723                     ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
  724                     ;
  725                     SGTASC:
  726 01E0 370A                   LTI     A,0AH   ; CHECK / HEX > 9
  727 01E2 4607                   ADI     A,41H-3AH ; BIAS (+7)
  728 01E4 4630                   ADI     A,30H   ; BIAS (+30H)
  729 01E6 B8                     RET
  730                     ;
  731                     ; ++++++++++++++++++++++++++++++++++++++++++++
  732                     ; +                                          +
  733                     ; +      DATA  SEARCHING  ( HL ) ( DE )      +
  734                     ; +                                          +
  735                     ; +        INPUT : HL <= SERCH DATA ADDRS    +
  736                     ; +                DE <= TABLE TOP ADDRS     +
  737                     ; +        OUTPUT : DE <= SEARCH TABLE ADDRS +
  738                     ; +                                          +
  739                     ; ++++++++++++++++++++++++++++++++++++++++++++
  740                     ;
  741                     SEARCH:
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                        (                                        )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  742            0200            ORG     200H
  743 0200 2A                    LDAX    D           ; LOAD ! DATA-LENGTH
  744 0201 2700                  GTI     A,1-1       ; CHECK / LENGTH = 0
  745 0203 B8                    RET                 ; = 0
  746                            ;
  747 0204 1A                    MOV     B,A         ; STORE LENGTH
  748 0205 B2                    PUSH    D
  749 0206 B3                    PUSH    H           ; STORE SEARCH DATA ADDRS
  750                    SERCH1:
  751 0207 2C                    LDAX    D+
  752 0208 70FD                  EQAX    H+          ; CHECK / SEARCH DATA = T
ABLE
  753 020A C5                    GJMP    SERCH2      ; NOT =
  754                            ;
  755 020B 52                    DCR     B           ; DATA-LENGTH (-1)
  756 020C FA                    GJMP    SERCH1      ; NOT END-DATA
  757                            ;
  758 020D A3                    POP     H
  759 020E A2                    POP     D

  760 020F B9                    RETS                ; SEARCH SUCCESS !
  761                    ;
  762                    ;   **  FOR NEXT-TABLE DATA CHECK  **
  763                    ;
  764                    SERCH2:




  765 0210 A3                    POP     H
  766 0211 A6                    DMOV    EA,D
  767 0212 A2                    POP     D
  768 0213 7042                  EADD    EA,B
  769 0215 B6                    DMOV    D,EA        ; DE <= NEXT-TABLE DATA A
DDRS
  770 0216 4FCF                  GJMP    SEARCH
  771                            ;
  772                            EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

    773                    ;
    774                    ;
    775                    ; +++++++++++++++++++++++++++++++++++++++++++++++++
    776                    ; +
    777                    ; +          INTERPOLATION  POLYNOMIAL




    778                    ; +
    779                    ; +        INPUT      ACC <- X
    780                    ; +                   HL <= DATA TABLE TOP ADDRS
    781                    ; +        OUTPUT     EA - REG
    782                    ; +
    783                    ; +  F(N)= F(S) + ( F(S+10)-F(S) ) * (N-S) /10
    784                    ; +
    785                    ; +++++++++++++++++++++++++++++++++++++++++++++++++
    786                    ;
    787                    POLYNM:
    788           0300                 ORG      300H
    789 0300 440000                    LXI      EA,0
    790 0303 19                        MOV      EAL,A
    791 0304 690A                      MVI      A,10
    792 0306 483D                      DIV      A
    793 0308 B0                        PUSH     V          ; STACK <- N-S
    794                                ;
    795 0309 09                        MOV      A,EAL
    796 030A 4825                      SLL      A          ; ACC <- 2BYTE DATA TOP
DDRS
    797                                ;
    798 030C A7                        DMOV     EA,H
    799 030D 7041                      EADD     EA,A
    800 030F B7                        DMOV     H,EA
    801                                                    ; HL <-  F(S) STORE ADDR
    802 0310 4883                      LDEAX    H
    803 0312 B6                        DMOV     D,EA      ; DE <-  F(S)
    804 0313 488F02                    LDEAX    H+2       ; EA <-  F(S+10)
    805                                                    ; HL <- F(S) STORE ADDRS
    806 0316 A0                        POP      V
    807 0317 74E6                      DSUB     EA,D      ; F(S+10) - F(S)
    808 0319 480A                      SK       CY
    809 031B CC                        GJMP     POLYN1    ; IF F(S+10) > F(S)
    810                    ;
    811 031C 32                        INX      H
    812 031D 32                        INX      H
    813 031E 660A                      SUI      A,10
    814 0320 483A                      NEGA               ; 10- ((S+10)-N)
    815                                ;
    816 0322 B6                        DMOV     D,EA
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                          )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  817 0323 440000              LXI     EA,0
  818 0326 74E6                DSUB    EA,D      ; F(S) - F(S+10)
  819                  POLYN1:
  820 0328 B0                  PUSH    V
  821 0329 690A                MVI     A,10
  822 032B 483D                DIV     A         ; ( F(S+10) - F(S) ) / 1
  823 032D A0                  POP     V
  824 032E B5                  DMOV    B,EA
  825                  ;
  826                  ;     ( F(S+10)-F(S) ) * (N-S) / 10
  827                  ;
  828 032F 482F                MUL     C
  829 0331 B4                  PUSH    EA

  830                          ;
  831 0332 482E                MUL     B
  832 0334 09                  MOV     A,EAL
  833 0335 A1                  POP     B
  834 0336 6042                ADD     B,A
  835                  ;
  836 0338 4883                LDEAX   H
  837 033A 74C5                DADD    EA,B
  838 033C B8                  RET
  839                          EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE

                (                                    )

E STNO  ADRS  OBJECT    M    SOURCE STATEMENT

    840
    841                      ;  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
    842                      ;  S                                             S
    843                      ;  S            COMMON  SUBROUTINE               S
    844                      ;  S                                             S
    845                      ;  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS




    846                      ;
    847                      ;  **  0-CLEAR  PROCESS  **
    848                      ;
    849 033D 3418FF      HIMCLR: LXI      H,DMLCND+4
    850 0340 342CFF      RMNDCL: LXI      H,RMIND
    851 0343 3404FF      RSTCLR: LXI      H,BRSLT
    852 0346 340CFF      DIVCLR: LXI      H,BRMND
    853                          ;
    854 0349 6B03                MVI      C,4-1
    855                          ;
    856 034B 6900        RCLR0:  MVI      A,0
    857 034D 3D          RCLR1:  STAX     H+
    858 034E 53                  DCR      C
    859 034F FD                  GJMP     RCLR1
    860 0350 B8                  RET
    861                          ;
    862                      ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
    863                      ; S                                             S
    864                      ; S       COMMON  SUBROUTINE / END'             S
    865                      ; S                                             S
    866                      ; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
    867                            EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (5.2.1986              )  PAGE
                (                                        )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

    868                       ;
    869             0000              STACK   EQU     0       ;  PROGRAM  COUNTER
    870             0004              NTOP    EQU     4       ;  TABLE TOP ADDRS
    871                       ;
    872             FF00              ORG     OFF00H
    873                       ;
    874                       ;  BINARY  MULTIPLY




    875                       ;
    876 FF00        0002              BMLCND: DS      2
    877 FF02        0002              BMLIER: DS      2
    878 FF04        0004              BRSLT:  DS      4
    879                       ;
    880                       ;  BINARY  DIVISION
    881                       ;
    882 FF08        0004              DEND:   DS      4
    883 FF0C        0004              BRMND:  DS      4
    884 FF10        0004              SOR:    DS      4
    885                       ;
    886                       ;  DECIMAL  MULTIPLY
    887                       ;
    888 FF14        0004              DMLCND: DS      4
    889 FF18        0004              DMLIER: DS      4
    890 FF1C        0008              DRSLT:  DS      8
    891                       ;
    892                       ;  DECIMAL  DIVISION
    893                       ;
    894 FF24        0004              DIVIND: DS      4
    895 FF28        0004              DIVSOR: DS      4
    896 FF2C        0004              RMIND:  DS      4
    897                       ;
    898                       ; TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
    899                       ;
    900                             END

ERROR =     0
  (    0)
OPTIMIZATION COMPLETED,  TIMES =     2
```

```
** UPD7811      CROSS REFERENCE LIST ** (5.2.1986              )  PAGE

SYMBOL  DEF.  REF.   ( STATEMENT NUMBER )

BCDAD1  0258
BCDAD2  0259  0265
BCDADD  0255  0350   0427
BCDDIV  0384
BCDDV1  0406  0433
BCDDV2  0413  0420
BCDDV3  0424  0417
BCDLS   0462  0211   0409   0543
BCDLS1  0464  0467   0626
BCDML1  0329  0368

BCDML2  0340  0353
BCDML4  0357  0343
BCDMLT  0318
BCDRS   0446  0334
BCDRS1  0448  0361   0451
BCDSB1  0289
BCDSB2  0290  0298
BCDSUB  0286  0416
BFAD1   0046  0051
BFADD   0042  0230
BFDIV   0186
BFDIV2  0208  0236
BFDIV3  0215  0223
BFDIV4  0227  0220
BFMUL   0100
BFMUL1  0112  0131
BFMULS  0141  0121
BFSB1   0076  0081
BFSUB   0072  0219
BMLCND  0876  0142
BMLIER  0877  0110
BRMND   0883  0216   0228   0852
BRSLT   0878  0109   0851
BYTLS1  0511  0517
BYTLST  0505
BYTRS1  0485  0490
BYTRST  0478
DEND    0882  0209   0222
DIVCLR  0852  0200
DIVIND  0894  0407   0419
DIVSOR  0895  0388   0415   0426
DMLCND  0888  0349   0849
DMLIER  0889  0332
DRSLT   0890  0319   0348   0359
GETASC  0707
GETHEX  0667
HIMCLR  0849
```

```
** UPD7811      CROSS REFERENCE LIST ** (5.2.1986              )  PAGE

SYMBOL  DEF.  REF.   ( STATEMENT NUMBER )

MULSS   0149 0166
NTOP    0870
POLYN1  0819 0809
POLYNM  0787
RCLR0   0856 0321
RCLR1   0857 0859
RMIND   0896 0414   0425   0850
RMNDCL  0850 0398
RSTCLR  0851 0104
SBCDHX  0571 0553
SDGT0   0646 0650
SDGT1   0652 0647
SDIGIT  0637 0620
SEARCH  0741 0770
SERCH1  0750 0756
SERCH2  0764 0753
SGTASC  0725 0710   0715
SGTHEX  0684 0669   0674



SOR     0884 0190   0217   0229
STACK   0869 0015
START   0027
TBCDHX  0533
TBDHX1  0536 0560
THXBCD  0603
THXBD1  0615 0628
```

Square Wave Generation by the μPD7811 / 08 / C11 / C14

Contents:

Author: E. Goldberg
NEC Electronics USA

Person to contact: R. Cherrington
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Product Descriptions
μPD7807 / 08 / 09
μPD78C11 / C12 / C14

Related Products

μPD7807 / 08 / 09	8-Bit Single chip microcomputer	NMOS
μPD78C11 / C12 / C14	8-Bit Single chip microcomputer	CMOS

1. Introduction

This application note shows how to use the Timer/Event Counter of the μPD7810/7811 to generate two square waves (50 % duty cycle) which are out of phase by a variable amount. The phase shift of the waveforms can be altered by changing one constant in the program. Phase shifts of 0°—180° require a different initialization than shifts of 180°—360°. However, in both cases, incremental phase shift changes within the 180° boundaries are still varied by changing one constant in the program.

The resolution of the phase shift is determined by the clock input frequency of the μPD7810/7811, the resolution of the counters and the frequency of the square waves. In general, as the frequency of the square wave is increased, the resolution of the phase shift will decrease.

Conversely, as the frequency of the square wave decreases, the resolution of the phase shift increases. As the clock input frequency increases, the phase shift resolution increases and when it decreases, the phase resolution decreases. How this happens will be explained later in this note.

2. Operation of ETMM and EOM Registers

In order to write a program for the problem described in section 1., it is essential to understand how the ETMM and EOM registers operate and interrelate with each other. A block diagram of the Timer/Event Counter is shown in Figure 1 and a diagram of the CO0 output circuitry is shown in Figure 2. There is circuitry identical to Figure 2 for the CO1 output. The formats for the Timer/Event Mode Register (ETMM) and the Timer/Control Output Mode Register (EOM) are shown in Figures 3 and 4 respectively.

The CO0 output (Figure 2) is a master-slave type design. The level flip-flop (LV0 F/F) is the master portion of the CO0 output. It contains the next level (in the LV0 flip-flop) which will be output to be output latch (slave portion), which is the CO0 output. The CO0 output can be controlled using the ETMM register in combination with the EOM register. It can also be controlled using the EOM register only.

Referring to Figure 2, the CP0, CP1, and EIN inputs are generated by the timer circuitry. The LO0, LD0, LRE0 and LRE1 inputs are from the EOM Register. Referring to Figure 1, CP0 is generated when ECNT = ETM0, CP1 is generated when ECNT = ETM1, and EIN is generated if a falling edge of CI or TO is detected.

Controlling the CO0 output from only the EOM register is the easiest mode to unterstand. Bits 0—3 of the EOM control CO0 and bits 4—7 control CO1. Bits 1—3 specify whether the LV0 flip-flop (master stage) is to be set, reset or inverted. Bit 0 of EOM controls when the LV0 output latch will change.

Note: The bits LRE3, LRE2, LRE1, LRE0 function like a mono-flop, i.e. they go back to zero after having triggered the output flip-flops.

For example, if the Timer/Event Counter is programmed

```
MVI EOM, 09H
```

the LV0 flip-flop will be set by bits 3 and 2 = 10, and the level will be outputted to CO0 immediately since bit 0 = 1. However, if the counter were programmed as:

```
SET F/F                 MVI EOM,08H
OUTPUT                  MVI EOM,01H
```

Here, the LV0 flip-flop will be set when the SET F/F instruction is executed, but the output CO0 will not become high until the OUTPUT instruction is executed. This second type of sw procedure is recommended to prevent internal timing problems between setting the flip-flop and outputting its value. When using EOM only to control CO0, the EOM register must be programmed each time the CO0 output is to be changed and bit 0 must be 1.

The second and more complex application is to control the CO0 output using both ETMM and EOM. In this situation, how CO0 will be changed is determined by bits 1—3 of EOM. ETMM controls when and under what timer conditions CO0 and CO1 will change.

Bits 0 and 1 of ETMM select the ECNT input clock. The frequency of the internal 012 clock is:

$$12 \text{ frequency} = \frac{\text{XTAL frequency}}{12}$$

The frequency of the CI input clock is the 7810/7811 input crystal frequency divided by 3. However, when using the CI input as an event counter, signals must be two clock periods long (500 ns with 12 MHz crystal) to prevent noise errors. Bits 2 and 3 of ETMM control when ECNT is cleared. Bits 4 and 5 control what ECNT counter conditions will changed it. Bits 6 and 7 control CO1 in the same manner that bits 4 and 5 control CO0.

3. Programming Example

This section will show how to implement the two square waves described in section 1. Figure 5a shows the two square waves and their associated parameters. The square waves will be generated using CO1 and CO0 (Port PC7 and PC6 respectively). Output CO1 will be generated using the value in counter ETM1 as 1/2 the period and using the inversion bit LD1 of the EOM register. ECNT will count up until it reaches a value where ECNT = ETM1. At this point CP1 will be generated causing output CO1 to be inverted and ECNT will be cleared to 0000H and start counting up. When ECNT = ETM1, CO1 will be inverted again, thereby completing one period of the square wave. The period (t) of the square wave is:

$$t = 2 \text{ (ETM1 Times)}$$

The second square wave is outputted at CO0 and is phase shifted from CO1 by a time = ETM0 count. At the start, the outputs are programmed such that CO1 = 1 and CO0 = 0. ECNT counts up and when ECNT = ETM0, CP0 will be generated causing CO0 to be inverted (using bit LD0 of EOM). Each time ECNT = ETM0 CO0 will be inverted. This provides the phase shift between CO1 and CO0. Note that ECNT is only cleared to zero when ECNT = ETM1 and simultaneously the level of CO1 is changed.

4. Effects of Crystal Frequency and Counter Resolution

As stated in section 1., crystal frequency, square wave frequency and counter resolution very much affect the phase shift.

If the crystal frequency is 12 MHz, then the 12 clock has a period of 1 μsec. This means that the finest counter interval is 1 μsec and hence, the phase shift cannot be in increments of less than 1 μsec. If a square wave frequency of 10 KHz is desired (100 μs period), then the smallest phase shift resolution is:

$$\frac{1\ \mu s}{100\ \mu s} = \frac{x}{360} \qquad x = 3.6° \text{ minimum phase resolution}$$

If a 100 KHz square wave (10 μsec period) is desired, then the minimum phase shift resolution is:

$$\frac{1\ \mu s}{10\ \mu s} = \frac{x}{360} \qquad x = 36° \text{ minimum phase resolution}$$

Hence, phase shift resolution decreases as the square wave frequency increases.

If in the same examples the crystal frequency is 6 MHz, then the 012 clock period is 2 μsec. For the 10 KHz square wave, the minimum phase resolution is:

$$\frac{2\ \mu s}{100\ \mu s} = \frac{x}{360} \qquad x = 7.2^\circ \text{ minimum phase resolution}$$

Hence, phase shift resolution decreases as the crystal frequency decreases. And finally, phase shift resolution is limited by the counter resolution. If all 16 bits of the counter were used to generate 1/2 period of the square wave, then the absolute minimum phase shift resolution of the 7810/7811 would be:

$$\frac{1}{65536} = \frac{x}{180} \qquad x = 0.0027^\circ \text{ minimum phase resolution}$$

The program to generate the two square waves in Figure 5a is:

```
INITIALIZE:   DI
              MVI     A,0H          ; Clear and Stop
              MOV     ETMM,A        ; ECNT Counter
              MVI     A,0C0H        ; Set CO0 and CO1 of Port C to
              MOV     MCC,A         ; the Control Mode.
INITSQWV:     MVI     EOM,084A      ; Set CO1=1 and CO0=0 and
              MVI     EOM,0B7H      ; Select Inversion for CO0 and CO1.
LDCTR:        LXI     EA,1/2 PERIOD ; Load ETM1 with a Count
              DMOV    ETM1,EA       ; Equal to 1/2 Square Wave Period.
              LXI     EA,PHSFT      ; Load ETM0 with a Phase
              DMOV    ETM0,EA       ; Shift.
START:        MVI     A,0CH         ; Select 012 as ECNT Clock,
              MOV     ETMM,A        ; Clear ECNT when = ETM1, CO0
              EI                    ; Output Change t ECNT = ETM0,
              RET                   ; and CO1 Output Change at ECNT = ETM1.
                                    ; Count begins, enable Interrupts and REturn
                                    ; from Subroutine.
```

In this program, 1/2 PERIOD is the count value which generates half the period. PHSFT is the constant which generates the phase shift. It is required that:

$$\text{PHSFT} \leq \text{1/2 PERIOD}$$

If the phase shift is to be between 180°—360° as shown in Figure 5b, then the instruction at INITSQWV must be changed such both CO0 and CO1 ae set to a logic 1. This is shown in Figure 5b. To do this the instruction would be:

```
INITSQWV:     MVI     EOM,088H
              MVI     EOM,0BBH
```

This value PHSFT would be the incremental phase shift between 180°—360°.

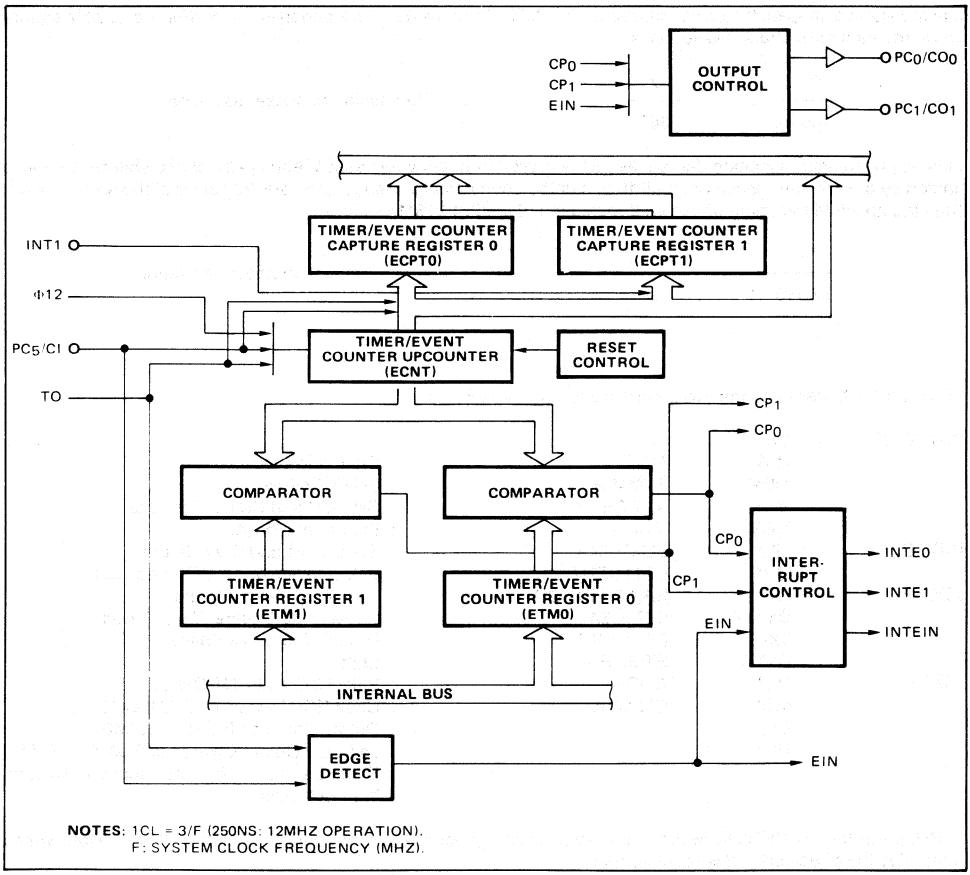


Figure 1. Block Diagram of Multifunction Timer/Event Counter (7809)

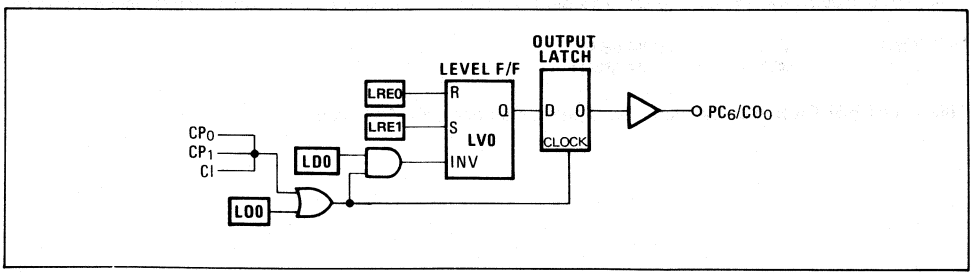


Figure 2. Block Diagram of CO0 Output Circuitry

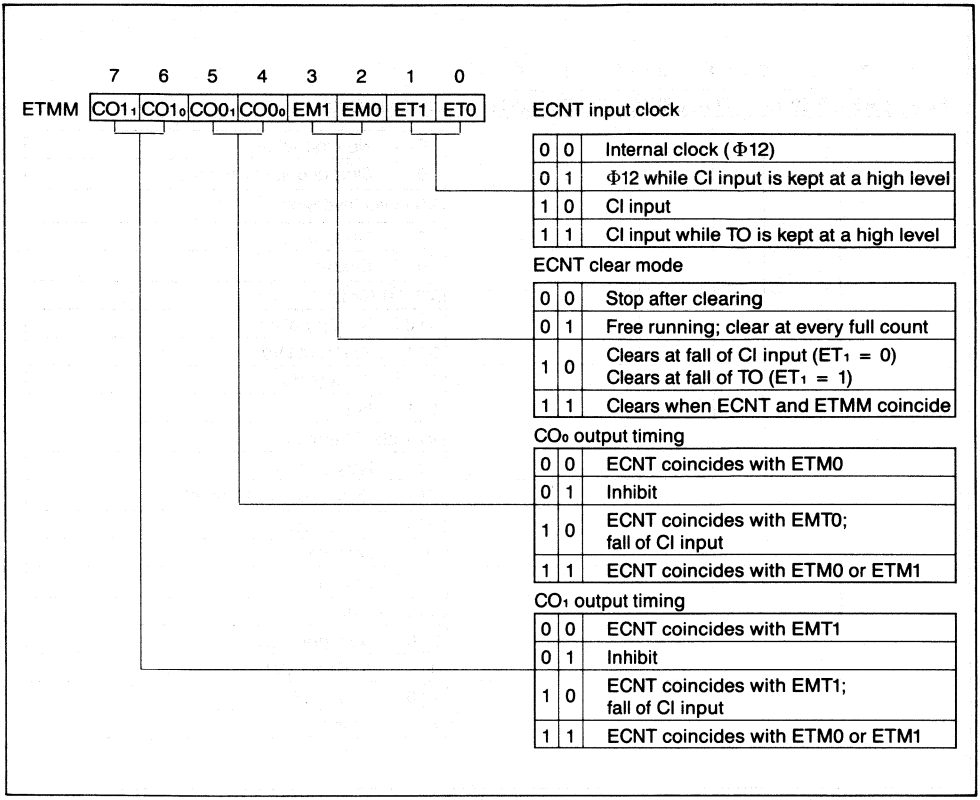


Figure 3. Timer/Event Counter Mode Register

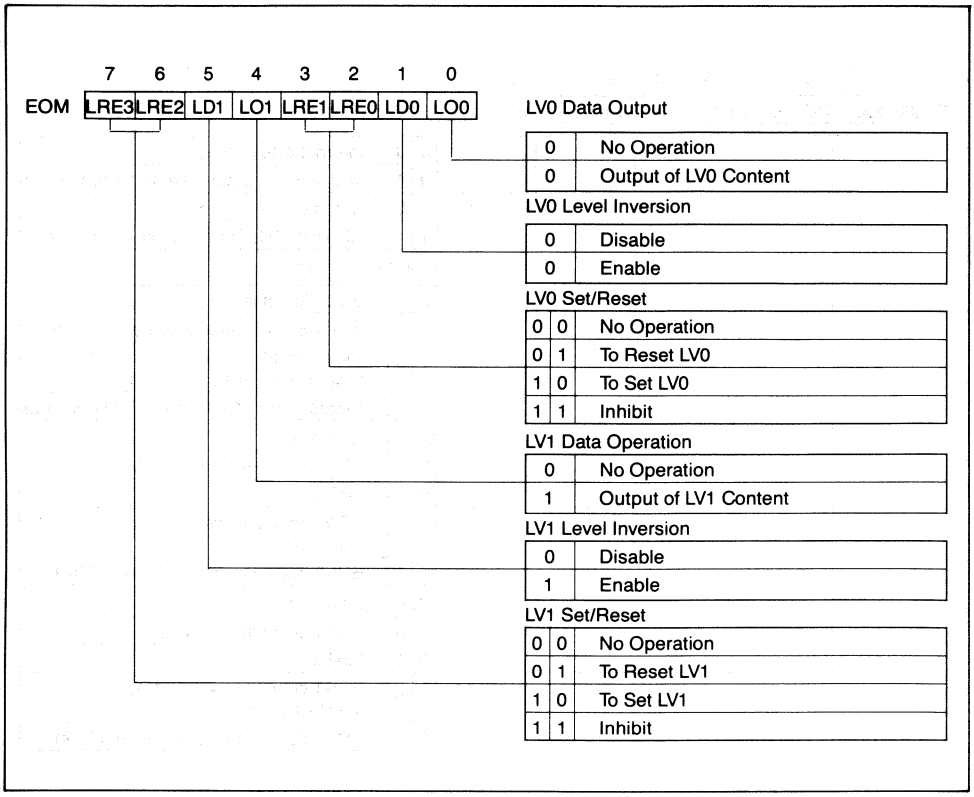


Figure 4. Timer/Event Counter Mode Register

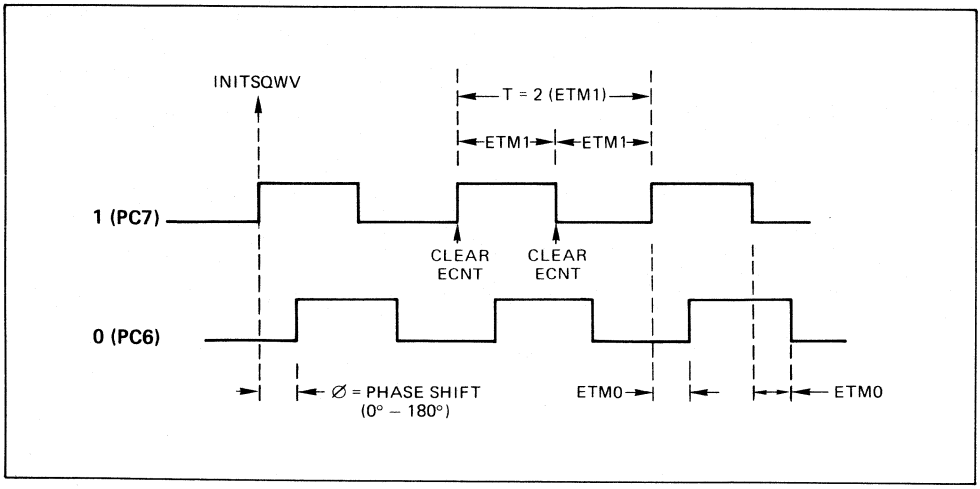


Figure 5a. Square Wave with Phase Shift = 0°—180°

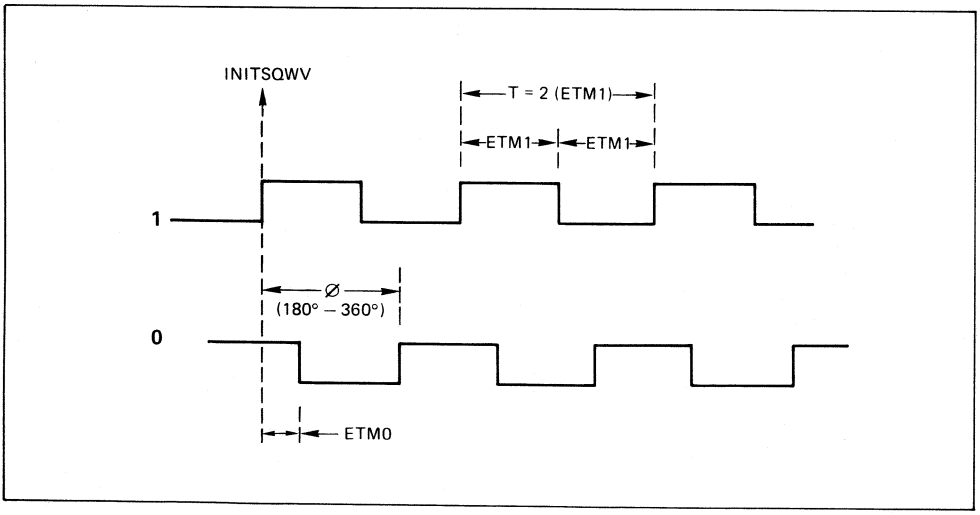


Figure 5b. Square Wave with Phase Shift = 180°—360°

Figure 5. Square Wave with Phase Shifts

PG-1000 PROM Programmer and the PG-1005 Adapter

Contents:

Author: Reiner Engels
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

PG-1000	Operation manual
PG-1005	Operation manual
EVAKIT 75X	Product description

Related Products

PG-1000	Programmer
PG-1005	Adapter for 75X
EVAKIT 75X	Development kit for 75X

1. Introduction

The PG-1000 is a PROM programmer unit with adapters for several programmable devices of the NEC μCOM 87 and 75X family. The connection to the a host computer is via a RS232C communication interface. Baud rates and formats can be selected with DIP-Switches on the PG-1000. Especially for some NEC EVAKITS it is possible to connect the PG-1000 directly via the second serial communication channel. This application note summarises all necessary information using the PG-1000, 1005 and EVAKIT 75X. For the host mode using MD-080/086 refer to the PG-1000 Operating manual.

2. Hardware Connections

2.1 Connection between PG-1000 and PG-1005

Before using the PG-1005

Please observe the following important precautions when using the PG-1005. NEC cannot assume any responsibility for malfunctioning or damage that may result from incorrect operation.

1. Unplug the power supply cord when connecting or disconnecting the PG-1005.
2. Do not turn off the power when a PROM is inserted. The PROM and/or the PG-1005 may be damaged.
3. Avoid using the PG-1005 in a noisy electrical surrounding.
4. Please take care that the operating temperature range (10 to 35 degree c.) and the humidity range (20 to 80 % RH) are not exceeded.

The PG-1005 consists of four parts.

1. The PG-1005 Adapter
2. The PG-1005-1 socket adapter for 64 pin shrink dip
3. The PG-1005-2 socket adapter for 64 pin flat pack

Connecting PG-1005 and PG-1000

1. Turn off the PG-1000 and unplug the power cord.
2. Pull out the lock pin located on the front panel to open the PG-1000.

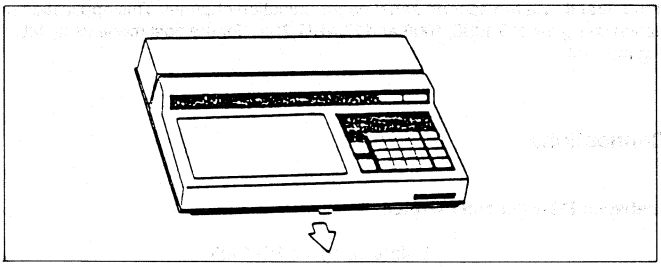


Figure 1.

3. Lift the case over to the upright position. If the personal module has been connected, remove the module.

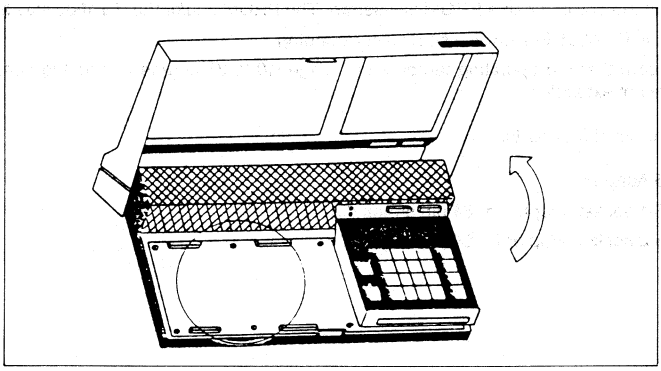
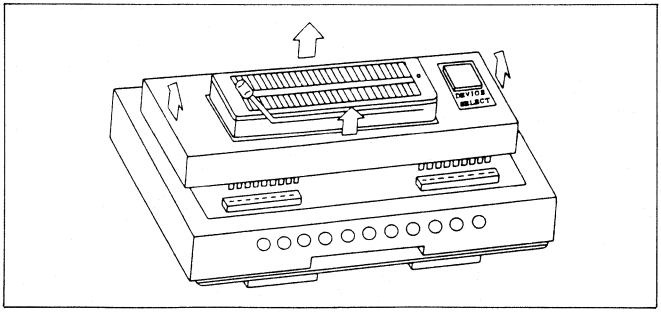


Figure 2.

4. If the socket adapter has been connected, remove it.



5. Fit together the PG-1000 and the PG-1005 by carefully looking to the four connectors.

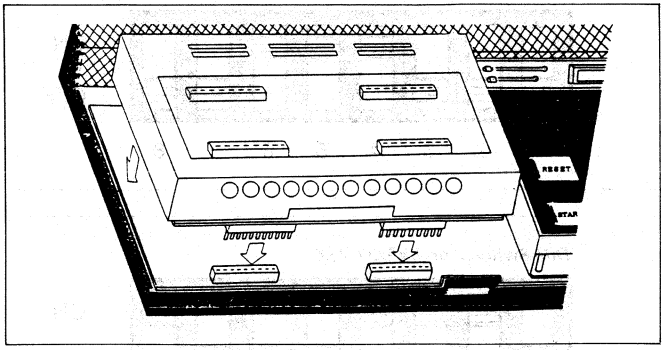


Figure 4.

6. Close the PG-1000.
7. Insert a socket adapter, observe carefully for a correct connection.

Comments for the socket adapter:

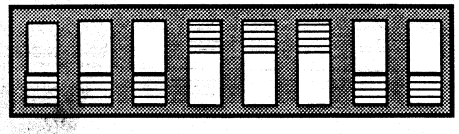
Device select bottom:	Read out the on silicon signature of the used device.	
Inserting the devices:	Shrink dip:	pin 1 is indicated with a dot on the direction indicator on the cover of the PG-1005-1.
	Lever Position:	insert = vertical fixed = horizontal
	Flat pack:	pin 1 is indicated open socket by pushing horizontal to the lever on the left side of the socket.

2.2 Connection between PG-1000 and EVAKIT 75X

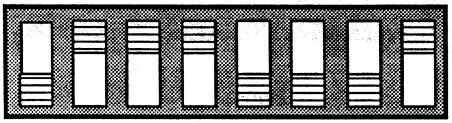
Please use the RS232C cable delivered with the EVAKIT and fix it to the serial channel of the PG-1000 and the serial channel 2 of the EVAKIT 75X.

2.2.1 Setting of the DIP-Switches on the PG-1000 and EVAKIT 75X when using the EVC communication program (IBM based)

DIP-Switch 1 on EVAKIT 75X

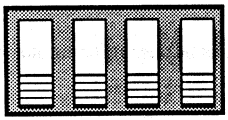


DIP-Switch 2 on EVAKIT 75X

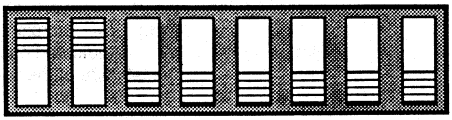


The EVAKIT CH2 is automatically set to the same baud rate etc. as CH1 after reset. The correct DIP-Switch setting of PG-1000 using the EVC software is as follows:

DIP-Switch 1



DIP-Switch 2

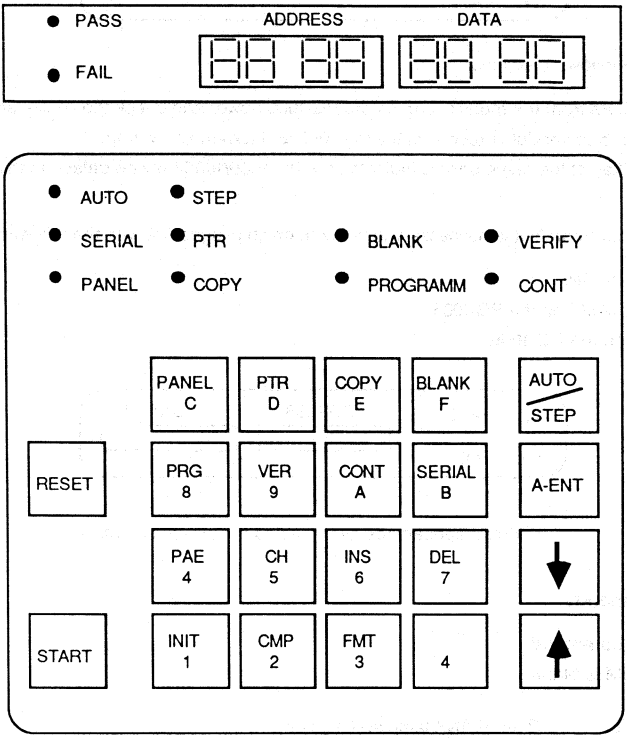


On the EVAKIT 75X two switches select between terminal and modem mode. These switches have to be set in the following positions:

CH1: modem mode
CH2: terminal mode

3. Command Description

3.1 Outline of front panel and keyboard



3.2 Description of stand alone mode

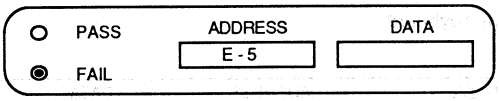
3.2.1 General

When the PG-1000 is turned on after a personal modul has been connected the panel display shows random data for a few seconds; then the AUTO and PANEL lamp turns on. The PG-1000 is always in stand alone mode after power up, which means that the device can be used without any terminal or host computer. The commands can be divided into two sections:

1. Control commands for data writing etc.
2. Panel commands to edit data in the memory buffer.

In the stand alone mode each operation can be terminated with the RESET key, which puts the system in the command wait state.

If an error occurs during the reset operation the following indication is given:

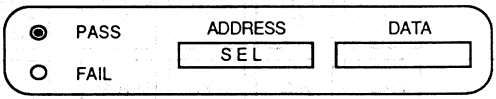


In this case perform following actions:

1. Remove the PROM from the module, turn off the PG-1000 power and check the socket adapter connections.
2. Connect the personal module properly to the PG-1000 and turn the power on.
3. Press the reset key. If the same error is indicated the PG-1000/1005 may be defect. In this case contact your dealer.

Before executing a commands it is neccessary to read the on chip signature with the following procedure:

1. Insert and fix a PROM
2. Push "Device select" on the PG-1005
3. Push "A-ENT" on the PG-1000



Correct appearance after reading the silicon signature

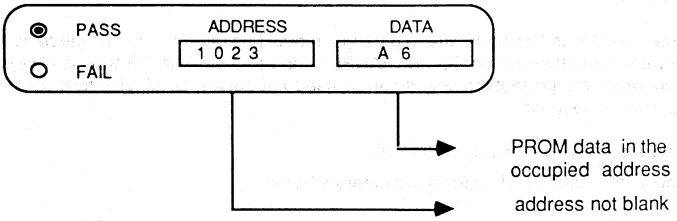
3.2.2 Control commands

1. **BLANK check command**

   Check if a PROM is blank

   Execution:
   a) Press **BLANK** (The BLANK lamp switches on)
   b) Press **START**

   If the PROM is blank the display shows the highest address and the contents "FF".
   The PASS lamp switches on.
   If data is written already in it, the following indication is given.



2. **COPY command**

Read all data from the PROM into the PG-1000 buffer memory.

Execution:
a) Press **COPY** (The COPY lamp switches on)
b) Press **START** (The display shows the load address and data)

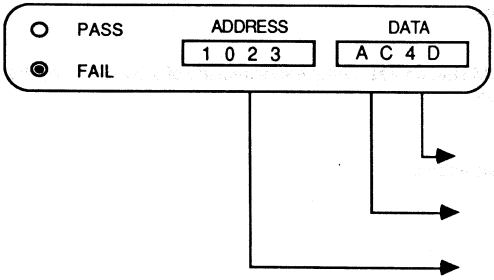
If the COPY command is terminated properly the PASS lamp switches on; in case of an error the FAIL lamp indicates some misoperation.

3. **PROGRAM command**

Write data from the PG-1000 buffer into the PROM. When the writing has been completed, the command performs a verification check.

Execution:
a) Pess **PRG** (The PROGRAM lamp switches on)
b) Press **START** (The display shows the just written address and data)

If the PROGRAM command is terminated properly the PASS lamp switches on; in case of an error the following indication can be seen on the panel.



4. **VERIFY command**

Compare contents of a written PROM with the contents of the memory to check proper programming.

Execution:
a) Press **VER** (The PROGRAM lamp switches on)
b) Press **START** (The display shows the checked address and data)

If the VERIFY command is terminated properly the PASS lamp switches on; in case of an error the a.m. error output can be seen on the panel.

5. **CONT command**

This command executes the BLANK, PROGRAM and VERIFY commands one after the other.

Execution:
a) Press **CONT** (The CONT lamp switches on)
b) Press **START** (The display shows the just written address and data)

During the execution of the CONT command, both the CONT and the lamp showing the currently executed command are lit. In case of an error to the description of the command just indicated; exit the CONT command by pressing the RESET key.

3.2.3 Panel commands

Panel commands are used to edit the data in the PG-1000 buffer memory, set the range of addresses etc. The setting of the correct address range for a selected PROM is automatically done by the silicon signature.

Points to note for all panel commands:

Enter the panel commands state by pressing the PANEL key.

For all panel commands the PANEL lamp must be lit.

For addresses 4 bytes must be entered, for data 2 bytes must be entered.

Overwrite wrong information by just typing the correct data.

If an control command is entered, the panel mode is automatically cleared.

1. **PAE command**

   Check or change the currently set parameters of PROM start and end address and the PG-1000 buffer memory start address.

   a) Check currently set values

   Press **PAE**

   The display shows the PROM start address. After each pressing of the A-ENT key the display shows one after the other following indication.

   PROM start address
   PROM end address
   PG-1000 buffer memory start address
   END

   b) Chande current values

   Press **PAE**

   The display shows the current PROM start address. Key in the new PROM start address and press **A-ENT** to set the new value or accept the current value by pressing **A-ENT**.

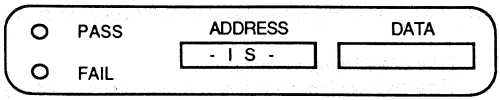
   The display shows the current PROM end address. Key in the new PROM end address and press **A-ENT** to set the new value or accept the current value by pressing **A-ENT**.

   The display shows the current PG-1000 buffer start address. Key in the new buffer start address and press **A-ENT** to set the new value or accept the current value by pressing **A-ENT**.

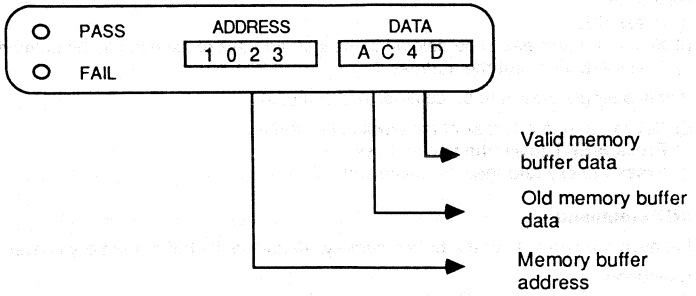
2. **CH command**

   Change or check the PG-1000 memory buffer data.

   Execution:
   a) Press **CH**. The panel shows following indication.



b) Key in the address of data to be changed or checked and set it with **A-ENT**. The display then shows:

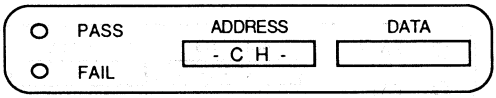


c) Key in the new data. Set the new data by pressing the ↑ or the ↓ key. The display then shows the address incremented or decremented by one and the belonging data, which can be modified or accepted by pressing ↑ or ↓.
d) Terminate the **CH**-command by pressing **RESET**.

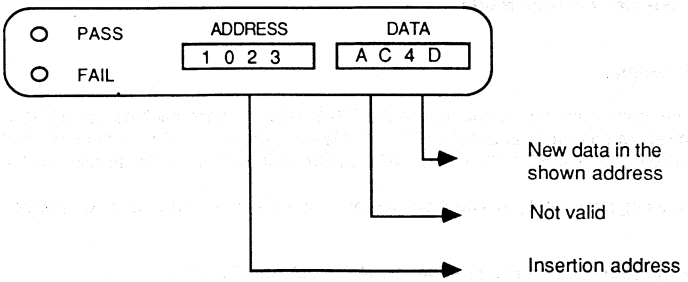
4. **INS command**

Insert new data into the data string in the buffer memory. In this mode, the address of all the data in and after the inserting address is incremented by one. Therefore, the last address of the buffer memory is deleted.

Execution:
a) Press **INS**. The panel gives the following indication.



b) Set the selected address with **A-ENT**.
c) Key in the data to be inserted. Only the low byte of the DATA panel shows the valid data.
d) Set the new value with ↑. The display shows the following indication.



5. **DEL command**

Delete single or block data in the buffer memory.

Execution:
a) Press **DEL**.
b) Key in the address to be deleted or the start address of the block to be deleted.
c) Press **A-ENT** to set the address.

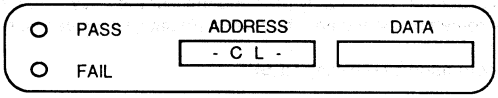
If only a single data is to be deleted, skip d) and e).

d) Key in the end address of the block to be deleted.
e) Press **A-ENT** to set the end address.
f) Press ↑ to execute the DEL command.

6. **INIT command**

Initialize the contents of the buffer memory. All data in the buffer memory is overwritten with 0FFH.

Execution:
a) Press **INIT**. The display shows following.

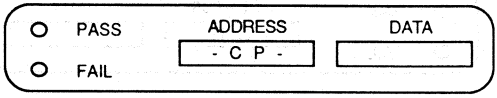


b) Press ↑ to execute the INIT command.

7. **CMP command**

Reverse all data in the buffer memory bitwise.

Execution:
a) Press **CMP**. The display shows following.



b) Press ↑ to execut the CMP command.

3.3 Console operation mode

3.3.1 General

This chapter describes how to handle the PG-1000 from a host machine using EVC communication software. The Baud rate of channel 1 and 2 of EVAKIT 75X is set to the same value after reset. For the correct dip-switch setting of EVAKIT 75X and PG-1000 using EVC communication software please refer to chapter 2.2.

When using a terminal pay attention that some control commands and up- or downloading of files are not possible.

3.3.2 Communication start between EVAKIT 75X and PG-1000

After power on, the PG-1000 is always in the stand alone mode. First one has to read the Silicon signature off the used device with the following sequence:

a) Press **DEVICE SELECT** on the PG-1005.
b) Press **A-ENT**.

If the device is well connected and o.k. the PASS lamp switches on.

Select the serial mode by pressing **SERIAL** on the PG-1000. To indicate the serial mode the PANEL lamp switches off and the SERIAL lamp switches on. Start the EVC communication software and wait for the command prompt (>). Start PROM programmer mode with **PGM**. Type in **I**.

Note: In the serial mode only big letters from the host are accepted.

The PG-1000 answers with the command prompt (*). The PG-1000 is now waiting for commands.

3.3.3 Command description

1. **A-command**

Command Format: ***A,s,e,r<CR>**
s: PROM start address
e: PROM end address
r: PG-1000 buffer start address

All inputs must be hexadecimal digits.

Function: Set the parameters for the PROM control instructions (W and V commands) to reduce inputs when using always the same parameters. After reading the silicon signature after startup the parameters are set to the min-max values of the used device.

Example:
***A0,3FF,0<CR>**
***W <CR>**
***W0,3FF,0 <CR>**

Both commands have the same function.

2. **Y-command**

Command Format: ***Y<CR>**

Output Format:
```
0000 1FFF 0000
     |    |
     |    PG-1000 buffer start address
     PROM start address
PROM end address
```

Function: Display the parameters set by the "A" command or the silicon signature reading.

Example:
***Y<CRH**
0000 1FFF 0000
*

3. **E-command**

Command Format: ***Er<CR>**

r: PG-1000 buffer memory address
If r is omitted, address "0" is assumed.

All inputs must be hexadecimal digits.

Function: Display the buffer address, data and a promt "-". The data can be changed in the following way.

a) Two hex inputs: change the data, increment the address by one and display the increment address contents.

b) Space: Display the next address contents without change.

c) Return: Terminate the E-command.

Example:

```
*E10<CR>
0010 55-12 55- <space> 55 78- <CR>
*
```

4. **F-command**

Command Format: ***Fs,e,d<CR>**

s: PG-1000 buffer memory start address
e: PG-1000 buffer memory end address
d: Initialization data

All inputs must be hexadecimal digits.

Function: Fill all memory addresses in the specified range with the same data.

Example:

```
*F30,40,AB<CR>          ;Fill 30H to 40H with 0abh
```

5. **O-command**

Command Format: ***Os,e<CR>**

s: PG-1000 buffer memory start address
e: PG-1000 buffer memory end address

All inputs must be hexadecimal digits.

Function: Display the contents of the buffer memory in the specified address. If "e" is omitted display only the data at "s". If "s" and "e" is omitted display only the contents of address 0.

Example:

```
*O100,105<CR>
0100 FF FF FF FF FF FF
*
```

6. **Z-command**

Command Format: ***Z<CR>**

Function: Checks if all data in the used PROM has been erased. If any data is not erased a "?" is displayed.

Example:

```
*Z<CR>
*                       ;All data erased

*Z<CR>'
?
*                       ;Not all data erased
```

7. **R-command**

Command Format: ***Rx,y<CR>**

x: Buffer memory start address
y: Buffer memory end address

All inputs must be hexadecimal digits.

Function: Read the contents of the PROM starting from address "0000" into the buffer memory from the specified start address to the end address.

If the end address is omitted, all data beginning from the start address is read into the PG-1000 buffer. If both, start and end address are omitted, the whole PROM contents is read into the PG-1000 buffer, starting from address "0".

Example:

```
*R<CR>                  ;Read the whole PROM contents
```

8. **W-command**

Command Format ***Ws,e,r<CRH**

s: PROM start address
e: PROM end address
r: PG buffer start address

All inputs must be hexadecimal digits.

Function: Write data from the PG-1000 buffer memory start address into the PROM area defined by the PROM start and end address.

If a parameter is omitted, it is assumed as to be set to "0".

If all parameters are omitted, the parameters set in the "A" command are used.

Example: ***W,0,100<CR>** ;same result
***W,,100**

9. **V-command**

Command Format ***Vs,e,r<CR>**

s: PROM start address
e: PROM end address
r: PG buffer start address

All inputs must be hexadecimal digits.

Function: Compare the data in the area between PROM start and end address with the data starting from the PG buffer start address.

If a parameter is omitted, it is assumed as to be set to "0".

If all parameters are omitted, the parameters set in the "A" command are used.

Example: ***A0,FF,1000<CR>**
***V<CR>**

and

***V0,FF,1000<CR>**

have the same result.

10. **L-command**

Command Format ***L<CR>**
PARTITION = **s,e**

s: EVAKIT buffer start address
e: EVAKIT buffer end address

All inputs must be hexadecimal digits.

Function: Load data from the specified address string in the EVAKIT 75X to the PG-1000 buffer memory starting at address "0".

Example: ***L<CR>**
PARTITION = **0,1FFF<CR>** ;Load 8k from EVAKIT

11. **P-command**

Command Format ***Ps,e<CR>**
BIAS = **d<CR>**

s: PG-1000 buffer start address
e: PG-1000 buffer end address
d: EVAKIT buffer start address

All inputs must be hexadecimal digits.

Function — Load data from the specified address string from the PG-1000 buffer memory to the EVAKIT 75X starting at the specified address.

Example:

```
*P0m1FFF<CR>
BIAS = 0<CR>          ;Load 8k from the PG-1000 to the EVAKIT buffer
```

3.3.4 System control character in console mode

Each control character must be pressed longer than 1 second.

CTRL-Z	Exit PGM mode (EVAKIT prompt is displayed)
CTRL-C	Exit EVC programm
CTRL-S	Temporary stops the display operation
CTRL-Q	Clear CTRL-S
Space	Stop display operation

3.3.5 Console command summary

A	Set the parameter of the W and V command
Y	Display the A command parameters
E	Change the contents of the PG-1000 buffer
F	Initialize the PG-1000 buffer
O	Show the contents of the PG-1000 buffer (hexdump)
Z	Check if PROM is blank
R	Read the contents of the PROM into the PG buffer
W	Shoot PROM with the PG buffer data
V	Compare the contents of the PROM and the EVAKIT
L	Load data from the EVAKIT 75X
P	Download data from the PG-1000 to the EVAKIT 75X
I	Set the PG-1000 to the console echo mode
T	Set the PG-1000 to the non echo mode (status after selecting)

4. Example for programming the 75P108 using PG-1005 adapter

a) Insert the 75P108
b) Push **DEVICE SELECT** on the PG-1005
c) Push **A-ENT** on the PG-1000
d) Push **SERIAL** on the PG-1000

Commands on the console

e) Start EVC communication software
C: **EVC<CR>**
f) 0>**PGM<CR>** ;Select echo mode
g) **I<CR>** (Must be a big letter.)
h) ***F0,1FFF,FF** ;Initialize the buffer memory
i) ***L<CR>** ;Load 8k program from the EVAKIT
PARTITION = **0,1FFF**
j) ***W<CR>** ;Shoot PROM and verify

5. Appendix

Error messages on the PG-1000

Ind.	Error type
E-1	Memory error
E-2	Parity error
E-3	Format error
E-4	Address entry error
E-5	PROM unit not inserted
E-6	Format unspecified
E-7	Paper tape verification error
E-11	Vpp (5V) incorrect
E-12	Vpp (25V or 21V) incorrect
E-13	Vcc (5V) incorrect
E-14	Reversed insertion or PROM not proper connected
E-15	Other errors

μPD7811 Microcomputer for CRT Terminal Control

This Application Note presents a basis CRT Controller Design using the NEC 8 Bit Microcomputer μPD7811 / C11 / C14 and the NEC Dual Port Video RAM μPD41264

Contents:

Person to contact: R. Cherrington
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Product Descriptions
μPD78C11 / C12 / C14
μPD7809

Related Products

μPD78C11 / C12 / C14	8-Bit Microcomputer	CMOS
μPD7809	8-Bit Microcomputer	NMOS

Introduction

This application note presents a very basic CRT controller design. This CRT terminal application uses simple and unique bit-mapped graphics without the use of a dedicated CRT or graphics display controller chip. The monochrome CRT terminal has a pixel resolution of 640 columns by 250 rows.

The following pages describe how to use the μPD7811 (one of the NEC μPD7800 high end single chip microcomputers) in a control environment. The μPD7811 controls data written to and transfered out of an NEC μPD41264 dual port video RAM (DRAM) as well as generates all the necessary synchronization and blanking for video displayed on a CRT. The μPD7811 also performs the necessary character processing from its internal USART and polls an ASCII encoded keyboard via one of its five on-board parallel ports. This application note includes μPD7811 source code, a hardware schematic, and software and hardware descriptions.

Traditional CRT Terminal Controller

Figure 1 shows that traditionally, CRT terminal controller designs included a CRT controller chip (CRTC), separate character ROM, screen RAM, USART for serial I/O, parallel I/O capability for keyboard input, USART input and output, and a controlling microprocessor providing data streams to the CRT controller, aided by interrupts and a DMA controller (DMAC). Interface hardware between the microprocessor and its peripherals, as well as the video dot clock sync and blanking circuitry, were required. This resulted in a fully functional, although dedicated, microcomputer system including the controlling firmware for the microprocessor.

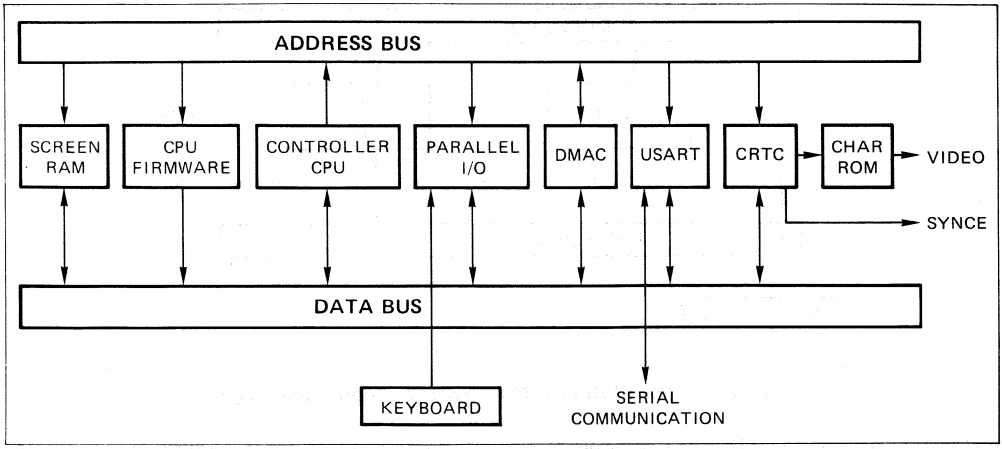


Figure 1. Block Diagram of a Traditional CRT Terminal Controller

Basic Operation of Traditional CRT Terminal Controller. Upon power-on, the CPU (controlling microcomputer) must initialize the CRTC, the USART, the DMAC, and clear the screen RAM (usually with ASCII 20H or blanks), as well as its own interrupt structure. The CRTC then begins demanding rows of character codes to be used as the basis for generating addresses to the character ROM. The character ROM has preprogrammed patterns for the display of each letter. The DMA controller provides data to the CRTC. The CPU has already sent the proper addresses and count to the DMA controller, usually after the CRTC has interrupted the CPU demanding more data.

In the mean time, the CPU will poll the USART checking for a character from the host system. The keyboard is also polled during this time for any key pressed. If, in either case, a character is available, it must be processed. The character is first checked for a control function like backspace, clear screen, or escape sequence, etc. If the character is a displayed character, its character code is written into the screen RAM in its correct screen position, as kept track of by the CPU. The cursor address is then updated and its new position is sent to the CRTC. The CRTC generates the necessary horizontal and vertical syncs and the active video signal.

Bit-Mapped Graphics Architecture

Many of the present state of the art CRT terminals are designed based upon raster scanned bit mapped graphics architecture. This architecture differs from the traditional CTRC/character ROM design in that an actual bit for bit (pixel for pixel) screen memory image is scanned and displayed continuously. This is in contrast to character display codes saved in a small screen RAM and used by a CRTC in conjunction with a character ROM. Bit-mapped techniques have become common as high density, low cost memory became available and high performance graphics display controllers (GDCs) were developed.

Bit-mapped designs allow for more flexibility as demands for high resolution graphics and mixed text/graphics displays increase. The two key functional times in this CRT terminal design are the μPD7811 microcomputer (controller CPU) and μPD41264 dual port video RAM (bit-mapped memory). Individually, both of these devices have unique features and, when combined, form the basis for a very simple, yet powerful design. The following is a brief description of each. Figure 2 shows the block diagram of a CRT terminal controller designed with the μPD7811 and μPD41264 VRAM.

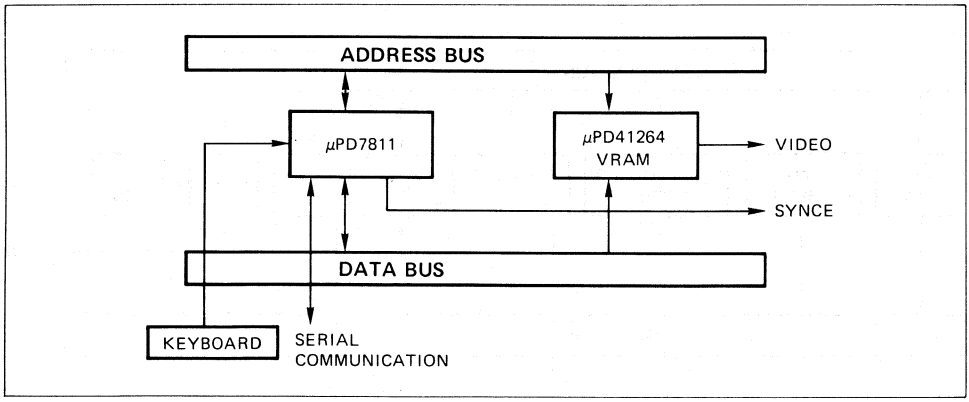


Figure 2. μPD7811 / VRAM CRT Terminal Controller Block Diagram

μPD7811 Controller CPU. The μPD7811 single chip microcomputer is a high end stand-alone microcomputer implemented from the basic 7800 architecture. The μPD7811 contains the following functional elements (figure 3):

[ ] Dual register set similar to that of a Z-80
— data swapping between registers and alternates during critical situations and in interrupt servicing
[ ] 8- and 16-bit accumulator
[ ] 8-bit vector register
[ ] Six 8-bit general purpose registers
[ ] 16-bit ALU
[ ] Internal data bus that allows operations on 8 or 16 bits

[ ] 40 I/O port lines
[ ] Full duplex USART
[ ] 8-channel / 8-input A/D converter
[ ] 16-bit multifunction timer/event counter
[ ] Two programmable 8-bit interval timers
— may be cascaded to create an 8-bit timer with an 8-bit prescaler
[ ] Two zero-crossing detection inputs

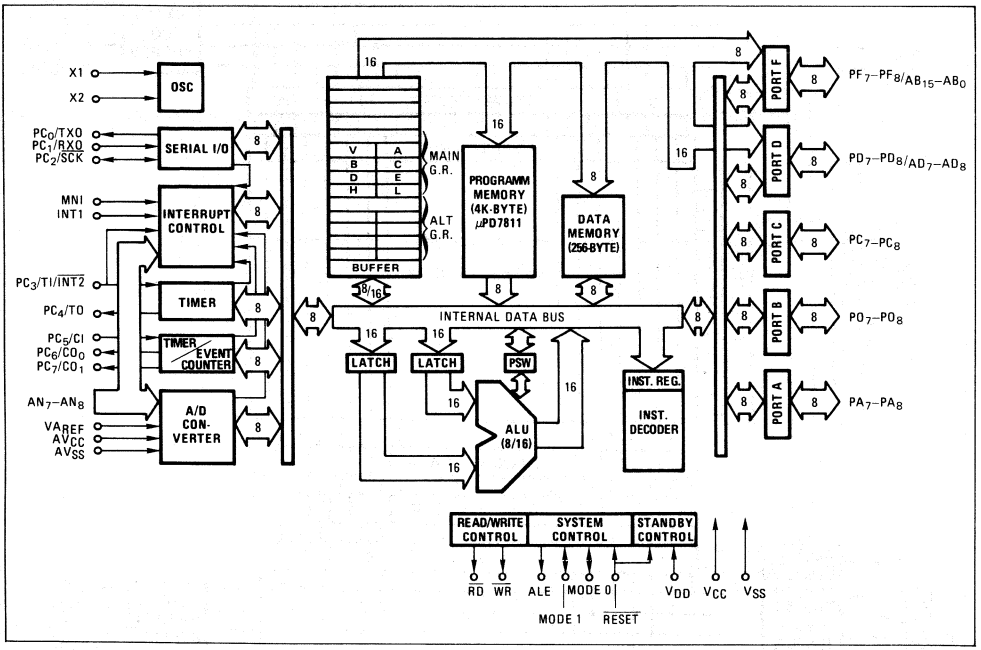


Figure 3. μPD7811 Block Diagram

The μPD7811 instruction set consists of 258 instructions featuring 16-bit data transfers, 16-bit shift and rotate operations, 8 by 8 multiply operations with 16-bit results, divide operations (16-bit dividend 8-bit divisor, 16-bit quotient, 8-bit remainder, and 16-bit index operations. Operating with a 12 MHz clock (15 MHz on the μPD7811H), the cycle time is 1 microsecond.

Twelve addressing modes further enhace programming efficiency and flexibility. Also, the μPD7811 has a 6-level prioritized, vectored interrupt structure, 256 bytes of RAM, 4K bytes of ROM, direct addressing of up to 64K bytes of external memory, and is available in 64-pin QUIP or 64-pin shrink-DIP packages. A CMOS version of the μPD7811, the μ78C11 is also available. A 16K byte ROM CMOS part, the μPD78C14 is under development. An additional package, the 64-pin flat pack is available for the CMOS part. Prototyping for this CRT controller was done using a μPD78PG11, which is a μPD7811 with a 28-pin JEDEC PROM socket mounted on top of the μPD7811. This convient configuration allowed testing of the software burned in a 2732 4K byte EPROM, without additional external memory interfacing.

μPD41264 Video RAM. The NEC μPD41264 is the natural choice of the raster scan memory because of its asynchronous internal data register with shift out control and random access memory port.

The NEC μPD41256 is a 262,144-bit dual port memory with a 64K x 4 dynamic RAM port and a 256 x 4 serial read port. The video memory is organized internally as 256 rows of 1024 dynamic memory cells. A by 4 data path, page mode access capability, and the column address strobe (CAS) before row address strobe (RAS) refresh function is combined with standard timing specifications to make the RAM port of this device a look-alike to the μPD41464 standard 64K x 4 DRAM. Figure 4 shows the functional block diagram of the μPD41264.

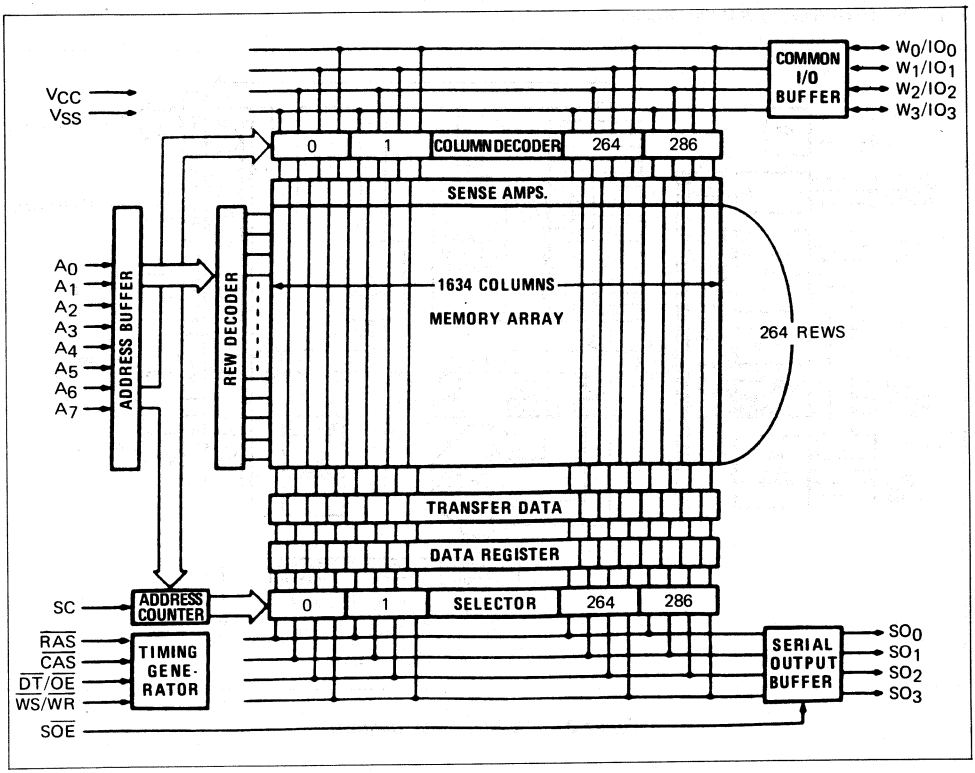


Figure 4. Functional Block Diagram of the μPD41264

A masking function called write-per-bit is added to this port. During write operations, mask data is sampled on the I/O lines at the time RAS goes low provided write enable (WE) is low at the same time. A mask data bit value of zero inhibits modification of the corresponding bit during that memory write cycle.

The 1024-bit serial register is organized as 256 words of 4 bits each and data is read out with each serial clock, 4 bits at a time. Data output from this serial register is not shifted, but selected through an 8- to 256-bit multiplexer which is controlled by a serial address counter. This counter is incremented with each shift clock (SC) clock cycle.

The serial register is updated from the memory array during the data transfer cycle, the only time in which the two ports must be synchronized. The data transfer cycle resembles a standard RAS/CAS access cycle where the RAS address defines the row to be transferred, and the CAS address defines which of 256 bits will be selected to begin serial output. This column pointer control is a unique feature of the video RAM that allows simplified smooth scrolling.

An added clock signal, data transfer (DT), is now required to first select the data transfer cycle. This happens when DT goes to a low level when RAS goes low (actually, the rising edge of DT causes the transfer to happen). The DT signal shares the same pin with the output enable (OE) pin. This update of the serial register can be accomplished in a single pixel display cycle during the active screen display time. This real time data transfer feature, together with the pointer control feature, allows more flexibility in the organization of video systems than ever before.

System Hardware

Figure 5 shows the hardware configuration for the μPD7811 / μPD41264 CRT terminal controller. Two key items are able to greatly reduce the amount of hardware and controlling software required in the design and operation of this CRT terminal controller. The first item is the amount of functionality in the μPD7811 and the second item is the dual port and internal counters in the μPD41264.

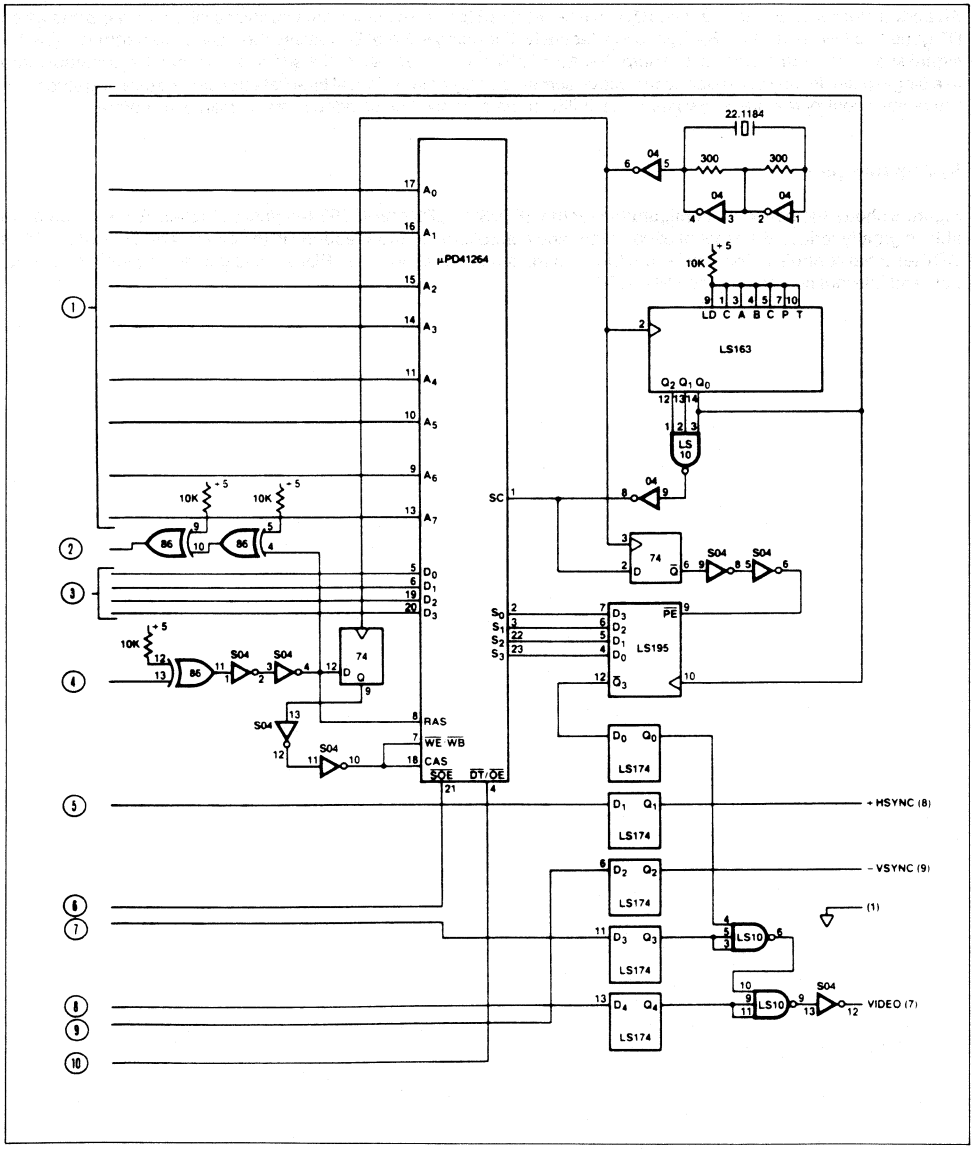


Figure 5. Configuration for μPD7811 / μPD41254 CRT Terminal Controller *(Right side)*

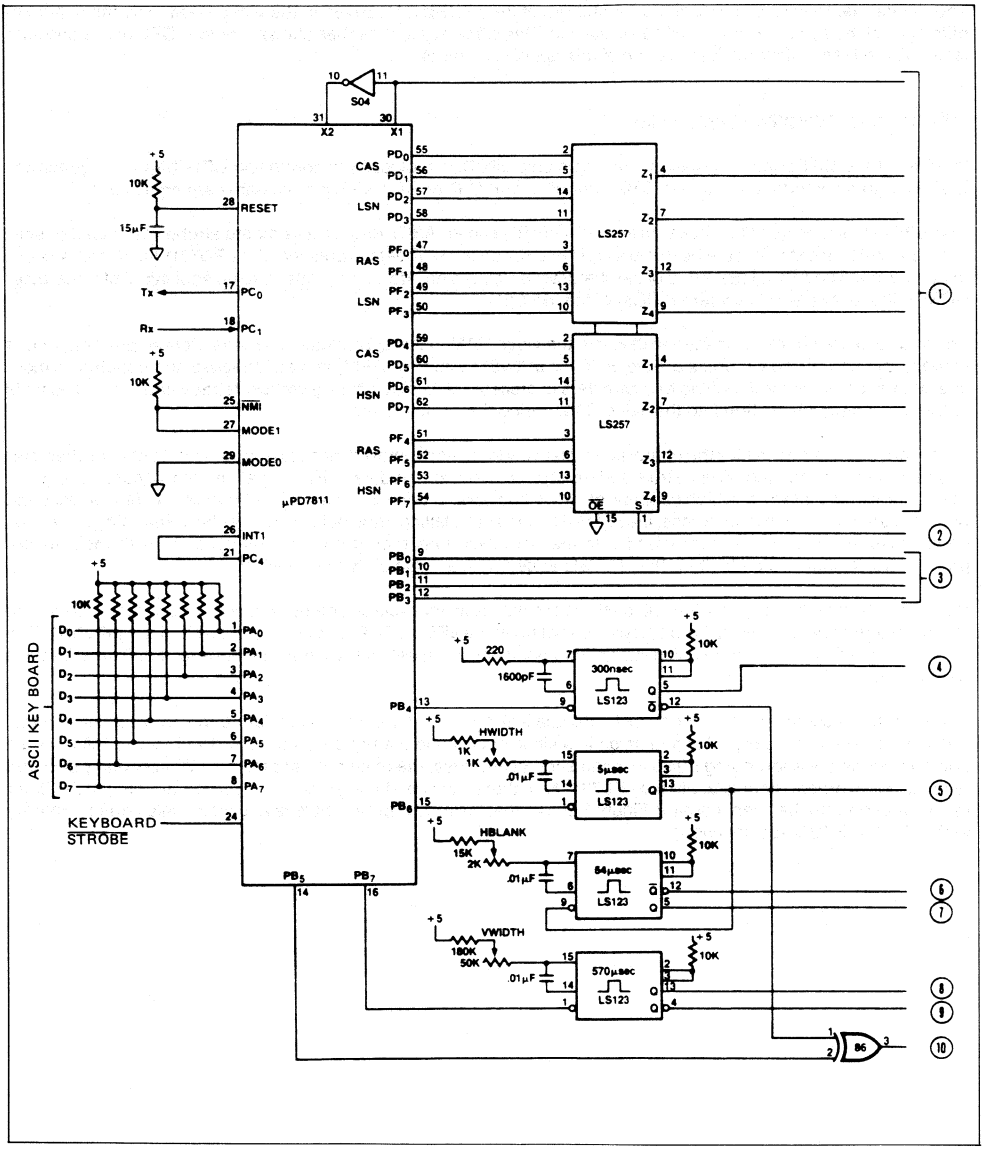


Figure 5. Configuration for μPD7811 / μPD41264 CRT Terminal Controller *(Left side)*

The design demonstrates most of the capabilities of the μPD7811. However, in the video RAM, only the minimum efficiency of the asynchronous dual port is utilized. This efficiency approaches almost 100 % in CPU accessing time (see "Video RAM Efficiency" at the end of this application note).

μPD7811 and μPD41264 Functionality

In a traditional CRT controller design, the single chip μPD7811 replaces the controlling CPU, baud rate generator, USART, parallel interface controller, DMA controller, and CRT controller chip or its hardware equivalent.

The μPD7811 scans its I/O devices, its internal USART, and an ASCII encoded keyboard strobe connected directly to the μPD7811. If data from the host is available or a key on the keyboard is pressed, the μPD7811 must then process the character. The ASCII HEX value of the character is used as an addressing index into an equivalent of a character ROM in the software in the internal ROM of the μPD7811.

The image of the character is then written into the video RAM in the appropriate locations. Screen position, end of screen, and end of line variables are stored in the μPD7811's internal RAM, and are checked and updated as more characters are processed. Polling for characters to process takes place during vertical retrace and, if a character is to be processed, it is written into the video RAM during the same retrace.

Since the processing time for the character is relatively long, horizontal retraces are suppressed until the character image is written into the video RAM and screen pointers are updated. Horizontal retrace occurs during the horizontal retrace interrupt which happens on the falling edge of timer zero which has been set up for a 64 usecond period. During the horizontal retrace, the μPD7811 sends out a falling edge pulse to a one-shot generating a five usecond pulse which, when synchronized with the dot clock, becomes the horizontal sync. As with most microprocessors, it would be difficult for the μPD7811 to generate a pulse of that short width.

During the horizontal retrace interrupt, a μPD41264 data transfer cycle is executed. A column address of zero and the current dot row address to be displayed are set up at the μPD41264 from port D and port F of the μPD7811. The μPD7811 sends out a low going pulse into another one-shot which will emulate the sequence for a 300 ns write cycle (RAS/-MUX-CAS/).

Vertical retrace occurs every 251 horizontal retrace interrupts, which are summed during the horizontal interrupts. The actual vertical sync signal is generated in a similar fashion to the horizontal sync and write video RAM, and a port line sends out a low going pulse to a one-shot. In this case, the one-shot generates a 570 μsecond pulse and relieves the overhead necessary in the μPD7811 for either a software timing loop or the reprogramming of the timer in conjunction with the event counter. Figure 6 shows that the resulting sync pulses and video data conform closely to the RS-170 NTSC specifications.

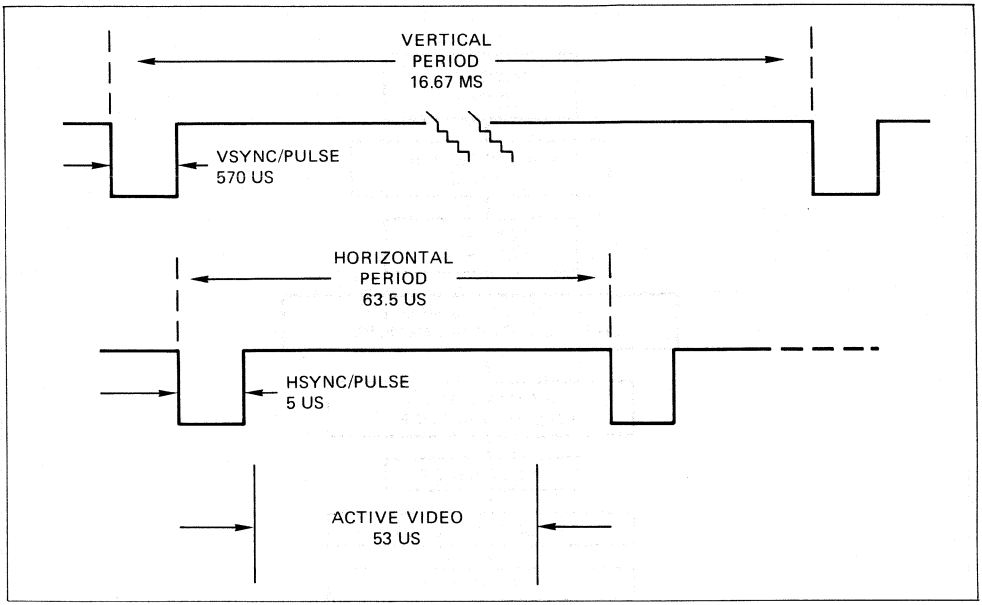


Figure 6. Approximate RS-170 Vertical and Horizontal Timing

The video interface coordinates the serial read data nibble from the video RAM with the vertical and horizontal sync signals with respect to the pixel or dot clock. A 22.1184 MHz crystal provides a clock to a 74LS163 counter, which in turn generates a /2 clock for the μPD7811. This 11.592 MHz clock, in conjunction with the divide by 384 internal divider and a /16 USART divider, results in an 1800 baud rate for the USART. This is accomplished with no software or hardware involved. Figure 5 shows that the 74LS163 also provides the basis for the μPD41264's shift clock (SC) and the parallel enable (PE) for the 74LS195 4-bit shift register, as well as its shift clock. All sync pulses and their video blanking counterparts are synchronized to the dot clock. The positive going horizontal sync and reverse video blanking were intended to display reverse video on an IBM PC monochrome monitor.

Controlling Software

The software listing and the flow chart in figure 7 show that upon a power-up reset, interrupts at the CPU are disabled via a DI instruction and a branch to the interrupt INT1 service routine is setup at interrupt vector location 10H.

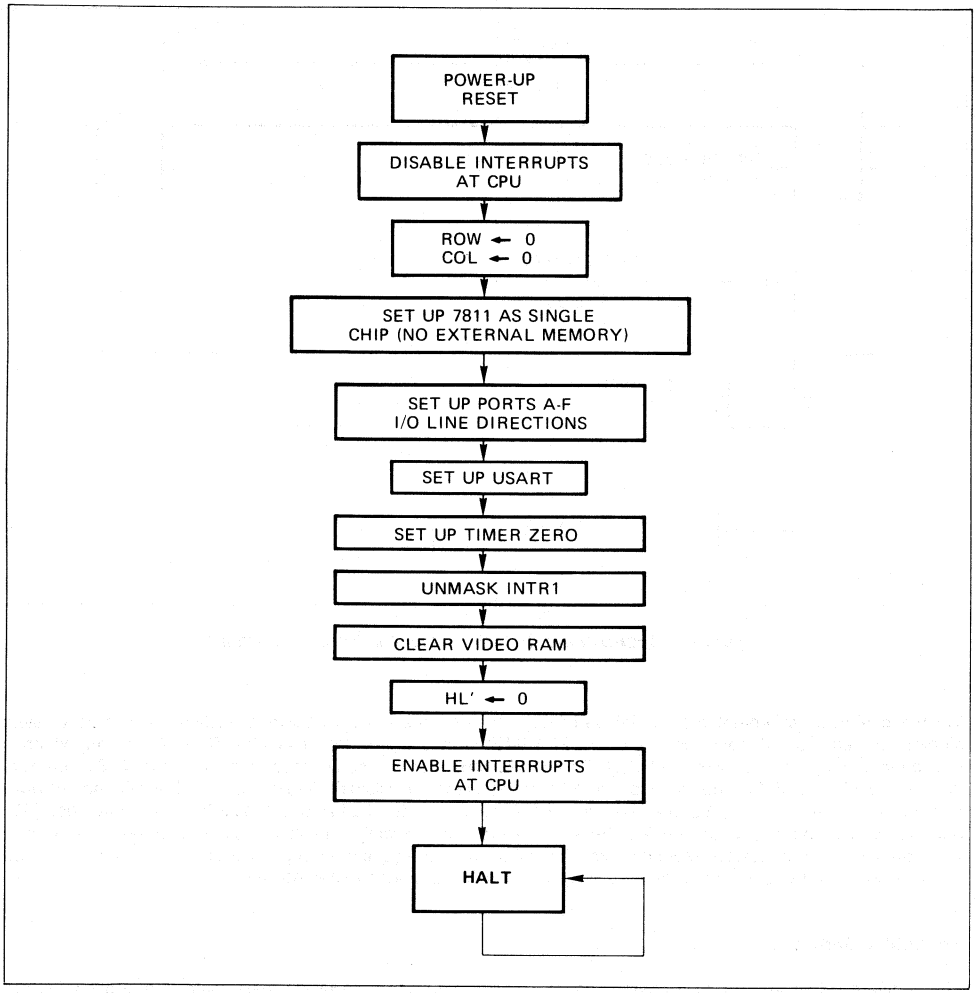


Figure 7. Power-Up Reset Flow Chart

Port Operation

In the μPD7811, the memory mapping (MM) register can designate the functions of ports D and F with respect to up to 60K bytes of external memory. Writing a 09H to the MM register sets up the μPD7811 as a single chip with ports D and F as output ports with the internal 256-byte RAM enabled. Mapping the μPD7811 as a single chip (no external memory) allows ports D and F to be available as I/O ports. Port D operates only as an 8-bit input or output port.

The remaining 8-bit ports A, B, C, and F have separately programmable lines that can be designated as input or output. Writing an FFH to port A's mode register (MA) sets port A connected to an ASCII encoded keyboard for all

inputs. Port B bits 0—3 write data to the μPD41264 video RAM. The remaining bits of port B generate write, vsync, hsync, and OE/DT pulses. Writing FFH to port B's mode register (MB) sets it for all output. Loading the mode C control register (MCC) with 33H enables the special function capability of port C. Port C, bit 0 is the USART serial data transmit, bit 1 is serial data receive, bits 2, 3, 6, and 7 are port lines, bit 4 is the timer zero output, and bit 5 is the counter input/falling edge interrupt 1 input.

Port D, previously set as output, is used as the column address port connected to both of the 74LS257 RAS/-CAS/ multiplexers. Since the μPD7811 is set up in a single chip mode, port F is available as a port and supplies the row address to the RAS/-CAS multiplexers. Writing 0 to the port F mode register (MF), sets the port to all output.

Baud Rate

The baud rate is established by using the μPD7811's internal /384 clock and the USART is initialized for a /16 clock, 8 bits/character, and no parity. This results in a baud rate of 1800 — a rate not commonly used, but available on most host systems — without the need for any additional external baud clock. If a higher baud rate is desired, additional circuitry is required. Note that final implementation of the software determines the character display rate with respect to the baud rate.

Writing a DH to the serial mode high (SMH) register, and a CEH to the serial mode low (SML) register selects the /384 baud clock input to the USART and the USART's initialization. These two serial mode registers set up the baud rate source (from either internal to the μPD7811 or from an external clock) and set the USART's operating characteristics.

Interrupts

Timer zero is set up to generate a square wave whose falling edge is used as the horizontal retrace interrupt line connected to the INT1 pin. Loading the timer mode register (TMM) with 0H sets up timer zero for square wave generation with the CPU clock/12 as its input. Loading a 1DH timer zero divider (TM0), results in a 64 microsecond period.

Writing an F7H and FFH to the interrupt mask registers (MKL and MHK), respectively, enables the INT1 interrupt mask bit and disables all other interrupt sources. Filling the entire RAM with all zeros then clears the video RAM. A zero clears the alternate register HL'. Next, interrupts at the CPU are enabled and the μPD7811 is then put into a HALT.

See the "NEC μPD7810/7811 NMOS Microcomputers User's Manual " for detailed information on programming the μPD7811.

The HALT mode is very important since it allows the CPU to be in a known and pre-defined state at the time of horizontal interrupt. Normally with microprocessors, when an interrupt occurs, the current instruction is allowed to finish before the interrupt servicing begins. This is irrespective of the type of instruction being currently executed or the current cycle the instruction execution is in.

Since the microprocessor is executing code continuously, it will be at any number of possible opcode or instruction phases at interrupt time and, with instructions varying widely in execution times, the interrupts will actually begin to be serviced a varying number of microseconds later with respect to the clock generating the interrupt signal.

The result of this in real time applications where the microprocessor is expected to generate signals in a precise timebase, is loss of signal consistency. If, at interrupt time, the CPU could be guaranteed to be executing the same instruction each time, this problem would be avoided. However, the μPD7811, as with normal microprocessors, has varying length instruction execution times. If the μPD7811 were to be halted after interrupts are enabled and returned to this halt state upon interrupt completion, an acceptably accurate interrupt generated signal could be generated.

When the μ7811 is in HALT mode, on-chip peripherals continue to function and any unmasked interrupts are able to release the chip from HALT mode. It is this technique, and the capability of the μPD7811 HALT mode, that generates a solid 64 microsecond horizontal sync pulse period.

Hsync interrupts

Every 64 microseconds a horizontal retrace interrupt occurs (except during vertical retrace and character processing) and is serviced by the hsync interrupt service routine. Figure 8 shows a flow chart for INT1 hsync interrupts.

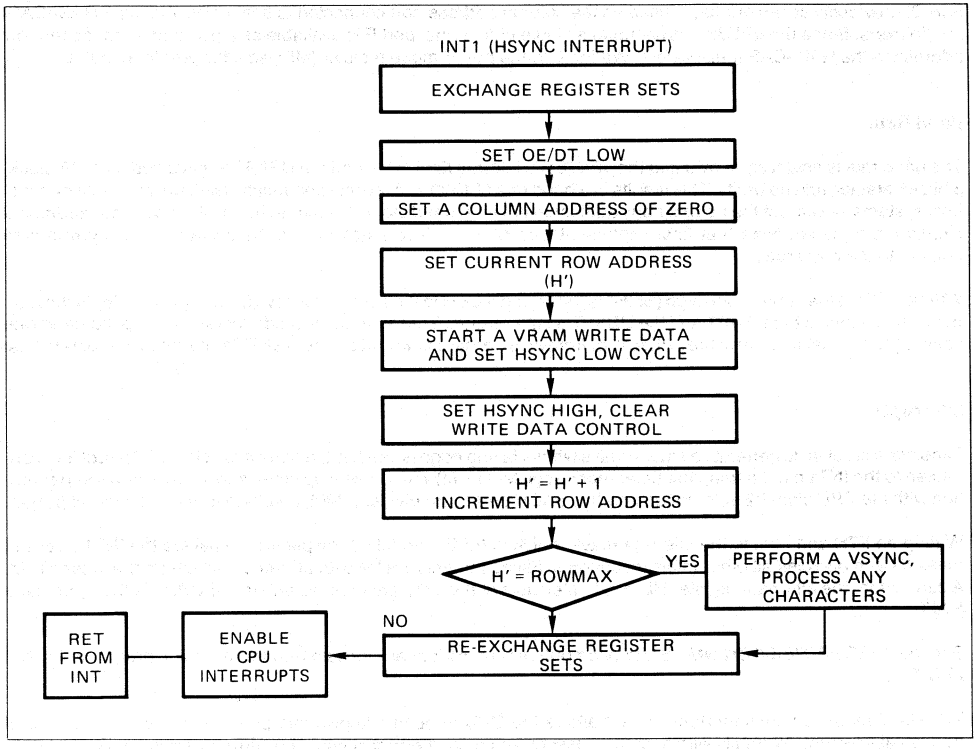


Figure 8. Flow Chart for INT1 Hsync Interrupts

This interrupt sets up a row of character bits to be sent to the CRT and stores this row value in the alternate register H', the register set used during the interrupt. The proper row address and column zero addresses are sent to the video RAM and a horizontal sync pulse is output. The display row is incremented and is checked for row 251, indicating the time for a vertical retrace. If not vertical retrace time is indicated, the CPU returns to a HALT state. If the current row pointer is at row 251, the processor sets up to generate a vertical retrace signal. During this time a check for a USART or keyboard character is made, and any subsequent character processing takes place. This is done during the vertical retrace time because there is not enough time between horizontal retraces for the μPD7811 to poll and process characters. Figure 9 shows the vertical sync flow chart.

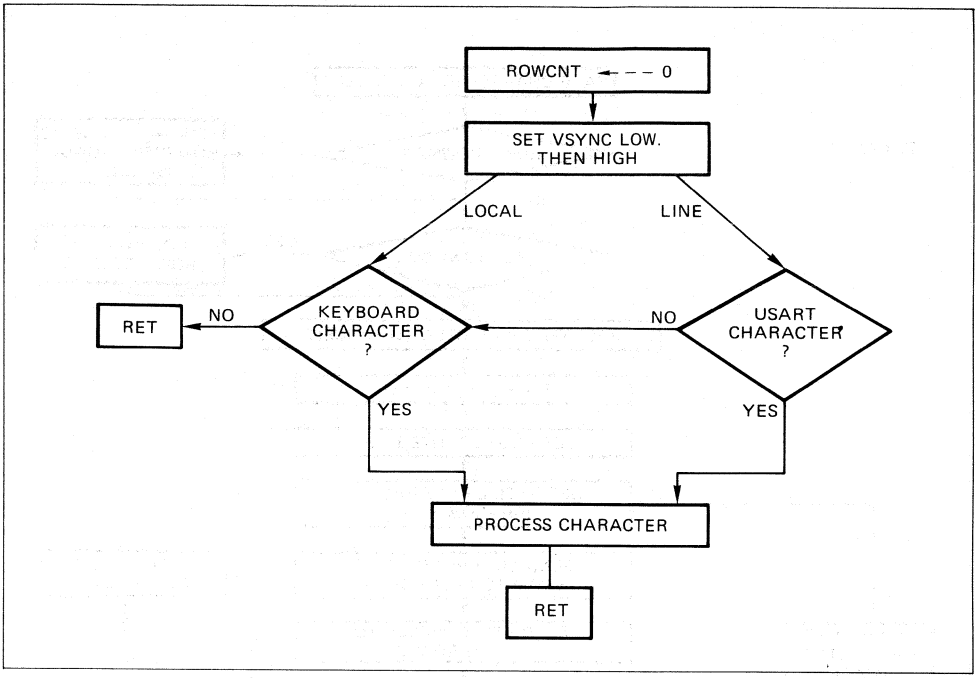


Figure 9. Vertical Sync Flow Chart

The HSYNC interrupt takes approximately 35 microseconds to execute and given 64 microseconds between successive interrupts, there is very little time to check and process characters. Actually, the horizontal SYNC is output about halfway between successive interrupt clocks, generating an apparently skewed HSYNC timing.

It is skewed with respect to the interrupt clock, but since both VSYNC and pixel data generation begin and are a function of HSYNC, all signals with respect to themselves are correct and are synchronized by the dot clock. If a character is detected from either the USART or the keyboard, it is first echoed back to the host via the USART, and then character processing begins.

Character Processing

Character processing in this design is very basic. For a specific design, more extensive processing and checking should be implemented and screen display pointers as described may be enhanced and modified for the specific application. Figure 10 shows a flow chart for character processing.

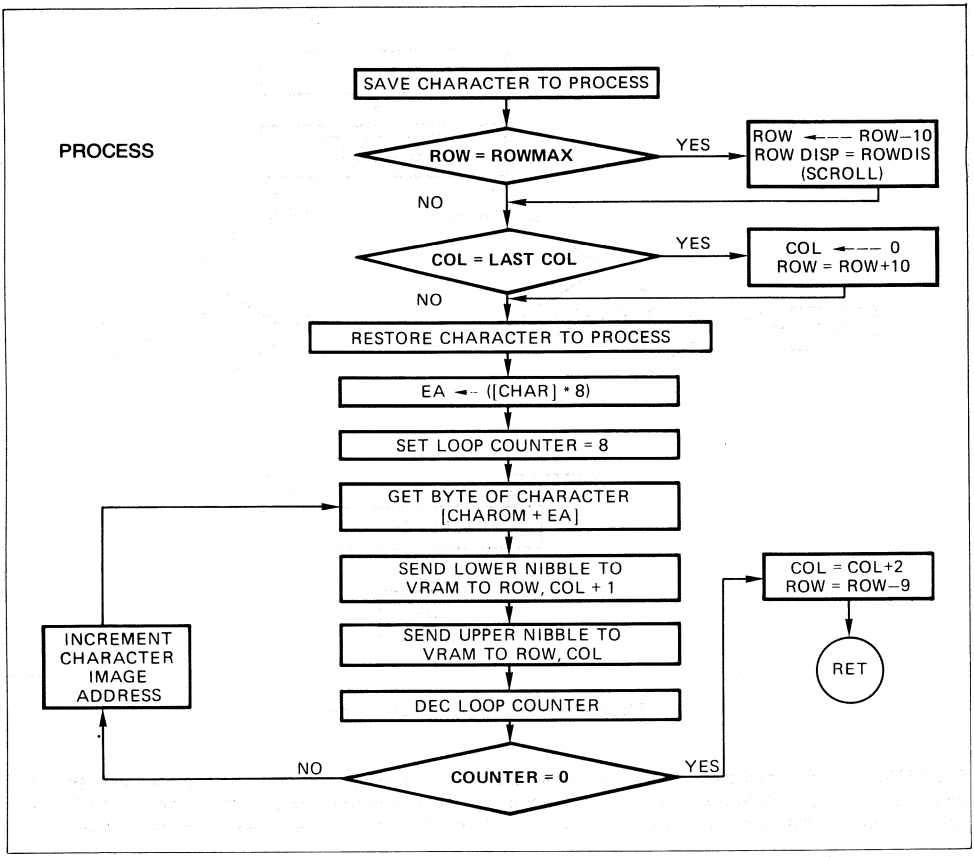


Figure 10. Character Processing Flow Chart

The routine PROCESS (in the accompanying software listing) checks for when the current ROW is egual to ROWMAX (indicating row 25). When the current ROW equals ROWMAX, the ROW pointer is adjusted by the character height and ROWDISP can be used by HSYNC to create a modulo 250 display pointer. This is provided as a video RAM starting display row location and is used as the basic scroll variable. After the row updates are made, the "end of line" or the "column" checks are performed by checking if the current COL is equal to COLMAX, while keeping in mind that the basic data unit sent to the video RAM is in nibble form.

If COL is equal to COLMAX, then the COL pointer must be set to the next character COL. Then the input character is multiplied by eight (the character images in ROM are setup as an 8 by 282 matrix) which provides the base index pointing to the character image pattern for the particular ASCII character. The character's upper left nibble to lower right bottom nibbles are then written to the video RAM as performed by the SCANCHR loop. When this loop is finished, ROW and COL are updated to point at the next character location (which can be used for a cursor).

Referring to the routine PROCESS, note how NEI and EQI check if an immediate byte is not equal (NEI) or equal (EQI) to the accumulator. Both these instructions are typical of the SKIP next instruction upon a specific comparison result. For example, in line 258 of the software listing, if ROWMAX is equal to the accumulator, the routine CALL SCROLL is executed. If ROWMAX is not equal to the accumulator, then the CALL SCROLL is skipped. This unique type of check and skip reduces code space and thus increases algorithm speed. The actual object code space used to implement the basic CRT control is 376 bytes which attests to the efficiency of the μPD7811. For comparison purposes, the character images in ROM occupy 1016 bytes and when added with the code space, this is 1392 out of 4092 bytes of ROM space available. This leaves much room for software enhancments and advanced bit mapped graphic features.

Video RAM Efficiency

The video RAM features fully asynchronous dual access capability to allow memory array operations independent of the serial read operation. In a video system, this capability results in an improvement in CPU access efficiency and overall system performance. Figure 11 shows that conventional DRAMs used in video systems limit the CPU's access to non-displayed periods during horizontal and vertical retrace (A). CPU efficiency, the percentage of the time during a frame display cycle that the CPU can update the video data, is approximately 16 %. A scheme to improve the efficiency of conventional DRAM video memory interleaves memory access cycles with display refresh cycles. At the expense of system complexity, CPU efficiency is boosted to 60 % (B).

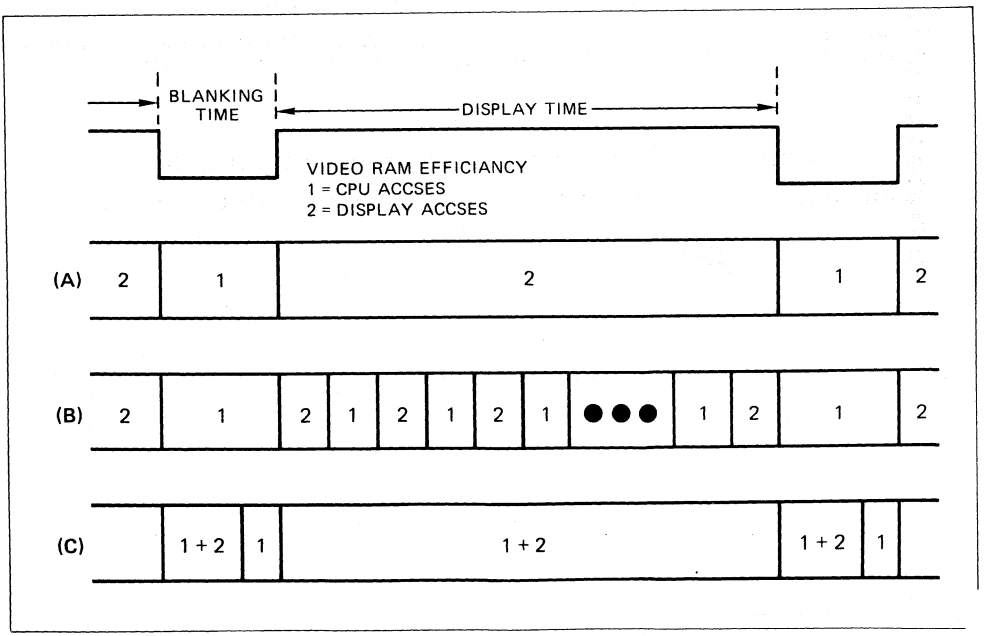


Figure 11. Video RAM Efficiency

Any video system that uses the μPD41264 dual port memory can instantly enjoy CPU efficiency of nearly 100 % a well as simplified design and fewer chips (C).

```
** UPD7811      ASSEMBLE LIST **

              (

E STNO ADRS OBJECT   M   SOURCE STATEMENT

     1                    ;       ************************************************************
     2                    ;       *                                                          *
     3                    ;       *       CONTROL PROGRAM FOR 7811 BASED CRT TERMINAL        *
     4                    ;       *                                                          *
     5                    ;       *                                                          *
     6                    ;       ************************************************************
     7
     8
     9                    ;               SYSTEM EQUATES
    10
    11           FFFF    STACK   EQU     0FFFFH  ;SET TOP OF STACK AT TOP OF MEMORY
    12           00FA    ROWMAX  EQU     250     ;MAXIMUM NUMBER OF DOT ROWS ON SCREEN DISPLAY
    13           0280    COLMAX  EQU     640     ;MAXIMUM NUMBER OF DOT COLUMNS ON SCREEN DISPLAY
    14
    15                   ;       S T A R T   O F   P R O G R A M
    16
    17
    18           0000            ORG     0       ;POWER ON RESET START ADDR LOC
    19
    20 0000 BA                   DI              ;PREVENT THE CPU FROM PROCESSING ANY SPURIOUS IN
    21 0001 D0                   JR      START   ;PROCEED TO START OF MAIN PROGRAM
    22
    23
    24
    25
    26           0010            ORG     10H     ;INTERRUPT INT1 VECTOR LOCATION
    27
    28 0010 4E44                 JRE     HSYNC   ;PROCESS A HORIZ SYNC DURING HORIZ BLANKING
    29
    30
    31                   EJECT
```

```
** UPD7811      ASSEMBLE LIST **

              (

E STNO ADRS OBJECT   M   SOURCE STATEMENT

     1                    ;       ************************************************************
     2                    ;       *                                                          *
     3                    ;       *       CONTROL PROGRAM FOR 7811 BASED CRT TERMINAL        *
     4                    ;       *                                                          *
     5                    ;       *                                                          *
     6                    ;       ************************************************************
     7
     8
     9                    ;               SYSTEM EQUATES
    10
    11           FFFF    STACK   EQU     0FFFFH  ;SET TOP OF STACK AT TOP OF MEMORY
    12           00FA    ROWMAX  EQU     250     ;MAXIMUM NUMBER OF DOT ROWS ON SCREEN DISPLAY
    13           0280    COLMAX  EQU     640     ;MAXIMUM NUMBER OF DOT COLUMNS ON SCREEN DISPLAY
    14
    15                   ;       S T A R T   O F   P R O G R A M
    16
    17
    18           0000            ORG     0       ;POWER ON RESET START ADDR LOC
    19
    20 0000 BA                   DI              ;PREVENT THE CPU FROM PROCESSING ANY SPURIOUS IN
    21 0001 D0                   JR      START   ;PROCEED TO START OF MAIN PROGRAM
    22
    23
    24
    25
    26           0010            ORG     10H     ;INTERRUPT INT1 VECTOR LOCATION
    27
    28 0010 4E44                 JRE     HSYNC   ;PROCESS A HORIZ SYNC DURING HORIZ BLANKING
    29
    30
    31                   EJECT
```

```
* UPD7811     ASSEMBLE LIST **


STNO ADRS OBJECT   M   SOURCE STATEMENT

  32
  33                START:
  34 0012 6900              MVI     A,0             ;SET UP TO INIT THE FOLLOWING VARIABLES:
  35 0014 707900FF          MOV     ROW,A
  36 0018 707901FF          MOV     COL,A
  37
  38 001C 6909              MVI     A,09H           ;SET UP MEMORY MAP REG AS A SINGLE CHIP
  39 001E 4DD0              MOV     MM,A            ;WITH PD=PF=OUT, INTERNAL RAM ENABLE
  40
  41 0020 04FFFF            LXI     SP,STACK        ;SET UP THE STACK POINTER AS PER EQUATE
  42
  43 0023 69FF              MVI     A,0FFH          ;SET UP PORT A
  44 0025 4DD2              MOV     MA,A            ;AS ALL INPUTS
  45
  46 0027 6900              MVI     A,0             ;SET UP PORT B
  47 0029 4DD3              MOV     MB,A            ;AS ALL OUTPUTS
  48
  49 002B 6933              MVI     A,33H           ;SET UP PORT C MODES: PC0-TX, PC1-RX
  50 002D 4DD1              MOV     MCC,A           ;PC2,PC3,PC6,PC7-PORTS, PC4-TO, PC5-CI
  51 002F 69FF              MVI     A,0FFH          ;SET UP PORT C PORTS AS ALL INPUTS
  52 0031 4DD4              MOV     MC,A            ;REFER TO SCHEMATIC FOR PORT C FUNCTIONALIT
  53
  54 0033 6900              MVI     A,0             ;SET UP PORT F
  55 0035 4DD7              MOV     MF,A            ;AS ALL OUTPUTS
  56
  57 0037 64810D            MVI     SMH,0DH         ;PHASE 384 CLK(@11.0592MHZ & /16 = 1800 BA
  58 003A 69CE              MVI     A,0CEH          ;X16 CLK, 8 BITS/CHAR, NO PARITY,
  59 003C 4DCA              MOV     SML,A           ;2 STOP BITS, TX AND RX ENABLE
  60
  61 003E 691D              MVI     A,1DH           ;SET UP TIMER ZERO
  62 0040 4DDA              MOV     TM0,A           ;FOR A 64US
  63 0042 648500            MVI     TMM,0           ;SQUARE WAVE
  64
  65 0045 6407F7            MVI     MKL,0F7H        ;KEEP ALL MASKABLE
  66 0048 6406FF            MVI     MKH,0FFH        ;INITR REQ MASKED EXCEPT INT1
  67 004B 40B400            CALL    CLRVRAM         ;CLEAR THE 41264 VIDEO RAM
  68
  69 004E 11                EXX                     ;SET UP ALT REG
  70 004F 6E00              MVI     H,0             ;H FOR COL DISPLAY COUNTER
  71 0051 11                EXX
  72 0052 AA                EI
  73                EJECT
```

** UPD7811 ASSEMBLE LIST **

```
E STNO ADRS OBJECT   M   SOURCE STATEMENT

    74
    75                   ;              M A I N  P R O G R A M  L O O P
    76
    77
    78                   WAIT:
    79 0053 483B                HLT                    ;HALT AND WAIT FOR INTERRUPT
    80 0055 FD                  JR      WAIT           ;RET FROM INTR AND THEN RE-HALT
    81                   EJECT
```

```
* UPD7811     ASSEMBLE LIST **        (                    )
             (                       )
STNO ADRS OBJECT   M   SOURCE STATEMENT

  82                 ;       ************************************************
  83                 ;       *                                              *
  84                 ;       *                  INT1                        *
  85                 ;       *     HORIZONTAL SYNC INTERRUPT ROUTINE        *
  86                 ;       *     GENERATED WHEN A TIMER HAS GENERATED     *
  87                 ;       *     A HORIZONTAL SYNC INTERVAL PULSE         *
  88                 ;       *                                              *
  89                 ;       * CHKS FOR ROWMAX HSYNC, AT WHICH POINT A VSYNC *
  90                 ;       * IS GENERATED. ALSO, BEFORE RET, A ROW OF DOTS *
  91                 ;       * IS OUTPUT TO THE SCREEN (I.E. A DATA TRANSFER *
  92                 ;       * CYCLE-PORT B ACTIVE).                        *
  93                 ;       *                                              *
  94                 ;       ************************************************
  95
  96
  97                 ;       PB4             PB5             PB6             PB7
  98                 ;       WR/START        OE/DT/ CNTL     HSYNC/          VSYNC/
  99
 100                 ;       PD      COL  ADDR
 101                 ;       PF      ROW  ADDR
 102
 103                HSYNC:
 104 0056 11                 EXX             ;USE ALTERNATE REG SET
 105
 106 0057 69DF               MVI     A,0DFH  ;BEFORE ANYTHING, SET OE/DT/
 107 0059 4DC1               MOV     PB,A    ;LOW
 108
 109 005B 6900               MVI     A,0     ;SET UP A COLUMN
 110 005D 4DC3               MOV     PD,A    ;ADDR OF ZERO ALWAYS
 111
 112 005F 0E                 MOV     A,H     ;COPY CURRENT ROW
 113 0060 4DC5               MOV     PF,A    ;INTO ACC AND SET ROW ADDR
 114
 115 0062 698F               MVI     A,8FH   ;SET UP TO INITIATE A WR VRAM PROCESS AND AN HSYNC/
 116 0064 4DC1               MOV     PB,A    ;WITH WR/START=OE/DT=HSYNC/=LOW, VSYNC=HI
 117
 118 0066 69FF               MVI     A,0FFH  ;SET HSYNC/
 119 0068 4DC1               MOV     PB,A    ;AND ALL OTHERS HI
 120
 121 006A 744601             ADI     H,1     ;ADVANCE THE DISPLAY ROW
 122
 123 006D 746EFA             NEI     H,ROWMAX;CHK IF A VERTICAL SYNC
 124 0070 407600             CALL    VSYNC   ;IS REQUIRED
 125
 126 0073 11                 EXX             ;RESTORE NORMAL
 127
 128 0074 AA                 EI              ;REENABLE INTRS
 129
 130 0075 62                 RETI
 131
 132                EJECT
```

```
** UPD7811     ASSEMBLE LIST **        (                    )

             (                              )

E STNO ADRS OBJECT   M   SOURCE STATEMENT

   133                 ;       ************************************************************************
   134                 ;       *                                                                      *
   135                 ;       * VERTICAL SYNC MODULE. REINIT DISPLAY PARAMETERS, GENERATE A         *
   136                 ;       * VSYNC/, THEN CHK FOR A USART OR KEYBRD CHAR-ECHO IT TO USER         *
   137                 ;       * AND PROCESS CHAR TO CRT DISPLAY.                                    *
   138                 ;       *                                                                      *
   139                 ;       ************************************************************************
   140
   141
   142 0076 BA         VSYNC:  DI                      ;SHUT OFF INTRS DURING CHAR WR TO VRAM
   143 0077 6900               MVI     A,0             ;SET UP TO
   144 0079 707904FF           MOV     ROWCNT,A        ;RESET THE HSYNC DISPLAY ROW COUNTER
   145
   146 007D 697F               MVI     A,7FH           ;SET UP TO SET VSYNC/ LOW
   147 007F 4DC1               MOV     PB,A            ;W/ ALL OTHERS INACTIVE
   148 0081 69FF               MVI     A,0FFH          ;NOW SET VSYNC/ AND
   149 0083 4DC1               MOV     PB,A            ;ALL OTHERS HI
   150
   151 0085 D2                 JR      KYBD            ;CHK FOR ANY I/O CHARS TO PROCESS
   152                 EJECT
```

```
UPD7811     ASSEMBLE LIST **        (                    )

            (                                )

STNO ADRS OBJECT   M   SOURCE STATEMENT

 153
 154
 155                    ;       ***************************************************************
 156                    ;       *                                                             *
 157                    ;       * MAIN USART AND KEYBOARD SCAN ECHO AND CHAR PROCESSING MODULE *
 158                    ;       *                                                             *
 159                    ;       ***************************************************************
 160
 161                    IOSCAN:
 162 0086 4849                  SKIT    FSR             ;CHK FOR USART RX INT REQ
 163 0088 CF                    JR      KYBD            ;IF NO USART CHAR, THEN CHK KYBD
 164 0089 4CD9                  MOV     A,RXB           ;READ CHAR FROM USART
 165 008B 077F                  ANI     A,07FH          ;REMOVE PARITY-KEEP LOWER 7 ASCII BITS
 166 008D 707905FF              MOV     USCHAR,A        ;SAVE COPY OF USART CHAR -  ECHO CHAR TO HO
 167 0091 40AE00                CALL    XMIT            ;ECHO CHAR TO HOST
 168 0094 40DF00                CALL    PROCESS         ;SEND CHAR FOR DISPLAY PROCESSING
 169 0097 B8                    RET
 170
 171 0098 4CC2          KYBD:   MOV     A,PC            ;CHK THE KEYBOARD STROBE LINE
 172 009A 0780                  ANI     A,80H           ;MASK OUT ALL BITS EXCEPT STROBE BIT #PC7
 173 009C 7780                  EQI     A,80H           ;CHK IF CHAR IS AVAIL (ACTIVE HI STROBE)
 174 009E B8                    RET                     ;IF NO KYBD CHAR, THEN RET
 175
 176 009F 4CC0                  MOV     A,PA            ;GET COPY OF KYBD CHAR
 177 00A1 077F                  ANI     A,7FH           ;STRIP OFF PARITY BIT
 178 00A3 40AE00                CALL    XMIT            ;ECHO CHAR TO HOST
 179 00A6 707906FF              MOV     KBCHAR,A        ;SAVE COPY OF CHAR IN MEM STORAGE
 180 00AA 40DF00                CALL    PROCESS         ;SEND CHAR FOR DISPLAY PROCESSING
 181 00AD B8                    RET
 182
 183
 184 00AE 484A          XMIT:   SKIT    FST             ;READ AND SKIP IF TX BUFFER EMPTY,I.E. TX RI
 185 00B0 FD                    JR      XMIT            ;LOOP UNTIL TX BUFFER EMPTY
 186 00B1 4DD8                  MOV     TXB,A           ;SEND CHAR TO USART
 187 00B3 B8                    RET
 188
 189
 190                    EJECT
```

```
** UPD7811      ASSEMBLE LIST **

              (

E STNO ADRS OBJECT   M   SOURCE STATEMENT

   191                  ;       ************************************************
   192                  ;       *                                              *
   193                  ;       * THIS ROUTINE SENDS WRITE ZEROS TO ALL THE    *
   194                  ;       * ADDRESSES IN THE 41264 VIDEO RAM, THUS CLEARING *
   195                  ;       * IT.                                         *
   196                  ;       ************************************************
   197
   198                  CLRVRAM:
   199 00B4 6F00                MVI     L,0             ;SET UP A START ROW ADDR OF ZERO
   200 00B6 6E00                MVI     H,0             ;SET UP A START COL ADDR OF ZERO
   201                  CLRLOOP:
   202 00B8 69F0                MVI     A,0F0H          ;SET UP DATA NIBBLE OF ALL ZEROS
   203
   204 00BA 40CC00              CALL    WRITERAM        ;WRITE TO 41264 L(ROW),H(COL),A0-3(DATA)
   205
   206 00BD 744601              ADI     H,1             ;INCREMENT THE COL ADDR UNTIL IT
   207 00C0 747E00              EQI     H,0             ;OVERFLOWS (FINISHES 256 COLS)
   208 00C3 F4                  JR      CLRLOOP         ;LOOP UNTIL DONE
   209 00C4 744701              ADI     L,1             ;INCREMENT THE ROW ADDR UNTIL IT
   210 00C7 747F00              EQI     L,0             ;OVERFLOWS (FINISHED 256 COLS)
   211 00CA ED                  JR      CLRLOOP         ;LOOP UNTIL DONE
   212
   213 00CB B8                  RET
   214                  EJECT
```

```
UPD7811     ASSEMBLE LIST **        (                    )

            (                                )

STNO ADRS OBJECT   M   SOURCE STATEMENT

 215                ;       ************************************************
 216                ;       *                                              *
 217                ;       * WRITE A NIBBLE OF DATA IN A REG LOW NIBBLE     *
 218                ;       * TO ROW ADDR IN L REG AND COL ADDR IN H REG     *
 219                ;       * TO THE 41264  (USING OE/ CONTROLLED WR CYCLE) *
 220                ;       *                                              *
 221                ;       ************************************************
 222
 223                ;       ENTER W/ UPPER NIBBLE OF REG A (DATA NIBBLE) SET TO F
 224
 225              WRITERAM:
 226
 227 00CC 4DC1               MOV     PB,A            ;SET UP DATA NIBBLE AT DATA PORT
 228                                                 ;WITH CONTROL LINES INACTIVE
 229 00CE 0F                MOV     A,L             ;SET UP THE
 230 00CF 4DC5              MOV     PF,A            ;ROW ADDR
 231
 232 00D1 0E                MOV     A,H             ;SET UP THE
 233 00D2 4DC3              MOV     PD,A            ;COL ADDR
 234
 235 00D4 4CC1              MOV     A,PB            ;READ CONTENTS OF PORT B - KEEP DATA INTACT
 236 00D6 07EF              ANI     A,0EFH          ;SET UP FOR AN OE/ CONTROLLED WR VRAM
 237 00D8 4DC1              MOV     PB,A            ;WITH DATA NIBBLE KEPT AT ZERO
 238
 239 00DA 69F0              MVI     A,0F0H          ;AND THEN INACTIVATE THE CONTROL LINES
 240 00DC 4DC1              MOV     PB,A            ;WITH DATA NIBBLE KEPT AT ZERO
 241
 242 00DE B8                RET
 243
 244              EJECT
```

```
** UPD7811     ASSEMBLE LIST **        (                    )

             (                                 )

E STNO ADRS OBJECT   M   SOURCE STATEMENT

   245                   ;       ************************************************************
   246                   ;       *                                                          *
   247                   ;       * PROCESS A CHARACTER: A CHAR FROM KEYBOARD OR USART      *
   248                   ;       * GOES THROUGH HERE TO BE USED AS INDEX INTO CHAR ROM     *
   249                   ;       * LOOKUP PROCEDURE. THAT CHARACTER'S BIT PATTERN IS THEN  *
   250                   ;       * SENT TO THE 41264 VRAM. ROW AND COL ARE CHKED AND UP-   *
   251                   ;       * DATED, SCROLL CONDITION IS CHKED FOR.                   *
   252                   ;       *                                                          *
   253                   ;       ************************************************************
   254
   255                   PROCESS:
   256 00DF B0                   PUSH    V               ;SAVE COPY OF INPUT CHAR
   257 00E0 706900FF             MOV     A,ROW           ;CHK IF THE CHAR ROW TO DISPLAY AT PRESEN
   258 00E4 67FA                 NEI     A,ROWMAX        ;IS AT THE END OF THE MAXIMUM ROW
   259 00E6 406401               CALL    SCROLL          ;IF AT ROW=25, THEN SCROLL THE DISPLAY
   260
   261 00E9 706901FF             MOV     A,COL           ;CHK IF ROW MUST POINT TO NEXT CHAR (NIBE
   262 00ED 77A0                 EQI     A,COLMAX/4      ;BY CHKING FOR THE LAST CHAR COL
   263 00EF D0                   JR      SAMELN          ;IF NOT AT LAST COL, THEN PROCEED
   264 00F0 6900                 MVI     A,0             ;IF AT LAST COL, THEN
   265 00F2 707901FF             MOV     COL,A           ;REINIT COL TO ZERO
   266 00F6 706900FF             MOV     A,ROW           ;SET UP TO POINT THE ROW LOC
   267 00FA 460A                 ADI     A,10            ;TO THE NEXT 10 PIXELS DOWN,
   268 00FC 707900FF             MOV     ROW,A           ;FORMING THE START OF THE NEXT CHAR LINE
   269
   270 0100 A0           SAMELN: POP     V               ;RESTORE ASCII CHAR INPUT
   271 0101 6A08                 MVI     B,8             ;SET UP TO MULTIPLY ASCII CODED CHAR
   272 0103 482E                 MUL     B               ;BY 8, AS INDEX INTO CHAR ROM, ANSWER IN
   273 0105 347901               LXI     H,CHAROM        ;POINT HL TO START OF CHAR ROM ADDR
   274 0108 6A08                 MVI     B,8             ;SET UP AN 8-ROW CHAR SCAN COUNTER IN REC
   275
   276 010A B1           SCANCHR:PUSH    B               ;SAVE LOOP COUNTER VALUE
   277
   278 010B AE                   LDAX    H+EA            ;GET CHAR BIT PATTERN FROM CHAR ROM
   279 010C B3                   PUSH    H               ;SAVE CHAR ROM START ADDR
   280 010D 1B                   MOV     C,A             ;SAVE COPY OF CURRENT BIT PATTERN BYTE
   281 010E 4831                 RLR     A               ;COPY THE
   282 0110 4831                 RLR     A               ;UPPER NIBBLE
   283 0112 4831                 RLR     A               ;INTO THE
   284 0114 4831                 RLR     A               ;LOWER NIBBLE WHICH WILL BE SENT FIRST
   285 0116 17F0                 ORI     A,0F0H          ;SET UPPER NIBBLE OF DATA TO ALL 1'S
   286 0118 706F00FF             MOV     L,ROW           ;GET CURRENT ROW ADDR INTO L
   287 011C 706E01FF             MOV     H,COL           ;GET CURRENT COL ADDR INTO H
   288 0120 40CC00               CALL    WRITERAM        ;WRITE LOWER NIBBLE INTO VRAM
   289
   290 0123 706901FF             MOV     A,COL           ;GET THE CURRENT COL ADDRESS
   291 0127 41                   INR     A               ;ADVANCE IT BY ONE
   292 0128 707901FF             MOV     COL,A           ;RESTORE UPDATED COL IN STORAGE
   293 012C 1E                   MOV     H,A             ;COPY IT INTO REG H
   294 012D 706F00FF             MOV     L,ROW           ;GET CURRENT ROW ADDR INTO L
   295 0131 0B                   MOV     A,C             ;GET ANOTHER COPY OF THE CHAR FROM CHAR ROM
   296
```

```
** UPD7811     ASSEMBLE LIST **        (                    )

             (                                  )

E STNO ADRS OBJECT   M   SOURCE STATEMENT

   297 0132 17F0             ORI     A,0F0H          ;SET UPPER NIBBLE OF DATA TO ALL 1'S
   298 0134 40CC00           CALL    WRITERAM        ;WRITE UPPER NIBBLE INTO VRAM
   299 0137 706901FF         MOV     A,COL           ;SET UP TO RESTORE THE ORIGINAL
   300 013B 51               DCR     A               ;COLUMN ADDR
   301 013C 707901FF         MOV     COL,A           ;BEFORE NEXT 2 NIBBLES GET DISPLAYED
   302
   303 0140 A3               POP     H               ;RESTORE CHAR ROM START ADDR
   304 0141 32               INX     H               ;POINT TO NEXT CHAR ROW BYTE IN CHAR ROM
   305 0142 706900FF         MOV     A,ROW           ;SET UP TO POINT THE ROW
   306 0146 41               INR     A               ;POINTER TO POINT
   307 0147 707900FF         MOV     ROW,A           ;TO NEXT DISPLAY ROW
   308
   309 014B A1               POP     B               ;RESTORE LOOP COUNTER VALUE
   310 014C 52               DCR     B               ;DECREMENT THE SCAN LOOP COUNTER
   311 014D 4FBB             JRE     SCANCHR         ;LOOP TILL DONE
   312
   313 014F 706901FF         MOV     A,COL           ;SET UP TO POINT THE COL LOC
   314 0153 4602             ADI     A,2             ;REQUIRED FOR THE NEXT CHAR
   315 0155 707901FF         MOV     COL,A           ;CHAR LOC ON SCREEN
   316
   317 0159 706900FF         MOV     A,ROW           ;SET UP TO POINT ROW TO NEXT
   318 015D 7609             SBI     A,9             ;CHAR LOCATION TO
   319 015F 707900FF         MOV     ROW,A           ;RIGHT
   320 0163 B8               RET
   321
   322 0164 706900FF SCROLL: MOV     A,ROW           ;SET UP TO ADJUST THE CHAR ROW POSITION
   323 0168 760A             SBI     A,10            ;UP BY ONE CHAR POSITION TO ACHIEVE A
   324 016A 707900FF         MOV     ROW,A           ;SINGLE ROW SCROLL
   325 016E 706902FF         MOV     A,ROWDISP       ;SET UP TO START THE DISPLAY ROW DOWN
   326 0172 460A             ADI     A,10            ;BY 1 CHAR POSITION
   327 0174 707902FF         MOV     ROWDISP,A
   328 0178 B8               RET
   329             EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (                    )

                (                                  )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   330
   331
   332                        ;        CHARACTER ROM BIT IMAGE AREA
   333
   334 0179 00000000    CHAROM: DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   335 0181 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   336 0189 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   337 0191 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   338 0199 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   339 01A1 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   340 01A9 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   341 01B1 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   342 01B9 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   343 01C1 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   344 01C9 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   345 01D1 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   346 01D9 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   347 01E1 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   348 01E9 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   349 01F1 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   350 01F9 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   351 0201 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   352 0209 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   353 0211 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   354 0219 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   355 0221 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
            00000000
   356 0229 00000000            DB      00H,00H,00H,00H,00H,00H,00H,00H
```

** UPD7811 ASSEMBLE LIST ** ( )

( )

```
E STNO ADRS OBJECT   M   SOURCE STATEMENT

          00000000
357 0231  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
358 0239  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
359 0241  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
360 0249  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
361 0251  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
362 0259  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
363 0261  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
364 0269  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
365 0271  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
366 0279  00000000        DB      00H,00H,00H,00H,00H,00H,00H,00H
          00000000
367 0281  08080808        DB      08H,08H,08H,08H,08H,08H,08H,00H
          08080800
368 0289  14141400        DB      14H,14H,14H,00H,00H,00H,00H,00H
          00000000
369 0291  14143E14        DB      14H,14H,3EH,14H,3EH,14H,14H,00H
          3E141400
370 0299  083C0A1C        DB      08H,3CH,0AH,1CH,28H,1EH,08H,00H
          281E0800
371 02A1  06261008        DB      06H,26H,10H,08H,04H,32H,30H,00H
          04323000
372 02A9  040A0A04        DB      04H,0AH,0AH,04H,2AH,12H,2CH,00H
          2A122C00
373 02B1  08080800        DB      08H,08H,08H,00H,00H,00H,00H,00H
          00000000
374 02B9  08040202        DB      08H,04H,02H,02H,02H,04H,08H,00H
          02040800
375 02C1  08102020        DB      08H,10H,20H,20H,20H,10H,08H,00H
          20100800
376 02C9  082A1C08        DB      08H,2AH,1CH,08H,1CH,2AH,08H,00H
          1C2A0800
377 02D1  0008083E        DB      00H,08H,08H,3EH,08H,08H,00H,00H
          08080000
378 02D9  00000000        DB      00H,00H,00H,00H,00H,08H,08H,04H
          00080804
379 02E1  0000003C        DB      00H,00H,00H,3CH,00H,00H,00H,00H
          00000000
380 02E9  00000000        DB      00H,00H,00H,00H,00H,00H,18H,00H
          00001800
381 02F1  00201008        DB      00H,20H,10H,08H,04H,02H,00H,00H
          04020000
```

```
** UPD7811      ASSEMBLE LIST **

                (

E STNO ADRS OBJECT    M   SOURCE STATEMENT

382 02F9 1C22322A           DB      1CH,22H,32H,2AH,26H,22H,1CH,00H
         26221C00
383 0301 080C0808           DB      08H,0CH,08H,08H,08H,08H,1CH,00H
         08081C00
384 0309 1C22201C           DB      1CH,22H,20H,1CH,02H,02H,3EH,00H
         02023E00
385 0311 3E201018           DB      3EH,20H,10H,18H,20H,22H,1CH,00H
         20221C00
386 0319 10181412           DB      10H,18H,14H,12H,3EH,10H,10H,00H
         3E101000
387 0321 3E021E20           DB      3EH,02H,1EH,20H,20H,22H,1CH,00H
         20221C00
388 0329 3804021E           DB      38H,04H,02H,1EH,22H,22H,1CH,00H
         22221C00
389 0331 3E201008           DB      3EH,20H,10H,08H,04H,04H,04H,00H
         04040400
390 0339 1C22221C           DB      1CH,22H,22H,1CH,22H,22H,1CH,00H
         22221C00
391 0341 1C22223C           DB      1CH,22H,22H,3CH,20H,10H,0EH,00H
         20100E00
392 0349 00000800           DB      00H,00H,08H,00H,00H,08H,00H,00H
         00080000
393 0351 00000800           DB      00H,00H,08H,00H,00H,08H,08H,04H
         00080804
394 0359 10080402           DB      10H,08H,04H,02H,04H,08H,10H,00H
         04081000
395 0361 00003E00           DB      00H,00H,3EH,00H,3EH,00H,00H,00H
         3E000000
396 0369 04081020           DB      04H,08H,10H,20H,10H,08H,04H,00H
         10080400
397 0371 1C222010           DB      1CH,22H,20H,10H,08H,08H,00H,08H
         08080008
398 0379 1C222A3A           DB      1CH,22H,2AH,3AH,1AH,02H,3CH,00H
         1A023C00
399 0381 08142222           DB      08H,14H,22H,22H,3EH,00H,00H,00H
         3E000000
400 0389 1E24241C           DB      1EH,24H,24H,1CH,24H,24H,1EH,00H
         24241E00
401 0391 1C220202           DB      1CH,22H,02H,02H,02H,22H,1CH,00H
         02221C00
402 0399 1E242424           DB      1EH,24H,24H,24H,24H,24H,1EH,00H
         24241E00
403 03A1 3E02020E           DB      3EH,02H,02H,0EH,02H,02H,3EH,00H
         02023E00
404 03A9 3E020203           DB      3EH,02H,02H,03H,02H,02H,02H,00H
         02020200
405 03B1 3C02023A           DB      3CH,02H,02H,3AH,22H,22H,3CH,00H
         22223C00
406 03B9 2222223E           DB      22H,22H,22H,3EH,22H,22H,22H,00H
         22222200
```

```
** UPD7811      ASSEMBLE LIST **          (                    )

                (                                  )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

407 03C1 1C080808           DB      1CH,08H,08H,08H,08H,08H,1CH,00H
         08081C00
408 03C9 70202020           DB      70H,20H,20H,20H,20H,22H,1CH,00H
         20221C00
409 03D1 22120A06           DB      22H,12H,0AH,06H,0A1H,22H,20H,00H
         A1222000
410 03D9 02020202           DB      02H,02H,02H,02H,02H,02H,3EH,00H
         02023E00
411 03E1 22362A2A           DB      22H,36H,2AH,2AH,22H,22H,22H,00H
         22222200
412 03E9 22262A32           DB      22H,26H,2AH,32H,22H,22H,22H,00H
         22222200
413 03F1 1C222222           DB      1CH,22H,22H,22H,22H,22H,1CH,00H
         22221C00
414 03F9 1E22221E           DB      1EH,22H,22H,1EH,02H,02H,02H,00H
         02020200
415 0401 1C222222           DB      1CH,22H,22H,22H,2AH,12H,2CH,00H
         2A122C00
416 0409 1E22221E           DB      1EH,22H,22H,1EH,0AH,12H,22H,00H
         0A122200
417 0411 3C02021C           DB      3CH,02H,02H,1CH,20H,20H,1EH,00H
         20201E00
418 0419 3E080808           DB      3EH,08H,08H,08H,08H,08H,08H,00H
         08080800
419 0421 22222222           DB      22H,22H,22H,22H,22H,22H,1CH,00H
         22221C00
420 0429 22222222           DB      22H,22H,22H,22H,22H,14H,08H,00H
         22140800
421 0431 22222222           DB      22H,22H,22H,22H,2AH,36H,22H,00H
         2A362200
422 0439 22221408           DB      22H,22H,14H,08H,14H,22H,22H,00H
         14222200
423 0441 22222214           DB      22H,22H,22H,14H,08H,08H,08H,00H
         08080800
424 0449 3E201008           DB      3EH,20H,10H,08H,04H,02H,3EH,00H
         04023E00
425 0451 1C040404           DB      1CH,04H,04H,04H,04H,04H,1CH,00H
         04041C00
426 0459 00020408           DB      00H,02H,04H,08H,10H,20H,00H,00H
         10200000
427 0461 38202020           DB      38H,20H,20H,20H,20H,20H,38H,00H
         20203800
428 0469 081C2A08           DB      08H,1CH,2AH,08H,08H,08H,08H,00H
         08080800
429 0471 00000000           DB      00H,00H,00H,00H,00H,00H,00H,7EH
         0000007E
430 0479 08901100           DB      08H,90H,11H,00H,00H,00H,00H,00H
         00000000
431 0481 00003C20           DB      00H,00H,3CH,20H,3CH,22H,3CH,00H
         3C223C00
432 0489 02021A26           DB      02H,02H,1AH,26H,22H,22H,1EH,00H
         22221E00
```

```
** UPD7811      ASSEMBLE LIST **          (                    )

                (                                  )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

433 0491 00003804             DB      00H,00H,38H,04H,84H,04H,30H,00H
         84043000
434 0499 20202C32             DB      20H,20H,2CH,32H,22H,22H,3CH,00H
         22223C00
435 04A1 00003824             DB      00H,00H,38H,24H,0BCH,84H,0B8H,00H
         BC84B800
436 04A9 3824040E             DB      38H,24H,04H,0EH,04H,04H,04H,00H
         04040400
437 04B1 0000BC22             DB      00H,00H,0BCH,22H,22H,3CH,20H,3CH
         223C203C
438 04B9 02021A26             DB      02H,02H,1AH,26H,22H,22H,22H,00H
         22222200
439 04C1 08000809             DB      08H,00H,08H,09H,08H,08H,90H,00H
         08089000
440 04C9 20002020             DB      20H,00H,20H,20H,20H,0A4H,24H,18H
         20A42418
441 04D1 02022212             DB      02H,02H,22H,12H,0AH,16H,22H,00H
         0A162200
442 04D9 88080808             DB      88H,08H,08H,08H,08H,08H,90H,00H
         08089000
443 04E1 0000362A             DB      00H,00H,36H,2AH,2AH,22H,22H,00H
         2A222200
444 04E9 00001A26             DB      00H,00H,1AH,26H,22H,22H,22H,00H
         22222200
445 04F1 00001824             DB      00H,00H,18H,24H,24H,24H,18H,00H
         24241800
446 04F9 00001E22             DB      00H,00H,1EH,22H,22H,1EH,02H,02H
         221E0202
447 0501 00001C22             DB      00H,00H,1CH,22H,22H,3CH,20H,00H
         223C2000
448 0509 00001A26             DB      00H,00H,1AH,26H,02H,02H,02H,00H
         02020200
449 0511 00003804             DB      00H,00H,38H,04H,18H,20H,1CH,00H
         18201C00
450 0519 08081C08             DB      08H,08H,1CH,08H,08H,08H,90H,00H
         08089000
451 0521 00002222             DB      00H,00H,22H,22H,22H,32H,4CH,00H
         22324C00
452 0529 00002222             DB      00H,00H,22H,22H,22H,14H,08H,00H
         22140800
453 0531 00002222             DB      00H,00H,22H,22H,2AH,3EH,14H,00H
         2A3E1400
454 0539 00002214             DB      00H,00H,22H,14H,08H,14H,22H,00H
         08142200
455 0541 00002222             DB      00H,00H,22H,22H,22H,3CH,20H,38H
         223C2038
456 0549 00003E10             DB      00H,00H,3EH,10H,08H,04H,3EH,00H
         08043E00
457 0551 18888903             DB      18H,88H,89H,03H,88H,89H,19H,00H
         88891900
```

```
** UPD7811      ASSEMBLE LIST **          (                    )

                (                                  )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

458 0559 08080808             DB      08H,08H,08H,08H,08H,08H,08H,08H
         08080808
459 0561 0C909121             DB      0CH,90H,91H,21H,90H,91H,0DH,00H
         90910D00
460 0569 00008C2B             DB      00H,00H,8CH,2BH,0B0H,01H,00H,00H
         B0010000
461 0571 00000000             DB      00H,00H,00H,00H,00H,00H,00H,00H
         00000000
```

```
** UPD7811     ASSEMBLE LIST **        (                    )

             (                              )

E STNO ADRS OBJECT   M   SOURCE STATEMENT

   462
   463                 EJECT
```

```
** UPD7811     ASSEMBLE LIST **        (                    ) PAGE   17

             (                              )

E STNO ADRS OBJECT   M   SOURCE STATEMENT

   464
   465                 ;       PROGRAM VARIABLE STORAGE AREA
   466
   467          FF00   ORG     0FF00H                  ;START VARIABLE LOCS AT 7811 RAM START AI
   468
   469 FF00 00         ROW:            DB      0       ;ROW OF CURRENT DISPLAY CHAR
   470 FF01 00         COL:            DB      0       ;COL OF CURRENT DISPLAY CHAR
   471 FF02 00         ROWDISP:        DB      0       ;ROW OF CURRENT HSYNC DOT LINE
   472 FF03 00         COLDISP:        DB      0       ;COL OF CURRENT HSYNC DOT LINE START POSI
   473 FF04 00         ROWCNT:         DB      0       ;IF=251, THEN PERFORM A VSYNC
   474
   475 FF05 00         USCHAR:         DB      0       ;STORAGE FOR A USART CHARACTER
   476 FF06 00         KBCHAR:         DB      0       ;STORAGE FOR A KEYBOARD CHARACTER
   477
   478                 END

ERROR =     0
  (    0)
REDUCIBLE CODE =     0
```

Examples of enhanced and unique Instructions supported by V20, V30, V40 and V50 Microprocessors

Contents:

Author: Bernd Peters
Application Department
NEC Electronis (Europe) GmbH

Related Documentation

Data Book
Microprocessors Peripherals
User's Manual
μPD70108/70116/70208/70216

Related Products

μPD70108/116	16-Bit Microprocessors	CMOS
μPD70208/216	16-Bit Microprocessors with peripherals	CMOS

1. Introduction

The instruction set of NEC's CMOS microprocessors V20, V30, V40 and V50 represents superset of instruction set supported by the μPD8086/8088 microprocessors.

The aim of this application note is to explain the operation/use of the additional instructions supported by the V-Series microprocessors. The additional instructions can be divided into three basic groups viz, the enhanced instructions supporting the μPD8080 emulation mode. The three groups of instructions can be listed as follows:

a) Enhanced Instructions

Instruction	Function
PUSH imm	Pushes immediate data onto stack
PUSH R	Pushes 8 general registers onto stack
POP R	Pops 8 general registers from stack
MUL imm	Executes 16-bit multiply of register or memory contents by immediate data
SHL imm8 SHR imm8 SHRA imm8 ROL imm8 ROR imm8 ROLC imm8 RORC imm8	Shifts/rotates register or memory by immediate value
CHKIND	Checks array index against designated boundaries
INM	Moves a string from an I/O port to memory
OUTM	Moves a string from memory to an I/O port
PREPARE	Allocates an area for a stack frame and copies previous frame pointers
DISPOSE	Frees the current stack frame on a procedure exit

b) Unique Instructions

Instruction	Frunction
INS	Insert bit field
EXT	Extract bit field
ADD4S	Adds packed decimal strings
SUB4S	Substracts one packed decimal string from another
CMP4S	Compares two packed decimal strings
ROL4	Rotates one BCD digit left through AL lower 4 bits
ROR4	Rotated one BCD digit right through AL lower 4 bits
TEST1	Tests a specified bit and sets/resets Z flag
NOT1	Inverts a specified bit
CLR1	Clears a specified bit
SET1	Sets a specified bit

c) Instrucxtions for Emulation of 8080

BRKEM	(Break for Emulation)
RETEM	(Return for Emulation)
CALLN	(Call Native Routine
RETI	(Return from Interrupt)

2. The Enhanced Instructions

2.1 PUSH/POP-Instructions

a) PUSH R-/POP R-Instructions

These instructions allow the contents of the general registers (AW, BW, CW, SP, BP, IX and IY) to be pushed onto or popped from the stack with a single instruction.

b) PUSH imm8 / PUSH imm16

These instructions allow immediate data to be pushed onto stack. The immediate 8-bit data will be treated as 16-bit data.

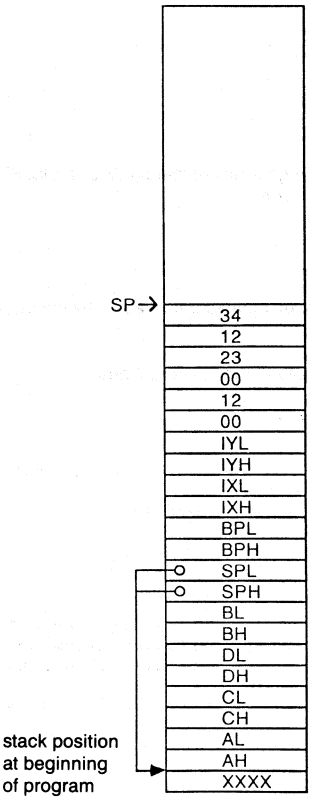
The following example demonstrates the use of these instructions.

Program:

```
;This program shows the use of the extended PUSH and POP instructions:
;
code    segment         para    public
        assume  ps:code
;
vari1   dw      ?
vari2   dw      ?
;
        push    r               ;load the general purpose registers
                                ;(aw,bw,cw,dw,sp,bp,ix,iy) on stack.
        push    012h            ;load 012h on stack.
        push    023h            ;
        push    01234h          ;load 01234h on stack.
;⊗
        mov     aw,dw
        mov     dw,aw
        mov     ix,dw
;
        pop     aw              ;this isn't an extended instruction!
        mov     vari1,aw
        pop     aw              ;this isn't an extended instruction!
        mov     vari2,aw
        pop     r               ;reload the registers from stack.
;
code    ends
        end
```

The following chart shows the assignment of the stack at point X in this program.

Contents of stack at point X



2.2 MUL-Instruction

This enhanced instruction allows use of immediate data as operand. The comments in the program explain the operations of MUL.

Instruction syntax: MUL reg., imm. / MUL reg., mem., imm. / MUL mem., imm.

Program example:

```
;This program shows the use of the extended Multiply instructions:
;
code    segment         para public
        assume  ps:code
;
memo1   dw      0233h
memo2   dw      0fffah                          ;= -6
;
        mov     aw,0120h
        mov     dw,0000h
        mov     cw,010h
        mov     bw,0ff01h                       ;= -ff
;
        mul     dw,aw,04                        ;120h * 04 = 480h = dw
        mul     dw,02                           ;480h * 02 = 900h = dw
        mul     ix,bw,05                        ;-ffh * 05 = -4fbh =>ix=fb05h
;
        mul     dw,memo1,05                     ;233h * 05 = affh = dw
        mul     bp,memo2,02                     ;-6 * 02 = -ch =>bp=fff4
;
        mul     iy,cw,0123h                     ;10h * 123h = 1230h = iy
        mul     aw,0200h                        ;120h * 200h = 24000h =>aw=4000h
;
        mul     bw,memo1,030h                   ;233h * 30h = 6990h = bw
;
code    ends
        end
```

2.3 Shift- and Rotate-Instructions

This instruction allows shift/rotate operation using immediate operand. Here the comments in the program example explain the results after execution of the instructions.

Instruction syntax: SHL / SHR / ROL / ROR / ROLC / RORC reg., imm8

Note: Once a register is changed, the changed content is used for the next instruction.

Program example:

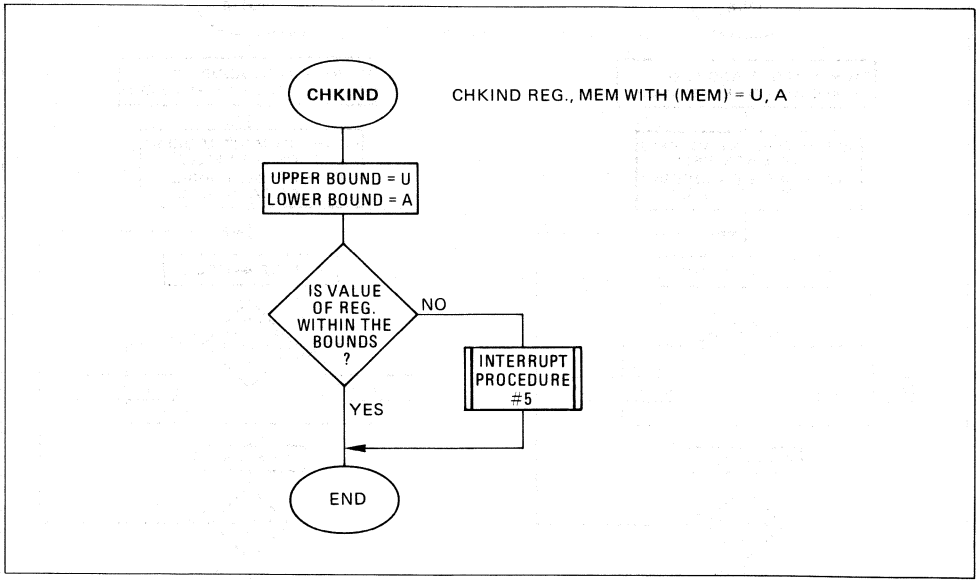
```
;This program shows the use of the extended SHIFT and ROTATE instructions.
;Here one can use immediate values for the operands
;
code    segment         para    public
        assume  ps:code
vari1   db      085h                    ; 1000.0101b
vari2   dw      01011h                  ; 0001.0000.0001.0001b
;
        mov     aw,0f125h               ; aw=1111.0001.0010.0101b
        mov     dw,0012h                ; dw=0000.0000.0001.0010b
        mov     cw,09001h               ; cw=1000.0000.0000.0001b
        mov     iy,01010h               ; iy=0001.0000.0001.0000b
;
        shl     ah,3                    ; =>aw=1000.1000.0010.0101b, CY=1
        shl     dw,15                   ; =>dw=0000.0000.0000.0000b, CY=1
;
        shl     vari1,04                ; =>vari1=0101.0000b, CY=0
        shl     vari2,0ah               ; =>vari2=0100.0100.0000.0000b,CY=0
;
        shr     cw,11                   ; =>cw=0000.0000.0001.0010b, CY=0
        shr     dl,04                   ; =>dw=0000.0000.0000.0000b, CY=0
;
        shr     vari1,02                ; =>vari1=0001.0100b, CY=0
        shr     vari2,08                ; =>vari2=0000.0000.0100.0100b, CY=0
;
        mov     ix,09012h               ; 1001.0000.0001.0010b
;
        shra    cl,03                   ; =>cw=0000.0000.0000.0010b, CY=0
        shra    ix,06                   ; =>ix=1111.1110.0100.0000b, CY=0
;
        shra    vari1,03                ; =>vari1=0000.0010b, CY=1
        shra    vari2,05                ; =>vari2=0000.0010b, CY=0
;
        rol     ah,03                   ; =>aw=0100.0100.0010.0101b, CY=0
        rol     iy,05                   ; =>iy=0000.0010.0000.0010b, CY=0
;
        rol     vari1,04                ; =>vari1=0010.0000b, CY=0
        rol     vari2,0ch               ; =>vari2=0010.0000.0000.0000b, CY=0
;
        ror     al,2                    ; =>aw=0100.0100.0100.1001b, CY=0
        ror     cw,04                   ; =>cw=0010.0000.0000.0000b, CY=0
;
        ror     vari1,02                ; =>vari1=0000.1000b, CY=0
        ror     vari2,20                ; =>vari2=0000.0010.0000.0000b, CY=0
;
        rolc    ch,4                    ; =>cw=0000.0001.0000.0000b, CY=0
        rolc    aw,14                   ; =>aw=0100.1000.1000.1001b, CY=0
;
        rolc    vari1,03                ; =>vari1=0100.0000b, CY=0
        rolc    vari2,10                ; =>vari2=0000.0000.0000.0100b, CY=0
;
        rorc    ch,5                    ; =>cw=0001.0000.0000.0000b, CY=0
        rorc    aw,3                    ; =>aw=0100.1001.0001.0001b, CY=0
;
        rorc    vari1,07                ; =>vari1=0000.0000b, CY=1

        rorc    vari2,13                ; =>vari2=0000.0000.0100.1000b, CY=0
;
code    ends
        end
```

2.4 CHKIND-Instruction (CHKIND reg16, mem32)

This instruction is used to verify that index values pointing to the elements of an array data structure are within the defined range. The lower limit of the array should be in memory location mem32, the upper limit in mem32+2. If the index value in reg16 does not lie between these limits when CHKIND is executed, a BRK# 5 will occur. This causes a jump to the location in interrupt vector # 5.

Instruction syntax: CHKIND reg16, mem32

The flow chart of the CHKIND-Instruction shows the following illustration:



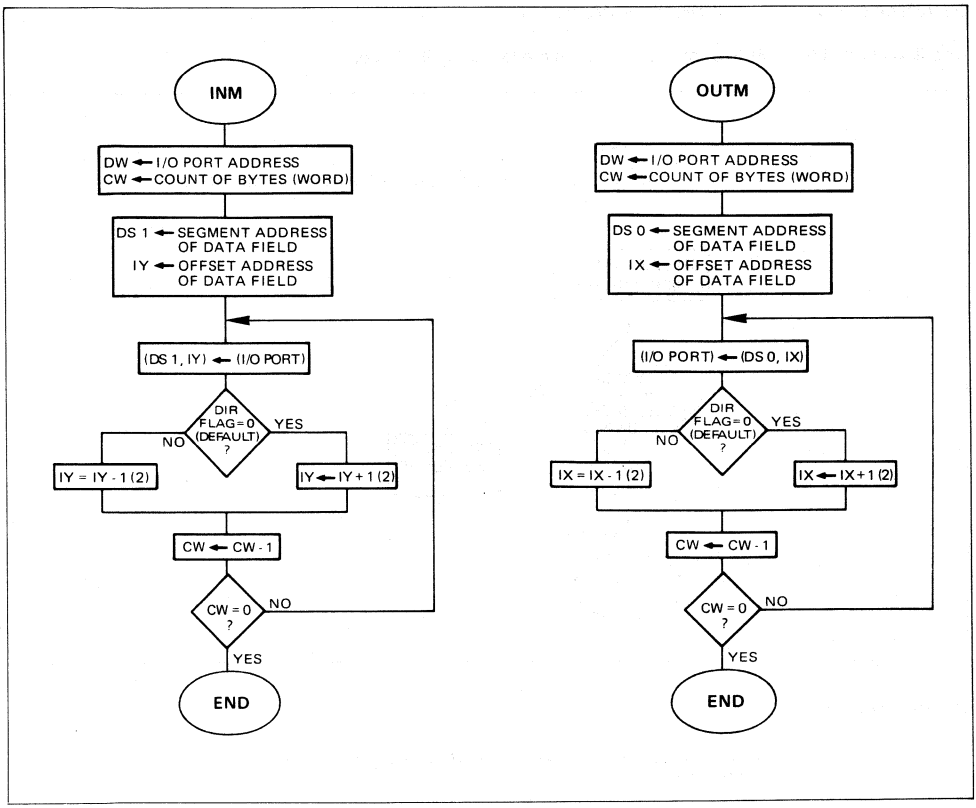
Program example:

```
;This program shows an easy example of the application for the
;CHKIND instruction:
;
code    segment         para    public
        assume  ps:code
;
bounds1 dw      5,130h          ;there is an array from value 5 to 130h.
bounds2 dw      110h,230h       ;another array from value 110h to 230h.
;
        mov     ix,129h
        chkind  ix,bounds1      ;OK, no interrupt no.5 .
        inc     ix
        inc     ix
        chkind  ix,bounds1      ;this causes an interrupt no.5 .
        chkind  ix,bounds2      ;OK
        mov     aw,23h
        chkind  aw,bounds1      ;OK
        chkind  aw,bounds2      ;interrupt no.5
;
code    ends
        end
```

2.5 INM/OUTM-Instruction (INM DST-block, DW/OUTM DW, SRC-block)

These instructions are used for multiple input/output operations, when preceded by a repeat prefix.

The flow chart below demonstrates the operation flow of these instructions.



Instruction syntax: OUTM DW, src-block / INM dst-block, DW

Program example:

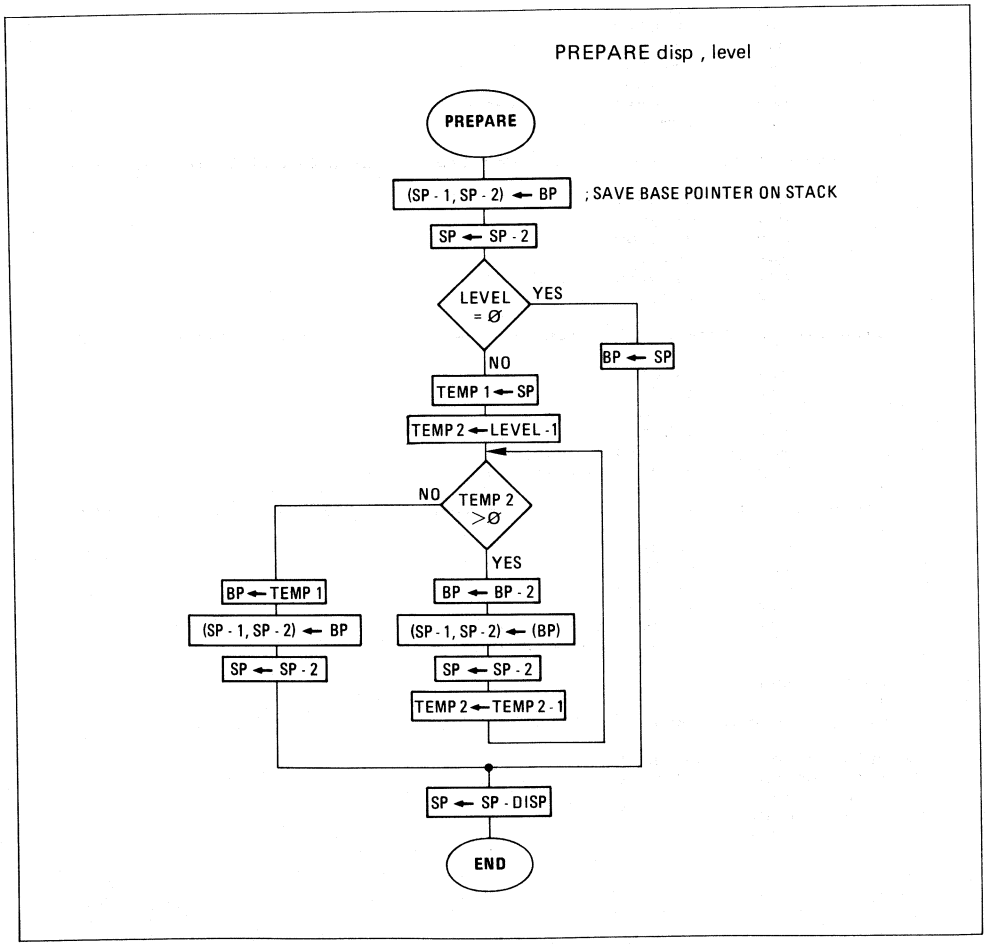
```
;This program shows the use of the PRIMITIVE INPUT/OUTPUT-instructions:
;
code    segment         para    public
        assume  ps:code,ds0:data,ds1:data
;
;transfer a block of 30 bytes from I/O device (address=0123h) to memory
;
        mov     dw,0123
        mov     cw,30
        mov     aw,code
        mov     ds1,aw
        mov     iy,offset byte_var
        rep     inm     byte_var,dw
;
;transfer a block of 30 words from I/O device (address=0321h) to memory
;
        mov     dw,0321
        mov     cw,30
        mov     aw,code
        mov     ds1,aw
        mov     iy,offset word_var
        rep     inm     word_var,dw
;
;Transfer a block of 30 bytes from memory to I/O device (address=0122h)
;
        mov     dw,0122
        mov     cw,30
        mov     aw,code
        mov     ds0,aw
        mov     ix,offset byte_var
        rep     outm    dw,byte_var
;
;Transfer a block of 30 words from memory to I/O device (address=0ffh)
;
        mov     dw,0ffh
        mov     cw,30
        mov     aw,code
        mov     ds0,aw
        mov     ix,offset byte_var
        rep     outm    dw,word_var
;
code    ends
;
data    segment para    public
;
byte_var db     30 dup(?)
word_var dw     30 dup(?)
data    ends
        end
```

2.6 PREPARE/DISPOSE
(PREPARE imm16, imm8 / DISPOSE)

This instruction is used to generate stack frames required to pass arguments/varuables during procedure calls. The use of this instruction offers comfortable programming as is otherwise supported by high level languages like Pascal and ADA.

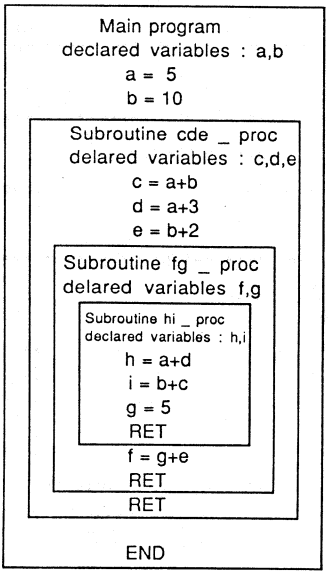
The stack frame consists of two areas. One area has a pointer that points to another frame which has variables that the current frame can access. The other is a local variable area for the current procedure.

The flow chart of PREPARE-operations is shown in the picture below.



To show the use of PREPARE and DISPOSE the following example has embedded subroutines which use global and local variables.

This structure shows the nested procedures and their operations for the subsequently program:



The corresponding program is shown in the example.

Program example:

```
;This program shows the use of PREPARE and DISPOSE instructions
;
;The mainprogram defines the variables a and b and assigns the values
;to them.
;                                              a = 5
;                                              b = 10
;
;In the procedure cde_proc will be evaluated :  c = a + b
;(defined variables:c,d and e)                  d = a + 3
;                                              e = b + 2
;
;In the procedure hi_proc:                      h = a + d
;(defined variables:h and i)                    i = b + c
;                                              g = 5
;
;In the procedure fg_proc                       f = g + e
;(defined variables:f and g)
;
;
```

```
a       equ     -2                              ;offset of the variables, relative
b       equ     -4                              ;to the corresponding BP
c       equ     -2
d       equ     -4
e       equ     -6
f       equ     -2
g       equ     -4
h       equ     -2
i       equ     -4
ab_pointer      equ     -2                      ;offset pointers for access
cde_pointer     equ     -4                      ;to the variables
fg_pointer      equ     -6
hi_pointer      equ     -8
;
;************************************ stack ************************************
;
stackp  segment         para    public
;
stack_frame     dw      50h dup (0000)          ;reserve and clear stack area
;
stackp  ends
;
;********************************** mainprogram ********************************
;
code    segment         para    public
        assume  ps:code,ss:stackp
;
        mov     aw,seg stack_frame              ;load stack segment
        mov     ss,aw                           ;
        mov     sp,0100h                        ;load stack pointer
        mov     bp,sp                           ;stack pointer=base pointer
        prepare 4,1                             ;generate the stack frame
                                                ;for the first level with two
                                                ;varibles
;after execution:
;sp=00f8h,bp=00feh and stack contents:  offset  contents
;                                       00f8h   0000h   =location of variable b
;                                       00fah   0000h   =location of variable a
;                                       00fch   00feh   =base pointer of level 1
;                                                        [00feh]=start base pointer
;                                       00feh   0100h   =saved start base pointer
;
        mov     [bp][a + ab_pointer],05         ;load variable a with 5
        mov     [bp][b + ab_pointer],10         ;load variable b with 10
;
;the stack contains:                    00f8h   000ah   =variable a
;                                       00fah   0005h   =variable b
;                                       00fch   00feh
;                                       00feh   0100h
;
        call    cde_proc                        ;goto subroutine cde_proc
        dispose                                 ;get the right stack and base
;                                               ;pointer value:sp=100h,bp=100h

        br      fin                             ;branch to end of program
;
;********************************** cde_proc ***********************************
;
cde_proc        proc    near
;
        prepare 6,2                             ;generate continous stack frame
                                                ;for the second level with three
                                                ;variables
;after execution:
;sp=00eah,bp=00f4h and stack contents:  00eah   0000h   =location of variable e
;                                       00ech   0000h   =location of variable d
;                                       00eeh   0000h   =location of variable c
;                                       00f0h   00f4h   =base pointer of level 2
;                                       00f2h   00feh   =base pointer of level 1
;                                       00f4h   00feh   ->[00fe]=start base pointer
;                                       00f6h   001bh   =return address from subroutine
;                                       00f8h to 00feh the same like above
;
```

```
        mov     iy,[bp][ab_pointer]                     ;iy = temporary base pointer for
                                                        ;variables a and b
        mov     aw,ss:[iy][a + ab_pointer]              ;load variable a in aw
        add     aw,ss:[iy][b + ab_pointer]              ;aw <- aw + b
        mov     [bp][c + cde_pointer],aw                ;store the result in c
        mov     aw,ss:[iy][a + ab_pointer]              ;aw <- a
        add     aw,3                                    ;aw <- aw + 3
        mov     [bp][d + cde_pointer],aw                ;store the result in d
        mov     aw,ss:[iy][b + ab_pointer]              ;
        add     aw,2                                    ;e = b + 2
        mov     [bp][e + cde_pointer],aw                ;
;
;the stack contains:                     00eah   000ch   =variable e
;                                       00ech   0008h   =variable d
;                                       00eeh   000fh   =variable c
;
        call    fg_proc                                 ;call subroutine fg_proc
        dispose                                         ;adjust stack and base pointer
                                                        ;sp=00f6h,bp=00feh
        ret
cde_proc        endp                                    ;end subroutine cde_proc
;
;************************************ fg_proc ************************************
;
fg_proc         proc    near
;
        prepare 4,3                                     ;generate the next continous stack
                                                        ;frame for the third level with two
                                                        ;variables
;after execution:
;sp=00dch, bp=00e6h and stack contents: 00dch   0000h   =location of variable g
;                                       00deh   0000h   =location of variable f
;                                       00e0h   00e6h   =base pointer of level 3
;                                       00e2h   00f4h   =base pointer of level 2
;                                       00e4h   00feh   =base pointer of level 1
;                                       00e6h   00f4h   ->[[00f4]]=start base pointer
;                                       00e8h   0046h   =return address from subroutine
;                                       00eah to 00feh are the same like above
;
        call    hi_proc
        mov     aw,[bp][g + fg_pointer]                 ;f = g + e
        mov     iy,[bp][cde_pointer]
        add     aw,ss:[iy][e + cde_pointer]
        mov     [bp][f + fg_pointer],aw
;
;the stack contains:                     00dch   0005h   =variable g
;                                       00deh   0011h   =variable f
;
        dispose                                         ;adjust stack and base
                                                        ;pointer: sp=00e8h,bp=00f4
        ret
fg_proc         endp
;
;************************************ hi_proc ************************************
;
hi_proc         proc    near
;
        prepare 4,4                                     ;generate the last stackframe
                                                        ;for the fourth level with two
                                                        ;variables
;after execution:
;sp=00cch,bp=00d8h and stack contents:  00cch   0000h   =location of variable i
;                                       00ceh   0000h   =location of variable h
;                                       00d0h   00d8h   =base pointer of level 4
;                                       00d2h   00e6h   =base pointer of level 3
;                                       00d4h   00f4h   =base pointer of level 2
;                                       00d6h   00feh   =base pointer of level 1
;                                       00d8h   00e6h   ->[[[00e6]]]=start base pointer
;                                       00dah   004fh   =return address from subroutine
```

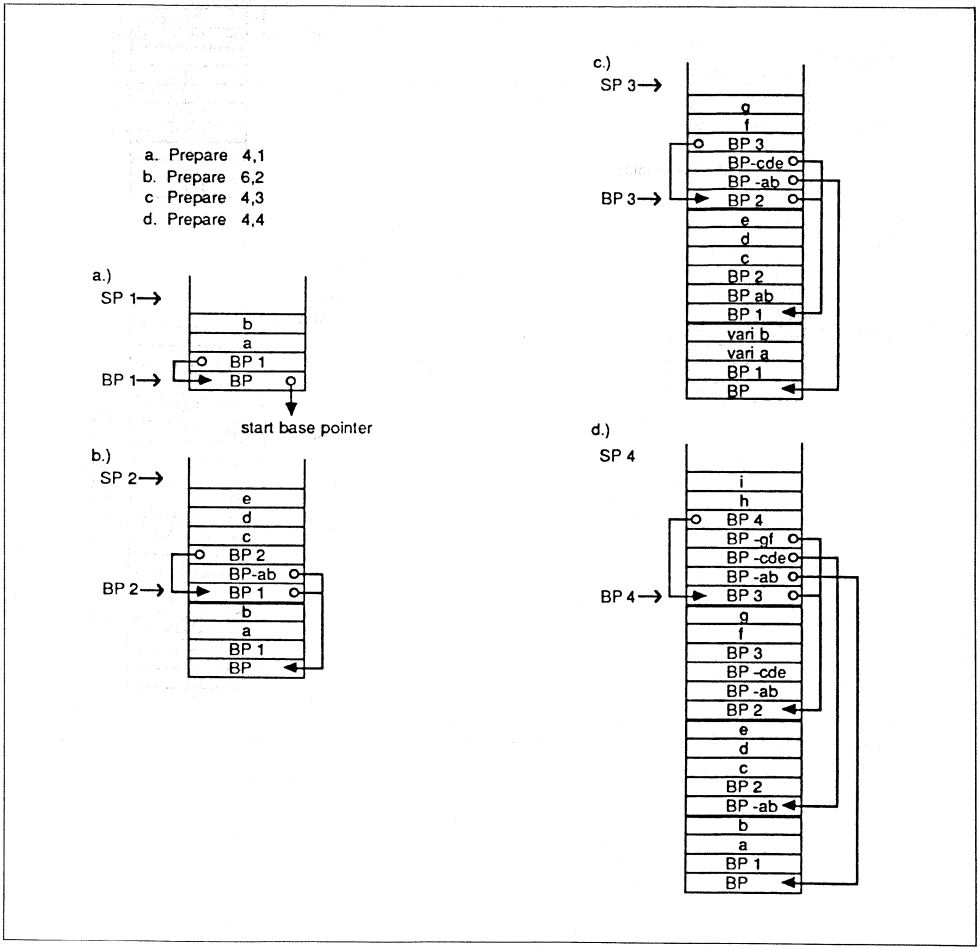
```
;                                       00dch to 0100h like above
;
        mov     iy,[bp][ab_pointer]
        mov     ix,[bp][cde_pointer]
        mov     aw,ss:[iy][a + ab_pointer]              ;h = a + d
        add     aw,ss:[ix][d + cde_pointer]             ;
        mov     [bp][h + hi_pointer],aw                 ;
        mov     aw,ss:[iy][b + ab_pointer]              ;i = b + c
        add     aw,ss:[ix][c + cde_pointer]             ;
        mov     [bp][i + hi_pointer],aw                 ;
        mov     iy,[bp][fg_pointer]                     ;g = 5
        mov     ss:[iy][g + fg_pointer],05              ;
;
;the stack contains:                    00cch   0019h   =variable i
;                                       00ceh   000dh   =variable h
;                                       00dch   0005h   =variable g of level 3
;
        dispose                                         ;adjust stack and base pointer
                                                        ;sp=00dah,bp=00e6h
        ret
hi_proc         endp
;
fin     label   near
;
code    ends
        end
```

The stack assignments is shown by the illustration below.

(Please regard to the comment of the program.)

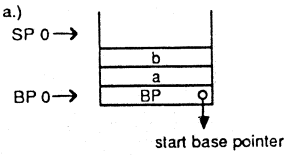
Note:

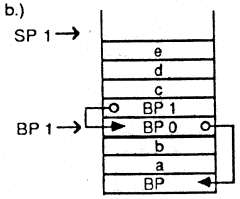
Picture a) shows the stack after the first prepare
Picture b) shows the stack after the second preprare
Picture c) shows the stack after the third prepare
Picture d) shows the stack after the fourth prepare

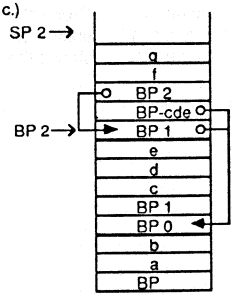


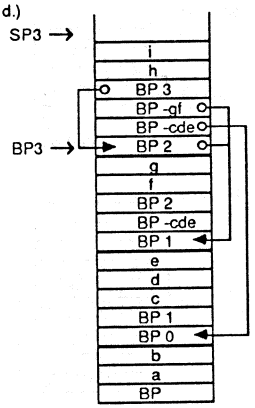
If the level counting is started from 0, the result is different:

a. Prepare 4,0
b. Prepare 6,1
c Prepare 4,2
d. Prepare 4,3









3. The Unique Instructions

3.1 INS/EXT-Instructions

Variable Length Bit Field Operation Instructions

This category has two instructions: INS (Insert Bit Field)
EXT (Extract Bit Field)

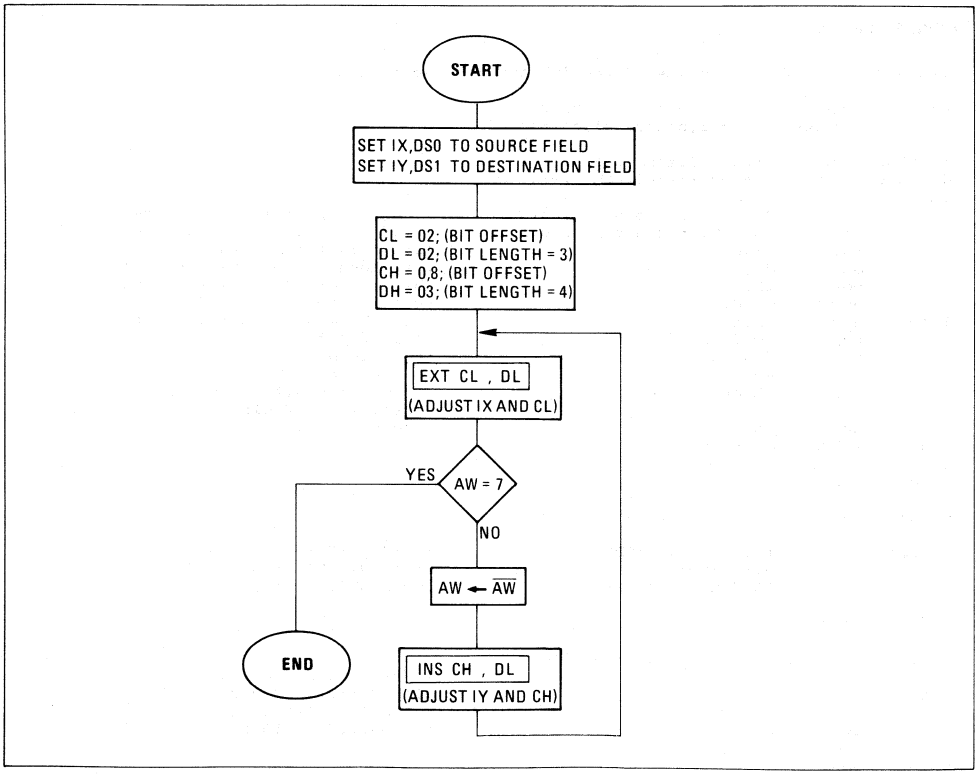
These instructions are highly effective for computer graphics and high-level languages. They can, e.g. be used for data structures such as packed arrays and record type data as used in PASCAL.

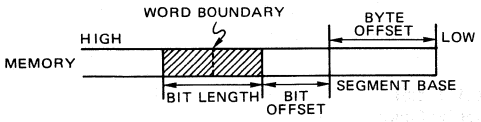
The assignment of the Start position and the operation of these instructions shows the illustration on the following page.

In the following program, a 3-bit-pattern will be extracted from the source field until this combination is '111'.

First these three bits are inverted and the inverted bits + bit#3 = 1 incorporate (as a 4-bit-combination) are stored in the destination field.

The following flow chart demonstrates the function of the corresponding program on the next side.

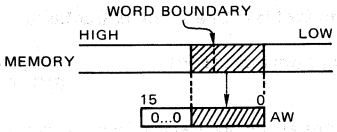




SEGMENT BASE: DS0, DS1
BYTE OFFSET: IX, IY
BIT OFFSET: REG8
BIT LENGTH: REG8, IMM4

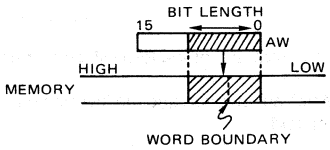
– EXT

EXTRACT BIT FIELD IN THE DS0 SEGMENT TO AW.
REMAINED FIELD IS FILLED WITH 0'S.



– INS

INSERT BIT FIELD IN AW TO THE DS1 SEGMENT.



Program example:

```
;This program shows the use of the INS and EXT instructions:
;
code    segment         para    public
        assume  ps:code,ds0:data1,ds1:data2
;
srcblock  dw    offset field1,seg field1
desblock  dw    offset field2,seg field2
;
        mov     ds0,ix, dword ptr srcblock      ;load the position of source field
        mov     ds1,iy, dword ptr desblock      ;load the position of destination field
        mov     cl,02                           ;bit offset = 2 bits for  EXT
        mov     dl,02                           ;bit length = 3 bits for  EXT
        mov     ch,08                           ;bit offset = 8 bits for  INS
        mov     dh,03                           ;bit length = 4 bits for  INS
loop:   ext     cl,dl                           ;read 3 bits of source field
        cmp     aw,007                          ;end of data
        be      fin                             ;then branch
        not     aw                              ;invert reg. AW
        ins     ch,dh                           ;load the 4 bits of AW to
                                                ;destination field
        br      loop                            ;end
fin     label   near
;
code    ends
;
data1   segment         para    public
;
field1  dw      1000110110101001b               ;10'001'101'101'010'01
        dw      0110110011000110b               ;'011'011'001'100'011'0
        dw      0000000011111111b               ;0000000011111'111
;
data1   ends
;
data2   segment         para    public
;                                               ;after program execution
field2  dw      0                               ;'1010'1101'00000000
        dw      0                               ;'1100'1101'1110'1010
        dw      0                               ;'1100'1100'1110'1011
;
data2   ends
        end
```

3.2 ADD4S-. SUB4S-, CMP-Instruction

The instructions described here process packed BCD data either as strings (ADD4S, SUB4S, CMP4S) or byte-format operands (ROR4, ROL4). Packed BCD strings may be from 1 to 254 digits in length.

When the number of digits is even, the zero and carry flags will be set according to the result of the operation. When the number of digits is odd, the zero and carry flags may not be set correctly in this case, (CL = odd), the zero flag will not be set unless there is a carry out of the upper 4 bits of the highest byte. When CL is odd, the contents of the upper 4 bits of the highest byte of the result are undefined.

Program example:

```
;This program shows the use of the BCD-instructions:
;
;       ADD4S,SUB4S and CMP4S
;
code    segment         para public
        assume  ps:code
;
        mov     aw,ps                           ;set code-segment to
        mov     ds0,aw                          ;DS0 and DS1,too
        mov     ds1,aw                          ;
        mov     ix,offset str0                  ;point to
        mov     iy,offset str1                  ;BCD strings
        mov     cl,08                           ;8 digits
        cmp4s                                   ;compare str0 with str1
        add4s                                   ;str0 + str1 => str1
        cmp4s                                   ;
        sub4s                                   ;str0 - str1 => str1
        sub4s                                   ;
        sub4s                                   ;
        halt
;
str0    dw      4321h,0765h                     ;BCD=07654321
str1    dw      4321h,0765h                     ;BCD=07654321
;
code    ends
        end
```

The program execution

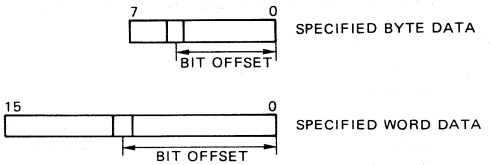
after CMP4S:	str. 0 = 07654321	CY	= 0
	str. 1 = 07654321	Z	= 1
after ADD4S:	str. 0 = 07654321	CY	= 0
	str. 1 = 15308642	Z	= 0
after CMP4S:	str. 0 = 07654321	CY	= 0
	str. 1 = 15308642	Z	= 0
after SUB4S:	str. 0 = 07654321	CY	= 0
	str. 1 = 07654321	Z	= 0
after SUB4S:	str. 0 = 07654321	CY	= 0
	str. 1 = 0	Z	= 1
after SUB4S:	str. 0 = 07654321	CY	= 1
	str. 1 = 92345679	Z	= 0

3.3 TEST1-, NOT-, CLR-, SET1-Instruction

These bit manipulation instructions test, invert, clear or set a specific bit in a register or memory location.

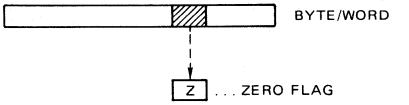
See the following illustrations:

OPERATION FOR A BIT IN THE BYTE/WORD

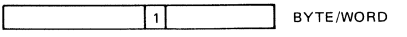


BYTE/WORD DATA: (MEM), REG
BIT OFFSET. CL, IMM3/IMM4

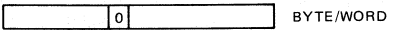
– TEST1
TEST THE SPECIFIED BIT DATA AND AFFECT FLAGS



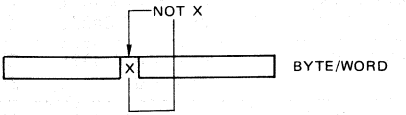
– SET1
SET THE SPECIFIED BIT TO 1



– CLR1
CLEAR THE SPECIFIED BIT TO 0



– NOT1
COMPLEMENT THE SPECIFIED BIT DATA



Program example:

```
;This program shows the use of the Bit Manipulation instructions:
;
;       TEST1, NOP1, CLR1 and SET1
;
;
code    segment         para public
        assume  ps:code
;
memory1 dw      0101100010100010b                       ;=58a2h
memory2 db      01110101b                               ;=75h
;
        mov     aw,0010010101111101b                    ;=257dh
        mov     cl,03                                   ;bit no.4
;
        test1   ah,cl                                   ;bit no.4 = 0, =>Z = 1
        not1    ah,cl                                   ;=> bit no.4 = 1, aw = 2d7dh
        clr1    ah,cl                                   ;=> bit no.4 = 0, aw = 257dh
        set1    ah,cl                                   ;=> bit no.4 = 1, aw = 2d7dh
;
        test1   memory2,cl                              ;bit no.4 = 0, =>Z = 1
        not1    memory2,cl                              ;=> bit no.4 = 1, memory2 = 7dh
        clr1    memory2,cl                              ;=> bit no.4 = 0, memory2 = 75h
        set1    memory2,cl                              ;=> bit no.4 = 0, memory2 = 7dh
;
        test1   al,01                                   ;bit no.2 = 0, =>Z = 1
        not1    al,01                                   ;=> bit no.2 = 1, aw = 2d7fh
        clr1    al,01                                   ;=> bit no.2 = 0, aw = 2d7dh
        set1    al,01                                   ;=> bit no.2 = 1, aw = 2d7fh
;
        test1   memory2,00                              ;bit no.1 = 1, =>Z = 0
        not1    memory2,00                              ;=> bit no.1 = 0, memory2 = 7ch
        clr1    memory2,00                              ;=> bit no.1 = 0, memory2 = 7ch
        set1    memory2,00                              ;=> bit no.1 = 1, memory2 = 7dh
;
        mov     cl,0fh                                  ;bit no.16
;
        test1   aw,cl                                   ;bit no.16 = 0, =>Z = 1
        not1    aw,cl                                   ;=> bit no.16 = 1, aw = ad7fh
        clr1    aw,cl                                   ;=> bit no.16 = 0, aw = 2d7fh
        set1    aw,cl                                   ;=> bit no.16 = 1, aw = ad7fh
;
        test1   memory1,cl                              ;bit no.16 = 0, =>Z = 1
        not1    memory1,cl                              ;=> bit no.16 = 1, memory1 = d8a2h
        clr1    memory1,cl                              ;=> bit no.16 = 0, memory1 = 58a2h
        set1    memory1,cl                              ;=> bit no.16 = 1, memory1 = d8a2h
;
        test1   aw,0ah                                  ;bit no.10 = 1, =>Z = 0
        not1    aw,0ah                                  ;=> bit no.11 = 0, aw = a97fh
        clr1    aw,0ah                                  ;=> bit no.11 = 0, aw = a97fh
        set1    aw,0ah                                  ;=> bit no.11 = 1, aw = ad7f
;
        test1   memory1,05                              ;bit no.6 = 1, =>Z = 0
        not1    memory1,05                              ;=> bit no.6 = 0, memory1 = d882h
        clr1    memory1,05                              ;=> bit no.6 = 0, memory1 = d882h
        set1    memory1,05                              ;=> bit no.6 = 1, memory1 = d8a2h

;
code    ends
        end
```

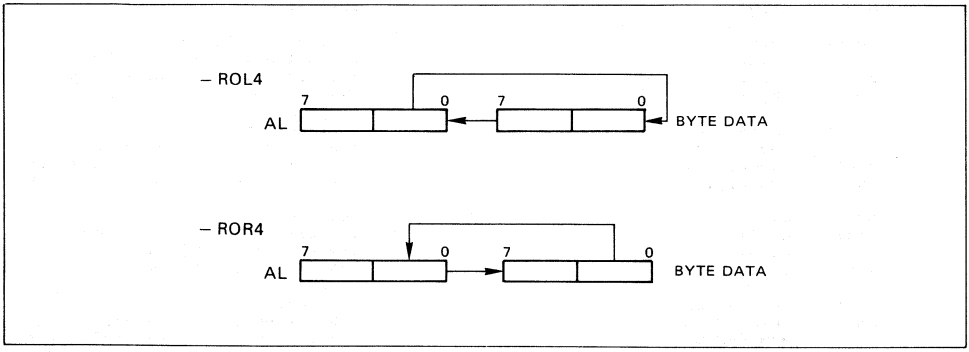
3.4 ROL4/ROR4-Instruction

These instructions treat the byte data of the register or memory operand, specified by the instruction, as BCD data and use the lower 4 bits of the AL register to rotate that data one BCD digit to the left or right respectively.

Instruction syntax: ROL4/ROR4 reg./mem.

● OPERATIONS FOR BCD IN AL

These instructions treat the byte data of a 8 bit register specified by the operand as a two digit BCD data and use the lower 4 bits of the A_L register to rotate that data one BCD digit to the left/right.



Programa example:

```
;This program shows the of the BCD-instruction:
;
;       ROL4 and ROR4
;
code    segment         para    public
        assume  ps:code
;
memory  db      ?
;
        mov     bl,93h
        mov     al,03h
        rol4    bl                      ;after this instruction bl contains
                                        ;33h and al=x9h
        mov     memory,12h
        mov     al,26h
        rol4    memory                  ;after this instruction memory contains
                                        ;26 and al=x1
        mov     cl,95h
        mov     al,29h
        ror4    cl                      ;after this instruction cl contains
                                        ;99 and al=x5
        mov     memory,45h
        mov     al,03h
        ror4    memory                  ;after this instruction memory contains
                                        ;34 and al=x5
;
code    ends
        end
```

4. μPD8080 Emulation Instructions

The V-Series standard microprocessors can operate in two modes. One is the native mode in which the CPU executes the V20—V50 instructions, valid for 16-bit environment.

The other is the emulation mode, in this operation mode, the processors emulate the μPD8080AF CPU fully. The mode flag (MD) in the PSW describes the CPU operation mode at any given time.

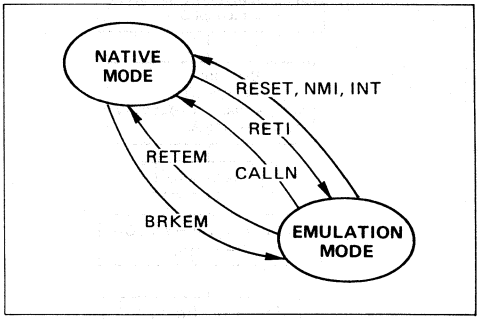
Native mode is selected when MD is 1 and emulation mode when MD is 0. MD is set and reset, directly and indirectly, by executing the mode manipulation instructions.

Two instructions are provided to switch operation from the native mode to the emulation mode and back.

BRKEM	(Break for Emulation)
RETEM	(Return from Emulation)

Two instructions are used to switch from the emulation mode to the native mode and back.

CALLN	(Call Native Routine)
RETI	(Return from Interrupt)



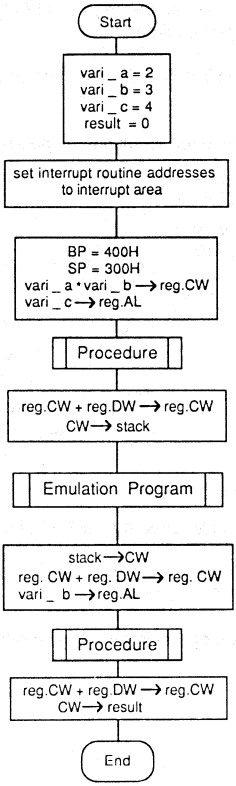
The system will return from the 8080 emulation mode to the native mode when the RESET signal is present, or when an external interrupt (NMI or INT) is present.

Program example: result = a * b + c ** 4 + (c + b + a ** 4) + b ** 4

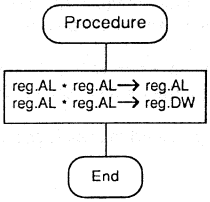
The term: (c + b + a ** 4) shall be calculated in a program in emulation mode, the remaining terms in native mode.

The program flow is described in the following flow charts.

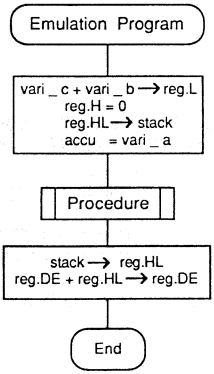
Main Prom (Native Mode):



Subroutine Procedure (Native Mode):



Emulation Program (Emulation Mode):



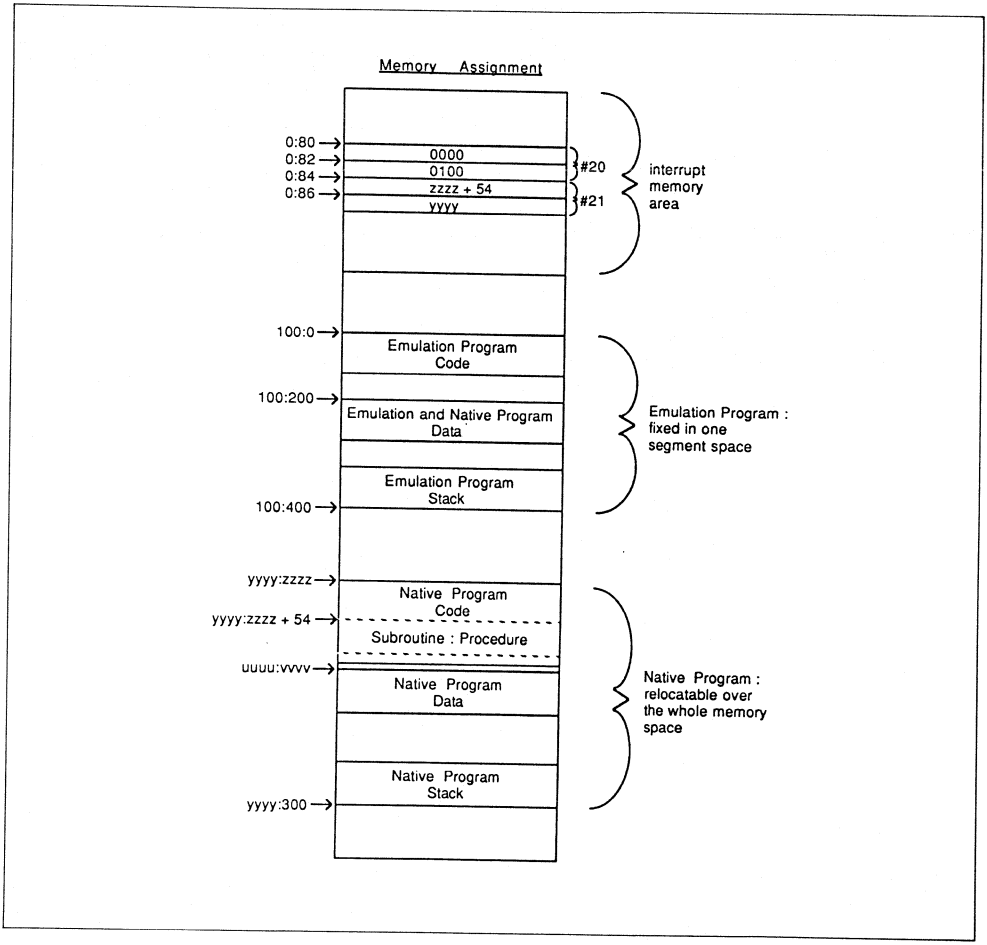
The corresponding program:

Note: The embedded 8080-Program is only comment.

```
;This program shows the use of the Mode Operation Instructions:
;
;       BRKEM,RETI,RETEM and CALLN
;
;This example evaluate the following term:
;
;       result=a * b + c**4 + (c + b + a**4) + b**4
;
;The result of the parenthesis will be evaluated in a program in
;Emulation Mode, but  the term a**4 will be calculated in the
;subroutine of the Native Program.
;
;values  of the variables:       a=2, b=3, c=4
;
offset_emu      equ     0000h                           ;offset address of Emulation program
segment_emu     equ     0100h                           ;segment address of Emulation program
;
;
data_common     segment para    at(0120h)               ;set datasegment to address 01200h
;
vari_a  db      02
vari_b  db      03
vari_c  db      04
;
data_common     ends
;
data_nativ      segment para    public
;
result  dw      0000                                    ;clear result variable
;
data_nativ      ends
;
;
code    segment         para    public
        assume  ps:code,ds0:data_common,ds1:data_nativ
;
        mov     aw,000                                  ;set ds1 to interrupt
        mov     ds1,aw                                  ;memory area.
        mov     iy,0080h                                ;set address pointer to interrupt #20.
        mov     [iy],offset_emu                         ;load offset segment of
                                                        ;Emulation program.
        add     iy,02                                   ;
        mov     [iy],segment_emu                        ;load segment address of
                                                        ;Emulation program.
        add     iy,02                                   ;
        mov     [iy],offset procedure                   ;load offset address of
                                                        ;procedure.
        add     iy,02                                   ;
        mov     [iy],code                               ;load segment address of
                                                        ;procedure.

        mov     aw,code                                 ;set segment address of ds1
        mov     ds1,aw                                  ;to the same of ps
        mov     ss,aw                                   ;and stack segment.
        mov     bp,0400h                                ;the base pointer is the stack pointer in
        mov     sp,0300h                                ;Emulation Mode.
```

```
        mov     al,vari_a               ;vari_a * vari_b -> cw
        mulu    vari_b                  ;
        mov     cw,aw                   ;
        mov     al,vari_c               ;vari_c -> al
        brk     21                      ;call procedure of al ** 4
                                        ;result in reg. dw
        add     cw,dw                   ;cw + vari_c**4 -> cw
        push    cw                      ;save cw on stack
        brkem   20                      ;call of Emulation program
                                        ;evaluation of the parenthesis value
                                        ;result in reg. dw

;
;Emulation program:
;
;       lda     122                     ;load vari_c in accu
;       mov     h,a                     ;load vari_c in reg. h
;       lda     121                     ;load vari_b in accu
;       add     h                       ;accu=vari_c + vari_b
;       mov     l,a                     ;load accu to reg. l
;       mvi     h,00                    ;clear reg. h
;       push    h                       ;save registerpair hl on stack
;       lda     120                     ;load vari_a in accu
;       calln   21                      ;call subroutine in Native Mode
;                                       ;to do accu**4
;       pop     h                       ;get registerpair hl from stack
;       dad     d                       ;add registerpair de to hl
;                                       ;result in hl
;       xchg                            ;de <--> hl, because result in reg. dw
;                                       ;for the Native program
;       retem                           ;back to Native program
;
        pop     cw                      ;get cw from stack
        add     cw,dw                   ;cw + ( ) -> cw
        mov     al,vari_b               ;b**4
        brk     21                      ;
        add     cw,dw                   ;cw + b**4
        mov     result,cw               ;save the result
        br      fin
;
;
procedure       proc    far             ;subroutine to evaluate
                                        ;al**4
        mulu    al
        mulu    al
        reti
procedure       endp
;
fin     label   near
;
code    ends
;
;
;Created in Berni's Wonderland Studios in spring '86.
;Many thanks to Bettina, Shyam and Helmut for their aim
;to Application Note No.9.

 end
```

The last chart shows the memory assignment for the above program:



μCOM relocatable and absolute Crossassembler Software (RA87 / ASM87)

Contents:

Author:

Heiner Tendyck
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

RA87 User's Manual

Related Products

μPD78C10	8-Bit Microcomputer	CMOS
μPD78C12	8-Bit Microcomputer	CMOS
μPD78C14	8-Bit Microcomputer	CMOS

1. Overview on RA87 and ASM87 Cross-Software

NEC provides absolute and relocatable Cross-Software for the μCOM87 series;

Absolute Cross-Software:	ASM87
supported microcomputers:	μPD7800, 01, 02 μPD78c05(a), c06(a) μPD7810, 11
supported operating systems:	CP/M-80, CP/M-86, MS-DOS, ISIS II

Note: For ASM87 NEC offers as well a magnetic tape with a FORTRAN source of ASM87 assembler. This can be compiled and implemented on computers supporting FORTRAN IV compilers.

Relocatable Cross-Software:	RA87, LNK87, LOC87, LIB87
supported microcomputers:	μPD7810 / 11 / c10 / c11 / c14(a), 10H, 11H μPD7807, 08, 09
supported operating systems:	CP/M80, CP/M-86, MS-DOS, ISIS II, UDI VAX/VMS, VAX/ULTRIX
Version number of the different O.S. are:	PC-DOS V3.0 [MS-DOS 2.11 compatible] CP/M-86 V1.0 ISIS II V4.3 CP/M-80 V2.2 VAX/VMS V4.1 VAX/ULTRIX BERKELY V4.2

The actual version numbers of the software tools are released by periodical updates to all NEC offices and application departments.

2. Software History of RA87/ASM87

a) The absolute Cross-Software 'ASM87' has not been changed since it's first release. Meanwhile a MS-DOS version of this S/W is available from stock of NEC.

It has been created out of the CPM86 version by using a shell converter program. Thus you need on your floppy disc both the ASM87.CMD and the ASM87.com file to run it under MS-DOS.

b) In Juni 1985 NEC EE released the new version of the relocatable crossassembler software package RA87 (V1.5 for all O.S.).

This has been updated for 16 bit O.S. by adding additional features and by eleminating known bugs. It's version no. is from 2.0 onward.

It has been updated for 8 bit O.S. by eliminating the known bugs. It's version number is 1.6 (derived from released software 1.5 of Juni 1985).

→ This application note describes all feature changes for RA87 V2.0 onward, compared to V1.5/1.6.

→ Furthermore it describes some features differences concerning SRC-code creation compared to ASM87 conventions.

→ It also includes NOTES for beginners in using this S/W.

3. Differences between ASM87 and RA87 SRC-code Conventions

a) Please note, that RA87 supports a symbol length of max 6 characters. ASM87 supports up to 8 characters.

b) The module size of the RA87 software is restricted to 64KByte source code. This should be enough for each relocatable assembler module. The ASM87 does not have this restriction.

c) ASM87 does not accept capital letters in command line. RA87 does.

d) ASM87 does optimize 'GJMP' instructions. RA87 does optimize GJMP, JMP and JRE if optimizer is switched on (by primary control command).

4. Application Notes for the user of RA87 S/W Package

a) Structure of the module head:

a1) Each source module requires some primary controls e.e. '$PC(C11)' as processor selection. 'XREF' as command to create a cross reference table etc.

a2) After these primary controls you have to input the 'name' of the module.

a3) After the input of primary controls and name of module the user is requested to input all 'public' and 'extrn' definitions. Furthermore V2.0 supports the 'extbit' definition for resolving external bit-manipulations (for 7809).

Comment lines together with primary controls and definition of name should be avoided.

Example:

```
$PC(C11)
$XREF
$DEBUG
$TITLE('DEMO OF MODULE HEAD')
$DATE('06-06-86')
$PAGELENGTH (60)
$PAGEWIDTH (80)

NAME MODUL

PUBLIC PUB1,PUB2,PUB3
EXTRN EXT1,EXT2,EXT3
EXTBIT BIT0,BIT1,BIT2

CSEG

START:  MVI A,09H
        MOV MM,A
.
.
.
```

b) Please note, that 'title', 'subtitle' and 'date' directives require apostroph.

c) RA87 assumes that all code labels are 16 bit. Therefore the application of an 8 bit instruction on a code label is not allowed. In this case the programmer must use a 'low' or 'high' operator.

d) The same applies on all external symbols. The assembler assumes that all external symbols are 16 bit!

```
Example:    extrn ext4
            start: nop
            mvi a, low start
            eqi a, low ext4
```

e) The user should avoid the calculation with reloc. addresses or externals. It is allowed to generate immediate positive offsets to addresses or external symbols.

If the user wants to apply operations on reloc. addresses, he should do a subtraction between this address and the start address of the reloc. segment.

```
Example:    cseg
   start:   nop
            mvi a,09h
            .
            .
            .
   lab1:    mov txb,a
            mov a, mod (lab1-start)
            .
            end
```

The subtraction will make the operand an absolute value, which can be used for all sorts of calculation.

f) CALT is a feature to implement 1Byte calls to often used subroutines and utilities.

For 7811 / 78c11 / 7809 the 'table' are is located to 80h-BFh. The operand of a 'CALT' instruction must be an even address. If this address happens to be odd, the assembler rounds up this 'TABLE' address to an even value.

g) GJMP: this means 'generic jump'.

The μCOM87 family features 1, 2 and 3Byte jumps. If the user codes GJMP, the RA87 tries to generate the smallest code. If possible RA87 optimizes as wall 'JMP' and 'JRE' instructions. If an optimization is not wanted, the user may specify 'optimize(0)' as primary control. The RA87 S/W does not support a searching throughout 'user' levels in CPM86 or 'subdirectories' in MS-DOS. All overlay files should be in same directory.

i) The 'errprn' file on the RA87 package disk is used to pick out the assembler errors and warnings even if assembly is finished already. You may apply it on the xxx.lst or xxx.prn file. You may output the result to printer, console or drive.

```
Example:    ERRPRN demo.prn lst:
```

This outputs all list lines including a warning or error message to printer.

Note: Inputting primary control $ep(lst:) will do the same automatically right after assembly.

5. Differences between RA87 V1.5/1.6 and V2.0 Onwards

a) V2.0 onwards does support the possibility to define 'equ' symbols and bitsymbols (for 7809) as 'public', respectively to define them as external in other modules by 'extrn' or 'extbit' directive.

b) The SET and EQU directive should be applied only on so called backward references. V2.0 will output an error if this rules is not applied.

c) The application of low/high operators in 16 bit instructions should be avoided. Using 16 bit instructions, we assume that user uses 16 bit operands. Error output for V2.0 onward.

d) The default naming conventions of RA87 package have been changed:

for	V1.5	V2.0
RA87 outputs	xxx.lst, xxx.obj	xxx.prn,xxx.rel
INK87 outputs	xxx.lnk,xxx.mp1	xxx.lnk,xxx.mp1
LOC87 outputs	xxx.hex,xxx.sym,xx.mp2	xxx.hex,xxx.sym,xxx.mp2

e) The logical device names have been changed:

	v1.5	v2.0
printer	:lp:	lst:
console	:co:	con:

f) From V2.0 onward Ra87 supports the use of parenthesis.

g) V1.5 optimizes just inside one segment. V2.0 optimizes inside one module.

h) From V2.0 onward the sw supports the so called 'naming' of segments. I.e. each segment can get a name.

Example:

```
ONE     CSEG    ;NAMED RELOC. CSEG
NOP
.
.
ENDS

TWO ORG 0C0H    ;NAMED ABS. SEGMENT
NOP
DB 'A'
DB 'B','C'
ENDS

THREE DSEG      ;NAMED RELOC DATA SEGMENT
LAB1: DS1
LAB2: DS2
ENDS

FOUR VSEG       ;NAMED RELOCATABLE BITSEGMENT
BIT0DBIT
BIT1DBIT
BIT2DBIT
ENDS
```

Reason of this update:

The naming of the segment will give you the additional feature to link together all same named segments to one segment block. So far RA87 linked together only unnamed CSEG/DSEG which is done with V2.0 (onward) as well.

As well you have the opportunity to locate 'named' segments (segment-blocks) to any desired address by 'segment' directive in locator (to be explained later).

i) V2.0 onward supports so called 'CALLT' segments. The user may implement several 'CALLT' reloc. segments by adding an attribute 'CALLT' to the 'CSEG' directive. The linker will combine all 'CALLT' codesegments to one table segment. The user should take care, that the table segment start addresses are even.

```
Example:    cseg calt
            tab0: dw util 1
            tab2: dw util 2
            tab3: dw util 3
```

The locator loc87 will check an overflow of the 'table' area 80h to 0bfh, if the final size of all 'callt' segments exceeds it.

Each module may contain one 'callt' segment.

j) For V2.0 onward the RA87 supports as well so called 'fixed' segments. The μCOM87 family supports two byte call by the 'CALF' instruction. The user may generate codesegments with the attribute 'FIXED'.

```
Example:    util cseg fixed      ;named reloc. 'fixarea' segment
   util4:   mov txb,a            ;serial buffer write
            ldax h+              ;get new serial data
            ret
            util5:   .
                     .
                     .
```

Thus the linker puts together all same named 'fixed' segments or all unnamed 'fixed' segments.

The locator will allocate the 'fixed' segments to address 800h upward. If any address of 'fixed' segments exceeds the area 800h to 0fffh loc87 will generate an error.

Each module may contain one 'fixed' segment.

k) From V2.0 onward RA87 supports relocatable bitsegments (vseg.). The linker links all relocatable bitsegments on Byte boundary. V1.5 supported just absolute bitsegments.

l) From V2.0 onward the meaning and priority of locator commands have been changed.

Loc87 V2.0 onward supports additional directives. Following is the priority of these directives during locating procedure:

```
1. gap                ;control of locator to implement wanted 'gaps'
2. org                (directive in src)
3. segment            ;segment directive to allocate named and unnamed
                      ;cseg and dseg
4. fixed/callt        (directive in src)
5. data               ;definition of default start address of dseg blocks
6. code               ;definition of default start address of cseg blocks
7. stack/stacksize    ;definition of start address for stacksegment SSEG
                      ;definition of max. SSEG size
```

*Examples about use of these directives:

1. loc87 demo.lnk to demo.hex gap (2000h 0feffh).

   This gap control keeps the area 2000h to 0feffh free of any code or data.

2. loc87 demo.lnk to demo.hex segment(cseg 0c0h, dseg 0ff00h, one 0500h, two 2000h).

   This will allocate all unnamed reloc. segments to 0c0h etc., all unnamed data segments to 0ff00h etc., the reloc. named segment 'one' to 500h, and the reloc. named segment 'two' to 2000h).

3. loc87 demo.lnk to demo.hex data(01000h) code(0c0h) stack(0ff00h) stacksize(01fh).

   This will propose to the locator to assign the start address 0c0h to reloc. codesegments, the start address 1000h to reloc. data segments, the stack base for all stack segments (SSEG) to 0ff00h, and control that the size of all SSEG does not exceed 20h.

Note: CODE and DATA may be overruled by a 'optimizing' feature of the locator i.e. loc87 tries to allocate large segments to small 'free' area between absolute segments. If you want well defined start addresses for CSEG/DSEG please use the 'SEGMENT' command.

Note: The SSEG size is evaluated by the RA87 by integrating on all push, pop, call, ret, reti instructions. It will indicate just a rough value about the required SSEG size, as real time conditions cannot be checked by RA87, of course.

6. Documentation

RA87 V1.5/V1.6:

This S/W is documented by the 'User's Manual RA87 Relocatable Assembler' of 7/85, V1.5.

RA87 V2.0 (onward):

This S/W is documented by the 'User's Manual RA87/RA310 Relocatable Assembler' of 8/86, V2.1 or later (It is valid for RA310 S/W as well).

An 'Info'-file on the delivered assembler FD informs about inconveniences or bugs of the current S/W versions (Demo-files are attached as well).

The μCOM-87 Floating Point Arithmetic Operation Package

Contents:

Author: NEC
KAWASAKI TECHNOLOGY CENTER/TOKYO

Person to contact: R. Cherrington
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

DBMICROCOM077V10	Data Book
μPD78C10/12/14	Product Description
μPD7807/08/09	Product Description

Related Products

μPD7807/08/09	8-Bit Microcomputer	NMOS
μPD78C10/11/12/14	8-Bit Microcomputer	CMOS

PART I
Fixed Point Arithmetic Operations

PREFACE

The μCOM87 series are NEC's original 8-bit, one-chip microcomputers having a 16-bit ALU. They are provided with a variety of 16-bit ALU multiplication/division instructions, 16-bit operation instructions, etc. This document describes basic software functions, including fixed point rule operation programs, so that the many powerful instructions may be fully utilized.

Also read Application Note (II) for floating point operation programs and Application Note (III) for programs useful in hardware functions.

Note: This application note is applicable to the following types of microcomputers:

μPD7811/7810/78PG11, μPD7811H/7810H/78PG11H, μPD78C14/78C11/78C10/78CG14, μPD7809/7808/7807/78P09

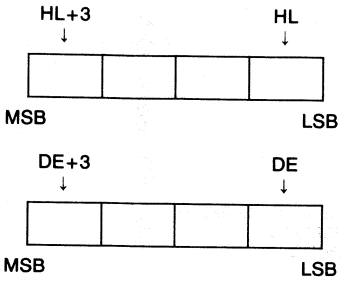
Chapter 1 Four Rules

1.1 Binary Operations

1.1.1 Binary Addition

32-Bit + 32-bit → 32-bit

a) Description of memory



b) Registers to be used

A, C, D, E, H, L

c) Input conditions

As shown in a) above,

HL pair register ← First address of area where 32-bits of augend is stored.

DE pair register ← First address of area where 32-bits of added is stored.

d) Output conditions

RET: The operation result overflows.

RETS: The normal operation result is stored in the four successive bytes of (HL, HL+1, . . ., HL+3) shown in a) above.

- **Programm List**

```
;  ++++++++++++++++++++++++++++++++++++++++++++++
;  +                                            +
;  +          BINARY  ADDITION                  +
;  +            32BIT <- 32BIT + 32BIT          +
;  +                                            +
;  +        INPUT : HL <= AUGEND TOP ADDRS      +
;  +                DE <= ADDIT TOP ADDRS       +
;  +                                            +
;  +        OUTPUT : RET .. OVERFLOW            +
;  +                 RETS .. NORMAL             +
;  +                                            +
;  ++++++++++++++++++++++++++++++++++++++++++++++
;
BFADD:
        CLC                                      ①
        MVI     C,4-1                            ②
        ;
BFAD1:  LDAX    H                                ③
        ADCX    D+                               ④
        STAX    H+                               ⑤
        ;
        DCR     C                                ⑥
        GJMP    BFAD1                            ⑦
        ;
        SKN     CY          ; CHECK / OVERFLOW ?
        RET                 ; OVERFLOW           ⑧
        RETS                ; NORMAL             ⑨
```

① The carry is cleared in advance.

② The byte counter is set to (4-1).

③ One byte of augend is loaded into the accumulator.

④ One byte of added is added to the accumulator and the addend address is increased by one.

⑤ The operation result is stored into the augend memory and the augend address is increased by one.

⑥ The count of the byte counter is decreased by one and is skipped at the end of operation.

⑦ The operation loops up to the end of operation.

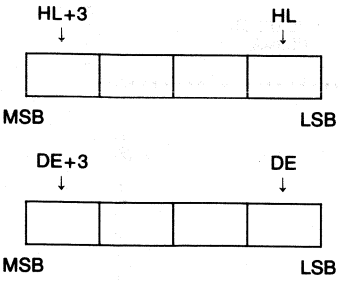
⑧ The operation result overflows.

⑨ The operation result is normal.

1.1.2 Binary Subtraction

32-Bit — 32-bit → 32-bit

a) Description of memory



b) Registers to be used

A, C, D, E, H, L

c) Input conditions

As shown in a) above,

HL pair register ← First address of area where 32-bits of minuend are stored.

DE pair register ← First address of area where 32-bits of subtrahend are stored.

d) Output conditions

RET: A borrow has resulted because subtrahend was smaller than minuend.

RETS: The normal operation result is stored in the four successive bytes of (HL, HL+1, . . ., HL+3) shown in a) above.

- **Programm List**

```
;  ++++++++++++++++++++++++++++++++++++++++++++++
;  +                                            +
;  +          BINARY  SUBTRACTION               +
;  +            32BIT <- 32BIT - 32BIT          +
;  +                                            +
;  +        INPUT : HL <= MINUEND TOP ADDRS     +
;  +                DE <= SUBTRC TOP ADDRS      +
;  +                                            +
;  +        OUTPUT : RET .. OVERFLOW            +
;  +                 RETS .. NORMAL             +
;  +                                            +
;  ++++++++++++++++++++++++++++++++++++++++++++++
;
BFSUB:
        CLC                                     ①
        MVI     C,4-1                           ②
        ;
BFSB1:  LDAX    H                               ③
        SBBX    D+                              ④
        STAX    H+                              ⑤
        ;
        DCR     C                               ⑥
        GJMP    BFSB1                           ⑦
        ;
        SKN     CY          ; CHECK / OVERFLOW ?
        RET                 ; BORROW            ⑧
        RETS                ; NORMAL            ⑨
```

① The borrow is cleared in advance.

② The byte counter is set to (4-1).

③ One byte of minuend is loaded into the accumulator.

④ One byte of subtrahend, including the carry, is subtracted from the accumulator and the subtrahend address is increased by one.

⑤ The operation result is stored into the minuend memory and the minuend address is increased by one.

⑥ The count of the byte counter is decreased by one and is skipped at the end of operation.

⑦ The operation loops up to the end of operation.

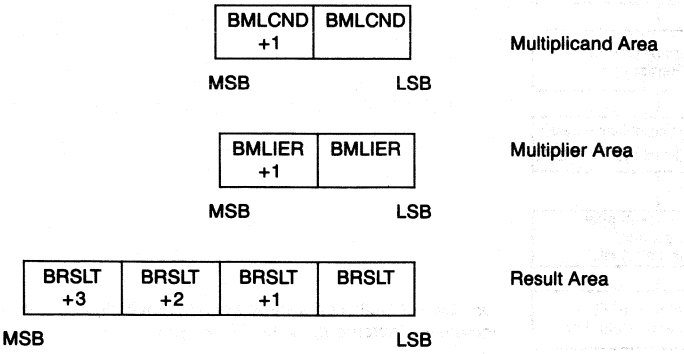
⑧ The operation result borrows.

⑨ The operation result is normal.

1.1.3 Binary Multiplication

16-Bit x 16-bit → 32-bit

a) Description of memory



b) Registers to be used

A, B, C, D, E, H, L, EA

c) Input conditions

16-bit of multiplicand and 16-bit of multiplier are stored in (BMLCND, BMLCND +1) and (BMLIER, BMLIER +1), respectively, shown in a) above.

d) Output conditions

RET: The normal operation result is stored in (BRSLT, BRSLT +1, . . ., BRESLT +3) shown in a) above.

e) Processing procedure

Because a multiplication instruction peculiar to the μCOM-87AD is used for this operation; 16-bits of multiplier is divided into upper 8-bits and lower 8-bits, and 16-bits of multiplicand is multiplied by 8-bits of multiplier twice.

① First, the result area (BRSLT, BRSLT +;, BRSLT +2, BRSLT +3) is 0-cleared for initialization.

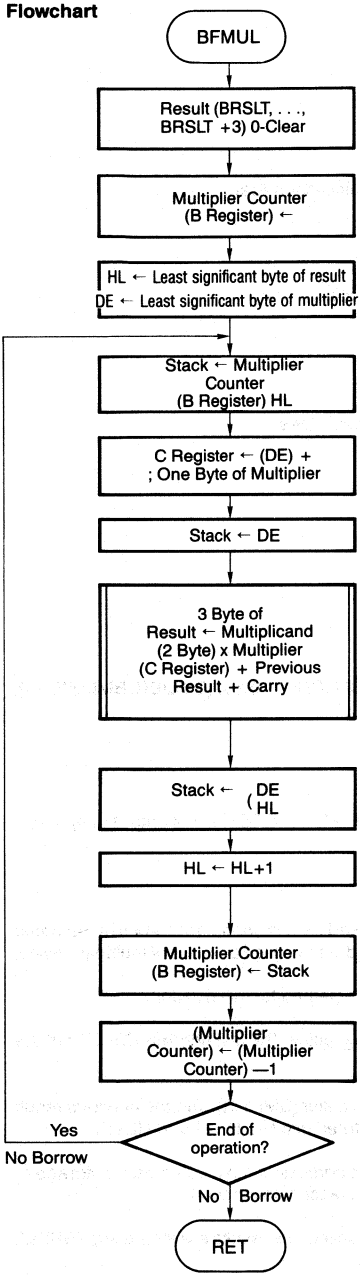
② To store the result of operation to the result area to be carried out in ③ below, the first address (BRSLT) of the result area is set by the HL register and is saved into the stack.

③ Multiplicand (BMLCND, BMLCND +1) is multiplied by one byte of multiplier (BMLIER), and the previous result (HL, HL+1, HL+2) is added to the product. After that, the result is stored into HL, HL+1 and HL+2.

④ Because the upper digit (BMLIER +1) serves as a multiplier, the first address of the result area for storage of result ③ (HL pair register) is unloaded from the stack and slightly increased (increment).

⑤ Steps ③ and ④ are carried out by using BMLIER as one byte of multiplier. The results are stored into (BRSLT, BRSLT +1, BRSLT +2, BRSLT +3).

Flowchart

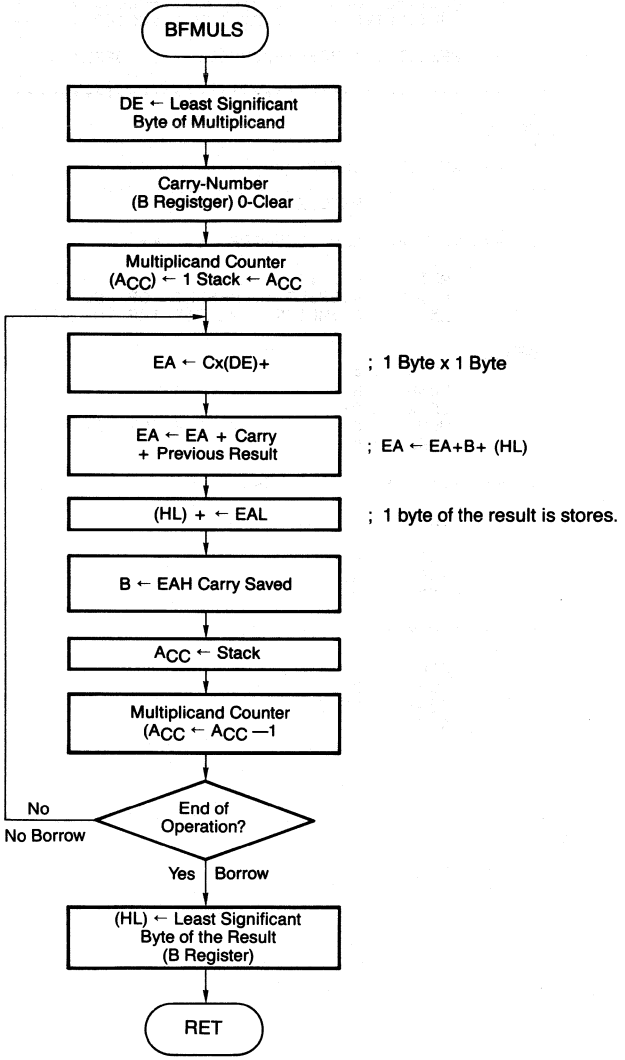


; Subroutine "RSTCLR"

; One byte of multiplier is loaded and the multiplier address is slightly increased (increment) by the DE register.

; Subroutine "BFMULS" (HL, HL+1, HL+2) ← Bytes of Multiplicand x 1 Bytre of Multiplier + Carry (B Register) + Previous Result (HL, HL+1, HL+2).

; Because the upper one digit serves as a multiplier, the result store address is set to an upper position by one word.


BFMULS
DE ← Least Significant Byte of Multiplicand
Carry-Number (B Registger) 0-Clear
Multiplicand Counter (ACC) ← 1 Stack ← ACC
EA ← Cx(DE)+
; 1 Byte x 1 Byte
EA ← EA + Carry + Previous Result
; EA ← EA+B+ (HL)
(HL) + ← EAL
; 1 byte of the result is stores.
B ← EAH Carry Saved
ACC ← Stack
Multiplicand Counter (ACC ← ACC —1
End of Operation?
No
No Borrow
Yes
Borrow
(HL) ← Least Significant Byte of the Result (B Register)
RET

- **Program List**

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +       BINARY  MULTIPLY                   +
;  +            32BIT <- 16BIT * 16BIT        +
;  +                                          +
;  +   MULTIPLICAND .. ( BMLCND+1,BMLCND )    +
;  +   MULTIPLIER   .. ( BMLIER+1,BMLIER )    +
;  +                                          +
;  +   RESULT ..( BRSLT+3,BRSLT+2,...,BRSLT )  +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
BFMUL:
;
;  **  RESULT  0CLEAR  **
;
        CALL    RSTCLR
;
;  **  MULTIPLICAND  COUNTER SET  **
;
        MVI     B,2-1
        LXI     H,BRSLT  ; RESULT TOP ADDRS
        LXI     D,BMLIER ; MULTIPLIER TOP ADDR
;
BFMUL1:
        PUSH    B
        PUSH    H
        LDAX    D+
        MOV     C,A      ;  MULTIPLIER SET
        PUSH    D
;
;  --  16BIT * 8BIT ->  24BIT --
;
        CALL    BFMULS
;
        POP     D
        POP     H
        INX     H
;
;  **  CHECK /  MULTIPLY END
;
        POP     B
        DCR     B
        GJMP    BFMUL1
        RET
```

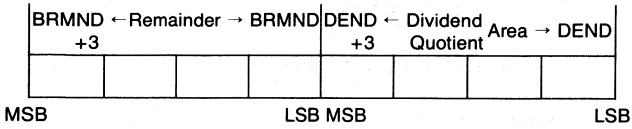
- **Program List**

```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                 S
; S --  24BIT <- 16BIT * 8BIT --    S
; S                                 S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BFMULS:
        LXI     D,BMLCND
        MVI     B,0
                ;  CY 0CLEAR
        MVI     A,2-1
;
;  --  8BIT * 8BIT + CY + RESULT
;
MULSS:
        PUSH    V
        LDAX    D+
        MUL     C
        EADD    EA,B
        LDAX    H
        EADD    EA,A
        MOV     A,EAL
        STAX    H+
        ;
        MOV     A,EAH
        MOV     B,A     ; CY STORE !
;
;  **  CHECK / 16BIT*8BIT  END ? '  **
;
        POP     V
        DCR     A
        GJMP    MULSS
        ;
        MOV     A,B
        STAX    H
        RET


;  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;  S                                               S
;  S           COMMON  SUBROUTINE                  S
;  S                                               S
;  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
;  **  0-CLEAR  PROCESS  **
;
HIMCLR: LXI     H,DMLCND+4
RMNDCL: LXI     H,RMIND
RSTCLR: LXI     H,BRSLT
DIVCLR: LXI     H,BRMND
        ;
        MVI     C,4-1
        ;
RCLR0:  MVI     A,0
RCLR1:  STAX    H+
        DCR     C
        GJMP    RCLR1
        RET
```

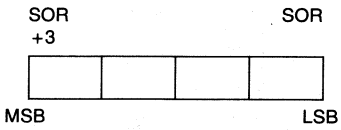
1.1.4 Binary Division

32-Bit ÷ 32-bit → Quotient 32-bit
Remainder 32-bit

a) Description of memory



; Dividend, Quotient, Remainder Data Areas



; Divisor Data Areas

b) Registers to be used

A, B, C, D, E, H, L, EA

c) Input conditions

32-bits of dividend and 32-bits of divisor are stored in the dividend (DEND, . . ., DEND +3) and the divisor (SOR, . . ., SOR +3) respectively. (Shown in a) above.)

d) Output conditions

RET: When divisor = 0

RETS: The normal operation result (quotient and remainder) is stored in (DEND, . . ., DEND +3) and (BRMND, . . ., BRMND +3) respectively. (Shown in a) above.)

e) Processing procedure

In this operation program the dividend (DEND, DEND +1, . . ., DEND +3) and remainder (BRMND, BRMND +1, . . ., BRMND +3) are 8-byte consecutive data. 4-bits of the dividend and 4-bits of the remainder each are left-shiftet. The most significant 4-bits of the dividend are transferred to the least significant area of the remainder, and the quotient is loaded into the least significant area of the dividend by 4-bits each from the minimum value.

The quotient of each digit is equal to the number of times a subtraction of divisor from remainder is repeated until a borrow occurs.

① The divisor (SOR, SOR +1, . . ., SOR +3) is 0-checked. If 0, the operation ends.

② The remainder area (BRMND, . . ., BRMND +3) is 0-cleared.

③ The quotient byte counter (B register) is set to 7.

④ The 8-bit consecutive quotient and remainder are left-shiftet by 4-bits.

⑤ The divisor is subtracted from the remainder. If a borrow occurs, the operation jumps to ⑦.

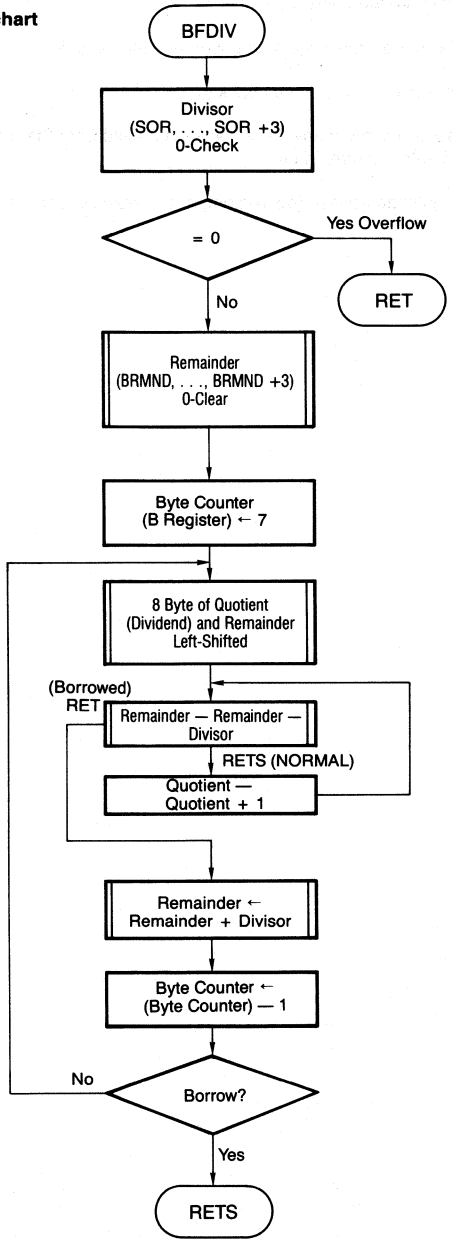
⑥ One is added of the quotient (DEND). Then, the operation jumps back to ⑤.

⑦ The result of ⑤ has become negative because one too many divisor was subtracted from the remainder. To reset the original state, the divisor is added to the remainder.

⑧ The count of the quotient byte counter is slightly decreased (decrement). If a borrow occurs, the operation ends.

If not, the operation jumps back to ④.

Flowchart



; Subroutine 'DIVCLR'

; Subroutine 'BCDLS'
The most significant 4 bits of dividend is set for the least significant 4 bits of remainder.

; Subroutine 'BFSUB'

; Subroutine 'BFADD'
The original state is reset after one too many divisor was subtracted from the remainder.

; End of operation?

- **Program List**

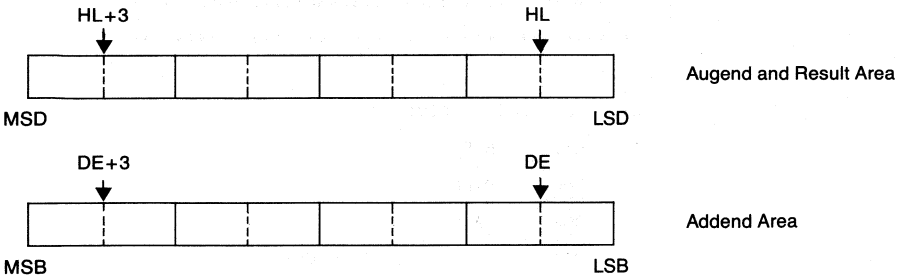
```
;  ++++++++++++++++++++++++++++++++++++++++++++++
;  +                                            +
;  +          BINARY  DIVISION                  +
;  +            32BIT <- 32BIT / 32BIT          +
;  +                                            +
;  +                                            +
;  +  DIVIDEND .. ( DEND+3,DEND+2,...,DEND )    +
;  +  SOR      .. ( SOR+3,SOR+2,...,SOR )       +
;  +  QUOTIENT .. ( DEND+3,DEND+2,...,DEND )    +
;  +  REMIND   .. ( BRMND+3,BRMND+2,...,BRMND ) +
;  +                                            +
;  ++++++++++++++++++++++++++++++++++++++++++++++
;
BFDIV:
;
;  **  CHECK FOR  / DIVISOR=0
;
        LXI     H,SOR
        LDEAX   H++
        DMOV    D,EA
        LDEAX   H
        DOR     EA,D
        SKN     Z
        RET             ; IF  OVERFLOW
;
;  **  QUOTIENT  0CLEAR  **
;
        CALL    DIVCLR  ;(Page 1-10)
;
;  **  BYTE-COUNTER  SET  **
;
        MVI     B,8-1
;
;  **  DIVIDEND,RMIND  1BYTE LEFT-SHIFT  **
;
BFDIV2:
        LXI     H,DEND
        MVI     C,8-1   ; SHIFT, BYTE-COUNTER SET
        CALL    BCDLS   ; SHIFT ! ;(Page 4-1)
;
;  **  SUBTRACT DIVISOR  FROM DIVIDEND  **
;
BFDIV3:
        LXI     H,BRMND
        LXI     D,SOR
        ;
        CALL    BFSUB  ;(Page 1-4)
        GJMP    BFDIV4          ; IF BORROW
                                ; IF NO BORROW
        INRW    DEND MOD 100H   ; 1-DIGIT QUOT (+1)
        GJMP    BFDIV3
;
;  **  IF BORROW , DIVISOR + DIVIDEND  **
;
BFDIV4:
        LXI     H,BRMND
        LXI     D,SOR
        CALL    BFADD  ;(Page 1-2)
        NOP
;
;  **  BIT COUNTER DECREMENT  **
;
        DCR     B       ; SKIP, IF END
        GJMP    BFDIV2
        ;
        RETS
```

1.2 Decimal Operations

1.2.1 Decimal Addition

8-Bit + 8-bit → 8-bit

a) Description of memory



b) Registers to be used

A, C, D, E, H, L

c) Input conditions

As shown in a) above,

HL pair register ← First address of area where 8-bits (4 byte) of augend is stored.

DE pair register ← First address of area where 8-bits of added is stored.

d) Output conditions

RET: The operation result overflows.

RETS: The normal operation result is stored in the four successive bytes of (HL, HL+1, . . ., HL+3) shown in a) above.

- **Programm List**

```
;
;  +++++++++++++++++++++++++++++++++++++++++++++
;  +                                           +
;  +          DECIMAL  SUBTRACTION             +
;  +       8-PLACE <- 8-PLACE - 8-PLACE        +
;  +                                           +
;  +    MINUEND .. ( HL+3,..,HL )              +
;  +    SUBTRACTER .. ( DE+3,..,DE )           +
;  +    RESULT  .. ( HL+3,..,HL )              +
;  +                                           +
;  +     OUTPUT : RET .. OVERFLOW              +
;  +              RETS .. NORMAL               +
;  +                                           +
;  +++++++++++++++++++++++++++++++++++++++++++++
;
BCDSUB:
        MVI     C,4-1
        ;
BCDSB1: STC                                     ①
BCDSB2: MVI     A,99H   ; FOR ADJUST            ②
        ACI     A,0
        SBBX    D+                              ③
        ADDX    H                               ④
        DAA             ; DECIMAL ADJUST        ⑤
        STAX    H+                              ⑥
        ;
        DCR     C                               ⑦
        GJMP    BCDSB2                          ⑧
        ;
        SK      CY      ; CHECK / OVERFLOW ?
        RET             ; BORROW                ⑨
        RETS            ; NORMAL                ⑩
```

① The carry is cleared in advance.

② The byte of augend is loaded into the accumulator.

③ One byte of addend is added to the accumulator and the addend address is increased by one..

④ The operation result is decimally adjusted. If there is a carry, it is set.

⑤ The operation result is stored in the augend area and the augend address is increased by one.

⑥ The count of the digit-number counter is decreased by one and is skipped at the end of operation.

⑦ The operation loops up to the end of operation.

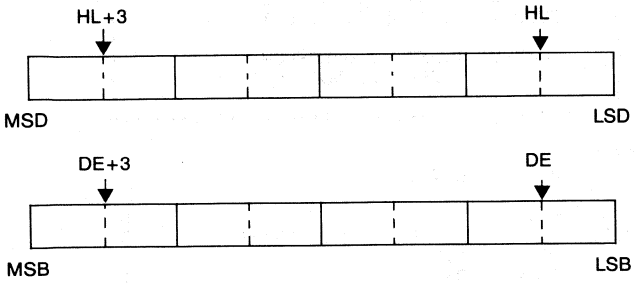
⑧ The operation result overflows.

⑨ The operation result is normal.

1.2.2 Decimal Subtraction

8-Bit — 8-bit → 8-bit

a) Description of memory



b) Registers to be used

A, C, D, E, H, L

c) Input conditions

As shown in a) above,

HL pair register ← First address of area where 8-bits (4 byte) of minuend are stored.

DE pair register ← First address of area where 8-bits (4 byte) of subtrahend are stored.

d) Output conditions

RET: A borrow has resulted because minuend was smaller than subtrahend.

RETS: The normal operation result is stored in the four successive bytes of (HL, HL+1, . . ., HL+3) shown in a) above.

- **Programm List**

```
;
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +          DECIMAL  SUBTRACTION            +
;  +       8-PLACE <- 8-PLACE - 8-PLACE       +
;  +                                          +
;  +    MINUEND .. ( HL+3,..,HL )             +
;  +    SUBTRACTER .. ( DE+3,..,DE )          +
;  +    RESULT  .. ( HL+3,..,HL )             +
;  +                                          +
;  +     OUTPUT : RET .. OVERFLOW             +
;  +              RETS .. NORMAL              +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
BCDSUB:
        MVI     C,4-1
        ;
BCDSB1: STC                                     ①
BCDSB2: MVI     A,99H   ; FOR ADJUST            ②
        ACI     A,0
        SBBX    D+                              ③
        ADDX    H                               ④
        DAA             ; DECIMAL ADJUST        ⑤
        STAX    H+                              ⑥
        ;
        DCR     C                               ⑦
        GJMP    BCDSB2                          ⑧
        ;
        SK      CY      ; CHECK / OVERFLOW ?
        RET             ; BORROW                ⑨
        RETS            ; NORMAL                ⑩
```

① The carry is preset for subtraction.

② Preparations are made for the complement representation of subtrahend.

③ The subtrahend is converted into 100's complement.

④ The complement of subtrahend is added to the minuend (addition).

⑤ Decimal adjustment is carried out.

⑥ The operation result is stored into the minuend area and the minuend address is increased by one.

⑦ The count of the digit-number counter is decreased by one.

⑧ The operation loops up to the end of operation.

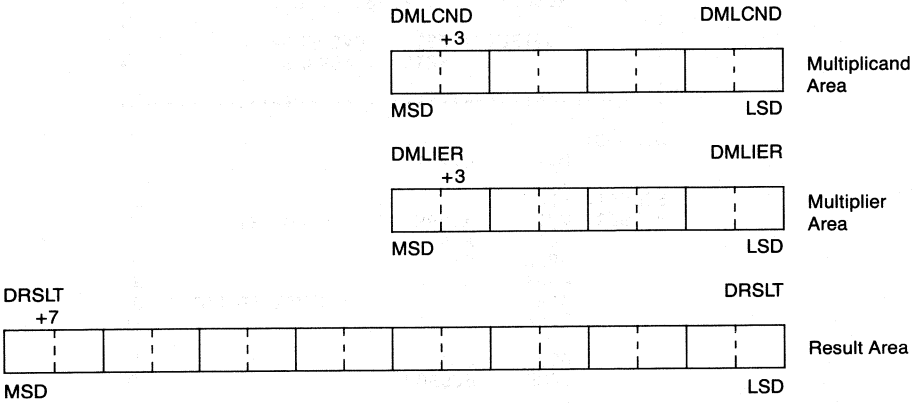
⑨ The operation result borrows.

⑩ The operation result is normal.

1.2.3 Decimal Multiplication

8-bit x 8-bit → 16-bit

a) Description of memory



b) Registers to be used

A, B, C, D, E, HL

c) Input conditions

8-bits of multiplicand and 8-bits of multiplier are stored in (DMLCND, DMLCND +3) and (DMLIER, DMLIER +3) respectively. (Shown in a) above.)

d) Output conditions

RET: The normal operations result is stored in the result (DRSLT, . . ., DRSLT +7), shown in a) above.

e) Processing procedure

In this operation program the multiplier is right-shifted, loads from LSD one digit at a time to serve as a counter for repeating the addition of multiplicand to the result. The result is right-shifted by one digit because a multiplier at a higher position by one word is added after completion of addition for one digit of multiplier (counter).

① The result area is 0-cleared.

② The digit counter (C register) is set to 7.

③ The digit counter is saved into the stack.

④ The multiplier is right-shifted and the LSD (counter) is loaded into the B register.

⑤ The count of the counter (B register) is slightly decreased (decrement).

⑥ If a borrow occurs, the addition ends and the operation jumps to ⑨.

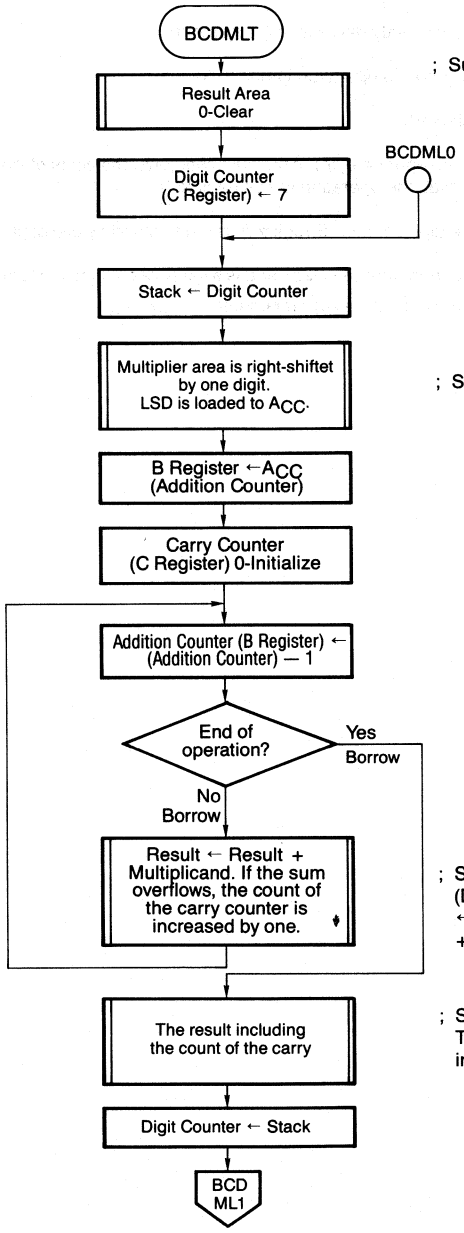
⑦ The carry counter (C register) is 0-cleared.

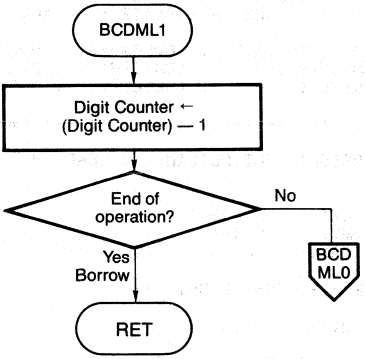
⑧ The multiplicand is added to the result (upper 4-byte). If the sum overflows, the count of the carry counter is slightly increased (increment). After that, the operation jumps back to ⑤.

⑨ The result, including the count of the digit counter (B register), is right-shifted by one digit.

⑩ The count of the digit counter is unloaded from the stack and is slightly decreased (decement).
If the result borrows, the operation ends. If not, the operation jumps back to ③.

Flowchart


BCDMLT
; Subroutine 'RCLRO'
Result Area 0-Clear
BCDML0
Digit Counter (C Register) ← 7
Stack ← Digit Counter
Multiplier area is right-shiftet by one digit. LSD is loaded to ACC.
; Subroutine 'BCDRS'
B Register ←ACC (Addition Counter)
Carry Counter (C Register) 0-Initialize
Addition Counter (B Register) ← (Addition Counter) — 1
End of operation?
Yes
Borrow
No
Borrow
Result ← Result + Multiplicand. If the sum overflows, the count of the carry counter is increased by one.
; Subroutine 'BCDADD'
(DRSLT + 4, . . ., DSRSLT + 7)
← (DRSLT + 4, . . ., DRSLT + 7)
+ (DMLCND, . . ., DMLCND +3)
The result including the count of the carry
; Subroutine 'BCDRS1'
The counter of the carry counter remains in MSD.
Digit Counter ← Stack
BCD ML1


BCDML1
Digit Counter ← (Digit Counter) — 1
End of operation?
No
BCD ML0
Yes Borrow
RET

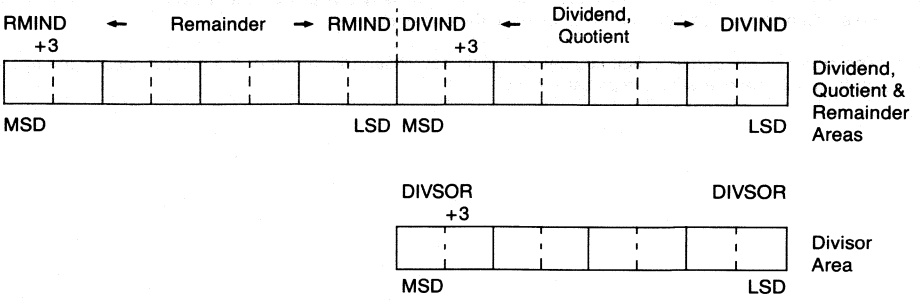
- **Program List**

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +        DECIMAL  MULTIPLY                 +
;  +      16 PLACE <- 8 PLACE * 8PLACE        +
;  +                                          +
;  +   MULTIPLICAND .. (DMLCND+3,..,DMLCND)   +
;  +   MULTIPLIER   .. (DMLIER+3,..,DMLIER)   +
;  +   RESULT       .. (DRSLT+7,...,DRSLT )    +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
;  **  RESULT,MULTIPLICAND  0CLEAR  **
;
BCDMLT:
        LXI     H,DRSLT
        MVI     C,8-1
        CALL    RCLR0  ; (Page 1-10)
;
;  **  PLACE COUNTER SET  **
;
        MVI     C,16/2-1
;
;  **  MULTIPLIER  RIGHT-SHIFT  **
;
BCDML1:
        PUSH    B       ; PLACE-COUNTER STORE
        ;
        LXI     H,DMLIER+3
        MVI     C,8/2-1
        CALL    BCDRS  ;(Page 3-5)
        PUSH    V
;
;  **  CHECK / MULTIPLIER=0'  **
;
        MVI     B,0
BCDML2:
        POP     V
        SUINB   A,1
        GJMP    BCDML4  ; IF  =0
;
;  **  RESULT + MULCND -> RESULT  **
;
        PUSH    V
        LXI     H,DRSLT+4
        LXI     D,DMLCND
        CALL    BCDADD ; SKIP ,IF NORMAL    ;(Page 1-16)
        INR     B      ; IF OVERFLOW
                       ; DIGIT ADDITION / NOT END
        GJMP    BCDML2
;
;  **  RESULT  RIGHT-SHIFT  WITH CARRY  **
;
BCDML4:
        MOV     A,B         ; LOAD  RESULT-MSD
        LXI     H,DRSLT+7
        MVI     C,16/2-1    ; SHIFT-COUNTER SET
        CALL    BCDRSI      ;(Page 3-5)

;
;  **  CHECK  / MULTIPLY END ?'  **
;
        POP     B
        DCR     C
        GJMP    BCDML1
        RET             ; END !
```

1.2.4 Decimal Division

8-bit ÷ 8-bit → Quotient 8-bit
Remainder 8-bit

a) Description of memory



b) Registers to be used

A, B, C, D, E, H, L, EA

c) Input conditions

8-bits of dividend and 8-bits of divisor are stored in the dividend (DIVIND, . . ., DIVIND +3) and the divisor (DIVSOR, . . ., DIVSOR +3) respectively. (Shown in a) above.)

d) Output conditions

RET: Because divisor = 0, overflows.

RETS: The normal operation result (quotient and remainder) is stored in (DIVIND, . . ., DIVIND +3) and (RMIND, . . ., RMIND +3) respectively. (Shown in a) above.)

e) Processing procedure

In this operation program the dividend (DIVIND, DIVIND +1, . . ., DIVIND +3) and remainder (RMIND, RMIND +1, .. ., RMIND +3) are 8-byte consecutive data.

One digit of the dividend and one digit of the remainder are each left-shiftet, the most significant digit of the dividend is transferred to the least significant area of the remainder, and the quotient is loaded into the least significant area of the dividend by one digit each from the minimum value.

The quotient of each digit is equal to the number of times a subtraction of divisor from remainder is repeated until a borrow occurs.

① The divisor (DIVSOR, DIVSOR +1, . . ., DIVSOR +3) is 0-checked. If 0, the operation ends.

② The remainder area (RMIND, . . ., RMIND +3) is 0-cleared.

③ The quotient digit counter (B register) is set to 7.

④ The 8-bit consecutive quotient and remainder are left-shiftet by one digit (4-bit).

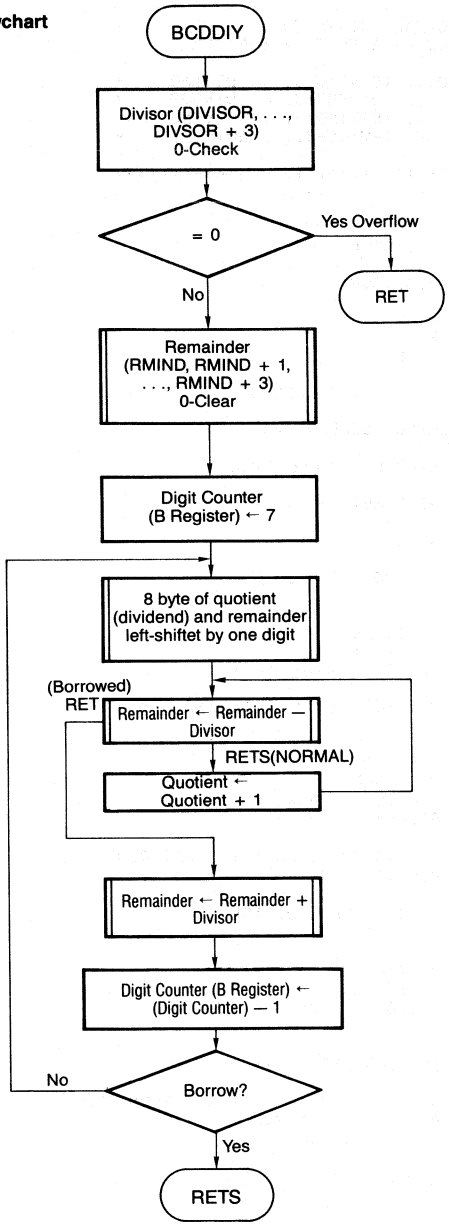
⑤ The divisor is subtracted from the remainder. If a borrow occurs, the operation jumps to ⑦.

⑥ One is added to the quotient (DIVIND). Then, the operation jumps back to ⑤.

⑦ The result of ⑤ has become negative because one too many divisor was subtracted from the remainder. To reset the original state, the divisor is added to the remainder.

⑧ The count of the quotient digit counter is slightly decreased (decrement). If a borrow occurs, the operation ends. If not, the operation jumps back to ④.

Flowchart


BCDDIY
Divisor (DIVISOR, . . ., DIVSOR + 3) 0-Check
= 0
Yes Overflow
RET
No
; Subroutine 'RMNDCL'
Remainder (RMIND, RMIND + 1, . . ., RMIND + 3) 0-Clear
Digit Counter (B Register) ← 7
8 byte of quotient (dividend) and remainder left-shiftet by one digit
; Subroutine 'BCDLS'
The MSD of dividend is loaded into the LSD area of remaindere.
(Borrowed) RET
Remainder ← Remainder — Divisor
; Subroutine 'BCDSUB'
RETS(NORMAL)
Quotient ← Quotient + 1
Remainder ← Remainder + Divisor
; Subroutine 'BCDADD'
The original state is reset after one too many divisor was subtracted from the remainder.
Digit Counter (B Register) ← (Digit Counter) — 1
No
Borrow?
End of operation?
Yes
RETS

- **Program List**

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +          DECIMAL  DIVISION               +
;  +       8-PLACE <- 8-PLACE / 8-PLACE       +
;  +                                          +
;  +     DIVIDEND .. (DIVIND+3,...,DIVIND )   +
;  +     DIVISOR  .. (DIVSOR+3,...,DIVSOR )   +
;  +     RSLT     .. (DIVIND+3,...,DIVIND )   +
;  +     REMIND   .. (RMIND+3,...,RMIND )     +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
BCDDIV:
;
;  **  CHECK / DIVISOR=0'  **
;
        LXI     H,DIVSOR
        LDEAX   H++
        DMOV    D,EA
        LDEAX   H
        DOR     EA,D
        SKN     Z
        RET             ; OVERFLOW
;
;  **  RESULT,REMIND  0CLEAR  **
;
        CALL    RMNDCL ; (Page 1-10)
;
;  **  DIGIT-COUNTER SET  **
;
        MVI     B,8-1
;
;  **  QUOTIENT,REMIND  LEFT-SHIFT  **
;
BCDDV1:
        LXI     H,DIVIND
        MVI     C,16/2-1
        CALL    BCDLS  ; (Page 4-1)
;
;  **  SUBTRACT DIVISOR  FROM  DIVIDEND  **
;
BCDDV2:
        LXI     H,RMIND
        LXI     D,DIVSOR
        CALL    BCDSUB  ; (Page 1-18)
        GJMP    BCDDV3
        ;
        INRW    DIVIND MOD 100H ; 1-DIGIT.QUOT (+1)
        GJMP    BCDDV2
;
;  **  IF BORROW , DIVISOR + DIVIDEND
;
BCDDV3:
        LXI     H,RMIND
        LXI     D,DIVSOR
        CALL    BCDADD  ; (Page 1-16)
        NOP
;
;  **  CHECK / DIVISION END ?  **
;
        DCR     B       ; SKIP , IF DIVISION END
        GJMP    BCDDV1  ; NOT END !
        RETS            ; END !
```

Chapter 2 Data Transfer

This chapter describes an example of data transfer from one selected area to another within the memory.

Example: As shown in Figure 2-1, 7-byte data at addresses 1000H to 1006H is transferred to addresses 1100H to 1106H in the memory.

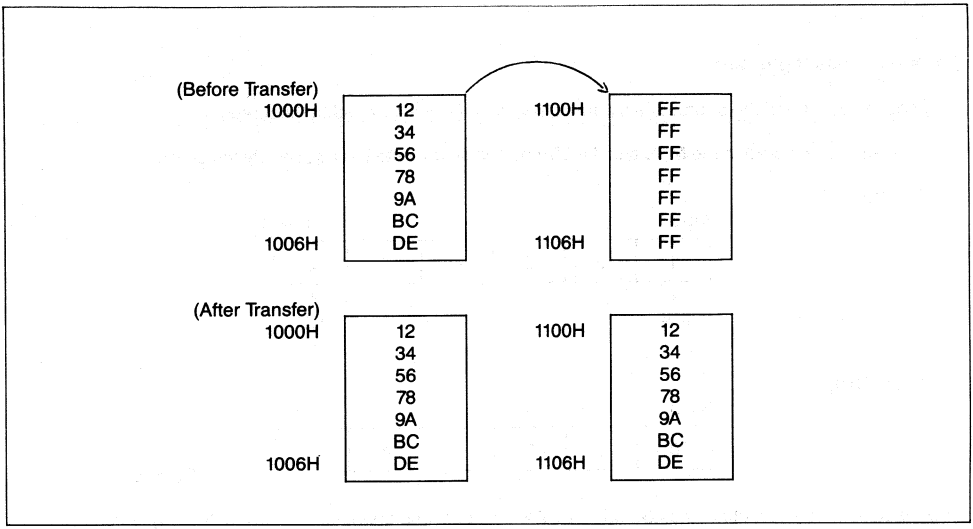


Figure 2-1 Data Transfer Diagram

The μCOM-87AD can carry out N byte transfer between memories using the BLOCK transfer instruction.

Because the transfer source address, the transfer destination address, and the number of bytes to be transferred are automatically selected by pair registers (HL, DE and C) when the BLOCK instruction is executed, it is necessary to set the first address of the transfer destination and the number of bytes to be transferred with those pair registers before transfer execution.

In this example 1000H and 1100H are preset into the HL, DE and C (7—1) pair registers respectively.

- **Program List**

```
ADRS1   EQU     1000H
ADRS2   EQU     1100H
        ;
        LXI     H,ADRS1
        LXI     D,ADRS2
        MVI     C,7-1
        BLOCK
```

Chapter 3 Shift Processing

The μCOM-87AD is provided with 1-bit unit shift instructions (RLL, RLR, etc.) and 4-bit unit shift instructions (RLD and RRD) for the registers (A_{CC}, B and C registers). This chapter describes the following two shift examples:

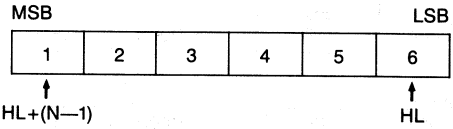
- Byte unit shift
- 4-bit unit shift

3.1 N-Byte Data Right Shift

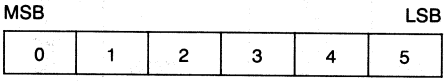
This section describes the subroutine for the right shift on N-byte data within the memory.

For right shift, 0 is stored in the MSB and the LSB data is output onto the A_{CC} as shown below.

(Before Shift)



(After Shift)



; ACC ← 6

Prior to calling the subroutine, the LSB address of N-byte data and the number of bytes, N (> 1), must be set into the HL pair register and the C register respectively.

- **Program List**

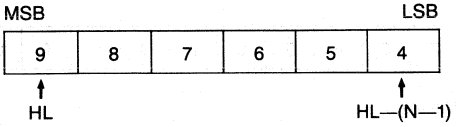
```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                           S
; S     N-BYTE DATA  RIGHT SHIFT              S
; S           HL - LSB                        S
; S           C-REG  <-  BYTE COUNTER         S
; S                                           S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BYTRST:
        DCR     C
        DCR     C
        ;
        LDAX    H
        PUSH    V
        ;
BYTRS1: INX     H
        LDAX    H-
        STAX    H+
        ;
        DCR     C
        GJMP    BYTRS1
        ;
        MVI     A,0
        STAX    H
        POP     V
        RET
```

3.2 N-Byte Data Left Shift

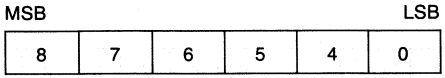
This section describes the subroutine for the left shift of N-byte data within the memory.

For left shift, 0 is stored in the LSB and the MSB data is output onto the A_{CC} as shown below.

(Before Shift)



(After Shift)



; ACC ← 9

Prior to calling the subroutine, the MSB address of N-byte data and the number of bytes, N (> 1), must be set into the HL pair register and the C register respectively.

- **Program List**

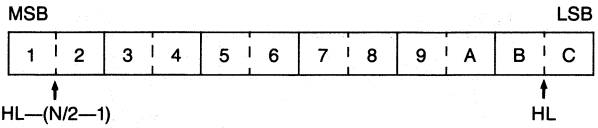
```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                           S
; S     N-BYTE DATA  LEFT SHIFT               S
; S                  HL <-  MSB               S
; S                  C-REG <-  BYTE COUNTER   S
; S                                           S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BYTLST:
        DCR     C
        DCR     C
        ;
        LDAX    H
        PUSH    V
BYTLS1:
        DCX     H
        LDAX    H+
        STAX    H-
        ;
        DCR     C
        GJMP    BYTLS1
        ;
        MVI     A,0
        STAX    H
        POP     V
        RET
```

3.3 N-Digit Data Right Shift

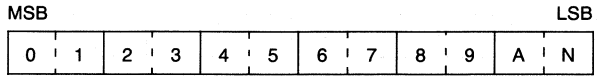
This section describes the subroutine for the 4-bit right shift on N-byte data within the memory.

For right shift, 0 is stored in the MSB and the LSB data is output onto the A_{CC} as shown below.

(Before Shift)



(After Shift)



; ACC ← 0 CH

Prior to calling the subroutine, the LSB address of N-byte BCD and the number of bytes (= N /2—1 where N = 2N' (1, 2, . . .)) must be set into the HL pair register and the C register respectively.

- **Program List**

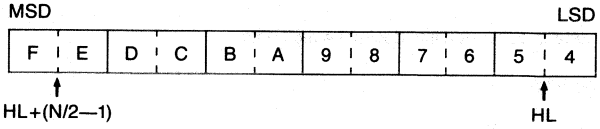
```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                           S
; S      N-DIGIT DATA  RIGHT-SHIFT            S
; S             HL <- MSD                     S
; S             C-REG <- DIGIT COUNTER        S
; S                                           S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BCDRS:  MVI     A,0
;
BCDRS1: RRD
        DCX     H
        DCR     C
        GJMP    BCDRS1
        RET
```

3.4 N-Digit Data Left Shift

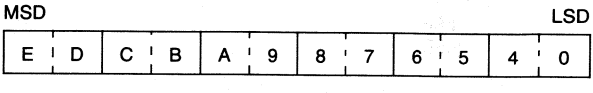
This section describes the subroutine for the 4-bit left shift of N-byte data within the memory.

For left shift, 0 is stored in the LSD and the MSD data is output onto the A_{CC} as shown below.

(Before Shift)



(After Shift)



; ACC ← 0 FH

Prior to calling the subroutine, the LSD address of N-byte BCD and the number of bytes (= N /2—1 where N = 2N' (1, 2, . . .)) must be set into the HL pair register and the C register respectively.

- **Program List**

```
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                               S
; S     N-DIGIT DATA  LEFT-SHIFT                  S
; S             HL <- LSD                         S
; S             C-REG <- DIGIT COUNTER            S
; S                                               S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
BCDLS:  MVI     A,0
;
BCDLS1: RLD
        INX     H
        DCR     C
        GJMP    BCDLS1
        RET
```

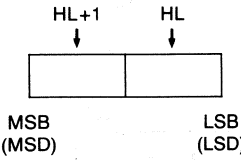
Chapter 4 Data Conversion

This chapter describes the procedure for converting numerical data representation formats between hexadecimal and decimal formats, and between hexadecimal and ASCII formats.

4.1 Conversion from Hexadecimal (HEX) to Decimal (BCD)

Hexadecimal 2 byte data is converted into decimal 4-digit data.

a) Description of memory



; Hexadecimal 2 Byte and Decimal 4-Digit Data Areas

b) Registers to be used

A, B. C, D, E, H, L, EA

c) Input conditions

The LSB address of the area where the hexadecimal 2 bytes of the input data is stored is set into the HL pair register. (As shown in a) above.)

d) Output conditions

RETS: The converted decimal 4-digit data is stored in (HL, HL +1).

RET: No conversion can be carried out because the hexadecimal data is greater then 270FH (= 9999).

e) Processing procedure

This conversion subroutine is carried out to obtain the converted value of decimal 4-digit (2 byte) data by using one digit each from the MSD.

To obtain the converted value of 9876 (= 9 x 1000 + 8 x 100 — 7 x 10 + 6), 1000, 100, 10 and 1, are set for the MSC, the third digit, the second digit and the first digit of the subtrahend, respectively.

The converted value is obtained for each digit from the MSD using the number of times which the subtrahend is subtracted from the hexadecimal input data until a borrow occurs (as the first digit of the converted value).

The procedure is:

① The subtrahend is set to 10000.

② The subtrahend is subtracted from the hexadecimal input data. If a borrow occurs, the operation ends because the difference cannot be represented by 2-byte data.

③ The count of the digit conter is set to 4.

④ The count of the digit counter is slightly decreased (decrement). If a borrow occurs, the operation ends.

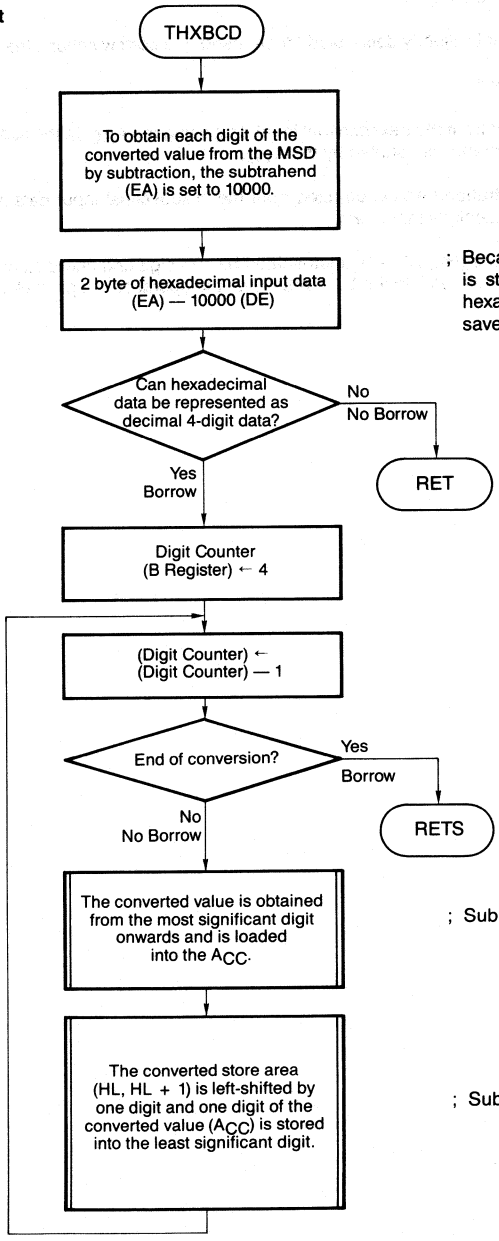
⑤ The subtrahend is divided by 10.

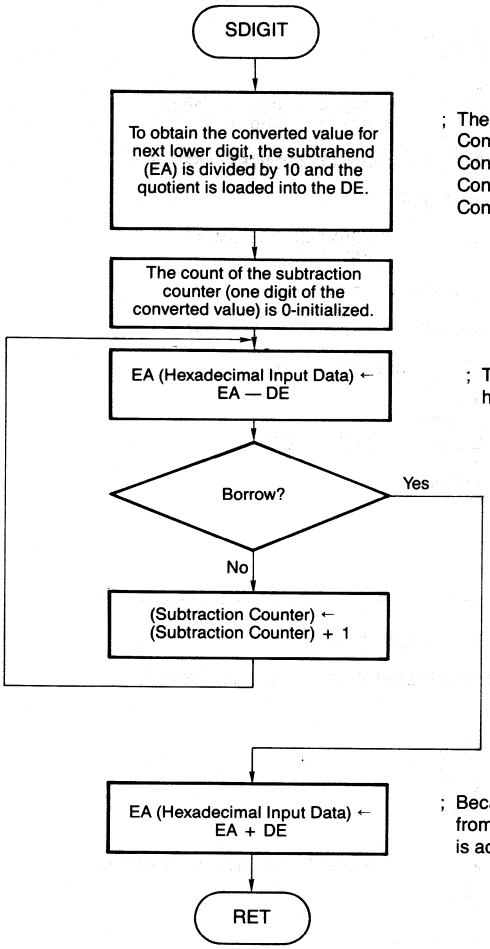
⑥ The subtrahend is subtracted from the hexadecimal input data using the A_{CC} as the subtraction counter until a borrow occurs and the difference is counted by the A_{CC}.

⑦ Because one too many subtrahend was subtracted from the hexadecimal input data in ⑥ above, the subtrahend is added to the hexadecimal input data.

⑧ The converted value store area (HL, HL +1) is left-shifted by one digit and one digit of the converted value (A_{CC}) obtained in ⑥ above is stored into the LSD. After that, the operation jumps back to ④.

Flowchart


THXBCD
To obtain each digit of the converted value from the MSD by subtraction, the subtrahend (EA) is set to 10000.
2 byte of hexadecimal input data (EA) — 10000 (DE)
; Because the converted value is stored into (HL, HL + 1), hexadecimal input data is saved into the EA.
Can hexadecimal data be represented as decimal 4-digit data?
No
No Borrow
RET
Yes
Borrow
Digit Counter (B Register) ← 4
(Digit Counter) ← (Digit Counter) — 1
End of conversion?
Yes
Borrow
RETS
No
No Borrow
The converted value is obtained from the most significant digit onwards and is loaded into the ACC.
; Subroutine 'SDIGIT'
The converted store area (HL, HL + 1) is left-shifted by one digit and one digit of the converted value (ACC) is stored into the least significant digit.
; Subroutine 'BCDLSI'



; The subtrahend is set as follows:
Conversion of 4th decimal digit: EA = 1000
Conversion of 3rd decimal digit: EA = 100
Conversion of 2nd decimal digit: EA = 10
Conversion of 1st decimal digit: EA = 1

; The hexadecimal input data (HL, HL + 1) has been saved into the EA.

; When this subroutine is terminated, the count of the subtraction counter is output as A_{CC}.

; Because one too many DE was subtracted from the hexadecimal input data, one DE is added.

- **Program List**

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +    TRANSFER / BCD  <-  HEX               +
;  +             (HL,HL+1)   (HL,HL+1)        +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
THXBCD:
        LXI     EA,10000
        DMOV    D,EA
        ;
        LDEAX   H
        DLT     EA,D        ; CHECK / HEX > 10000
        RET                 ; HEX > 10000 !
        ;
        MVI     B,4         ; DIGIT-COUNTER  SET !
;
;  **  CHECK / TRANS END  ?  **
;
THXBD1: SUINB   B,1         ; COUNTER DECREMENT
        RETS                ; END !
;
;  **  SUBROUTINE / HEX -> BCD  **
;
        CALL    SDIGIT  ; 1-DIGIT BCD -> ACC
;
;  **  RESULT * 10 + 1-HEX DATA  **
;
        PUSH    H
        MVI     C,4/2-1
        CALL    BCDLS1  ; ACC -> LSD
        POP     H
        GJMP    THXBD1
```

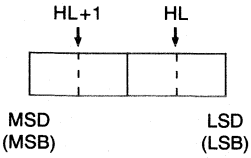
* Program List

```
;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTST
;  T                                            T
;  T    TRANSFORM   SUBROUTINE                  T
;  T                                            T
;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
;
SDIGIT:
        PUSH    EA
        DMOV    EA,D
        MVI     A,10
        DIV     A
        DMOV    D,EA
        POP     EA
        ;
        MVI     A,0         ; RESULT 1-DIGIT 0 INITIAL
SDGT0:  DSUBNB  EA,D
        GJMP    SDGT1
        ;
        INR     A
        GJMP    SDGT0
        ;
SDGT1:  DADD    EA,D        ; FOR ADJUST
        RET
```

4.2 Conversion from Decimal (BCD) to Hexadecimal (HEX)

Decimal 4-digit data is converted into hexadecimal 2-byte data.

a) Description of memory



; Hexadecimal 2 Byte and Decimal 4-Digit Data Areas

b) Registers to be used

A, B, C, H, L, EA

c) Input conditions

The LSD address of the area where the decimal 4 bytes of the input data is stored and is set into the HL pair register as shown in a) above.)

d) Output conditions

RETS: The converted hexadecimal 2-byte data is stored in (HL, HL +1), as shown in a) above.

RET: Input data cannot be represented in decimal notation.

e) Processing procedure

This conversion subroutine is carried out by 0-clearing the converted value store area, loading the decimal input data into the A_{CC} by one digit each from the MSD through one-digit left-shift and repeating calculations of "(store area) x 10 + A_{CC}" four times.

The procedure is:

① The count of the converted value store register (EA) is 0-cleared.

② The digit counter is set to (4—1).

③ The decimal input data is left-shiftet by one digit and the MSD is loaded into the A^{CC}.

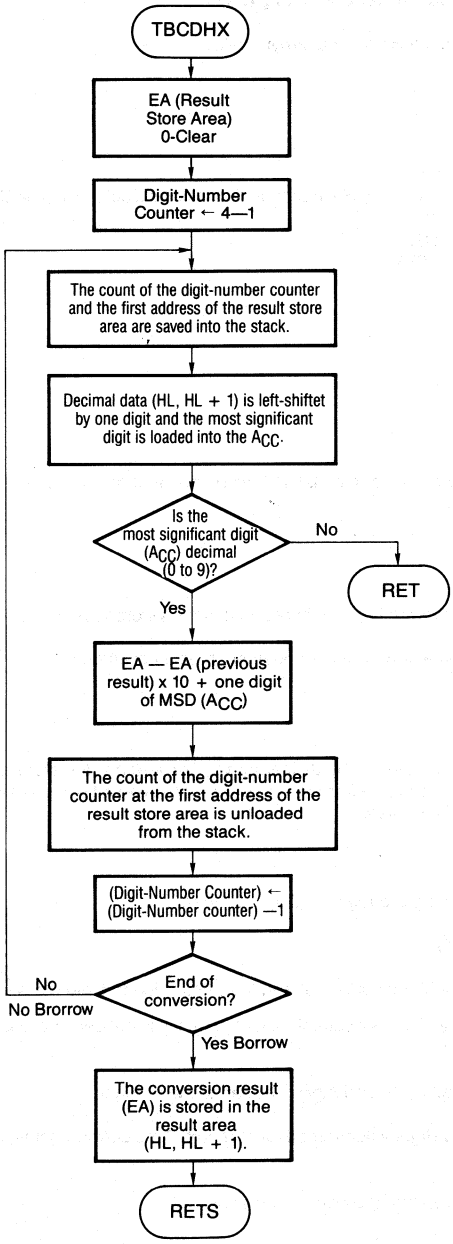
④ The MSD (A_{CC}) is checked to varify it is decimal data (0 to 9). If it is not decimal, the operation ends because no conversion can be carried out.

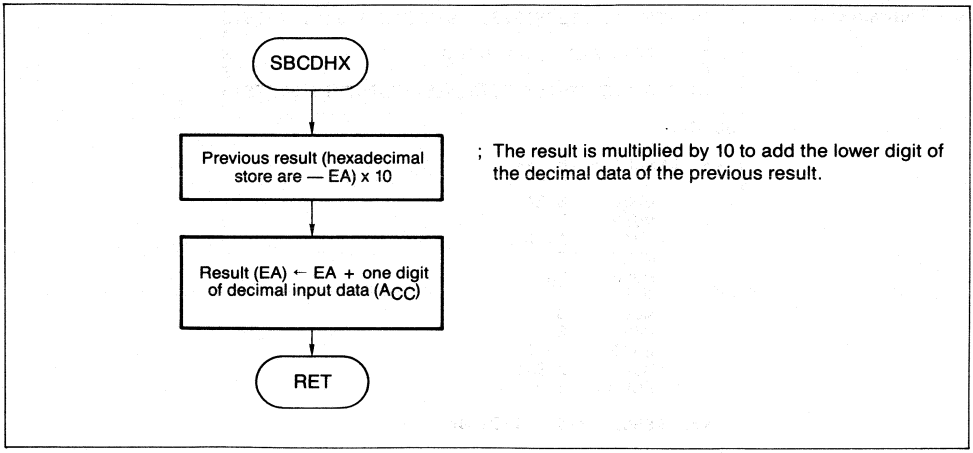
⑤ EA (previous result) is multiplied by 10 and A_{CC} is added to the product.

⑥ The count of the digit counter is slightly decreased (decrement). If no borrow occurs, the operation jumps back to ③.

⑦ The EA is stored in the store area and the conversion ends.

Flowchart


TBCDHX
EA (Result Store Area) 0-Clear
Digit-Number Counter ← 4—1
The count of the digit-number counter and the first address of the result store area are saved into the stack.
Decimal data (HL, HL + 1) is left-shiftet by one digit and the most significant digit is loaded into the ACC.
; Subroutine 'BCDLS'
Is the most significant digit (ACC) decimal (0 to 9)?
No
RET
Yes
EA — EA (previous result) x 10 + one digit of MSD (ACC)
; Subroutine 'SBCDHX'
The count of the digit-number counter at the first address of the result store area is unloaded from the stack.
(Digit-Number Counter) ← (Digit-Number counter) —1
No
No Brorrow
End of conversion?
Yes Borrow
The conversion result (EA) is stored in the result area (HL, HL + 1).
RETS



- **Program List**

```
;  ++++++++++++++++++++++++++++++++++++++++++++
;  +                                          +
;  +     TRANSFER * HEX <- BCD     *          +
;  +             (HL,HL+1)  (HL,HL+1)         +
;  +                                          +
;  ++++++++++++++++++++++++++++++++++++++++++++
;
TBCDHX:
        LXI     EA,0     ; RESULT  0CLEAR
        MVI     A,2*2-1  ; DIGIT COUNTER SET
TBDHX1:
        PUSH    V
        PUSH    H
;
;  **  OUTPUT  1-DIGIT FROM MSD  **
;
        MVI     C,4/2-1
        CALL    BCDLS  ;(Page 4-1)
        ;
;
;  **  CHECK / BCD DISPLAY ?
;
        LTI     A,9+1
        RET             ; NOT BCD
;
;  *S*  SUBROUTINE / BCD -> HEX  *S*
;
        CALL    SBCDHX
;
;  *S*  SUBROUTINE  END  *S*
;
        POP     H
        POP     V
        DCR     A        ; SKIP  IF TRANSFER END
        GJMP    TBDHX1   ; NOT END
        ;
        STEAX   H        ; STORE RESULT
        RETS
```

- **Program List**

```
;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
;  T                                             S
;  T     TRANSFER  SUBROUTINE                    S
;  T                                             S
;  TSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTSTS
;
SBCDHX:
;
;  **  RESULT * 10  **
;
        DMOV    B,EA
        PUSH    V
        MVI     A,10
        MUL     C
        PUSH    EA
        ;
        MUL     B
        POP     B
        EADD    EA,B
        MOV     A,EAL
        MOV     B,A
;
;  **  RESULT *10 + 1-DIGIT  **
;
        POP     V
        DMOV    EA,B
        EADD    EA,A
        RET
```

4.3 Conversion from ASCII to Hexadecimal (HEX)

This section describes the conversion of the ASCII 2 code (30H to 39H, 41H to 46H) to a hexadecimal 2 code (0 to FFH) using the following example:

Example: The ASCII 2 code stored into the BC pair register is converted into a hexadecimal 2 code (0 to FFH) and the result is stored in area 1 indicated by the HL register.

The program list and the program are described below.

- **Program List**

```
; +++++++++++++++++++++++++++++++++++++++++++++
; +                                           +
; +      TRANSFER : HEX  <=  ASCII            +
; +                 (2CODE)  (2CODE)          +
; +                                           +
; +           INPUT : BC REG <= ASCII         +
; +           OUTPUT:  (HL)   <= HEX          +
; +                                           +
; +++++++++++++++++++++++++++++++++++++++++++++
;
GETHEX:
        MOV     A,B       ; ASCII UPPER-CODE LOAD   ①
        CALL    SGTHEX    ; GET ! HEX ICODE         ②
        RET               ; ERREGULAR ASCII !       ③
        ;
        RLD                                         ④
        MOV     A,C       ; ASCII LOWER-CODE LOAD   ⑤
        CALL    SGTHEX    ; GET ! HEX  2TH CODE     ⑥
        RET               ; ERREGULAR ASCII !       ⑦
        RLD                                         ⑧
        RETS              ; POSSIBLE


; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                           S
; S      SUBROUTINE / GET  HEX 1-CODE(ACC)    S
; S                                           S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
SGTHEX:
        SUINB   A,'0'     ; CHECK / ASCII > 30H      ⑨
        RET               ; ERROR , SO < 30H
        ;
        GTI     A,'9'-'0'                           ⑩
                          ; CHECK / 30H < ASCII < 39H
        RETS
        SUINB   A,'A'-'0'                           ⑪
                          ; CHECK / ASCII > 41H
        RET
        ADINC   A,0AH     ; CHECK / ASCII < 46H     ⑫
        RET               ;
        RETS
```

① The upper 1 code of ASCII is loded into the accumulator.

② The upper 1 code of ASCII is converted into a hexadecimal 1 code.

③ Because the ASCII 1 code is not a numerical code ('0' to 'F'), no conversion can be carried out.

④ The hexadecimal 1 code is stored in the lower 4 bits of (HL).

⑤ The ASCII lower 1 code is loaded into the accumulator.

⑥ The ASCII lower 1 code is converted into a hexadecimal 1 code.

⑦ Because the ASCII 1 code isnot a numerical code ('0' to 'F'), no conversion can be carried out.

⑧ The hexadecimal 1 code is left-shiftet by 4 bits and the hexadecimal 1 code obtained in ⑥ above is stored in the lower 4 bits of (HL).

⑨ 30H (0 in hexadecimal notation) is subtracted from the ASCII code to check whether the ASCII 1 code is 30H or more. If so, the operation is skipped. If not, no conversion is carried out because the data is not a numerical code ('0' to 'F') and the operation ends.

⑩ The ASCII code is checked to be in the range from 30H to 39H (o to 9 in hexadecimal notation). If so, the conversion ends. If not, the operation is skipped.

⑪ Whether the ASCII code is 41H (A in hexadecimal notation) or more is checked. If so, the operation is skipped. If not, no conversion is carried out because the data is not a numerical code ('0' to 'F') and the operation ends.

⑫ Whether the ASCII code is 46H (F in hexadecimal notation) or less is checked. If so, the ASCII is converted into A to F and the operation is skipped. If not, no conversion is carried out.

4.4 Conversion from Hexadecimal (HEX) to ASCII

This section describes the conversion of the hexadecimal 2 code (o to FF) to an ASCII 2 code (30H to 39H, 4H to 46H) using the following example:

Example: The hexadecimal 2 code (0 to FF) in area 1 indicated by the HL pair register is converted into an ASCII 2 code and the result is stored into the BC pair register.

The program list and the program are described below.

- **Program List**

```
; ++++++++++++++++++++++++++++++++++++++++++++
; +                                          +
; +     TRANSFER / ASCII <=  HEX             +
; +                                          +
; +        INPUT : (HL) <= HEX 2-CODE        +
; +        OUTPUT : BC REG <= ASCII 2-CODE   +
; +                                          +
; ++++++++++++++++++++++++++++++++++++++++++++
;
GETASC:
        MVI     A,0                          }
        RLD                ; HEX UPPER CODE  LOAD} ①
        CALL    SGTASC                       ②
        MOV     B,A        ; STORE RESULT    ③
        ;
        MVI     A,0                          }
        RLD                ; HEX LOWER CODE  LOAD} ④
        CALL    SGTASC                       ⑤
        MOV     C,A        ; STORE RESULT    ⑥
        RET


; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
; S                                          S
; S     GET /  ASCII 1-CODE ( BC REG )       S
; S                                          S
; SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
;
SGTASC:
        LTI     A,0AH      ; CHECK / HEX > 9 ⑦
        ADI     A,41H-3AH  ; BIAS (+7)       ⑧
        ADI     A,30H      ; BIAS (+30H)     ⑨
        RET
```

① The hexadecimal code memory is shifted by 4 bits and the hexadecimal upper code is loaded into the accumulator.

② The hexadecimal upper 1 code is converted into the ASCII 1 code.

③ The ASCII upper 1 code is loaded into the B register.

④ The hexadecimal code memory is shifted by 4 bits and the hexadecimal lower code is loaded into the accumulator.

⑤ The hexadecimal lower 1 code is converted into the ASCII 1 code.

⑥ The ASCII lower 1 code is loaded into the C register.

⑦ The hexadecimal 1 code is checked to be 9 or less. If so, the operation is skipped.

⑧ The hexadecimal code will be converted into 30 to 3F if a 30H is added in ⑨ below. Because the ASCII codes for A to F will consequently become 41 to 46, 7 is added before the addition of 30H.

⑨ Bias 30H is added to HEX code.

Chapter 5 Comparison

This chapter describes the following processing operations:

- Comparing the size of two numerical data items and branching them according to the comparison result
- Data retrieval
- Setting/resetting and testing the software flags of the memory in bit units

5.1 16-Bit (2 Byte) Data Comparison

This section describes the following three types of comparison operations:

EQUAL ; =
GREATER THAN ; >
LESS THAN ; <

The following 16-bit operation instructions are for the μCOM-87AD to carry out the above three types of operations. Data from the extended accumulator and the pair register can be compared. If the necessary conditions are satisfied, the 16-bit operation instructions will skip the following instructions:

DGT	EA.H	①
DLT	EA.H	②
DNE	EA.H	③
DEQ	EA.H	④

① EA is compared with the HL pair register. When EA > HL, the next instruction is skipped.

② EA is compared with the HL pair register. When EA < HL, the DLT instruction is skipped.

③ EA is compared with the HL pair register. When EA = HL, the DNE instruction is skipped.

④ EA is compared with the HL pair register. When EA = HL, the DEQ instruction is skipped.

5.2 Data Retrieval

This section describes the procedure for detecting data in a memory area from the preset table as shown in the figure.

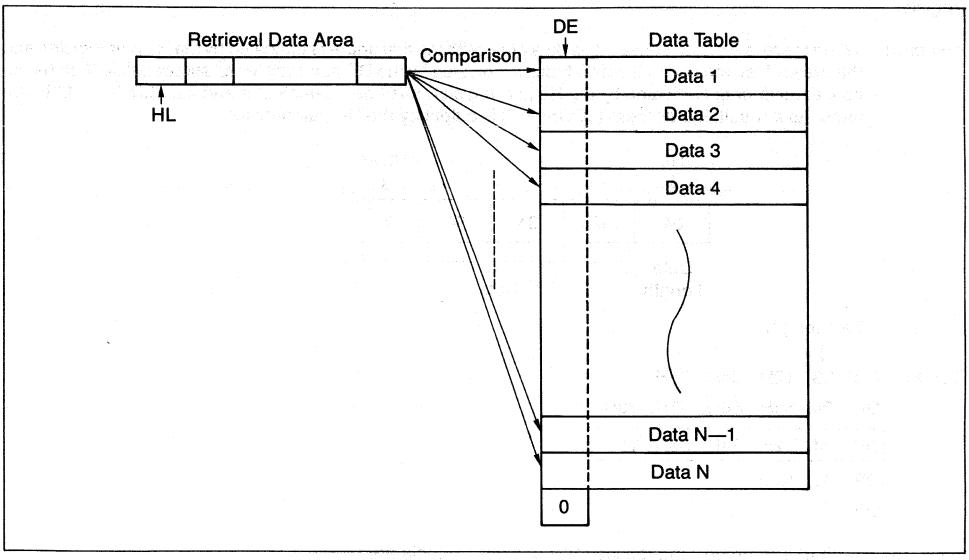


Figure 5-1 Data Retrieval Operation

The data table and retrieval data area have the following formats:

The data length may ormay not be fixed between retrieval data and data table. In this section each data length is unspecified.

Thus, the data format is set as follows to check in order, the data length, and the data contents for data retrieval.

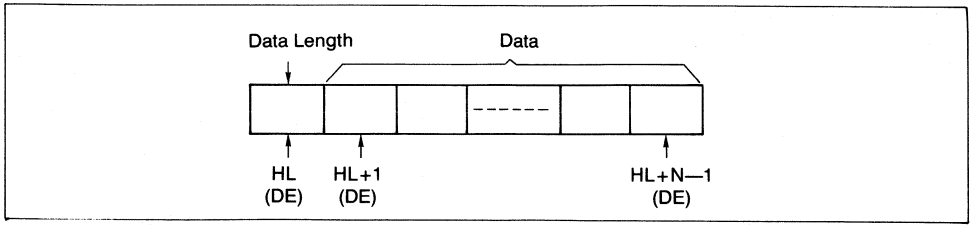


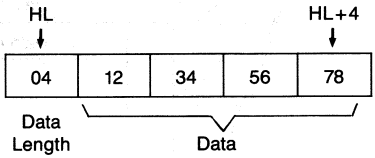
Figure 5-2 Retrieval Data Format

The data table terminates when 0 is set as data length at the first address of the data area as shown in Figure 5-2.

For data retrieval, the first addresses of the retrieval data area and the table must be preset by the HL and DE pair registers respectively.

When data is detected from the table, the first address of the corresponding data area is output by the DE pair register.

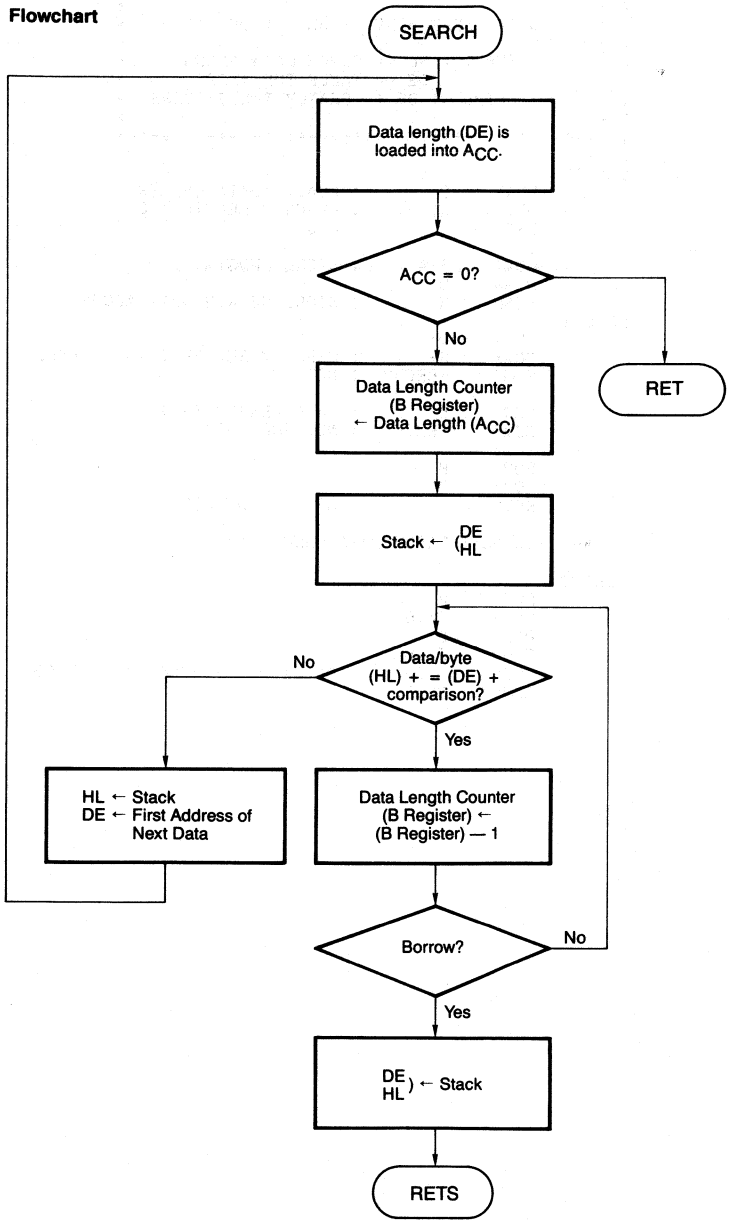
Example: When there is a 5-byte consecutive data area, with the first address indicated by the HL pair register, and the preset first address of the data table is indicated by the DE pair register as shown below then 6-byte consecutive data (indicated by the HL pair register) has been detected at address (LOOK + 9) in the table. As a result, the address (LOOK + 9) is output by the DE pair register.




```
          Data length
               ↓
LOOK:  DB   03, 12H, 34H, 78H
       DB   04, 55H, 66H, 77H, 88H
       DB   04, 12H, 34H, 56H, 78H
       DB   01, ABH
       DB
```

Next, the flowchart and processing routine related to this example are shown:

Flowchart


SEARCH
Data length (DE) is loaded into ACC.
ACC = 0?
No
RET
Data Length Counter (B Register) ← Data Length (ACC)
Stack ← (DE / HL)
No
Data/byte (HL) + = (DE) + comparison?
Yes
HL ← Stack
DE ← First Address of Next Data
Data Length Counter (B Register) ← (B Register) — 1
Borrow?
No
Yes
(DE / HL) ← Stack
RETS

- **Program List**

```
; ++++++++++++++++++++++++++++++++++++++++++++
; +                                          +
; +     DATA  SEARCHING  ( HL ) ( DE )       +
; +                                          +
; +       INPUT : HL <= SERCH DATA ADDRS     +
; +               DE <= TABLE TOP ADDRS      +
; +       OUTPUT : DE <= SEARCH TABLE ADDRS  +
; +                                          +
; ++++++++++++++++++++++++++++++++++++++++++++
;
SEARCH:
        LDAX    D           ; LOAD ! DATA-LENGTH
        GTI     A,1-1       ; CHECK / LENGTH = 0
        RET                 ; = 0
        ;
        MOV     B,A         ; STORE LENGTH
        PUSH    D
        PUSH    H           ; STORE SEARCH DATA ADDRS
SERCH1:
        LDAX    D+
        EQAX    H+          ; CHECK / SEARCH DATA = TABLE
        GJMP    SERCH2      ; NOT =
        ;
        DCR     B           ; DATA-LENGTH (-1)
        GJMP    SERCH1      ; NOT END-DATA
        ;
        POP     H
        POP     D
        RETS                ; SEARCH SUCCESS !
;
;  **  FOR NEXT-TABLE DATA CHECK  **
;
SERCH2:
        POP     H
        DMOV    EA,D
        POP     D
        EADD    EA,B
        DMOV    D,EA        ; DE <= NEXT-TABLE DATA ADDRS
        GJMP    SEARCH
```

5.3 Memory Bit Test and SET/RESET

a) Memory bit test

The memory and immediate data are compared (AND) and a bit test is carried out with the Z flag.

Thus, there are two instructions available: the OFFIW instruction, which skips the next instruction when Z flag = "1", and the ONIW instruction, which skips the next instruction when Z flag = "0".

The bits of ONIW and OFFIW immediate data which correspond to the memory bits to be checked are set at 1 and 0 respectively.

- ONIW ADRS, 00000011B (1—1)
  Next instruction
  Instruction for which the skip conditions are satisfied

  ⋮

  In (1—1), when memory bits 0, 1 = "1" (AND result ≠ 0), the next instruction is skipped.

  ⋮

- OFFIW ADRS, 11000000B (1—2)
  Next instruction
  Instruction for which the skip conditions are satisfied

  ⋮

  In (1—2), when memory bits 6, 7 = "0" (AND result = 0), the next instruction is skipped.

b) Memory bit test (when one bit is tested)

When one is tested, the BIT instruction particular to the μCOM-87AD can be used.

- BIT 1,ADRS (1—3)
  Next instruction
  Instruction for which the skip conditions are satisfied

  In (1—3), when memory bit 1 is set (1), the next instruction is skipped.?

c) Memory bit set/reset

The memory is ORed (ORIW) or ANDed (ANIW) with the immediate data and the result is set/reset in bit units.

- ORIW ADRS, 00000011B (1—4)
- ANIW ADRS, 00011111B (1—5)

  In (1—4), ADRS is ORed with 00000011B and ADRS bits 0 and 1 are set.

  In (1—5), ADRS is ANDed with 00011111B and ADRS bits 7, 6 and 5 are reset.

Chapter 6 Conditional Branching Operations

This chapter describes examples of multi-branching operations using a jump instruction for setting the jump destination address with a pair register or an extended accumultor.

a) JEA instruction

The jump destination address is determined by the extended accumulator.

The following is an example of multi-branching based on the vertical accumulation effects of the HL pair register.

```
JHL1:   LXI   H,ADRS1
JHL2:   LXI   H,ADRS2                          ] Vertical Accumulation Effects
        ;
        ;
        ;
JHL10;  LXI   H,ADRS10
        DMOV  EA,H          ; ES <— HL         ; EA ← HL
        JEA                                    ; Branching by EA
```

b) JB instruction

The jump destination address is determined by the BC pair register.

The following is an example of multi-branching by setting an address into the BC pair register by the TABLE instruction.

```
SLL     A                  ①
TABLE                      ②
JB                         ③
DW      CG00               ④
DW      CG01
DW      CG02
DW      CG03
;
;
;
DW      CG7F
```

① The accumulator data is left-shifted and is doubled so that the address which is loaded into the BC pair register when the TABLE instruction is executed will not overlap.

② The accumulator data is added to the current program counter data and 2 is added to the sum. The total address data and the total address data plus 1 are set into the C and B register respectively.

$$\begin{pmatrix} ADRS \leftarrow PC + A + 2 \\ C \leftarrow (ADRS) \\ B \leftarrow (ADRS + 1) \end{pmatrix}$$

③ The operation jumps to the address of the BC register data set by the TABLE instruction.

Chapter 7 Table Reference Operations

This chapter describes an interpolation method as an application example of table reference.

If the fx function and fx value (f_0, f_{10}, f_{20}, . . ., f_{260}) at 260 points X (X = 0, 10, 20, . . ., 260) spaced at equal intervals between [0] and [260] are given, an fx value at point N between [0] and [255] is obtained using the interpolation method.

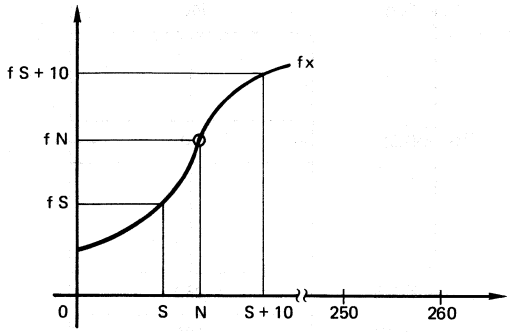


Figure 7-1. fx Function

Interpolation procedure:

① S = INT (N/10)*10 is calculated and the location of N in each region divided by 26 equal-interval points is determined by [S, S + 10].

② The f_N value is calculated using the following equation with S obtained in ①.

$$f_N = f_S + \frac{(f_{S+10} - f_S) * (N - S)}{10}$$

Next, an interpolation program example is described.

Example: Interpolation program

In this example, point N is set by an accumulator and an interpolation is carried out to obtain $f_{(N)}$ using f_0, f_{10}, f_{20}, . . ., f_{260} (in 2-byte units) in the memory table. As shown in figure 1-8-2, the first address of the table is preset by the HL pair register and S is obtained using the previously mentioned procedure ①.

Because each data unit (f_0, f_{10}, f_{20}, . . ., f_{260}) in the table is in 2-byte units, S is doubled and added to the first address resulting with the data address at which f_{50} is stored being indetermined.

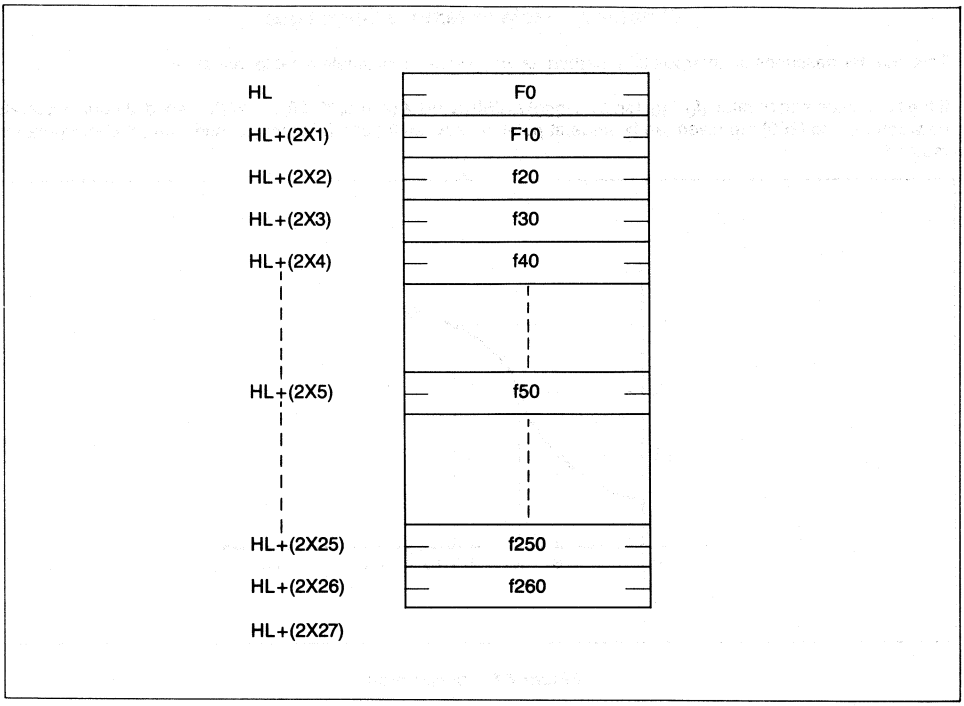
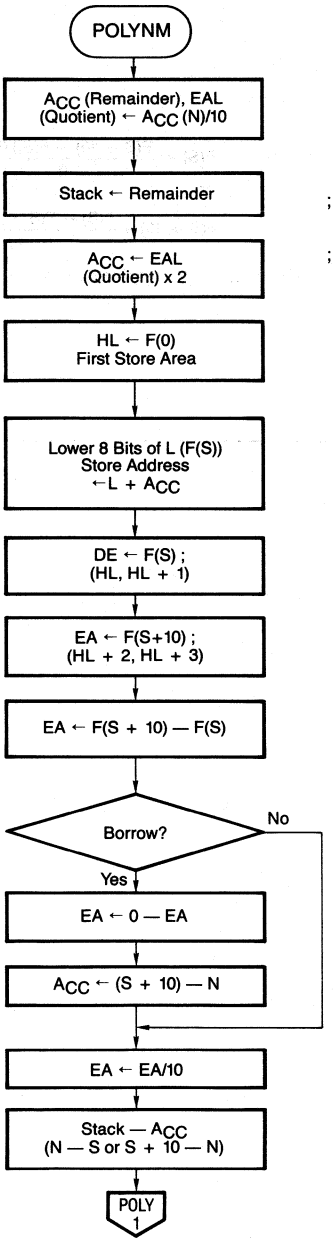
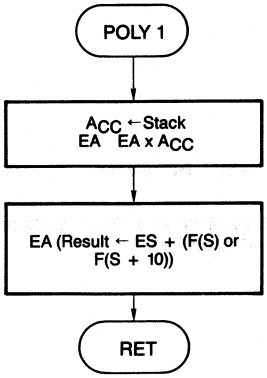


Figure 7-2. Table Data RAM Map

Flowchart


POLYNM
ACC (Remainder), EAL (Quotient) ← ACC (N)/10
Stack ← Remainder
; (N—S) ← Remainder
ACC ← EAL (Quotient) x 2
; Lower Address of F(S) First Store Area ← Quotient x 2
HL ← F(0) First Store Area
Lower 8 Bits of L (F(S)) Store Address ←L + ACC
DE ← F(S) ; (HL, HL + 1)
EA ← F(S+10) ; (HL + 2, HL + 3)
EA ← F(S + 10) — F(S)
Borrow?
No
Yes
EA ← 0 — EA
ACC ← (S + 10) — N
; S + 10 — N ← N — S
EA ← EA/10
; | F(S + 10) — F(S) | /10
Stack — ACC (N — S or S + 10 — N)
POLY 1



; $F(S) + \frac{(F(S + 10) - F(S))\,(N - S)}{10}$

or

$F(S + 10) + \frac{F(S) - F(S + 10)\,(S + 10 - N)}{10}$

- **Program List**

```
; ++++++++++++++++++++++++++++++++++++++++++++++++
; +                                              +
; +          INTERPOLATION  POLYNOMIAL           +
; +                                              +
; +        INPUT      ACC <- X                   +
; +                   HL <= DATA TABLE TOP ADDRS  +
; +        OUTPUT     EA - REG                   +
; +                                              +
; +  F(N)= F(S) + ( F(S+10)-F(S) ) * (N-S) /10   +
; +                                              +
; ++++++++++++++++++++++++++++++++++++++++++++++++
;
POLYNM:
        LXI     EA,0
        MOV     EAL,A
        MVI     A,10
        DIV     A
        PUSH    V           ; STACK <- N-S
        ;
        MOV     A,EAL
        SLL     A           ; ACC <- 2BYTE DATA TOP ADDRS
        ;
        DMOV    EA,H
        EADD    EA,A
        DMOV    H,EA
                            ; HL <-  F(S) STORE ADDRS
        LDEAX   H
        DMOV    D,EA        ; DE <-  F(S)
        LDEAX   H+2         ; EA <-  F(S+10)
                            ; HL <- F(S) STORE ADDRS
        POP     V
        DSUB    EA,D        ; F(S+10) - F(S)
        SK      CY
        GJMP    POLYN1      ; IF F(S+10) > F(S)
;
        INX     H
        INX     H
        SUI     A,10
        NEGA                ; 10- ((S+10)-N)
        ;
        DMOV    D,EA
        LXI     EA,0
        DSUB    EA,D        ; F(S) - F(S+10)
POLYN1:
        PUSH    V
        MVI     A,10
        DIV     A           ; ( F(S+10) - F(S) ) / 10
        POP     V
        DMOV    B,EA
;
;    ( F(S+10)-F(S) ) * (N-S) / 10
;
        MUL     C
        PUSH    EA
        ;
        MUL     B
        MOV     A,EAL
        POP     B
        ADD     B,A
;
        LDEAX   H
        DADD    EA,B
        RET
```

PART II
Floating Point Arithmetic Operations

PREFACE

The μCOM87 series are NEC's original 8-bit, one-chip microcomputers having a 16-bit ALU. This application note describes the floating point operation programs, so that the many powerful instructions of the μCOM87 series may be fully utilized.

Also read Application Note (I) for details of the software including the fixed point four rules, and Application Note (III) for details of the internal hardware operation functions.

Note: This application note is applicable to the following types of microcomputers:

μPD7811/7810/78PG11, μPD7811H/7810H/78PG11H, μPD78C14/78C11/78C10/78CG14, μPD7809/7808/7807/78P09

Chapter 1 Floating Point Operation Subroutines

1.1 Binary Floating Point Number Format

In this operation program, the binary floating point number is represented as 8 bits of the exponential part and 24 bits of the fixed point part. The decimal point is positioned to the left of the MSB of the fixed point part.

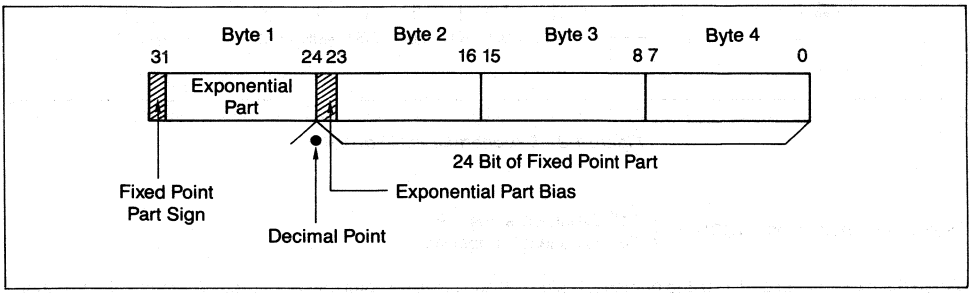


Figure 1-1 Binary Floating Point Number Format

Numerical value = (Sign) | (fixed point part) | x 2 (exponential Part)

Sign (23 bits) = { 0; Positive
1; Negative

The details of each part are described below.

1.1.1 Fixed Point Part

The fixed point part is represented as a 24-bit positive binary number.

Because the decimal floating point binary number is normalized and the MSB 623 bits) of the fixed point part is always set to 1, the sign information of the fixed point part may be included in the MSB.

Normalization is carried out to adjust the exponential part so that the most significant bit (MSB) (23 bits) of the fixed point part is set to 1. A normalization example is described below.

Example:

Before Normalization		After Normalization	
1234.567890	→	0.123456789	(4 was added to the exponential part.)
0.00012345	→	0.123456000	(3 was subtracted from the exponential part.)

1.1.2 Exponential Part

The exponential part is represented as an 8-bit binary number (two's complement for a negative binary number) plus a bias of 80H.

The real exponential value and the exponential part format have the following relationship:

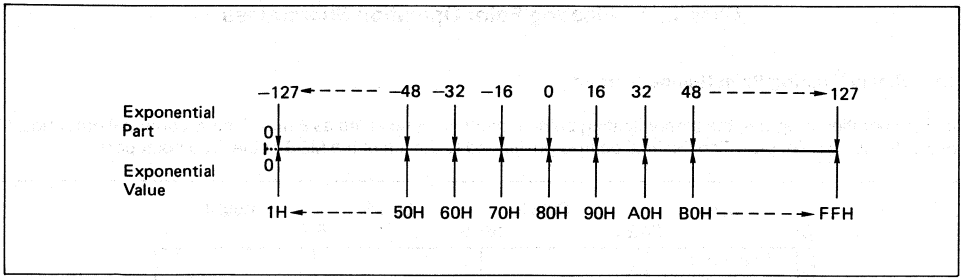


Figure 1-2 Exponential Part Format

MSB of exponential part (bit31) = { "1" Exponent is positive. / "0" Exponent is negative.

When the exponential part = 0, the numerical value is 0 and the fixed point part is ignored. Therefore, the exponential part is set to 0 to represent numerical value 0.

1.2 Variables

This section describes the most important variables within this operation program. Refer to the description of each subroutine for details of all other variables.

1.2.1 Binary Floating Point Accumulator (B.F.P.ACC)

All operations are carried out by the B.F.P.ACC.

The B.F.P.ACC consists of consecutive 5-byte memories (see figure 1-3).

The following are the details of each part.

(1) Exponential part (ACCE)

Refer to subsection 1.1.2

(2) Sign part

This part indicates whether the fixed point part is positive or negative.

ACCS = 80H . . . Positive
00H . . . Negative

(3) Ficed point parts (ACC1, ACC2, ACC3)

ACC1 to ACC3 are respresented as a 3-byte absolute value. Thus, the MSB of ACC1 is always 1 (normalization).

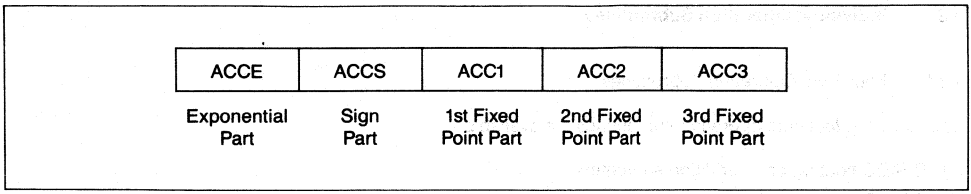


Figure 1-3 B.F.P.ACC Memory Configuration

1.2.2 Operand (OPE.)

The OPE. plays an important role for operations along with the B.F.P.ACC. It is mainly used for operations between two variables.

The OPE. consists of consecutive 4-byte memories (WSE, WS1, WS2, WS3). (See figure 1-4.)

Refer to sections 1.1 and 1.2 for details concerning each part.

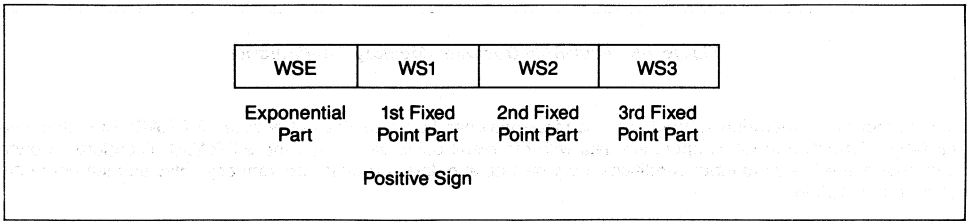


Figure 1-4 Operand Memory Configuration

1.2.3 Addition/Subtraction Flag (SF)

For addition/subtraction, the SF between B.F.P.ACC and OPE. ist set.

SF = { 0 ; Addition (because two signs are the same)
80H; Subtraction (because two signs are different)

1.2.3 Sign Flag (FSN)

After checking the B.F.P.ACC sign, the sign flag is set.

FSN = { 0 ; B.F.P.ACC is 0.
1 ; B.F.P.ACC is positive.
0FFH; B.F.P.ACC is negative.

1.3 Individual Operation Subroutines

1.3.1 Four-Rule Operation Subroutines

The following four-rule operation subroutines are available:

(1) BFADD floating point addition subroutine
(2) BFSUB floating point subtract subroutine
(3) BFMUL floating point multiply subroutine
(4) BFDIV floating point division subroutine

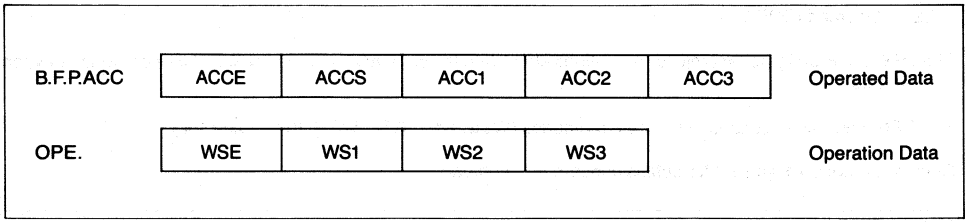


Figure 1-5 Four-Rule Operation Memory Configuration

In these four-rule operation subroutines, each four-rule operation is carried out between B.F.P.ACC operated data (in 5-byte format) and OPE. operation data with the result being stored into the B.F.P.ACC. Therefore, in each subroutine, the following input conditions are provided, and data is transferred from any selected memory to the OPE. for operations.

Input conditions:

The first 4-byte memory address where operation data is stored is indicated by the HL pair register and the operation is stored in the B.F.P.ACC.

Output conditions:

RET: The operation result overflows. The CY flag is set to "1".

RETS: The normal result is stored in the B.F.P.ACC. The CY flag is set to "0". If the result is 0, the Z flag is set to "1". If the result is not 0, the Z flag is set to "0" respectively.

If it is previously known that the operation result will not overflow, the NOP instruction is necessary after the four-rule operation subroutines are called.

1.3.1.1 BFADD (Addition) Subroutine

(1) Processing operation

The 4-byte consecutive binary floating point OPE. data is added to the B.F.P.ACC data and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the four-rule operation input conditions.

(3) Output conditons

Refer to the previous description of the four-rule operation output conditions.

(4) Subroutines used

Refer to Table 4-1.

(5) Stack depth

2 max.

(6) Coding order

```
        ≀
LXI H, ——      ; First address of OPE. 4-byte data
CALL BFADD     ; Addition
GJMP ERROR     ; Overflow processing
        ≀
```

(7) Processing time

Approx. 1.6 ms max.
(oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC	9BH, 80H, FFH, FFH, FFH
OPE.	82H, 7FH, FFH, FFH

(8) Processing procedure

If the B.F.P.ACC and the OPE. are added, the four patterns as shown in Table 1-1 are available according to the signs of the B.F.P.ACC (X) and the OPE. (Y).

Table 1-1 Operation Patterns Accoding to X and Y Signs

Pattern	X's Sign	Y's Sign	Sign After Operation	Operation Flag SF	Operation (Absolute Value)
1	Positive	Positive	Positive	0	X + Y
2	Positive	Negative	—	80H	X — Y
3	Negative	Positive	—	80H	Y — X
4	Negative	Negative	Negative	0	X + Y

SF: In Table 1-1, the SF is set to X's sign ~~∀~~ $\overline{\text{Y's sign}}$ (for the 7th bit only).

Thus, SF = { 80H: Subtraction
0 : Addition

Processing procedure:

(a) The B.F.P.ACC is 0-checked. If 0, the operation ends.

(b) The OPE. is 0-checked. If 0, the operation ends.

(c) The larger sign of the exponential part is set as the post-operation sign.

(d) The larger fixed point part of the exponential part is set into ACC1 to ACC3. The smaller fixed point part is set into WS1 to WS3.

(e) The addition and subtraction for the fixed point part are branched by the SF flag for each operation. In this case, addition and subtraction operations are not carried out for between the fixed point parts. Instead, it is necessary to carry out the digit alignment of the smaller exponential part to the larger exponential part as shown in example (1). In the digit alignment, the smaller exponential part is right-shifted by the exponential difference (remaining in WS1 to WS4).

Example (1) Digit alignment of exponential part

$0.123400000X10^{4}+0.567800000X10^{-2}$

→ $0.123400000X10^{4}+0.000000567X10^{4}$

→ $(0.123400000+0.000000567)X10^{4}$

If the right shift causes the exponential part to overflow, the overflow digit is regarded as having been omitted. If the exponential difference is 25 or more, the smaller value is ignored as being negligible with respect to the larger value and the operation ends.

(f) In the case of pattern 1 or 4 in Table 3-1 concerning the addition/subtraction of the exponential parts after digit alignment, when an overflow occurs as in example (2), the exponential part is right-shifted by 1 bit, and 1 is set at the most significant bit and 1 is added to the exponent.

Example (2)

$0.999900000X10^{1}+0.001234000X10^{1}$

→ $1.001134000X10^{1}$

→ $0.100113400X10^{2}$

In the case of pattern 2 or 3, the absolute value of the exponential part is subtracted. If a borrow occurs, it is represented as a complement and its sign is inverted.

(g) The WS4 most significant bit is half-adjusted and the result is normalized.

The flowchart is shown below:

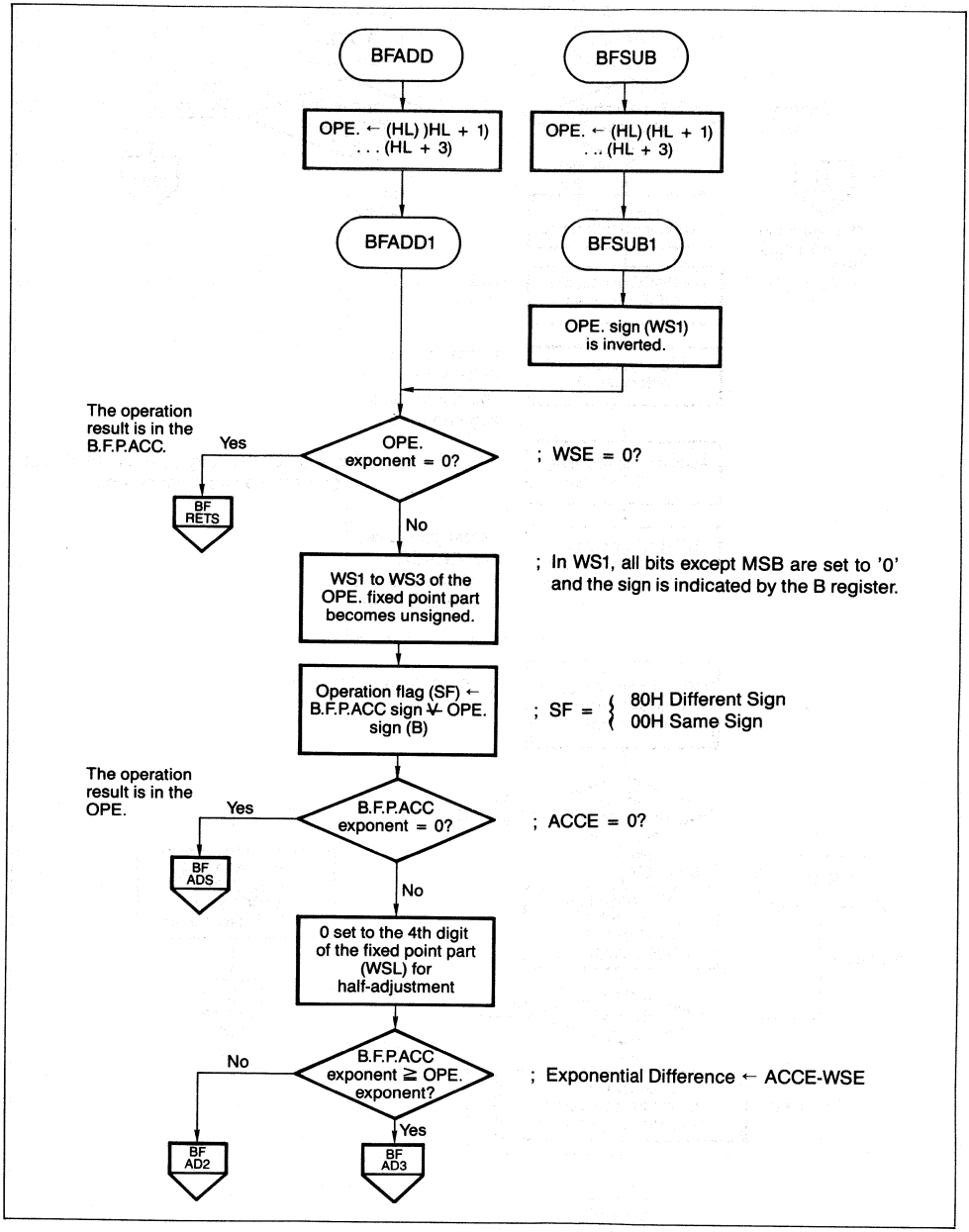
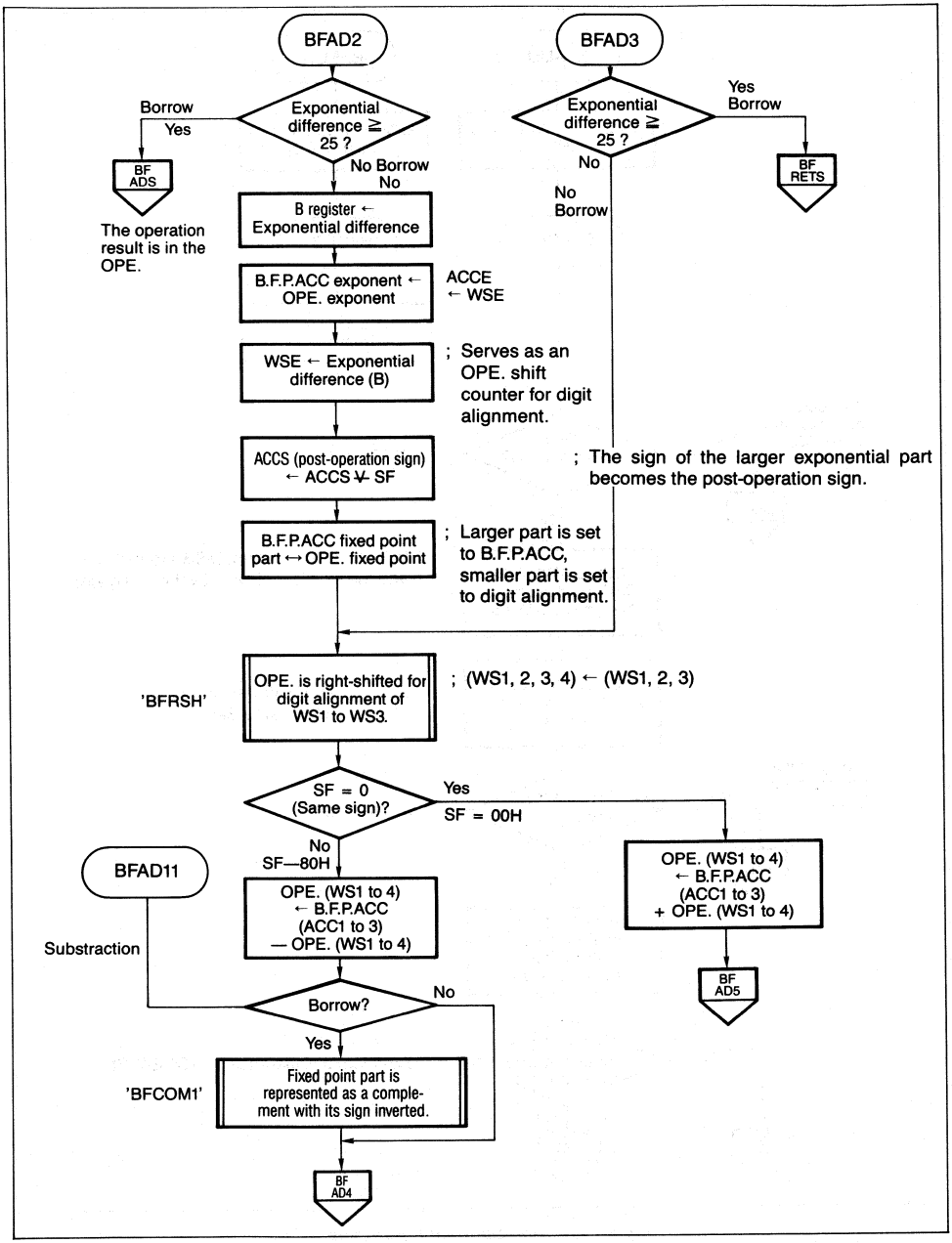
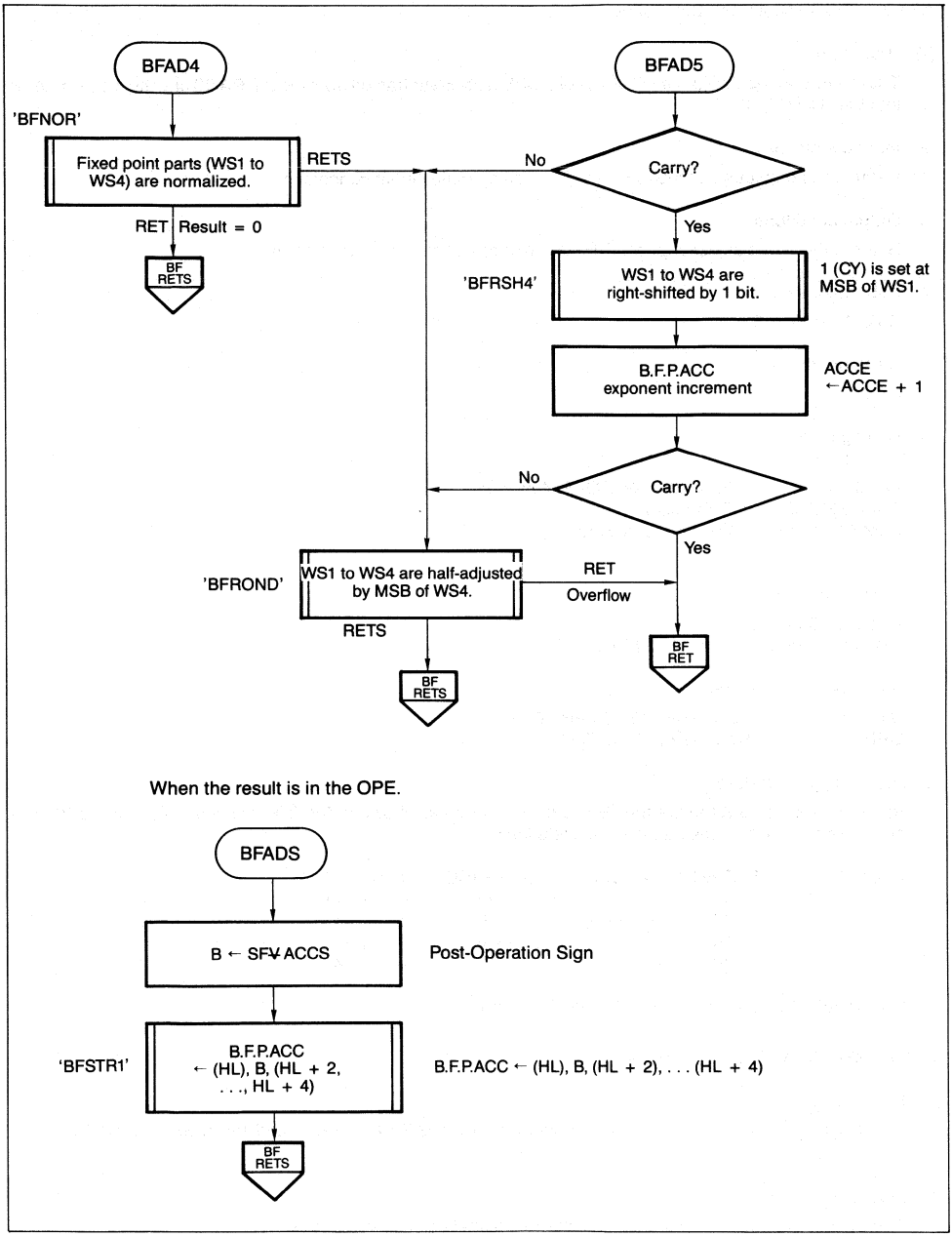


Figure 1-6 Add/Subtract Subroutine


BFAD2
BFAD3
Exponential difference ≧ 25 ?
Borrow
Yes
No Borrow
No
BF ADS
The operation result is in the OPE.
B register ← Exponential difference
B.F.P.ACC exponent ← OPE. exponent
ACCE ← WSE
WSE ← Exponential difference (B)
; Serves as an OPE. shift counter for digit alignment.
ACCS (post-operation sign) ← ACCS ⊻ SF
; The sign of the larger exponential part becomes the post-operation sign.
B.F.P.ACC fixed point part ↔ OPE. fixed point
; Larger part is set to B.F.P.ACC, smaller part is set to digit alignment.
'BFRSH'
OPE. is right-shifted for digit alignment of WS1 to WS3.
; (WS1, 2, 3, 4) ← (WS1, 2, 3)
SF = 0 (Same sign)?
SF = 00H
SF—80H
BFAD11
Substraction
OPE. (WS1 to 4) ← B.F.P.ACC (ACC1 to 3) — OPE. (WS1 to 4)
OPE. (WS1 to 4) ← B.F.P.ACC (ACC1 to 3) + OPE. (WS1 to 4)
Borrow?
BF AD5
'BFCOM1'
Fixed point part is represented as a complement with its sign inverted.
BF AD4
BF RETS


BFAD4
'BFNOR'
Fixed point parts (WS1 to WS4) are normalized.
RETS
RET
Result = 0
BF RETS
BFAD5
No
Carry?
Yes
'BFRSH4'
WS1 to WS4 are right-shifted by 1 bit.
1 (CY) is set at MSB of WS1.
B.F.P.ACC exponent increment
ACCE ← ACCE + 1
No
Carry?
Yes
'BFROND'
WS1 to WS4 are half-adjusted by MSB of WS4.
RET
Overflow
RETS
BF RET
BF RETS
When the result is in the OPE.
BFADS
B ← SFV ACCS
Post-Operation Sign
'BFSTR1'
B.F.P.ACC ← (HL), B, (HL + 2, . . ., HL + 4)
B.F.P.ACC ← (HL), B, (HL + 2), . . . (HL + 4)
BF RETS

1.3.1.2 BFSUB (Subtract) Subroutine

(1) Processig

The 4-byte consecutive binary floating point OPE. data is subtracted from the B.F.P.ACC and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the four-rule operation input conditions.

(3) Output conditons

Refer to the previous description of the four-rule operation output conditions.

(4) Subroutines used

Refer to Table 1-3

(5) Stack depth

2 max.

(6) Coding order

```
      ≀
LXI H, ——      ; First address of OPE. 4-byte data
CALL BFSUB     ; Subtraction
GJMP ERROR     ; Overflow processing
      ≀
```

(7) Processing time

Approx. 1.6 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC	9BH, 80H, FFH, FFH, FFH
OPE.	82H, 7FH, FFH, FFH

(8) Processing procedure

In the subtraction subroutine the OPE. sign is inverted as shown in the following example and the BFADD subroutine is used for the subsequent operations.

Example:

B.F.P.ACC	—	OPE.	→	B.F.P.ACC	+	(—OPE.)
10	—	4	→	10	+	(—4)
10	—	(—4)	→	10	+	4
—10	—	4	→	—10	+	(—4)
—10	—	(—4)	→	—10	+	4

For the BFSUB flowchart, refer to the BFADD flowchart.

1.3.1.3 BFMUL (Multiply) Subroutine

(1) Processing

The 4-byte consecutive OPE. data is multiplied by the B.F.P.ACC value and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the four-rule operation input conditions.

(3) Output conditons

Refer to the previous description of the four-rule operation output conditions.

(4) Subroutines used

Refer to Table 1-4.

(5) Stack depth

4 max.

(6) Coding order

```
      ≀
LXI H, ——      ; First address of OPE. 4-byte data
CALL BFMUL     ; Multiplication
GJMP ERROR     ; Overflow processing
      ≀
```

(7) Processing time

Approx. 2.5 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC	9BH, 80H, FFH, FFH, FFH
OPE.	82H, 7FH, FFH, FFH

(8) Processing procedure

(a) The OPE. (multiplicand) is 0-checked. If 0, the result also becomes 0.

(b) Addition is carried out for the exponential part. The OPE. is converted into an unsigned fixed point.

(c) Multiplication of fixed point part (3 bytes x 3 bytes).

The fixed point part multiply subroutine (BFMX) has a memory configuration as shown in figure 1-7.

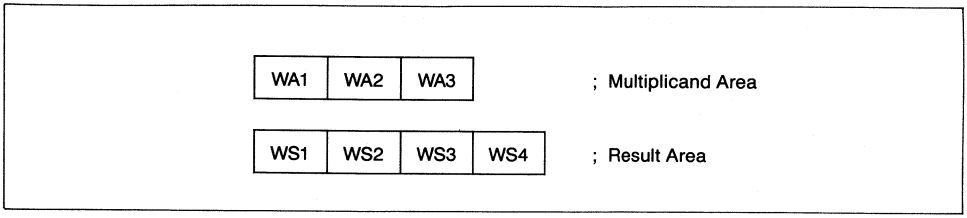


Figure 1-7 Memory Configuration for Fixed Point Part Multiply Subroutine

The following procedure is used to carry out the multiplication of 1 byte of the multiplier by 3 bytes of the multiplicand 3 times:

(i) First, multiplicands (WS1, WS2, WS3) are saved into (WA1, WA2, WA3) and the result areas (for WS1, WS2, WS3) are 0-cleared.

(ii) "(WA1, WA2, WA3) x 1 byte of multiplier (ACC3) + previous result (WS1, WS2, WS2)" is calculated and the result is stored into the B register (WS1, WS2, WS3).

(iii) The results of (ii) (B register, WS1, WS2, WS3) are each right-shifted by 1 byte and are stored into (WS1, WS2, WS3, WS4).

After that, (ii) and (iii) operations are carried out using ACC2 and ACC1 as 1 byte of the multiplier. The results are stored into (WS1, WS2, WS3, WS4).

(d) The half-adjustment and normalization are carried out for (WS1, WS2, WS3, WS4).

The flowcharts are shown in figures 1-8 through 1-10.

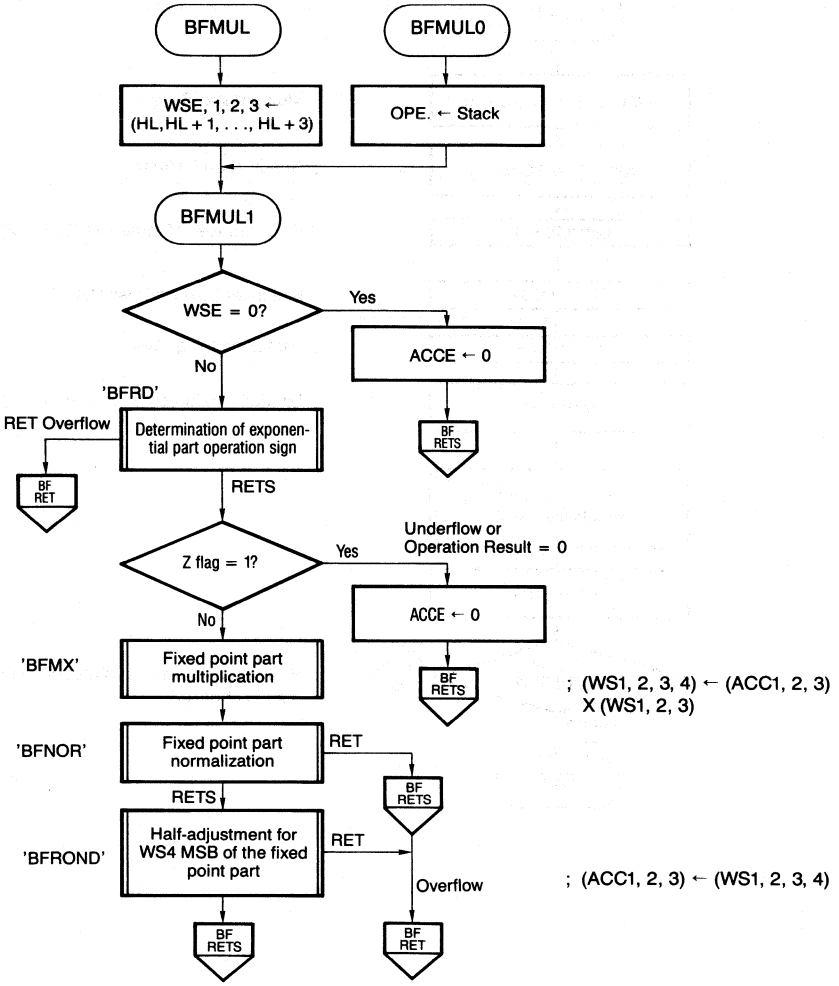


Figure 1-8 Multiply Subroutine

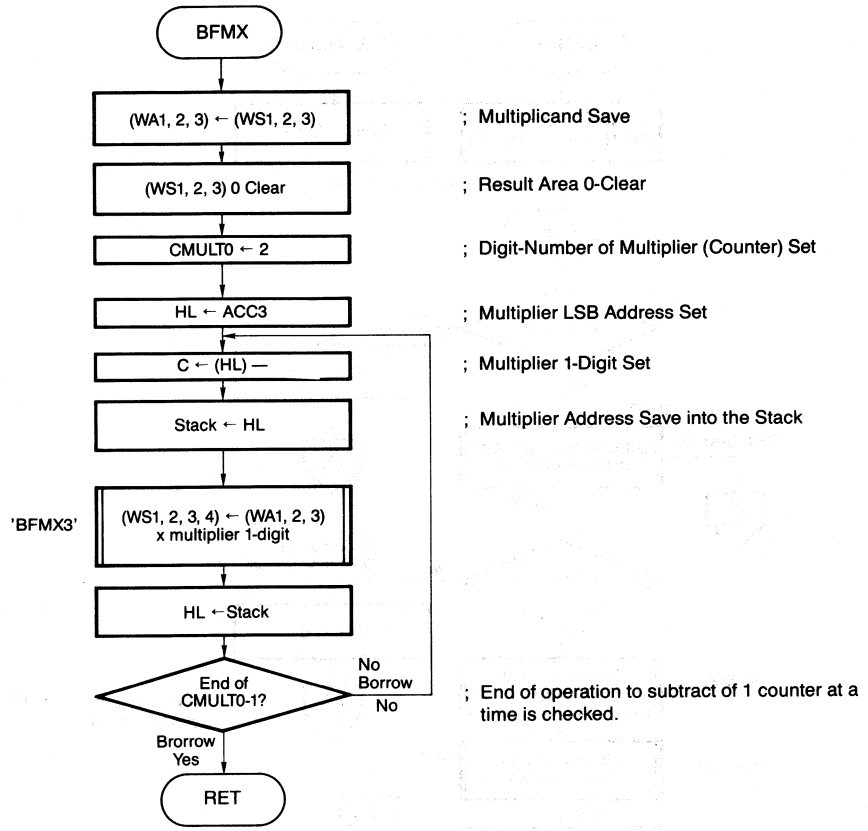
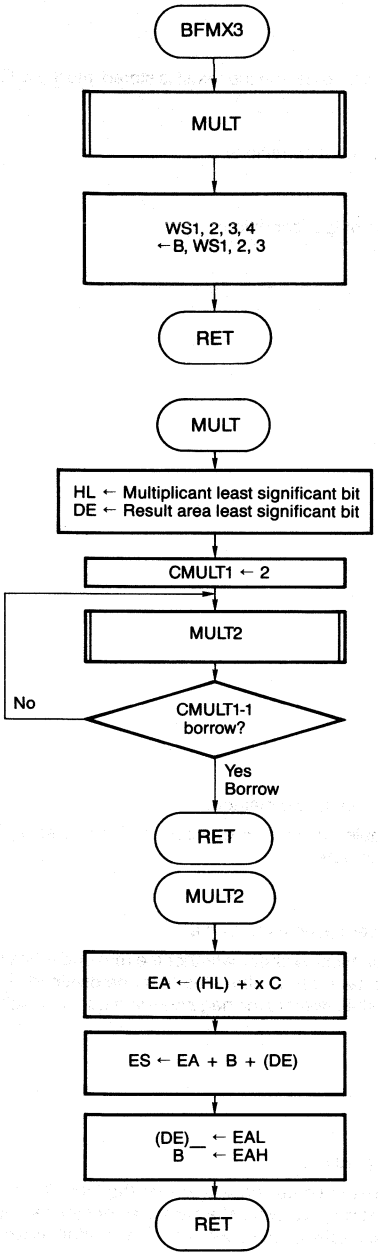


Figure 1-9 Fixed Point Part Subroutine



; B, WS1, WS2, WS3 ← (WA1, 2, 3) x Multiplier 1 Byte

'BFRSH0'

; Multiplicand Digit-Number (Counter) Set

; (WA1, 2, 3) x Multiplier 1 Byte + Previous Result

; End of (WA1, 2, 3) x Multiplier 1 Byte?

; B ; Carry from the previous digit
; (DE) ; Result up to the previous operation

Figure 1-10 Fixed Point Part (WA1, 2, 3) x Multiplier 1 Byte + Previous Result

1.3.1.4 BFDIV (Division) Subroutine

(1) Processing

The B.F.P.ACC value is divided by the 4-byte consecutive OPE. data and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the four-rule operation input conditions.

(3) Output conditons

Refer to the previous description of the four-rule operation output conditions.

(4) Subroutines used

Refer to Table 1-5.

(5) Stack depth

2 max.

(6) Coding order

```
      ≀
LXI H, ——      ; First address of OPE. 4-byte data
CALL BFDIV     ; Division
GJMP ERROR     ; Overflow processing
      ≀
```

(7) Processing time

Approx. 3.5 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC	9BH, 80H, FFH, FFH, FFH
OPE.	82H, 7FH, FFH, FFH

(8) Processing procedure

In the BFDIV subroutine, division is carried out using an irreparable method.

A reparable method and irreparable method are available for division operations. For reference, the two methods are described using a decimal number as an example.

(Reparable method)

The dividend is subtracted by the divisor until its remainder becomes negative.

Each time subtraction is carried out, one is added to the quotient to check whether the remainder has become negative. If the remainder is negative, too many divisors have been subtracted from the dividend. Thus, the previous remainder and quotient are recovered by adding the divisor to the negative remainder and subtracting 1 from the quotient.

This operation is carried out for each digit.

Example (1) shows the reparable method.

(Irreparable method)

Unlike the reparablemethod, no recovery operation is carried out.

The dividend is subtracted by the divisor until its remainder becomes negative. After that, the divisor is right-shifted by 1 digit and is added until the remainder becomes positive. Next, the divisor is right-shiftet and is subtracted until the remainder becomes negative. In the irreparable method, subtraction and addition are carried out alternately.

In the decimal notation the processing rate is the same with the reparable and irreparable methods. In the binary notation, the irreparable method is two times faster than the reparable method because in the former the subtraction is carried out without recovery operations and the next digit is added or subtracted depending on the occurrence or nonoccurrence of a borrow. Example (2) shows the irreparable method.

Example (1) Reparable Method

184659 ÷ 789 = 234 + remainder 33

			△Q	Quotient	
	184659	Dividend			
—)	78900	Divisor			
	105759		+100	100	
—)	78900				
	26859		+100	200	
—)	78900				
	52041	Negative remainder	+100	300	
+)	78900				
	26850		—100	200	Recovery Operation
—)	7890				
	18969		+10	210	
—)	7890				
	11079		+10	220	
—)	7890				
	3189		+10	230	
—)	7890				
	—4701	Negative remainder	+10	240	
+)	7890				
	3189		—10	230	Recovery Operation
—)	789				
	2400		+1	231	
—)	789				
	1611		+1	232	
—)	789				
	822		+1	233	
—)	789				
	33		+1	234	
—)	789				
	—756	Negative remainder	+1	235	
+)	789				
	33	Remainder	—1	234	Recovery operation

Example (2) Irreparable Method

184659 ÷ 789 = 234 + remainder 33

			△Q	Quotient
	184659	Dividend		
—)	78900	Divisor	+100	100
	105759			
—)	78900		+100	200
	26859			
—)	78900		+100	300
	—52041	Negative remainder		
+)	78900			
	—44151		—10	290
—)	7890			
	—36261		—10	280
—)	7890			
	—28371		—10	270
—)	7890			
	—20481		—10	260
—)	7890			
	—12591		—10	250
+)	7890			
	—4701		—10	240
—)	7890			
	3189	Positive remainder	—10	230
—)	789			
	2400		+1	231
—)	789			
	1611		+1	232
—)	789			
	822		+1	233
—)	789			
	33	Remainder	+1	234

Processing procedure:

(a) The OPE. and the B.F.P.ACC are 0-checked. If OPE. = 0, the data overflows.
The OPE. value is converted into an unsigned fixed point part.

(b) The complements of the OPE. exponential parts are added to each other.

(c) The fixed point part division is carried out using the following procedure:

(i) Both dividend and divisor are 24-bit. As shown below, the result of the fixed point part division shows that a carry from the decimal point is necessary. Thus, the OPE. value is first subtracted from the B.F.P.ACC value to obtain the most significant bit of quotient.

The next digit operation will jump to addition if quotient = 0, to subtraction if quotient = 1, and 1 will set into the quotient. The divisor is set to 1/2 and is stored into (WS1, WS2, WS3, WS4).

Example: 0.413 ÷ 0.211 → 1.957 Incorrect
→ 0.1957 x 10 Correct

(ii) Operations with the irreparable method are carried out in accordance with the flowchart in figure 1-11.

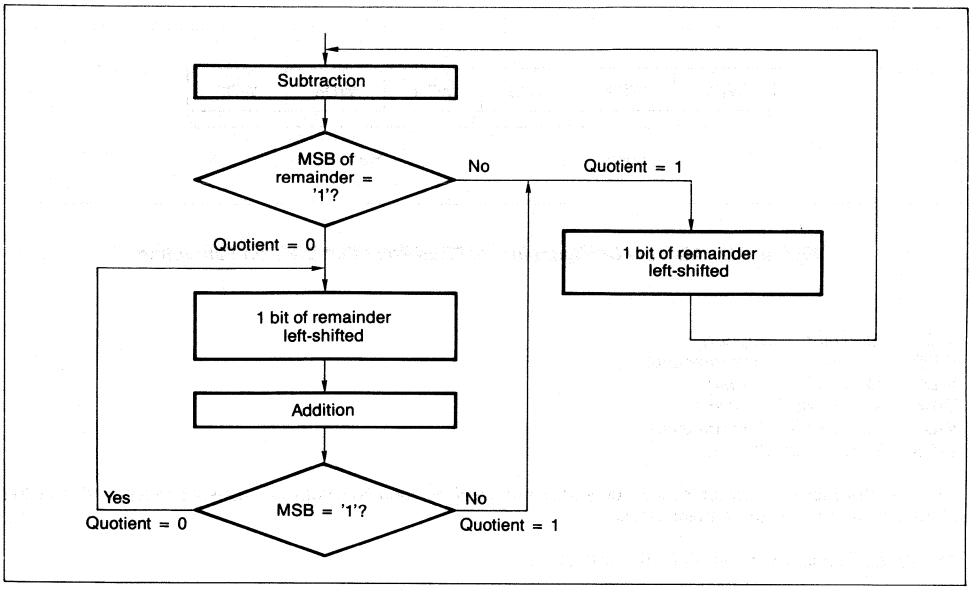


Figure 1-11 Irreparable Method Flowchart

Because the MSB of the remainder is cleared as a result of the subtraction and addition (see figure 1-11), the remainder is left-shifted by 1-bit for the next bit operation. At the same time the MSB is provided with a quotient.

As a result, the following will become clear:

① If the remainder does not become negative after a subtraction, MSB = 0. Thus, the inverted MSB value is the quotient.

If the remainder becomes negative, MSB = 1. Thus, the inverted MSB value is the quotient.

② If the remainder does not become positive after an addition, MSB = 1. If the remainder becomes positive, MSB = 0. Thus, both inverted MSB values are the quotient.

Because the inverted MSB is the quotient in both ① and ②, "1" is set to the MSB of WS1 when dividing the divisor by 2 in (i) above.

Next, the memory configuration for fixed point part division subroutine memory is shown in figure 1-12.

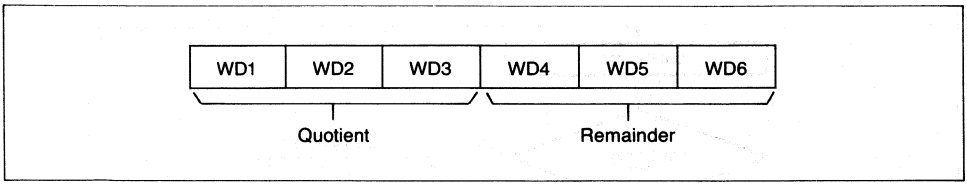


Figure 1-12 Memory Configuration for Fixed Point Part Division Subroutine

WD1	Quotient	(upper)
WD2	Quotient	(intermediate)
WD3	Quotient	(lower)
WD4	Remainder	(upper
WD5	Remainder	(intermediate)
WD6	Remainder	(lower)

Because the quotient is set for the MSB of WD4 as a result of the operation, both the quotient and remainder are left-shifted by 1-bit of 6-bytes simultaneously.

The division ends when 1 is set for the MSB of WD1.

Figures 1-13 and 1-14 show the flowcharts.

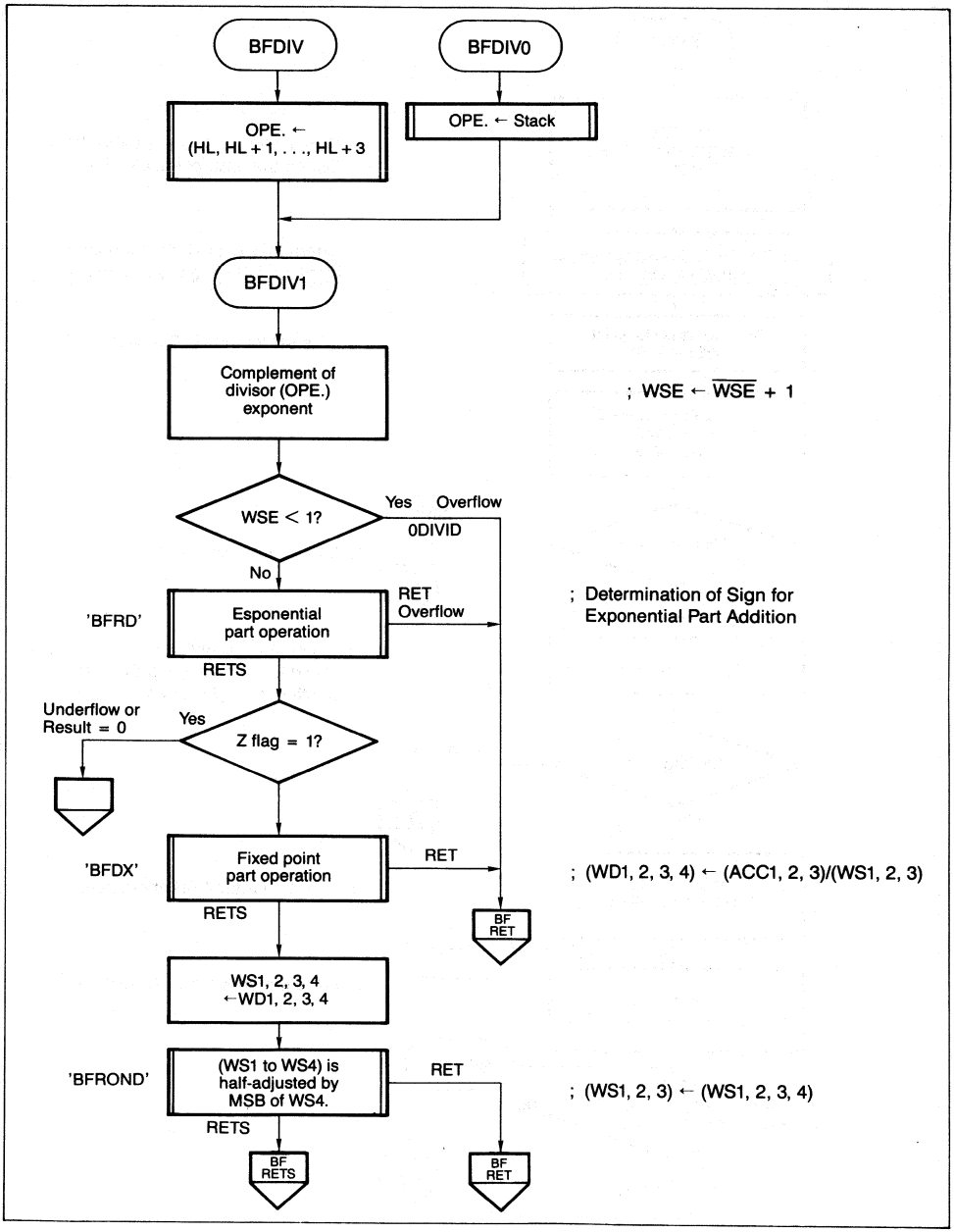
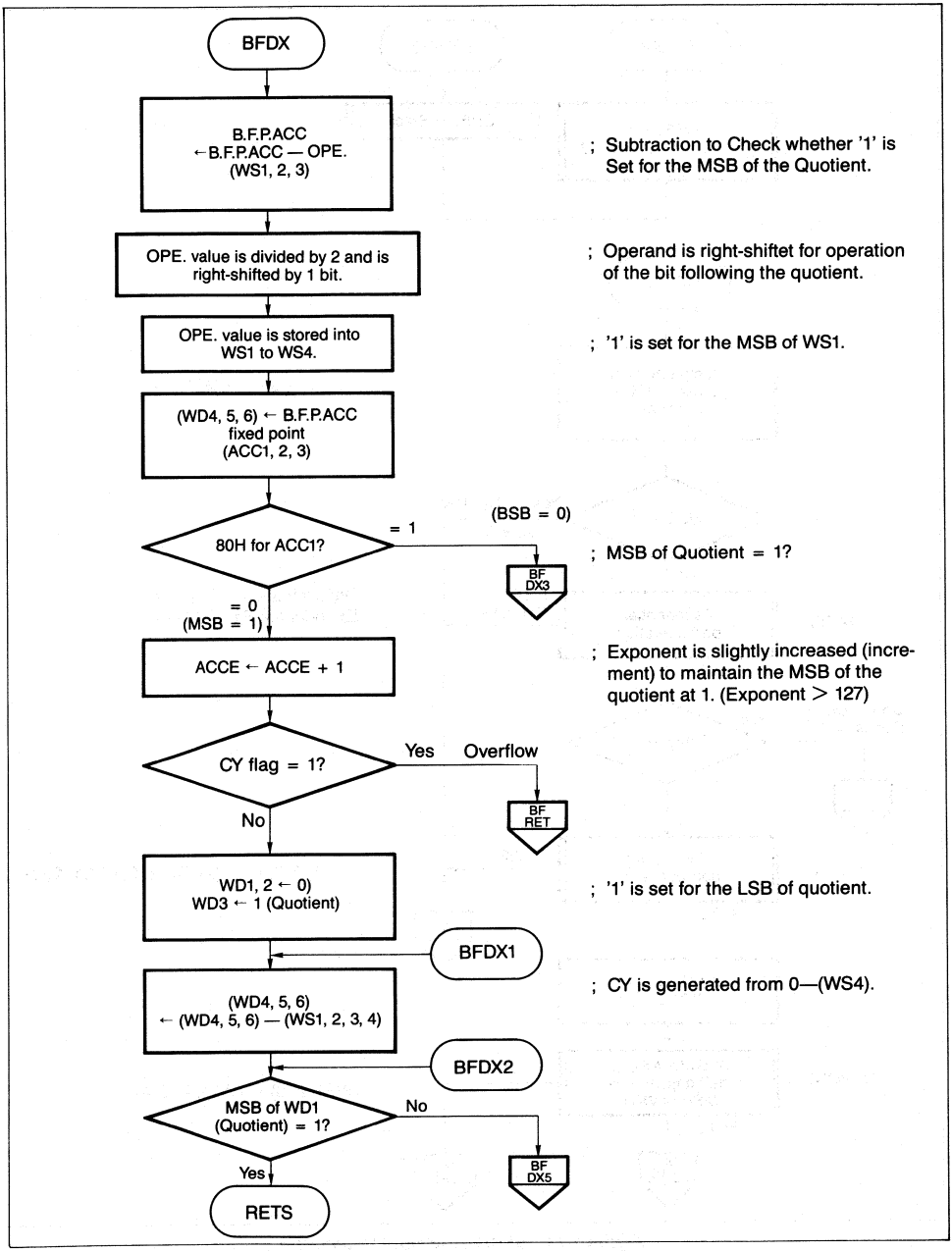


Figure 1-13 BFDIV (Division) Subroutine


BFDX
B.F.P.ACC ← B.F.P.ACC — OPE. (WS1, 2, 3)
; Subtraction to Check whether '1' is Set for the MSB of the Quotient.
OPE. value is divided by 2 and is right-shifted by 1 bit.
; Operand is right-shiftet for operation of the bit following the quotient.
OPE. value is stored into WS1 to WS4.
; '1' is set for the MSB of WS1.
(WD4, 5, 6) ← B.F.P.ACC fixed point (ACC1, 2, 3)
80H for ACC1?
= 1
(BSB = 0)
BF DX3
; MSB of Quotient = 1?
= 0 (MSB = 1)
ACCE ← ACCE + 1
; Exponent is slightly increased (increment) to maintain the MSB of the quotient at 1. (Exponent > 127)
CY flag = 1?
Yes
Overflow
BF RET
No
WD1, 2 ← 0) WD3 ← 1 (Quotient)
; '1' is set for the LSB of quotient.
BFDX1
(WD4, 5, 6) ← (WD4, 5, 6) — (WS1, 2, 3, 4)
; CY is generated from 0—(WS4).
BFDX2
MSB of WD1 (Quotient) = 1?
No
BF DX5
Yes
RETS

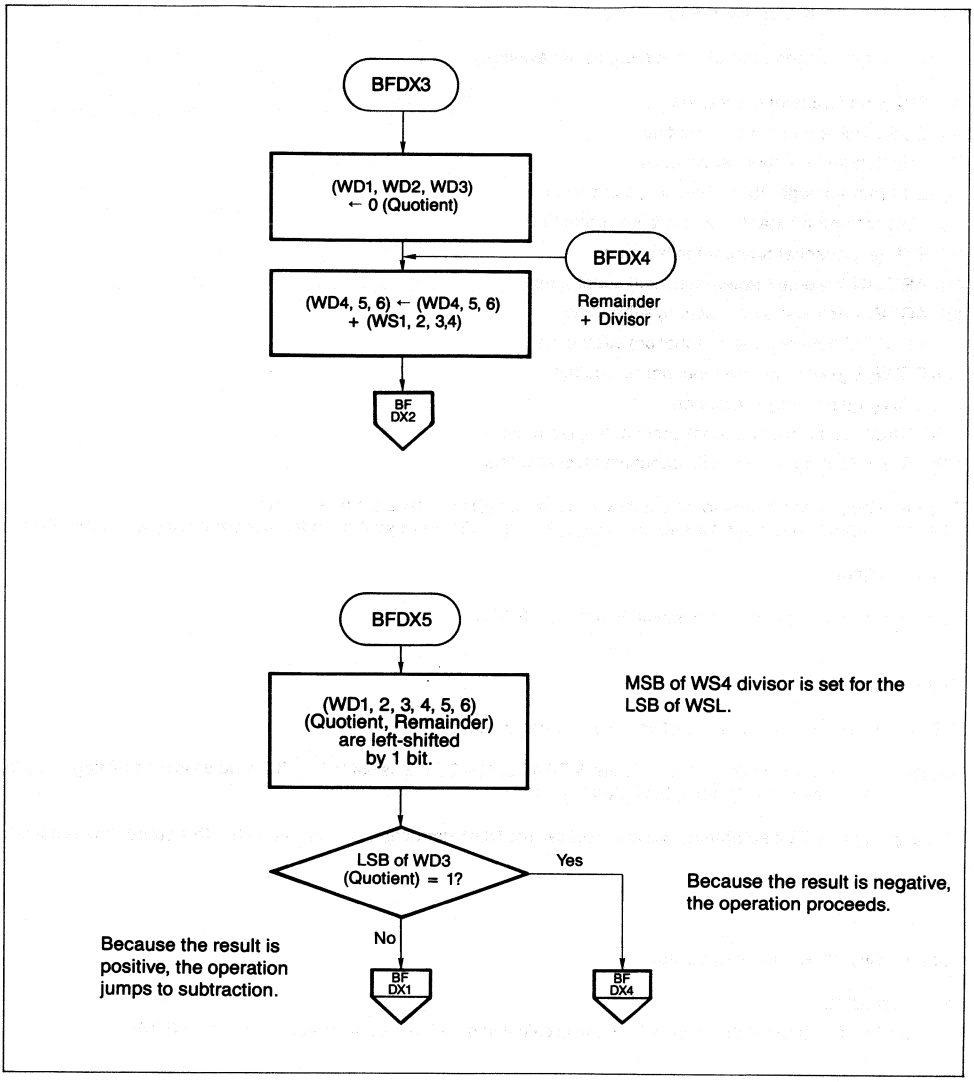


Figure 1-14 BFDX (Fixed Point Part Division) Subroutine

1.3.2 Functional Operation Subroutines

The following function operation subroutines are available:

(1) SIN (sine function) subroutine
(2) COS (cosine function) subroutine
(3) TAN (tangent function) subroutine
(4) LNX (natural logarithmic function) subroutine
(5) LOG (common logarithmic function) subroutine
(6) EXP (exponential function) subroutine
(7) ARCTAN (reverse tangent function) subroutine
(8) ARCSIN (reverse sine function) subroutine
(9) ARCCOS (reverse cosine function) subroutine
(10) POWER (power square function) subroutine
(11) SQR (square root) subroutine
(12) TRACA (polar to orthogonal coordinates) subroutine
(13) TRACB (orthogonal to polar coordinates) subroutine

The input/output conditions for subroutines (1) through (9) and)11) are set as follows.
The input/output conditions for subroutines (10) through (13) are described in the corresponding sections below.

Input conditions:

The binary floating point data is stored into the B.F.P.ACC.

Output conditions:

RET; The operation result overflows. The CY flag is set to "2".

RETS; The normal result is stored into the B.F.P.ACC. The CY flag is set to "0". If the result is 0, the Z flag is set to "1". If the result is not 0, the Z flag is set to "0".

Because (7) ARCTAN subroutine will not overflow, the NOP instruction is necessary after this subroutine is called.

1.3.2.1 SIN (Sine Function) Subroutine

(1) Processing
The SIN function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions
Refer to the previous description of the functional operation input conditions.

(3) Output conditons
Refer to the previous description of the functional operation output conditions.

(4) Subroutines used
Refer to Table 1-6.

(5) Stack depth

13 max.

(6) Coding order

```
      ≀
CALL SIN      ; SIN(X)
GJMP ERROR    ; Jumps to the over flow processing operation.
      ≀
```

(7) Processing time

Approx. 30 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC— 82H, 00H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

The algorithm is given as

$$\text{SIN } X = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \cdots \quad (3.1)$$

(unit: radian)

The series, up to the fifth term, are calculated.

Processing procedure:

(a) If data is negative, —X is changed into X because SIN (—X) = —SIN (X).

(b) To substitute in equation (3.1), data is scaled between 0 and 1 using $\theta = x/2\pi - \text{INT}(x/2\pi)$ from $\text{SIN}(2n\pi\ x) = \text{SIN}(x)$. (See Table 1-2.)

Table 1-2 Scaling

θ	X
0 to 1/4	0 to π/2
1/4 to 1/2	π/2 to π
1/2 to 3/4	π to 3/2π
3/4 to 1	3/2π to 2π

(c) The data is further scaled between —1/4 and 1/4.

(I) $\theta \leftarrow 1/4 - \theta$

If θ is negative, $(X > \pi/2)$, $\theta \leftarrow \theta + 1/2$

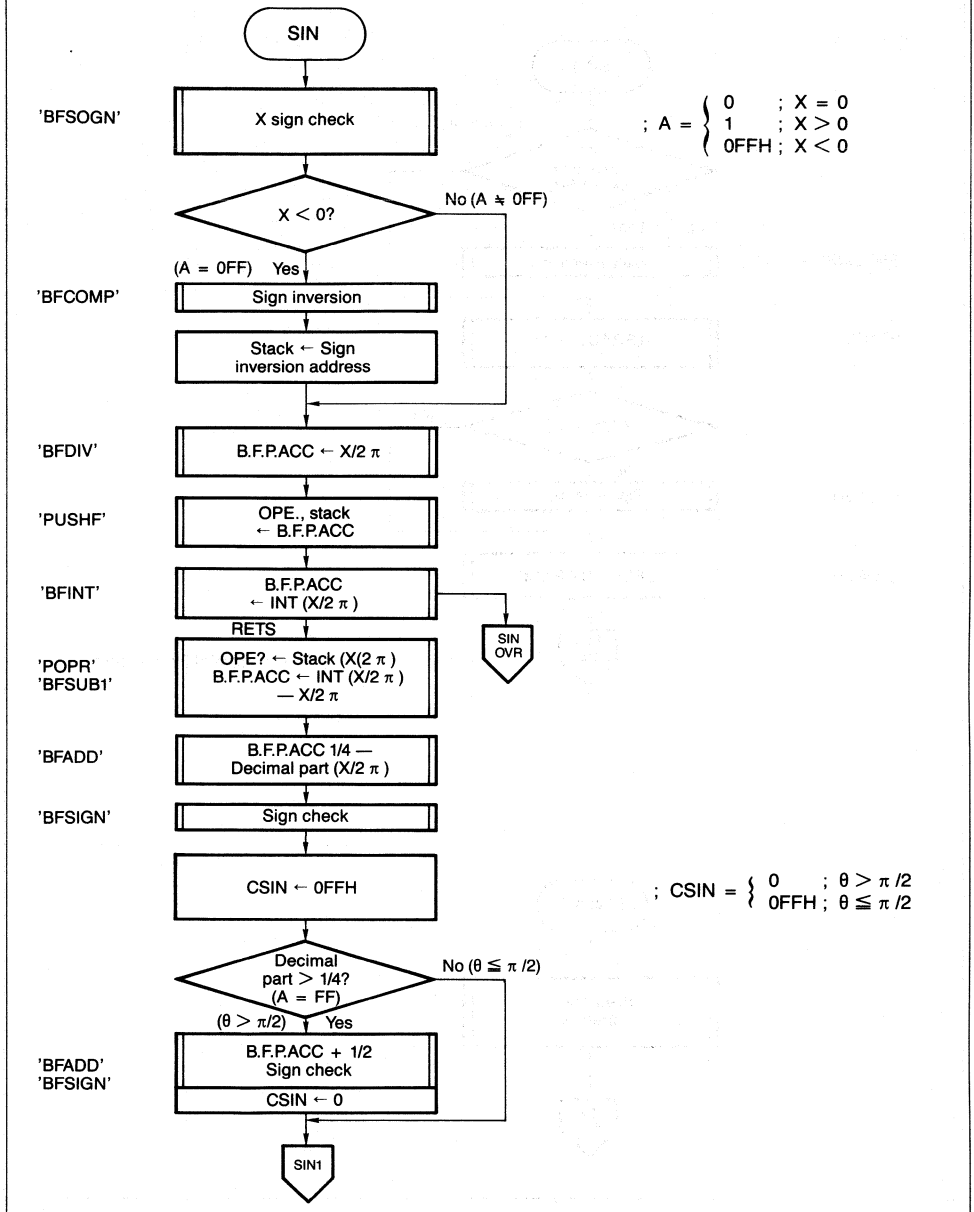
(II) If θ is negative, $(X > 3/2\pi)$, $\theta \leftarrow -\theta$

(III) $\theta \leftarrow \theta + 1/4$

(IV) If $X > \pi/2$, $\theta \leftarrow -\theta$

(d) $X = 2\pi\theta$ is substituted in equation (3.1).
In this subroutine, 2π is included into the coefficient to the series.

(e) If $X < 0$, the sign of the result obtained in (a) to (d) is inverted.
Figure 1-15 shows the flowchart.


SIN
'BFSOGN'
X sign check
; A = { 0 ; X = 0
1 ; X > 0
0FFH ; X < 0
X < 0?
No (A ≠ 0FF)
(A = 0FF) Yes
'BFCOMP'
Sign inversion
Stack ← Sign inversion address
'BFDIV'
B.F.P.ACC ← X/2 π
'PUSHF'
OPE., stack ← B.F.P.ACC
'BFINT'
B.F.P.ACC ← INT (X/2 π)
SIN OVR
RETS
'POPR'
'BFSUB1'
OPE? ← Stack (X(2 π)
B.F.P.ACC ← INT (X/2 π)
— X/2 π
'BFADD'
B.F.P.ACC 1/4 —
Decimal part (X/2 π)
'BFSIGN'
Sign check
CSIN ← 0FFH
; CSIN = { 0 ; θ > π /2
0FFH ; θ ≦ π /2
Decimal part > 1/4? (A = FF)
No (θ ≦ π /2)
(θ > π/2) Yes
'BFADD'
'BFSIGN'
B.F.P.ACC + 1/2
Sign check
CSIN ← 0
SIN1

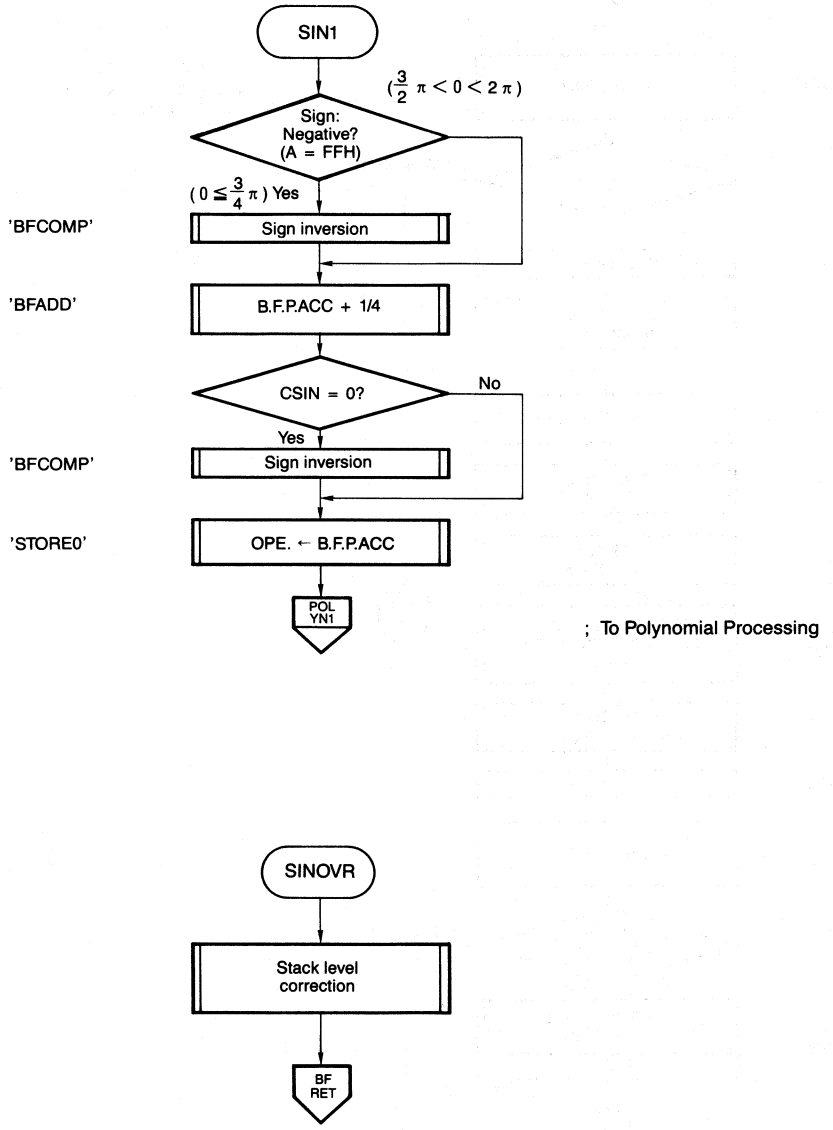


Figure 1-15 SIN Subroutine

1.3.2.2 COS (Cosine Function) Subroutine

(1) Processing

The COS function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditons

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-7.

(5) Stack depth

13 max.

(6) Coding order

```
      ≀
CALL COS      ; COS(X)
GJMP ERROR    ; Jumps to the over flow processing operation.
      ≀
```

(7) Processing time

Approx. 30 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC— 81H, 00H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

The following equation is used:

$$\cos X = \sin\left(\frac{\pi}{2} + \pm x\right) \qquad \text{(unit: radian)}$$

Processing procedure:

(a) If the result overflows, use the following equation:

$$\cos X = \sin\left(\frac{\pi}{2} - X\right)$$

If the result does not overflow, use the following equation:

$$\cos X = \sin\left(\frac{\pi}{2} + X\right)$$

(b) The SIN value (in parentheses (a) above) is calculated and the SIN subroutine is used for the subsequent operations.

Figure 1-16 shows the flowchart.

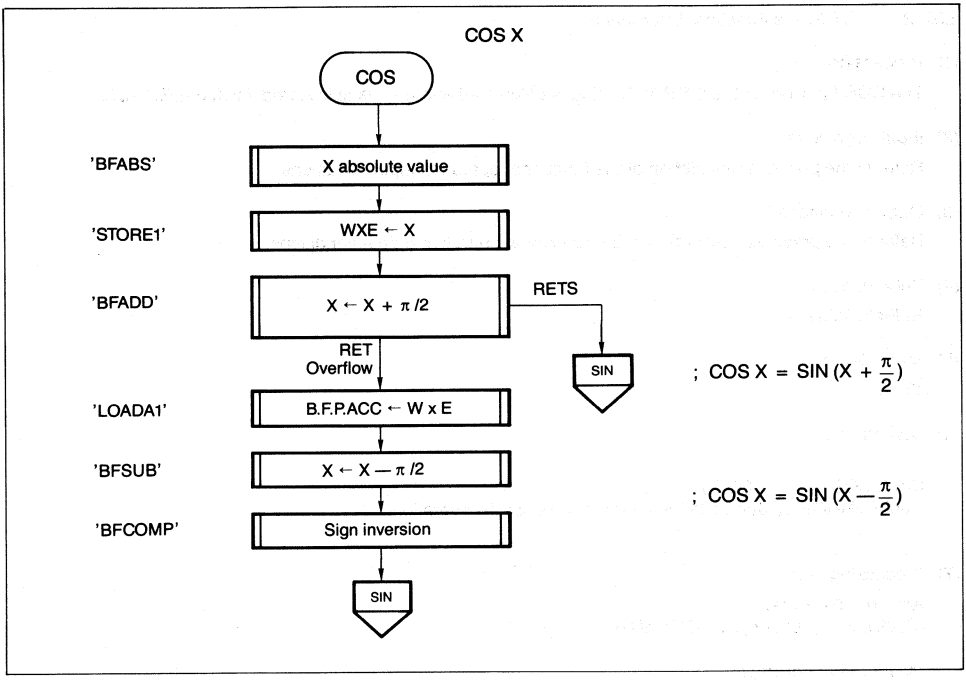


Figure 1-16 COS Subroutine

1.3.2.3 TAN (Tangent Function) Subroutine

(1) Processing

The TAN function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditons

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-8.

(5) Stack depth

16 max.

(6) Coding order

```
      ≀
CALL TAN      ; TAN(X)
GJMP ERROR    ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 65 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC— 81H, 00H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

The following equation is used:

$$TAN\ X = \frac{SIN\ X}{COS\ X}$$ (unit: radian)

Processing procedure:

(a) COS X is calculated. If the result overflows, the operation ends.

(b) SIN X is calculated. If the result overflows, the operation ends.

(c) TAN X is calculated by dividing SIN X by COS X.
Figure 1-17 shows the flowchart.

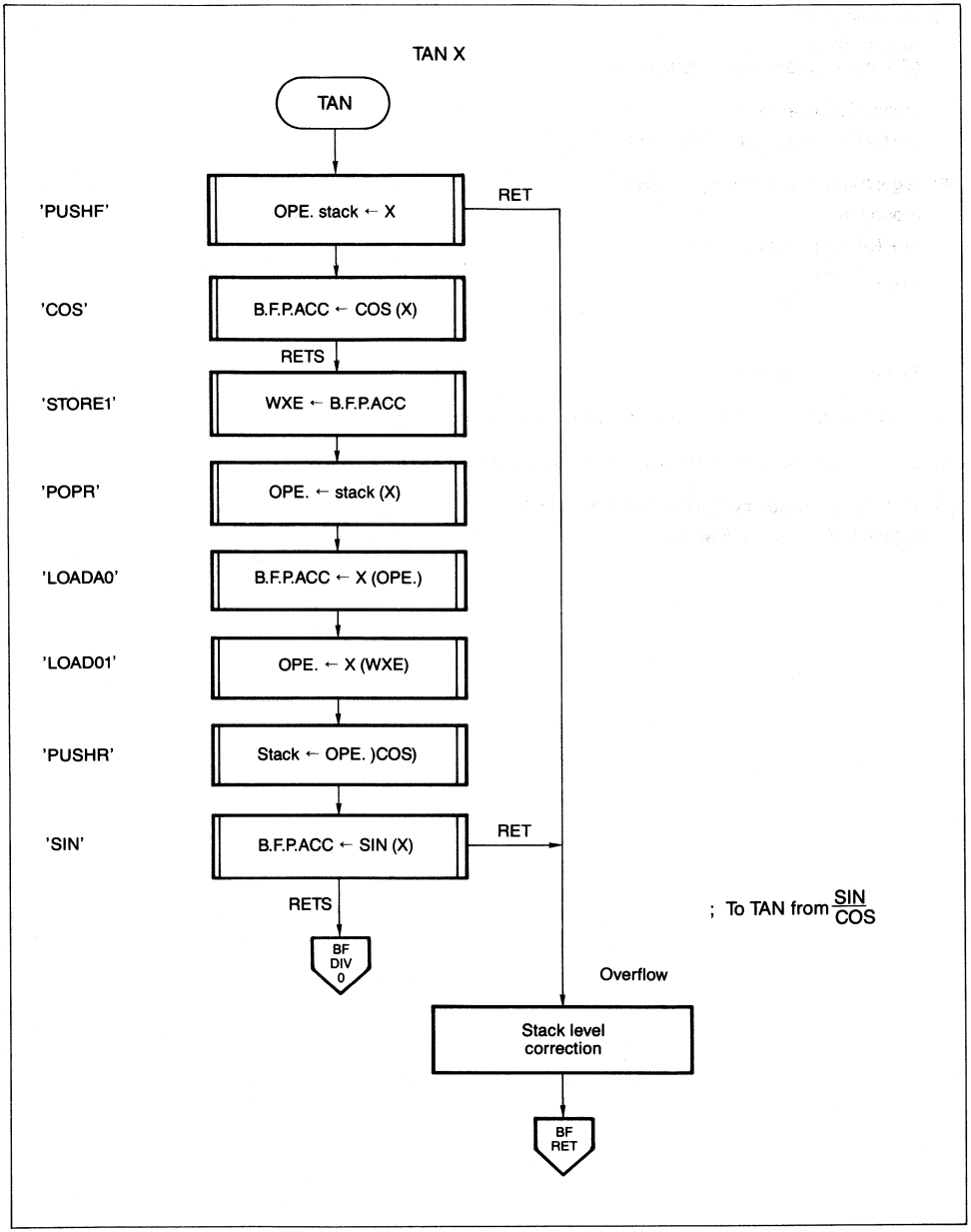


Figure 1-17 TAN Subroutine

1.3.2.4 LNX (Natural Logarithmic Function) Subroutine

(1) Processing

The LN function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditons

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-9.

(5) Stack depth

13 max.

(6) Coding order

```
      ≀
CALL LNK      ; LN(X)
GJMP ERROR    ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 16 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC— FFH, 80H, FFH, FFH, FFH

(8) Algorithm and procedure

Algorithm:

The following expansions are available for the LN logarithmic function:

$$LN(1 + X) = X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \cdots (3.2)$$

$$LN(1 - X) = -X - \frac{X^2}{2} - \frac{X^3}{3} - \frac{X^4}{4} + \cdots (3.3)$$

These equations can be expanded in the range of $-1 \leqq X \leqq 1$.

If $Y = \frac{X - 1}{X + 1} \cdots (3.4)$ is set, $-1 < Y < 1$ for $0 < X < \infty$.

If $X = \frac{1 + Y}{1 - Y} \cdots (3.5)$ is set, $LNX = LN(1 + Y) - LN(1 - Y) = 2\left(Y + \frac{Y^3}{3} + \frac{Y^5}{5} + \cdots\right) \cdots (3.6)$

from equations (3.2), (3.3), and (3.5). Thus, equation (3.6) can be used for any $X (>0)$.

Procedure:

(a) In equation (3.4), Y varies only slightly if X is too large or too small. Thus, the processing is carried out with the exponential part (ACCE) of X set to 80H (2^0).

(b) If the B.F.P.ACC value is set to X_1,

① When ACCE ≧ 80H, $LNX = LN(X_1 \cdot 2^{ACC-80H}) = LNX_1 + (ACCE - 80H) \cdot LN2$ . . (3.7)

② When ACCE < 80H, $LNX = LN(X_1 \cdot 2^{80H-ACCE}) = LNX_1 - (ACCE - 80H) \cdot LN1$ . . (3.8)

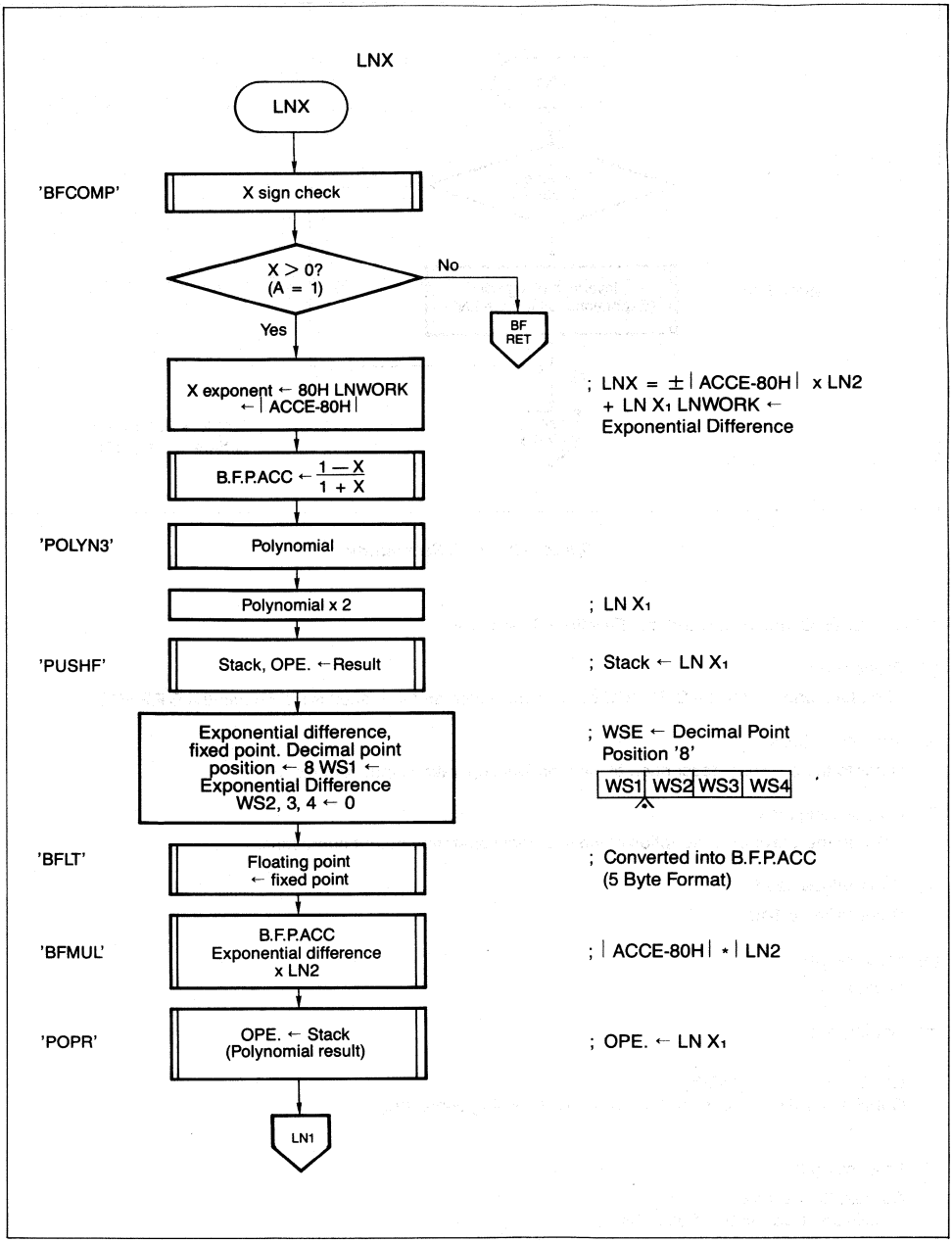
(I) LNX_1 is calculated from equations (3.4) and (3.6).

(II) "ACCE — 80H" is calculated.
The exponential difference (binary fixed point number) between the exponential part of X and 80H (2^0) is converted to B.F.P.ACC (binary floating point number)

(III) "(ACCE — 80H) x LN2" is calculated.

(IV) ① When X ≧ 80H, "result of (I) + result of (III)" is calculated in accordance with equation (3.7).
② When X < 80H, "result of (I) — result of (III)" is calculated in accordance with equation (3.8).

The result is stored into the B.F.P.ACC. Figure 1-18 shows the flowchart.


LNX
LNX
'BFCOMP'
X sign check
X > 0?
(A = 1)
No
Yes
BF
RET
X exponent ← 80H LNWORK
← | ACCE-80H |
; LNX = ± | ACCE-80H | x LN2
+ LN X1 LNWORK ←
Exponential Difference
B.F.P.ACC ← 1 — X / 1 + X
'POLYN3'
Polynomial
Polynomial x 2
; LN X1
'PUSHF'
Stack, OPE. ←Result
; Stack ← LN X1
Exponential difference,
fixed point. Decimal point
position ← 8 WS1 ←
Exponential Difference
WS2, 3, 4 ← 0
; WSE ← Decimal Point
Position '8'
WS1 WS2 WS3 WS4
'BFLT'
Floating point
← fixed point
; Converted into B.F.P.ACC
(5 Byte Format)
'BFMUL'
B.F.P.ACC
Exponential difference
x LN2
; | ACCE-80H | * | LN2
'POPR'
OPE. ← Stack
(Polynomial result)
; OPE. ← LN X1
LN1

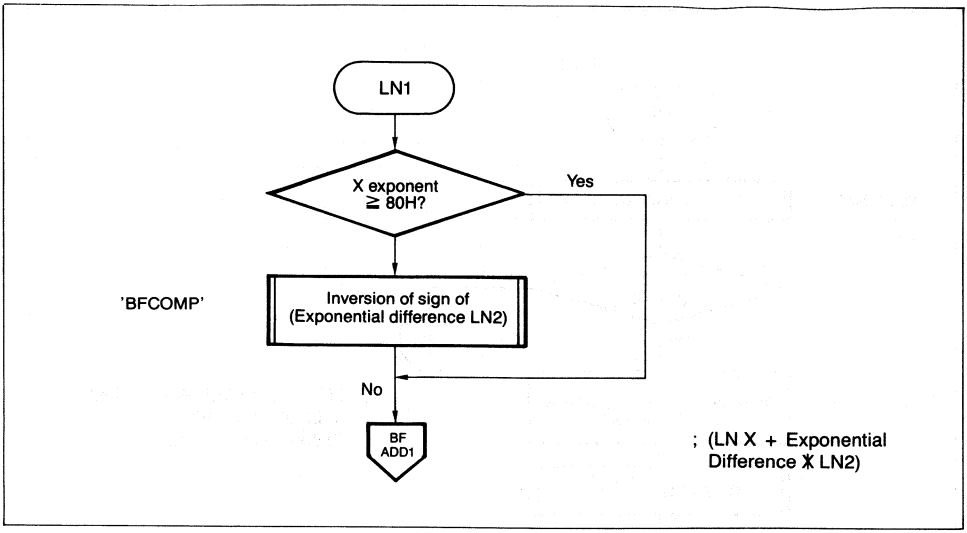


Figure 1-18 LNX Subroutine

1.3.2.5 LOG (Common Logarithmic Function) Subroutine

(1) Processing

The LOG function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditons

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-10.

(5) Stack depth

14 max.

(6) Coding order

```
      ≀
CALL LOG      ; LOG(X)
GJMP ERROR    ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 20 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC— FFH, 80H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

$LN\ X = \frac{LN\ X}{LN\ 10}$ "where X > 0" is used.

Processing procedure:

(a) LN X is calculated. If the result overflows, the operation ends.

(b) LOG X is calculated from $\frac{LN\ X}{LN\ 10}$.

Figure 1-19 shows the flowchart.

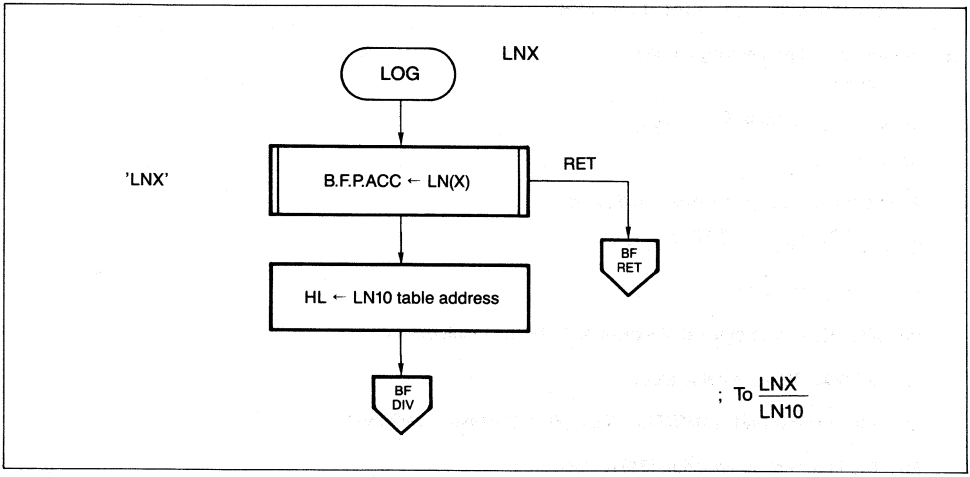


Figure 1-19 LOG Subroutine

1.3.2.6 EXP (Exponential Function) Subroutine

(1) Processing

The EXP function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditons

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-11.

(5) Stack depth

12 max.

(6) Coding order

```
      ≀
CALL EXP      ; EXP(X)
GJMP ERROR    ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 28 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC — 86H, 80H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

$EXP(X) = 2^{(X \times Log_2(E))}$ . . . (3.8)

where E = _____

2^Y is obtained using the following equation.

$2^Y = 2^{INT(Y)} \times 2^{(Y-INT(Y))}$ . . . (3.9)

Processing procedure:

(a) X✱LOG₂E is calculated. If exponent > 89, the result overflows.

(b) INT(X✱LOG₂E) is calculated.

(c) If the integer part (TMPWSL) is 7E or 7FH, the result overflows.

(d) The latter half of equation (3.9) is calculated.

$2^{(Y-INT(Y))} = EXP(X-LN2 \times (INT(X \times LOG_2E)))$ . . . (3.10)

The value in the EXP parentheses of equation (3.10) is substituted in the following equation.

$$EXP(Z) = 1 + \frac{Z}{1!} + \frac{Z^2}{2!} + \frac{Z^3}{3!} + \cdots$$

(e) The result of (d) is multiplied by $2^{INT(Y)}$. In other words, INT(Y) is added to the exponential part of the result of (d).

(f) If X < 0, the result of the following calculation is reversed.

EXP (—| X|) = 1/EXP (| X|)

Figure 1-20 shows the flowchart.

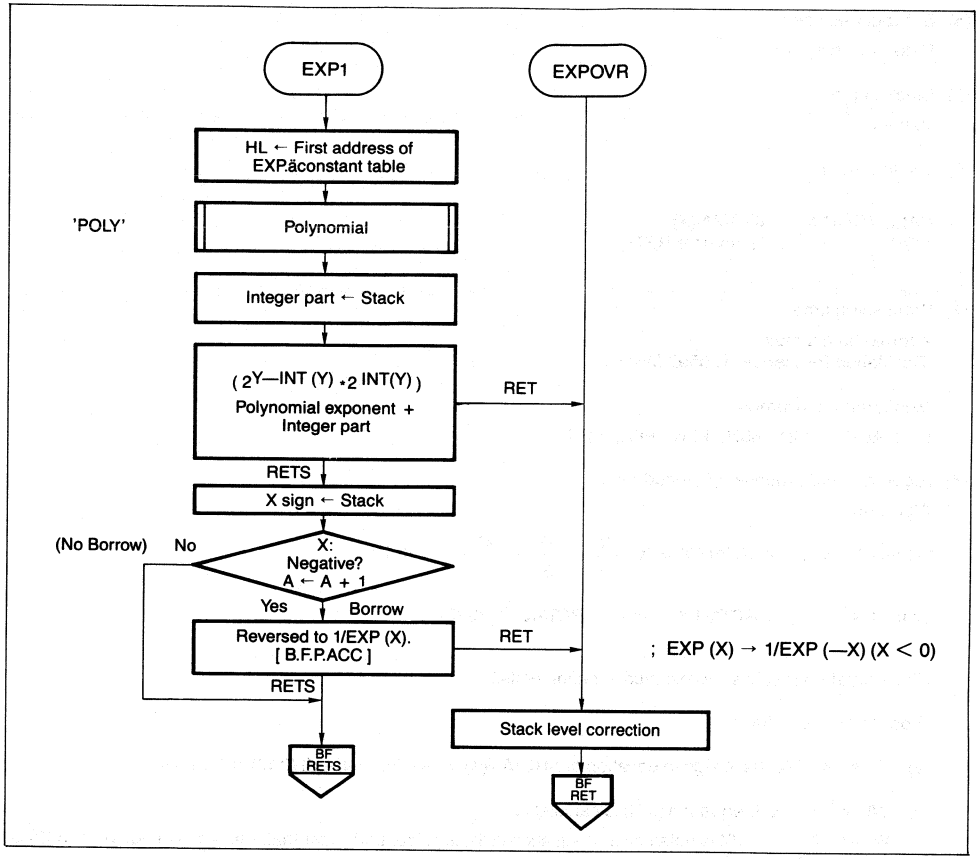


Figure 1-20 EXP Subroutine

1.3.2.7 ARCTAN (Reverse Tangent Function) Subroutine

(1) Processing

The ARCTAN function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditons

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-12.

(5) Stack depth

16 max.

(6) Coding order

```
      ≀
CALL ARCTAN  ; ARCTAN(X)
NOP          ; Dummy for RETS.
      ≀
```

(7) Processing time

Approx. 30 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC — 90H, 80H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

When $|X| \leqq 1$, $\text{ARCTAM } X = X - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + \ldots$ (3.11)

When $|X| > 1$, $\text{ARCTAN } X = \pi/2 - \text{ARCTAN } \frac{1}{|x|}$ (3.12)

(The operation result is represented in radian units.)

Processing procedure:

(a) When $X < 0$, the X sign in equation of ARCTAN(X) = —ARCTAN(—X) (3.13) is reversed.

(b) When $|X| \leqq 1$, equation (3.11) is calculated.
When $|X| > 1$, 1/X is obtained and equation (3.11) is calculated. After that, the result of equation (3.12) is subtracted from $\pi/2$.

(c) If $X < 0$ in (a) above, the sign of the result of (b) is reversed from equation (3.13).

Figure 1-21 shows the flowchart.

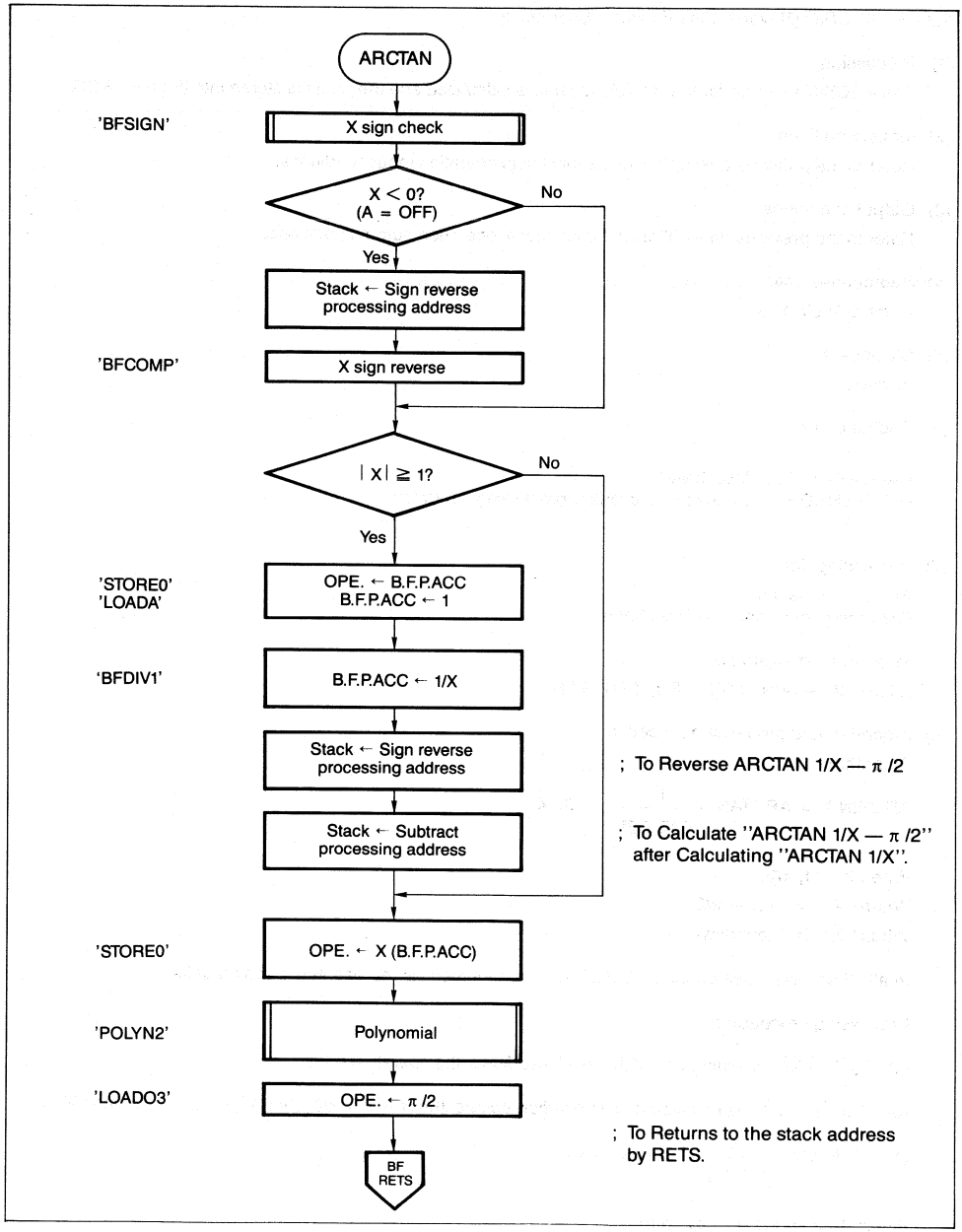


Figure 1-21 ARCTAN Subroutine

1.3.2.8 ARCSIN (Reverse Sine Function) Subroutine

(1) Processing

The ARCSIN function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditions

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-13.

(5) Stack depth

21 max.

(6) Coding order

```
      ≀
CALL ARCSIN  ; ARCSIN(X)
GJMP ERROR   ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 75 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC — 80H, 80H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

$$\text{ARCSIN } X = \text{ARCTAN } \frac{1}{\sqrt{1 - X^2}} \quad \ldots (3.14)$$

When $X = 1$, $\pi/2$.
When $|X| = -1$, $-\pi/2$.
When $|X| > 1$, overflow.

In all other cases, use equation (3.14). The operation result is represented in radian units.

Processing procedure:

(a) "$\sqrt{1 - X^2}$" is calculated. If the result overflows, the operation ends.

(b) "$X/\sqrt{1 - X^2}$" is calculated. If an overflow occurs, the result = $\pi/2$ when $X = 1$ or $-\pi/2$ when $X = -1$.

(c) "$\text{ARCTAN } \frac{X}{\sqrt{1 - X^2}}$" is calculated.

Figure 1-22 shows the flowchart.

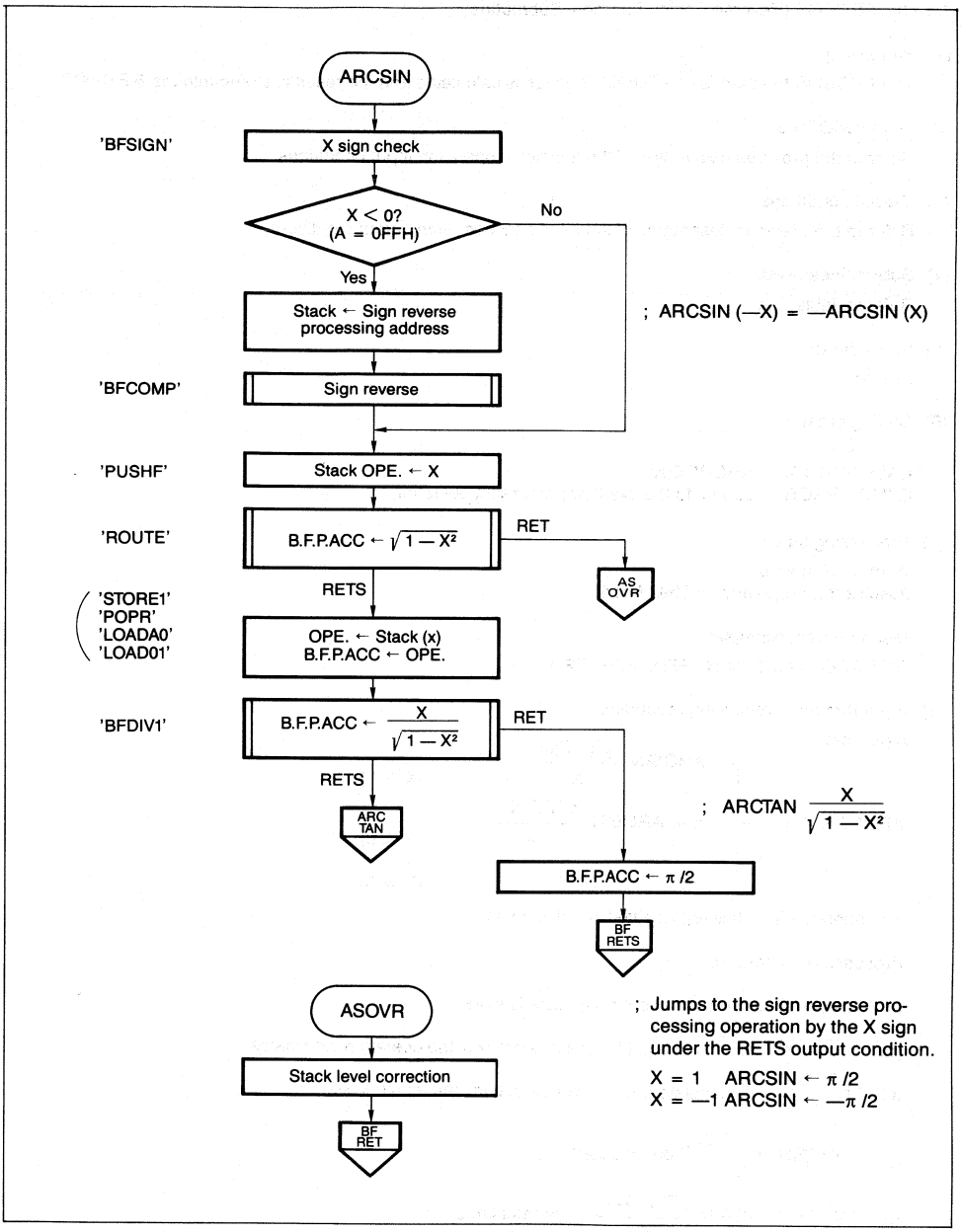


Figure 1-22 ARCSIN Subroutine

1.3.2.9 ARCCOS (Reverse Cosine Function) Subroutine

(1) Processing

The ARCCOS function for the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditions

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-14.

(5) Stack depth

21 max.

(6) Coding order

```
      ≀
CALL ARCCOS ; ARCCOS(X)
GJMP ERROR  ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 70 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC — 80H, 80H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

$$\text{ARCCOS } X = \begin{cases} \text{ARCTAN } \dfrac{\sqrt{1-X^2}}{X} & (X > 0) \\ \pi + \text{ARCTAN } \dfrac{\sqrt{1-X^2}}{X} & (X < 0) \\ \pi/2 & (X = 0) \end{cases}$$

The operation result is represented in radian units.

Processing procedure:

(a) The X sign is checked and the sign data is stored.

(b) "$\sqrt{1-X^2}$" is calculated. If the result overflows, the operation terminates.

(c) "$\dfrac{\sqrt{1-X^2}}{X}$" is calculated. If an overflow occurs, the result = $\pi/2$.

(d) "ARCTAN $\dfrac{\sqrt{1 \times X^2}}{X}$" is calculated.

(e) If $X < 0$, "ARCTAN $\dfrac{\sqrt{1-X^2}}{X} + \pi$" is calculated.

Figure 1-23 shows the flowchart.

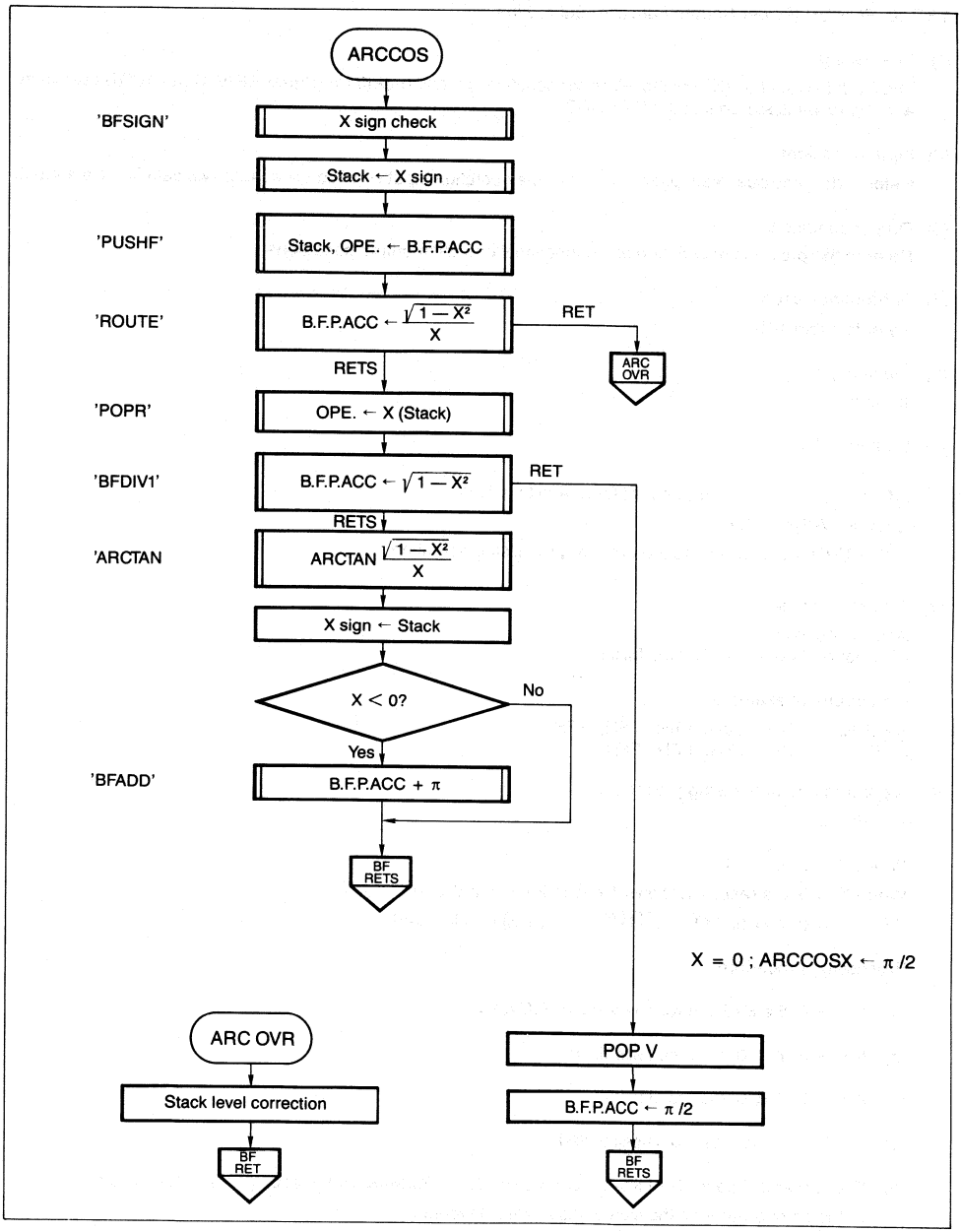


Figure 1-23 ARCCOS Subroutine

1.3.2.10 POWER (Power Square Function) Subroutine

(1) Processing

The POWER function (X^Y) for the 4-byte consecutive operand data (Y) and the B.F.P.ACC value (X) is calculated and the result is stored into the B.F.P.ACC.

(2) Input conditions

Refer to the previous description of the four-rule operation input conditions because two data items are input.

(3) Output conditions

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-15.

(5) Stack depth

16 max.

(6) Coding order

```
      ≀
LXI H, ____    ; First address of OPE. 4-byte data (Y)
CALL POWER     ; X^Y
GJMP ERROR     ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 46 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC	FFH, 80H, FFH, FFH, FFH
OPE.	10H, 7FH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

When Y = 0, $X^Y = 1$.
When X = 0, the result overflows if Y < 0. $X^Y = 0$ if Y ≧ 0.
When X ≠ 0, Y = 0, "$X^Y = e^{Y*LN(X)}$ . . . (3.15) is calculated.

Processing procedure:

(a) If Y = 0, the EXP subroutine is used. EXP(0) = 1.

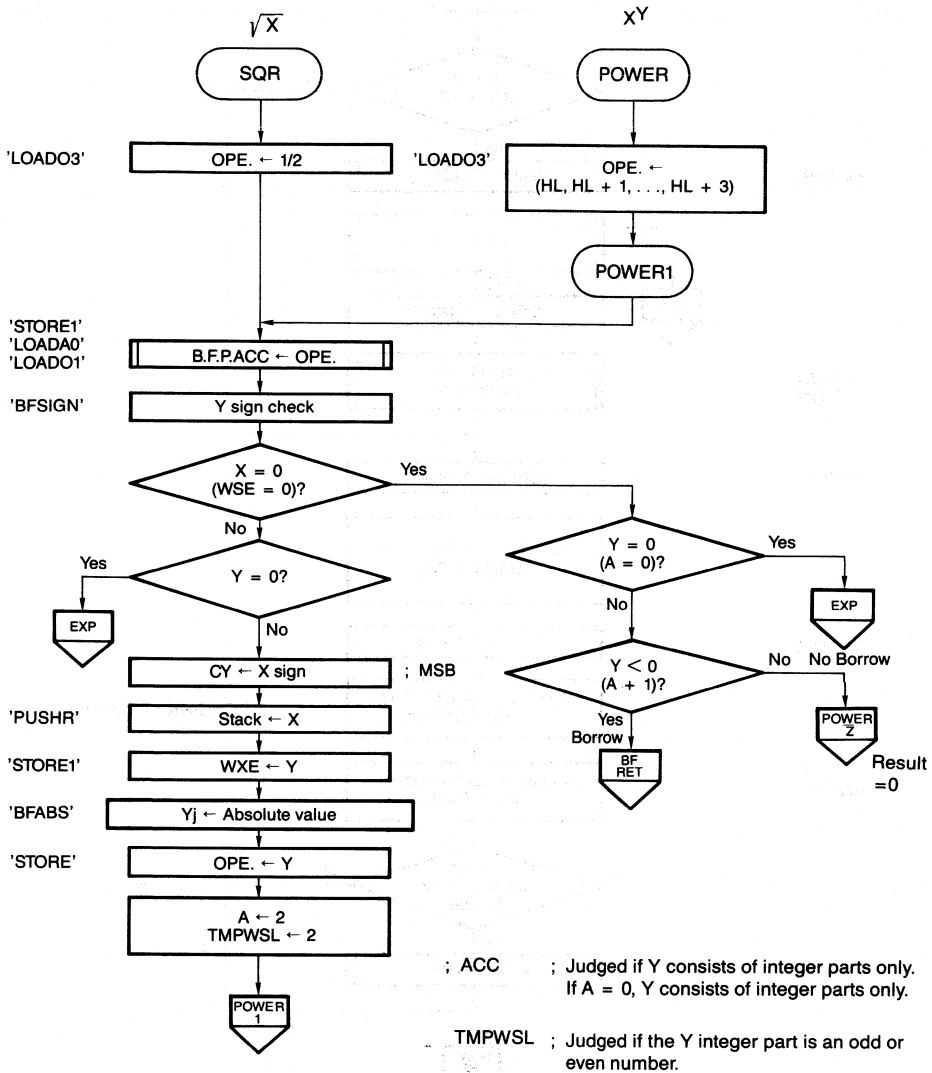
(b) If X = 0, Y < 0, the result overflows.

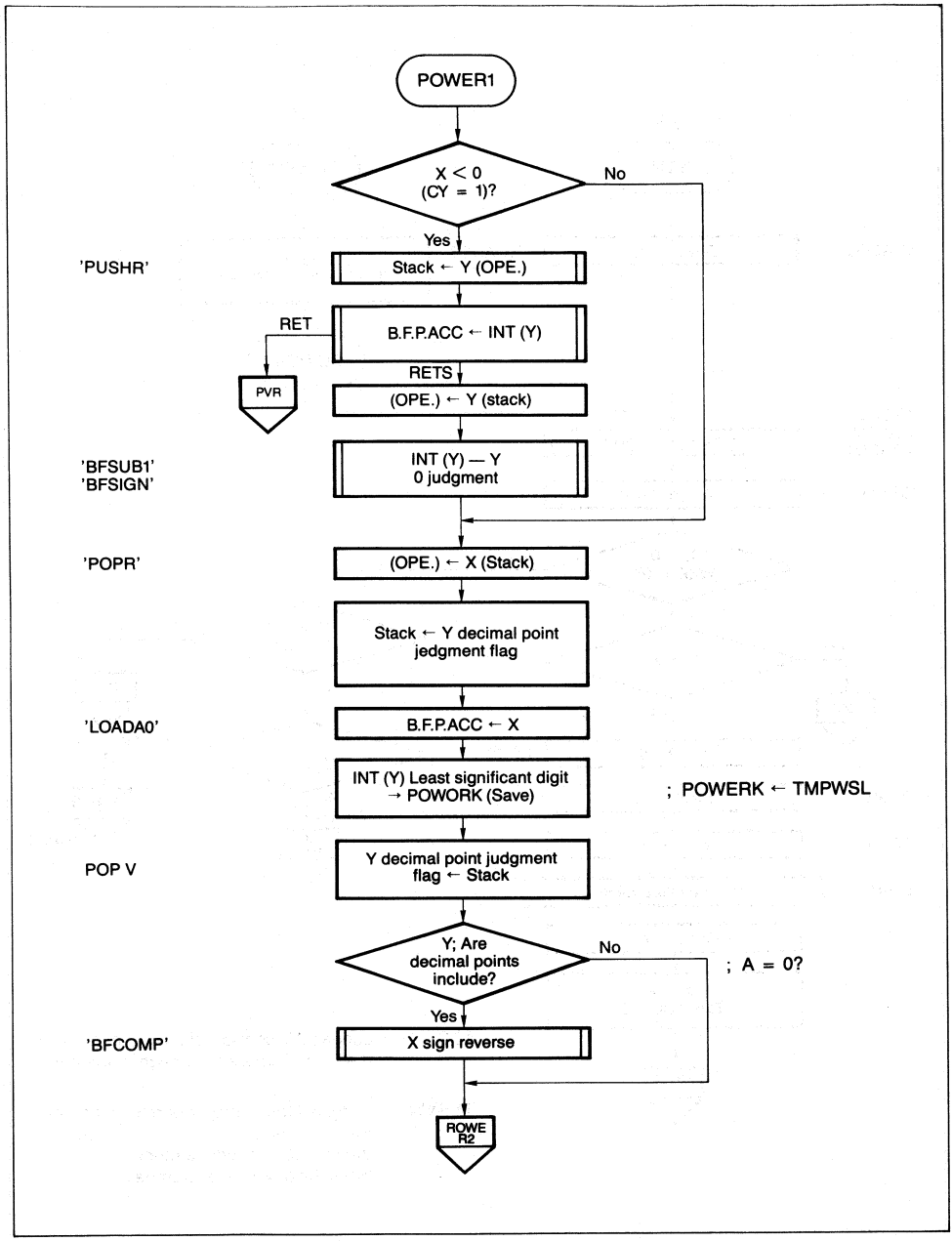
(c) If X = 0, Y ≧ 0, the result is "0".

(d) If X > 0, equation (3.15) is calculated.

(e) If X < 0 and Y consists of integer parts only, $|X|^Y$, followed by equation (3.15) re calculated.
If Y is an odd number, the sign of the result is reversed.
If X < 0 and Y consists of integer and other parts, the result overflows.

Figure 1-24 shows the flowchart.


√X
SQR
'LOADO3'
OPE. ← 1/2
XY
POWER
'LOADO3'
OPE. ←
(HL, HL + 1, . . ., HL + 3)
POWER1
'STORE1'
'LOADA0'
'LOADO1'
B.F.P.ACC ← OPE.
'BFSIGN'
Y sign check
X = 0
(WSE = 0)?
Yes
No
Yes
Y = 0?
EXP
No
CY ← X sign
; MSB
'PUSHR'
Stack ← X
'STORE1'
WXE ← Y
'BFABS'
Yj ← Absolute value
'STORE'
OPE. ← Y
A ← 2
TMPWSL ← 2
POWER 1
Y = 0
(A = 0)?
Yes
EXP
No
Y < 0
(A + 1)?
No
No Borrow
POWER Z
Result =0
Yes
Borrow
BF RET
; ACC ; Judged if Y consists of integer parts only.
If A = 0, Y consists of integer parts only.
TMPWSL ; Judged if the Y integer part is an odd or even number.
When bit 0 = 1, odd number.
When bit 0 = 0, even number.


POWER1
X < 0 (CY = 1)?
No
Yes
'PUSHR'
Stack ← Y (OPE.)
RET
B.F.P.ACC ← INT (Y)
RETS
PVR
(OPE.) ← Y (stack)
'BFSUB1'
'BFSIGN'
INT (Y) — Y 0 judgment
'POPR'
(OPE.) ← X (Stack)
Stack ← Y decimal point jedgment flag
'LOADA0'
B.F.P.ACC ← X
INT (Y) Least significant digit → POWORK (Save)
; POWERK ← TMPWSL
POP V
Y decimal point judgment flag ← Stack
Y; Are decimal points include?
No
; A = 0?
Yes
'BFCOMP'
X sign reverse
ROWE R2

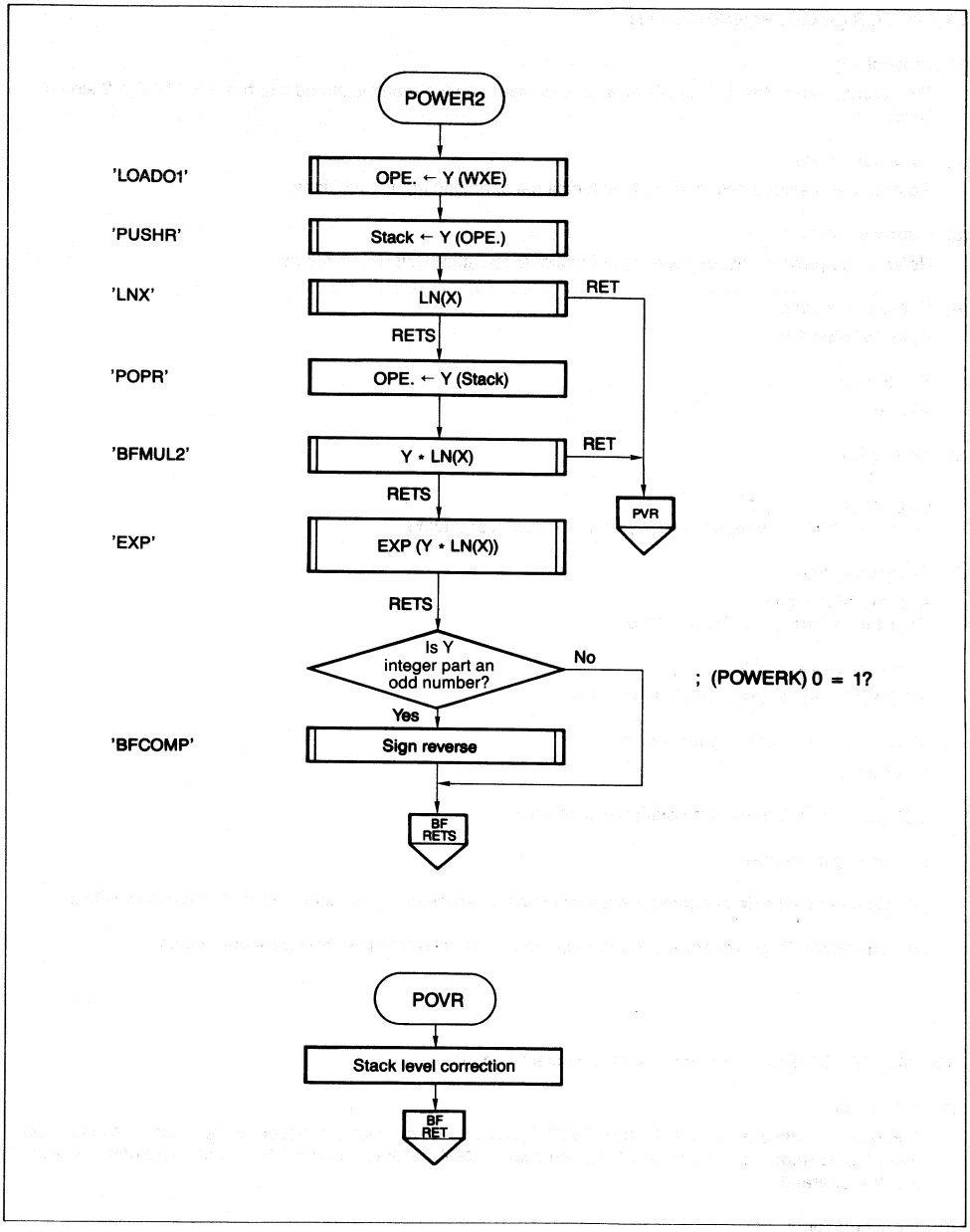


Figure 1-24 POWER and SQR Subroutine

1.3.2.11 SQR (Square Root) Subroutine

(1) Processing

The square root of the B.F.P.ACC value is calculated and the result is stored into the B.F.P.ACC. 1/2 stored in OPE.

(2) Input conditions

Refer to the previous description of the functional operation input conditions.

(3) Output conditions

Refer to the previous description of the functional operation output conditions.

(4) Subroutines used

Refer to Table 1-15.

(5) Stack depth

16 max.

(6) Coding order

```
      ≀
CALL SQR     ; √X
GJMP ERROR   ; Jumps to the overflow processing operation.
```

(7) Processing time

Approx. 46 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC FFH, 80H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

$SQR(X) = X^{1/2}$ (Where X is positive) is calculated.

Processing procedure:

(a) Since the left side of equation in above (8) is the power square function, 0.5 is stored into the OPE.

(b) The POWER (power square function) subroutine is used for the subsequent operations.

1.3.2.12 TRACA (Polar to Orthogonal Coordinates) Subroutine

(1) Processing

The polar coordinates (R, T) with the B.F.P.ACC data (R) and the 4-byte consecutive operand data (T) are converted into orthogonal coordintes (X, Y), and coordinate X is stored into B.F.P.ACC, and coordinate Y is stored into the operand.

(2) Input conditions

Refer to the previous description of the four-rule operation input conditions because two data items are input.

(3) Output conditions

RETS: The converted coordinates (X, Y) are stored into the B.F.P.ACC operand. The CY flag is set to "0".

RET: The conversion result overflows. The CY flag is set to "1".

(4) Subroutines used

Refer to Table 1-16.

(5) Stack depth

16 max.

(6) Coding order

```
      ≀
LXI H, _____    ; First address of 4-byte data (T)
CALL TRACA      ; Coordinate conversion
GJMP ERROR      ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 65 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

```
B.F.P.ACC  81H, 80H, FFH, FFH, FFH
OPE.       80H, 7FH, FFH, FFH
```

(8) Algorithm and processing procedure

Algorithm:

The following equations are used:

$$\begin{cases} X = R \times COS(T) \\ Y = R \times SIN(T) \end{cases}$$ are used. (T in radian units.)

Figure 1-25 shows the flowchart.

Polar coordinates (γ , θ) → Orthogonal coordinates (X, Y)

$$\begin{cases} X = \gamma \cos\theta \\ Y = \gamma \sin\theta \end{cases}$$

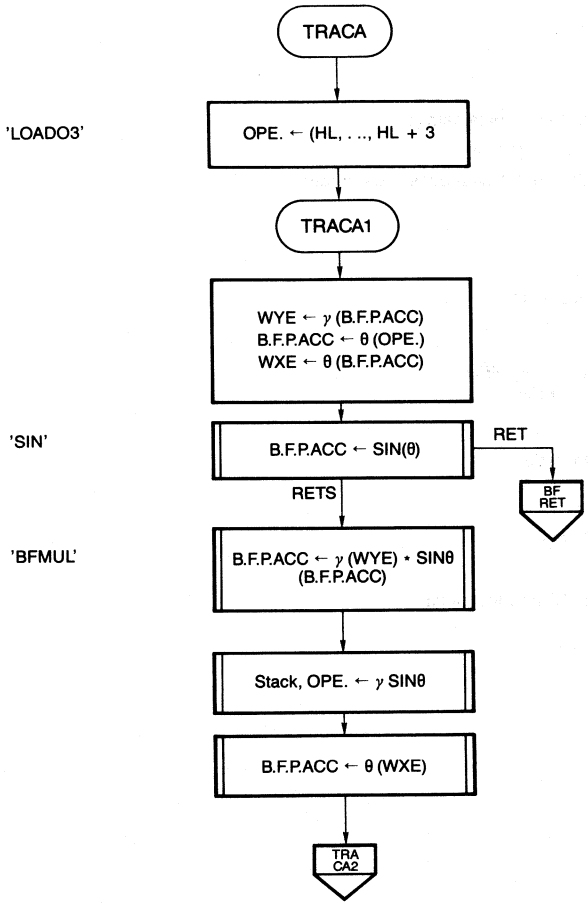
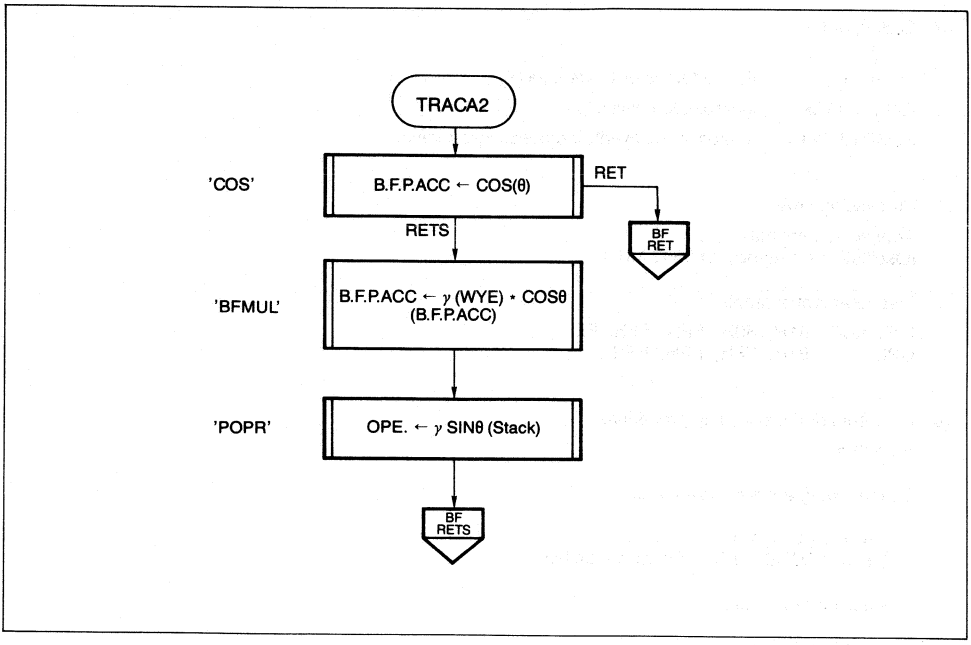


Figure 1-25 TRACA Subroutine



1.3.2.13 TRACB (Orthogonal to Polar Coordinates) Subroutine

(1) Processing

The orthogonal coordinates (X, Y) with the B.F.P.ACC data (X) and the 4-byte consecutive operand data (Y) are converted into polar coordinates (R, T), with the R result being stored into the B.F.P.ACC, and the T result stored into the operand.

(2) Input conditions

Refer to the previous description of the four-rule operation input conditions because two data items are input.

(3) Output conditions

RET: The conversion result overflows. The CY flag is set to "1".

RETS: The converted coordinates (X, Y) are stored into the B.F.P.ACC and operand respectively. The CY flag is set to "0".

(4) Subroutines used

Refer to Table 1-17.

(5) Stack depth

19 max.

(6) Coding order

```
    ≀
    LXI H, _____    ; First address of 4-byte data (Y)
    CALL TRACE      ; Coordinate conversion
    GJMP ERROR      ; Jumps to the overflow processing operation.
    ≀
```

(7) Processing time

Approx. 100 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC 81H, 80H, FFH, FFH, FFH
OPE. 81H, 7FH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

The following equation are used:

$$\begin{cases} R = \sqrt{(X^2 + Y^2)} \\ T = \text{ARCTAN}(Y/X) \quad (T \text{ in radian units.}) \end{cases}$$

Processing procedure:

(a) Y/X is calculated.

If the result overflows,
$T = -\pi/2$ when $Y < 0$.
$T = \pi/2$ when $Y > 0$.

(b) ARCTAN (Y/X) is calculated.

If $X < 0, Y < 0, T = \text{ARCTAM}(Y/X) - \pi$.
If $X < 0, Y > 0, T = \text{ARCTAN}(Y/X) + \pi$.

(c) $X^2 + Y^2$ is calculated.

(d) $R = \text{SQR}(X^2 + Y^2)$.

Figure 1-26 shows the flowchart.

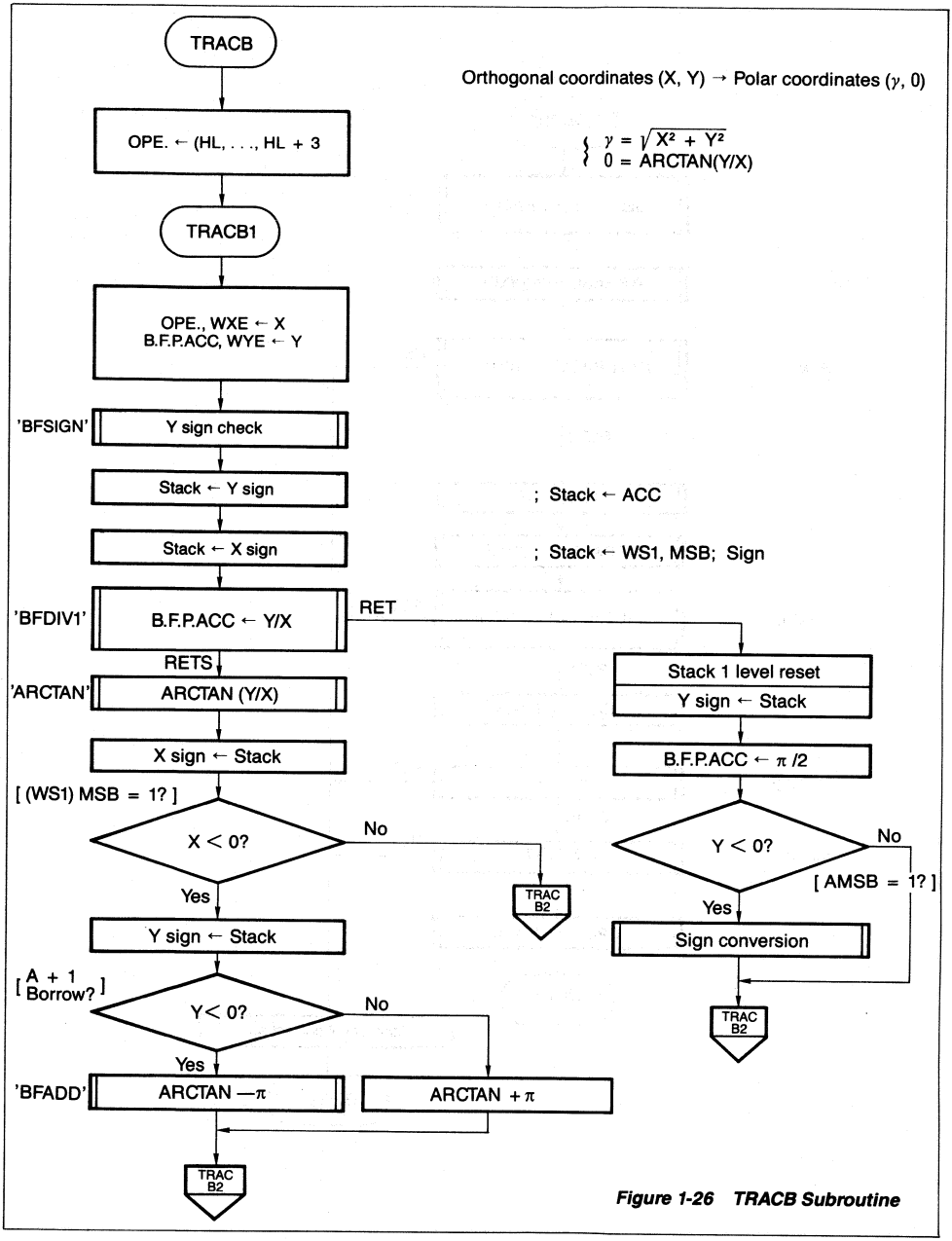
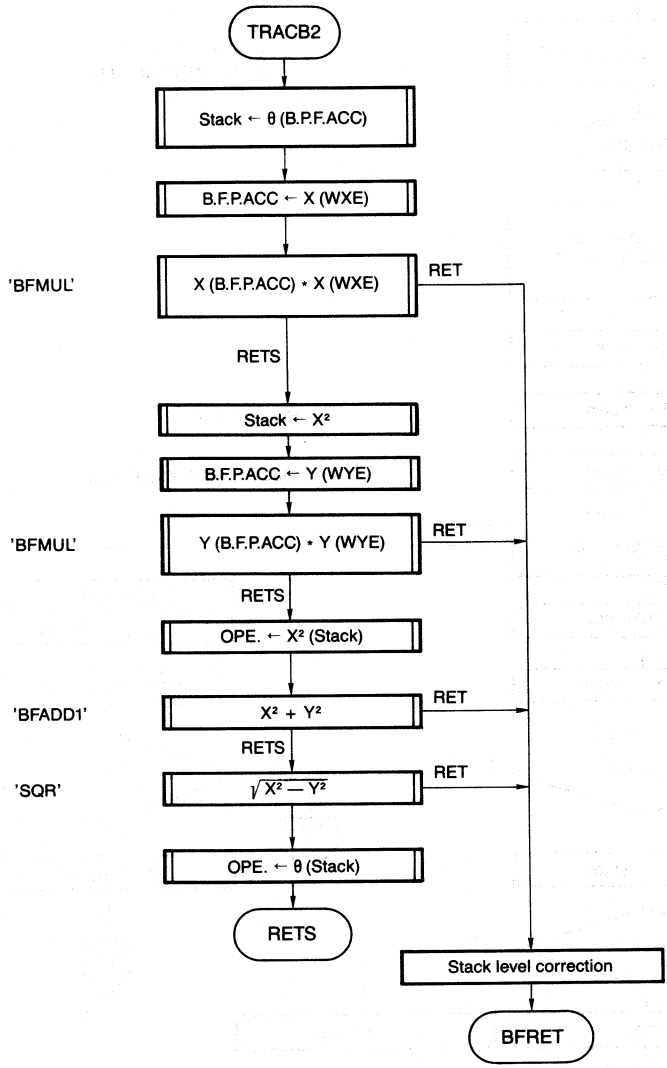


Figure 1-26 TRACB Subroutine


TRACB2
Stack ← θ (B.F.P.ACC)
B.F.P.ACC ← X (WXE)
'BFMUL'
X (B.F.P.ACC) * X (WXE)
RET
RETS
Stack ← X²
B.F.P.ACC ← Y (WYE)
'BFMUL'
Y (B.F.P.ACC) * Y (WYE)
RET
RETS
OPE. ← X² (Stack)
'BFADD1'
X² + Y²
RET
RETS
'SQR'
√X² — Y²
RET
OPE. ← θ (Stack)
RETS
Stack level correction
BFRET

1.3.3 Conversion Subroutine for Character Constant to Binary Floating Point Number

The following conversion subroutines are available:

(1) BFINP (character constant to binary floating point number) subroutine.

(2) BFOUT (binary floating point number to character constant) subroutine.

1.3.3.1 BFINP (Character Constant to Binary Floating Point Number) Subroutine

(1) Processing

The character constant addressed by the HL pair register data is converted into a binary floating point number and the result is stored into the B.F.P.ACC.

(2) Input conditions

The first address of the character constant is stored into the HL register.

(3) Output conditions

RET: The character constant value is too large. The flag is set to "1".

RETS: The character constant has been converted correctly and the conversion result stored into B.F.P.ACC. The CY flag is set to "0".
If the conversion result is 0, the Z flag is set to "1" and in all other cases the Z flag is set to "0".

(4) Subroutines used

Refer to Table 1-18.

(5) Stack depth

5 max.

(6) Coding order

```
      ≀
LXI H, ____    ; First address of character constant
CALL BFINP     ; Conversion
GJMP ERROR     ; Jumps to the overflow processing operation.
      ≀
```

(7) Processing time

Approx. 388 ms max.
(Oscillation frequency: 11.0592 MHz)

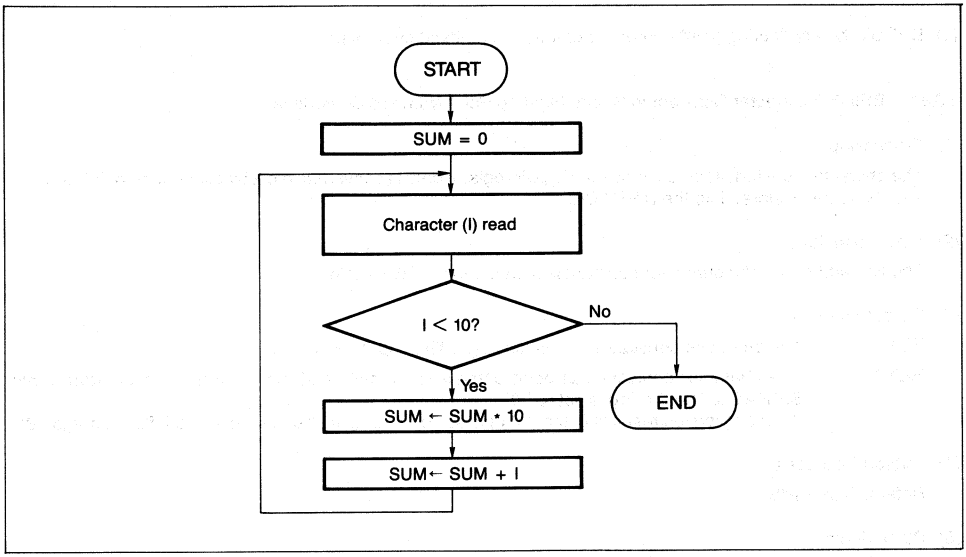
Measurement example:

2.9 . . . . . 9E — 38
└──────────┘
37 bits

(8) Algorithm and processing procedure

Algorithm:

The flow of decimal to binary conversion is shown below:



Because the above values are calculated by the fixed-point part operation, only the adjustment of the decimal point is necessary.

ADDRS	TMP1	TMP2	TMP3	VAE	VA1	VA2	VA3

ADDRS: Address at which the character constant is stored.

TMP1: Position of decimal point for character constant
After the decimal point flag (TMP2) has been set, 1 is subtracted from each digit.

TMP2: Decimal point flag for character constant
TMP2 = 0: Decimal point is not indicated.
TMP2 ≠ 0: Decimal point is indicated.

TMP3: Sign flag
TMP3 = 80H : Positive
TMP3 = 00H : Negative

WXE, WX1, WX2, WX23;
The operation result is temporarily saved.

Processing procedure:

(a) The fixed-point part sign is set for TMP3 from the first address of the character constant. If no sign is indicated, the flag is positive; so the positive sign flag is initialized for TMP3.

(b) The operation proceeds to (e) below for exponential sign (E), or the decimal point flag (TMP2) is set for the character (.).

(c) The B.F.P.ACC value is multiplied by 10 and the result is saved into WXE onwards.

(d) Characters (0 to 9) are converted into B.F.P.ACC values.

If the decimal point flag (TMP2 has previously been set, the decimal point position (TMP1) is subtracted by 1 from each digit. In other words, (12 x 10 + 3) is calculated if a numerical value is "12.3", and 0FFH(—1) is set for TMP1.

(e) "WXE (previous result) + B.F.P.ACC value" is stored into the B.F.P.ACC.

(f) The exponent is converted into a binary number.

If the converted value is negative, it is represented as a two's complement.

(g) "Exponent obtained in (f) above + TMP1" is calculated and the decimal point position is set.

(h) If the result of (g) is negative, the B.F.P.ACC value is repeatedly divided by 10 until the result of (g) becomes 0.

If the result of (g) is positive, the B.F.P.ACC value is repeatedly multiplied by 10 until the result of (g) becomes 0.

Example:

5 digits

0. 1 2 3 4 5 E4

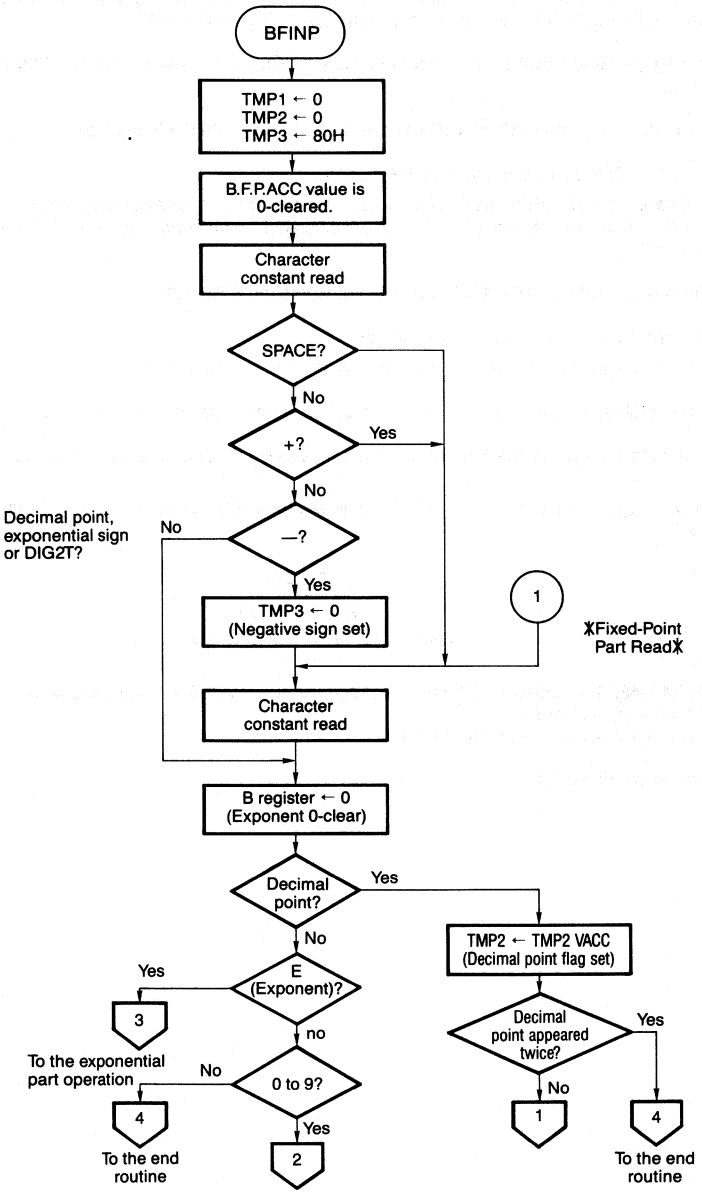
$(1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10 + 5)$

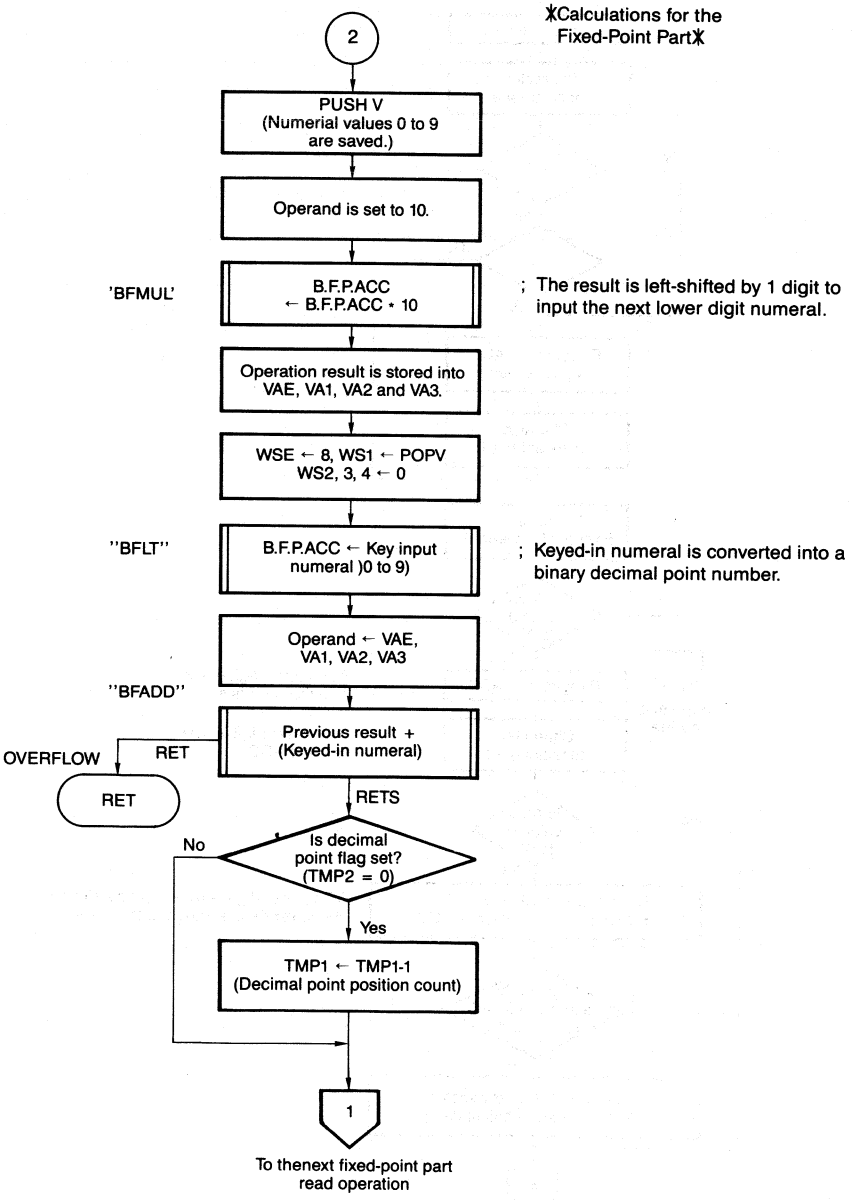
is calculated.

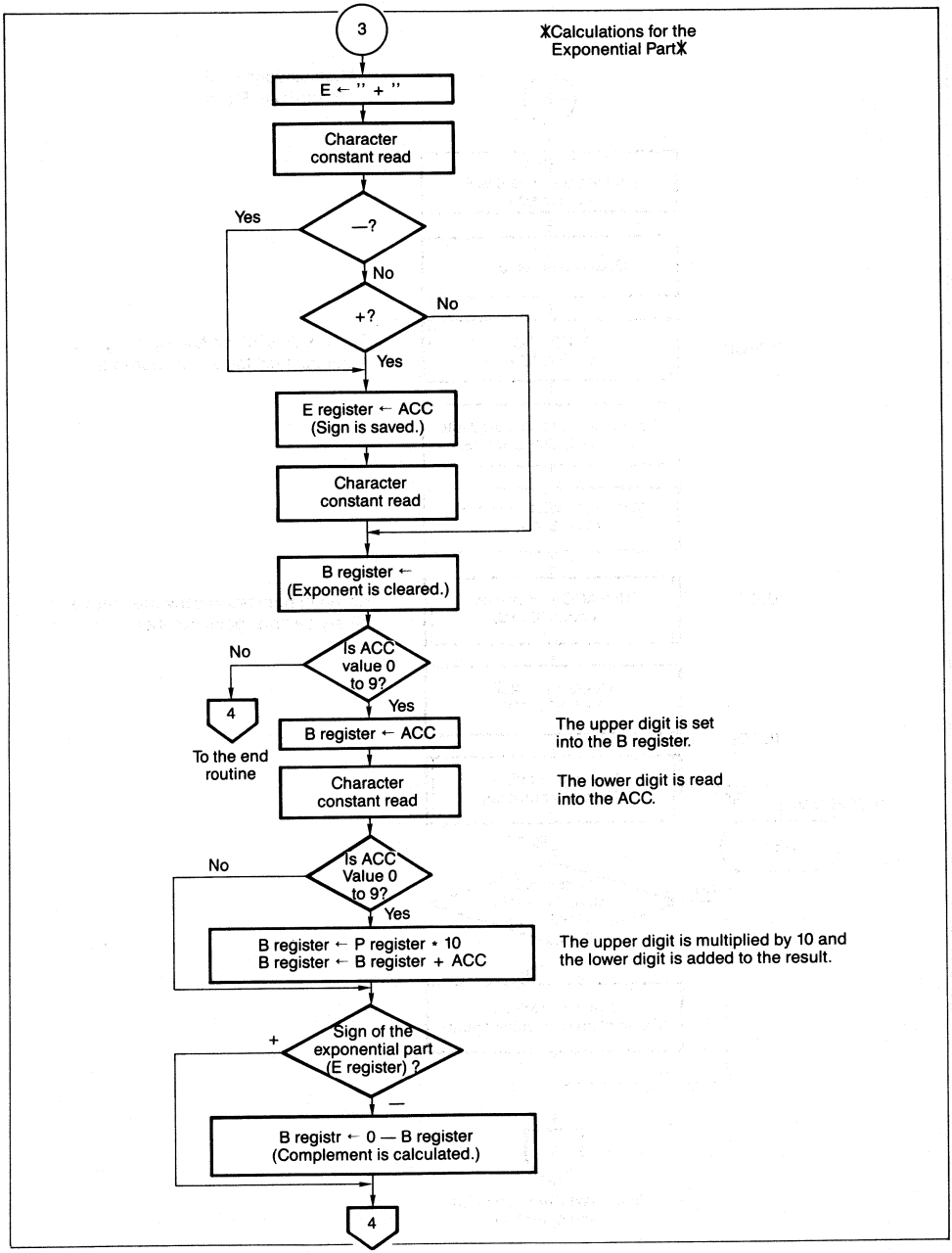
Because $0.12345E4 = 12345 \times 10^{-1}$, the calculated value is divided by 10 and the result remains in the B.F.P.ACC as a numerical value.
In this case, 0FFH(4—5) is preset for TMP1.

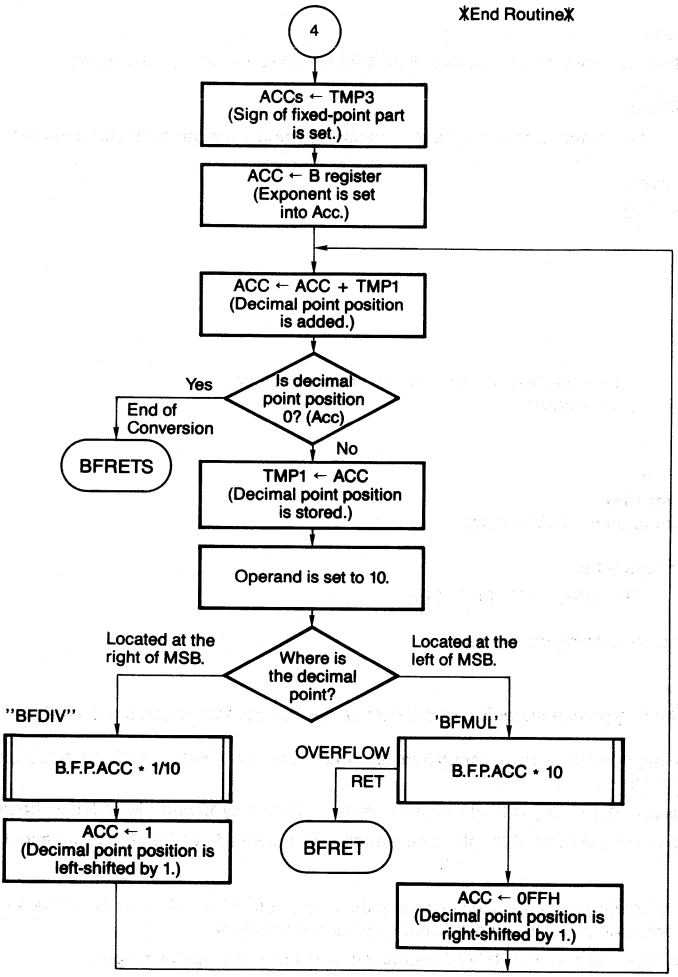
Figure 1-27 shows the flowchart.

Figure 1-27 BFINP Subroutine




Calculations for the Fixed-Point Part
2
PUSH V
(Numerial values 0 to 9 are saved.)
Operand is set to 10.
'BFMUL'
B.F.P.ACC ← B.F.P.ACC * 10
; The result is left-shifted by 1 digit to input the next lower digit numeral.
Operation result is stored into VAE, VA1, VA2 and VA3.
WSE ← 8, WS1 ← POPV
WS2, 3, 4 ← 0
"BFLT"
B.F.P.ACC ← Key input numeral)0 to 9)
; Keyed-in numeral is converted into a binary decimal point number.
Operand ← VAE, VA1, VA2, VA3
"BFADD"
Previous result + (Keyed-in numeral)
OVERFLOW
RET
RET
RETS
No
Is decimal point flag set? (TMP2 = 0)
Yes
TMP1 ← TMP1-1
(Decimal point position count)
1
To thenext fixed-point part read operation


3
※Calculations for the Exponential Part※
E ← '' + ''
Character constant read
Yes
—?
No
+?
No
Yes
E register ← ACC (Sign is saved.)
Character constant read
B register ← (Exponent is cleared.)
No
Is ACC value 0 to 9?
Yes
4
To the end routine
B register ← ACC
The upper digit is set into the B register.
Character constant read
The lower digit is read into the ACC.
No
Is ACC Value 0 to 9?
Yes
B register ← P register * 10
B register ← B register + ACC
The upper digit is multiplied by 10 and the lower digit is added to the result.
+
Sign of the exponential part (E register) ?
—
B registr ← 0 — B register (Complement is calculated.)
4


4
XEnd RoutineX
ACCs ← TMP3
(Sign of fixed-point part is set.)
ACC ← B register
(Exponent is set into Acc.)
ACC ← ACC + TMP1
(Decimal point position is added.)
Is decimal point position 0? (Acc)
Yes
End of Conversion
BFRETS
No
TMP1 ← ACC
(Decimal point position is stored.)
Operand is set to 10.
Where is the decimal point?
Located at the right of MSB.
Located at the left of MSB.
"BFDIV"
B.F.P.ACC • 1/10
ACC ← 1
(Decimal point position is left-shifted by 1.)
'BFMUL'
OVERFLOW
RET
BFRET
B.F.P.ACC • 10
ACC ← 0FFH
(Decimal point position is right-shifted by 1.)

1.3.3.2 BFOUT (Binary Floating Point Number to Character Constant) Subroutine

(1) Processing

The B.F.P.ACC value (binary floating point number) is converted into a character constant and is stored at the memory location addressed ba the HL pair register data. The character constant consists of 13 characters.

(2) Input conditions

The first address at which the character constant is stored into the HL register is set.

(3) Output conditions

RET: The binary number has been correctly converted into the character constant.

(4) Subroutines used

Refer to Table 1-19.

(5) Stack depth

6 max.

(6) Coding order

```
      ≀
LXI H,          ; First address at which character constant is stored.
CALL BFOUT      ; Conversion
      ≀
```

(7) Processing time

Approx. 160 ms max.
(Oscillation frequency: 11.0592 MHz)

Measurement example:

B.F.P.ACC FFH, 80H, FFH, FFH, FFH

(8) Algorithm and processing procedure

Algorithm:

The binary floating point number is converted into a decimal number according to the following concept:

(a) The floating point is set in the range from 0.1 to 1.0 in decimal notation (with the ACCE exponent set in the range from 2^{-2} to 2^{1}),

When exponent (ACCE) $\geqq 2^{1}$, the vinary number is divided by 10 and 1 is added to the decimal exponent.

When exponent (ACCE) $< 2^{-2}$, the binary number is multiplied by 10 and 1 is subtracted from the decimal exponent.

(b) Since the floating point obtained in (a) is in the range from 0.1 to 1.0, one digit of the decimal integer part can be obtained by multiplying the floating point position by 10.

The variables used for the BFOUT subroutine have the following meanings:

TMP1: The number of digits for the left side of tghe decimal point is set.

TMP2: The decimal exponent is set in binary notation. The negative part is represented by a complement.

When the floating point is set in the range from 2^{-2} to 2^{1}, 1 is added for division by 10 and 1 is subtracted for multiplication by 10. After that, the decimal point position is obtained.

If TMP2 ≧ 8, the exponential part is used, as shown in example (1) below, because a decimal number can be represented up to seven characters only (excluding the exponential part).

Example (1):

If a decimal number consists of more than 7 characters:

98765432.(incorrect) 9.876543E7(correct)

8 Digits 1 6

If TMP2 < 8, a decimal point is within the specified number of characters. Thus, the decimal exponential part is 0.

TMP3: The number of digits ot the right of the decimal point is set. When 0FFH is set, the operation ends.

WXE, WX1, WX2, WX3:
Areas where the B.F.P.ACC value is saved.

Processing procedure:

(a) The B.F.P.AC C value is saved in four bytes from WXE onwards so that it will not be destroyed.

(b) The B.F.P.ACC value is set in the range from 0.1 to 1.0. A decimal exponent is set for TMP2.

(c) TMP1, TMP2 and TMP3 are set.

- When TMP2 ≧ 8,
  As shown in example (1) above, the exponent (TMP2) consists of digits obtained by subtracting 1 from TMP2. TMP1 consist of 1 digit and TMP3 consists of 6 digits.
- When TMP2 < 8,
  The decimal point is within the specified number of characters as shown in example (2) below. Thus, the decimal point position is set at (7—TMP2) for TMP3 and (TMP2—1) for TMP1. The exponent is 0 for TMP2.

Example (2):

If a decimal point is within the specified number of characters:

3.141593 . . . . (Exponential part is not included.)

—1234567

(d) The B.F.P.ACC value is multiplied by 10.

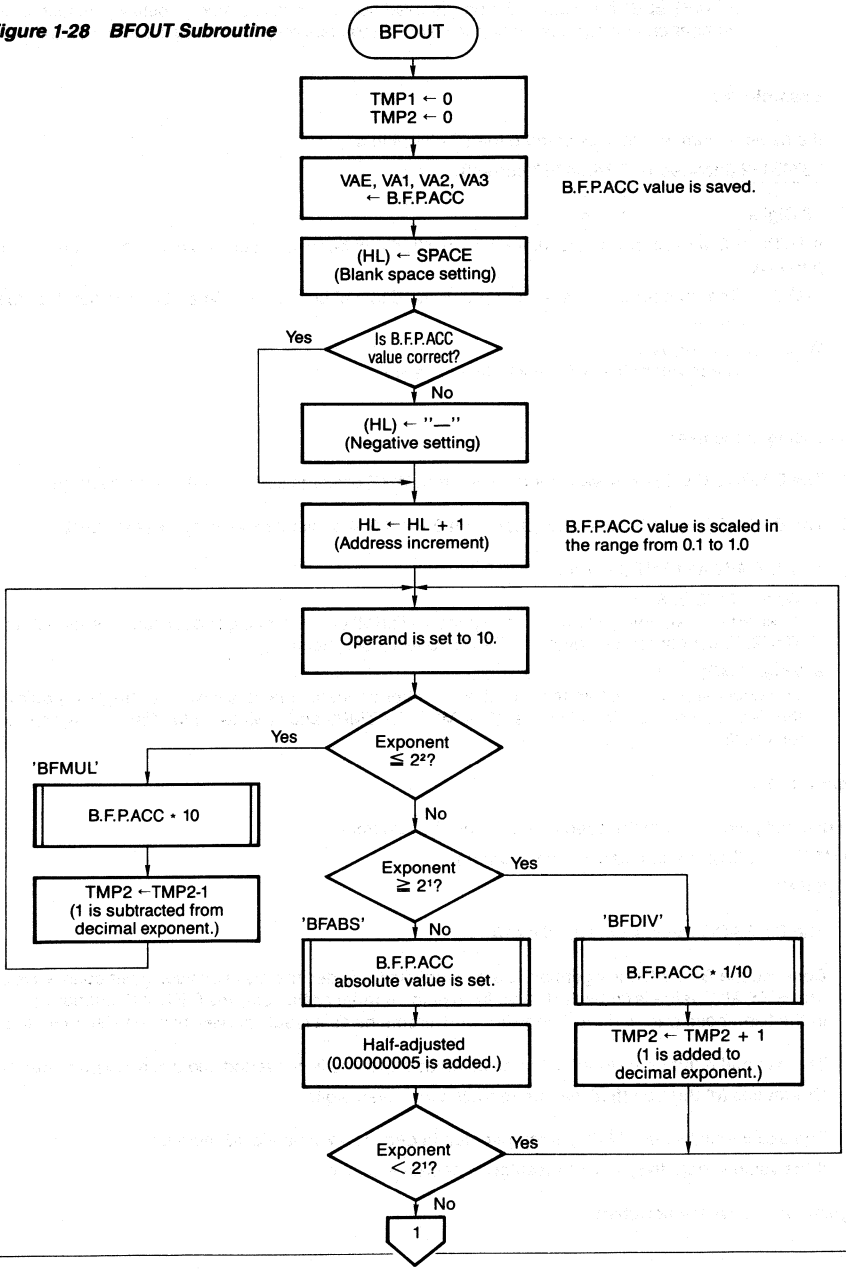
(e) Because the B.F.P.ACC integer part is a decimal digit, the decimal point position (shift counter) is set to 8. The B.F.P.ACC value is converted into a fixed point number so that only the B.F.P.ACC integer part remains in the fixed-point part and the A register, which remains as a result, serves as a one decimal digit.

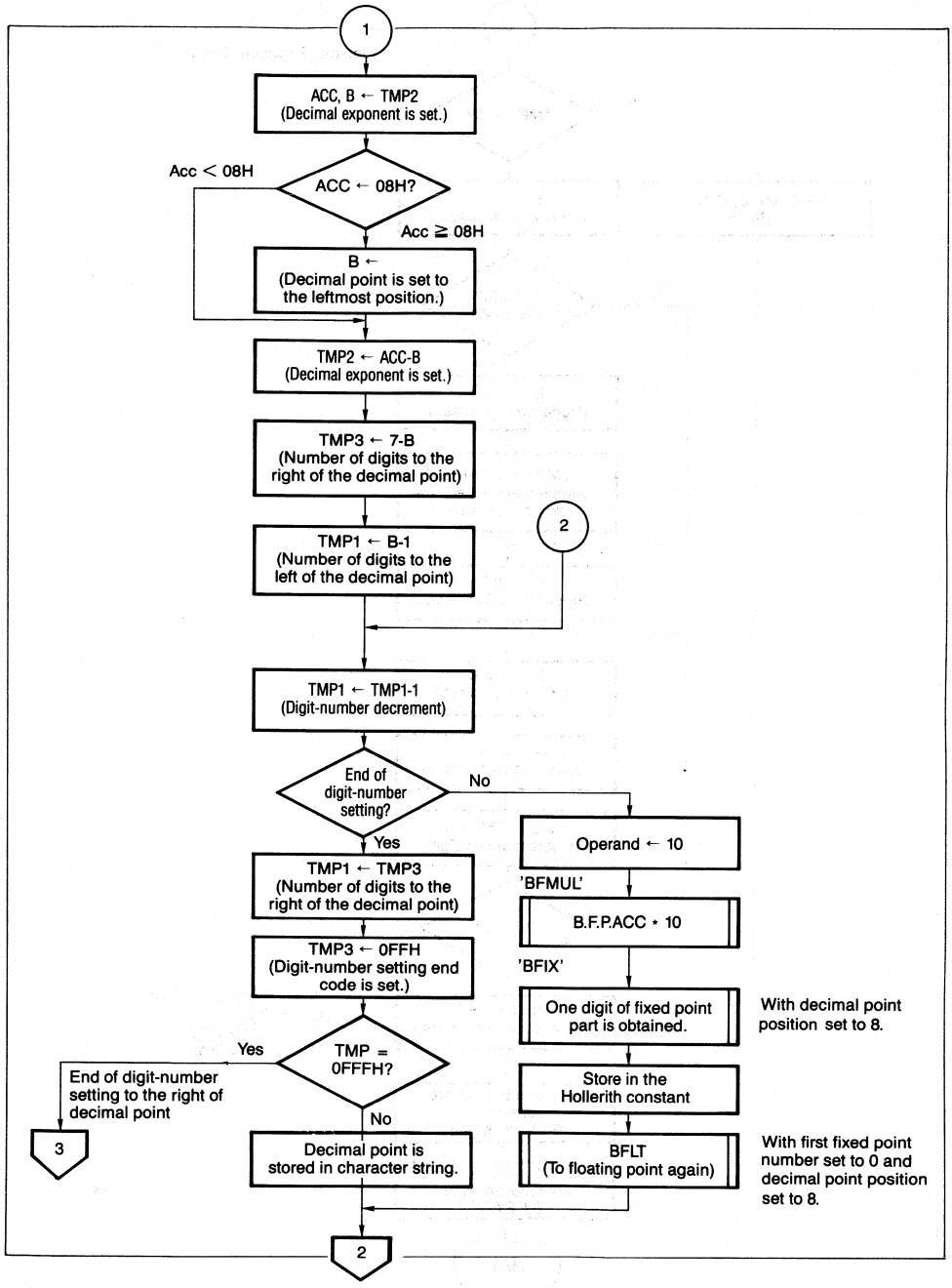
(f) The decimal number is stored at the selected address and is converted into a floating point number.
Operations (d) through (f) above are repeated for seven digits.

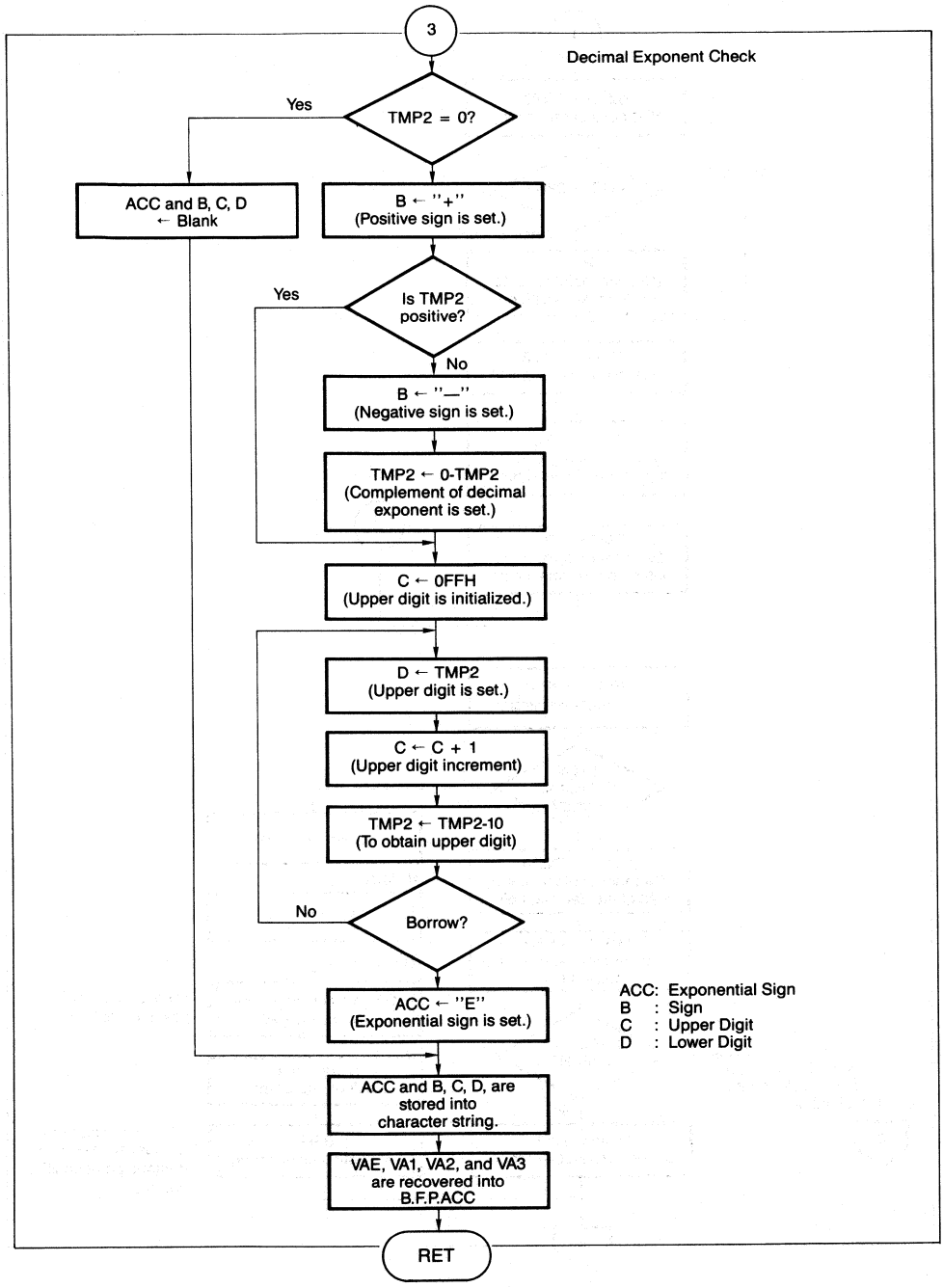
(g) The exponential value (TMP2) is represented in decimal exponential (E) notation.
If the value is negative, it is represented in negative notation.

Figure 1-28 shows the flowchart.

Figure 1-28 BFOUT Subroutine




1
ACC, B ← TMP2
(Decimal exponent is set.)
Acc < 08H
ACC ← 08H?
Acc ≧ 08H
B ←
(Decimal point is set to the leftmost position.)
TMP2 ← ACC-B
(Decimal exponent is set.)
TMP3 ← 7-B
(Number of digits to the right of the decimal point)
2
TMP1 ← B-1
(Number of digits to the left of the decimal point)
TMP1 ← TMP1-1
(Digit-number decrement)
End of digit-number setting?
No
Yes
TMP1 ← TMP3
(Number of digits to the right of the decimal point)
TMP3 ← 0FFH
(Digit-number setting end code is set.)
TMP = 0FFFH?
Yes
End of digit-number setting to the right of decimal point
3
No
Decimal point is stored in character string.
Operand ← 10
'BFMUL'
B.F.P.ACC * 10
'BFIX'
One digit of fixed point part is obtained.
With decimal point position set to 8.
Store in the Hollerith constant
BFLT
(To floating point again)
With first fixed point number set to 0 and decimal point position set to 8.
2


3
Decimal Exponent Check
TMP2 = 0?
Yes
ACC and B, C, D ← Blank
B ← "+" (Positive sign is set.)
Is TMP2 positive?
Yes
No
B ← "—" (Negative sign is set.)
TMP2 ← 0-TMP2 (Complement of decimal exponent is set.)
C ← 0FFH (Upper digit is initialized.)
D ← TMP2 (Upper digit is set.)
C ← C + 1 (Upper digit increment)
TMP2 ← TMP2-10 (To obtain upper digit)
Borrow?
No
ACC ← "E" (Exponential sign is set.)
ACC and B, C, D, are stored into character string.
VAE, VA1, VA2, and VA3 are recovered into B.F.P.ACC
RET
ACC: Exponential Sign
B : Sign
C : Upper Digit
D : Lower Digit

1.4 Configurations of Subroutines used for each Operation

Tables 4-1 through 4-17 list the configurations of operation subroutines and the number of bytes used (decimal).

In each table, the subroutine names used for operation subroutines are listed in the "level 1" column and the subroutines used at level 1 are listed in the "level 2" column. Similarly, the subroutine names up to the lowest level are listed.

Within parentheses in the table, the number of bytes necessary for use with a single subroutine is indicated.

Table 1-3 BFADD and BFSUB Subroutine Configurations

Subroutine Name \ Level	1	2
BFADD (278) BFSUB (286)	BFRSH (27)	
	BFRSH4 (13)	
	BFCOM1 (21)	
	BFNOR (27)	
	BFSTR1 (14)	
	BFROND (39)	BFRNDR (15)
		BFSTR2 (9)

Table 1-4 BFMUL Subroutine Configurations

Subroutine Name \ Level	1	2	3	4
BFMUL (266)	BFRD (48)			
	BFMX (99)	BFMX3 (66)	MULT (29)	MULT2 (12)
			BFRSH0 (27)	
	BFNOR (27)			
	BFROND (39)	BFRNDR (15)		
		BFSTR2 (9)		

Table 1-5 BFDIV Subroutine Configurations

Subroutine Name \ Level	1	2
BFDIV (281)	BFRD (48)	
	BFDX (136)	BFRSH4 (13)
		BFLSH1 (9)
	BFROND (39)	BFRNDR (15)
		BFSTR2 (9)

Table 1-6 SIN Subroutine Configurations

Subroutine Name \ Level	1	2	3	4
SIN (1047)	BFSIGN (15)			
	BFADD (263)	Same as with BFADD subroutine		
	BFSUB (270)	Same as with BFSUB subroutine		
	BFDIV (266)	Same as with BFDIV subroutine		
	BFCOMP (6)			
	STORE0 (21)			
	PUSHF (39)			
	POPR (12)			
	BFINT (130)	BFLT (14)		
		BFIX2 (108)	BFCOM1 (21)	
			BFRSH (27)	
	BFCOMX (10)	BFCOMP (6)		
	POLYN1 (614)			

Table 1-7 COS Subroutine Configurations

Subroutine Name \ Level	1	
COS (1076)	BFABS (4)	
	STORE1 (21)	
	BFADD (263)	Same as with BFADD subroutine
	LOADA1 (25)	
	BFSUB (270)	Same as with BFSUB subroutine
	BFCOMP (6)	
	SIN (1047)	Same as with SIN subroutine

Table 1-8 TAN Subroutine Configurations

Subroutine Name \ Level	1	
TAN (1103)	PUSHF (39)	
	POPR (12)	
	STORE1 (21)	
	LOADA0 (27)	
	LOADO1 (14)	
	SIN (1047)	Same as with SIN subroutine
	COS (1076)	Same as with COS subroutine
	BFDIV (266)	Same as with BFDIV subroutine

Table 1-9 LNX Subroutine Configurations

Subroutine Name \ Level	1	
LNX (880)	BFSIGN (15)	
	BFADD (263)	Same as with BFADD subroutine
	BFSUB (270)	Same as with BFSUB subroutine
	BFMUL (251)	Same as with BFMUL subroutine
	BFDIV (266)	Same as with BFDIV subroutine
	BFCOMP (6)	
	BFLT (14)	
	POLYN3 (612)	
	STORE0 (21)	
	LOADA (23)	
	PUSHF (39)	

Table 1-10 LOG Subroutine Configurations

Subroutine Name \ Level	1	
LOG (889)	LNX (880)	Same as with LNX subroutine
	BFDIV (266)	Same as with BFDIV subroutine

Table 1-11 EXP Subroutine Configurations

Subroutine Name \ Level	1	2	3
EXP (927)	BFSIGN (15)		
	BFABS (4)		
	PUSHF (39)		
	POPR (12)		
	STORE0 (21)		
	LOADA (23)		
	BFSUB (270)	Same as with BFSUB subroutine	
	BFMUL (251)	Same as with BFMUL subroutine	
	BFDIV (266)	Same as with BFDIV subroutine	
	BFINT0 (429)	BFSIGN (15)	
		BFSUB (270)	
		BELT (14)	
		BFIX2 (108)	BFCOM1 (21)
			BFRSG (27)
	POLY (566)		

Table 1-12 ARCTAN Subroutine Configurations

Subroutine Name \ Level	1	2	
ARCTAN (954)	STORE0 (21)		
	LOADA (23)		
	LOADO3 (14)		
	BFSIGN (15)		
	BFCOMP (6)		
	BFDIV (266)	Same as with BFDIV subroutine	
	POLYN2 (608)		
	BFSUB (270)	Same as with BFSUB subroutine	
	BFCOMX (10)	BFCOMP (6)	

Table 1-13 ARCSIN Subroutine Configurations

Subroutine Name \ Level	1	2	
ARCSIN (1747)	PUSHF (39)		
	POPR (12)		
	STORE1 (21)		
	LOADA0 (27)		
	LOADA (27)		
	LOADO1 (14)		
	BFSIGN (15)		
	ROUTE (1620)	BFMUL (251)	Same as with BFMUL subroutine
		BFSUB (270)	Same as with BFSUB subroutine
		SQR (1075)	Same as with SQR subroutine
		BFCOMP (6)	
	BFDIV (266)	Same as with BFDIV subroutine	
	BFCOMP (6)		
	ARCTAN (954)	Same as with ARCTAN subroutine	

Table 1-14 ARCCOS Subroutine Configurations

Subroutine Name \ Level	1	2	
ARCCOS (1762)	BFSIGN (15)		
	PUSHF (39)		
	ROUTE (1620)	BFMUL (251)	Same as with BFMUL subroutine
		BFSUB (270)	Same as with BFSUB subroutine
		SQR (1075)	Same as with SQR subroutine
		BFCOMP (6)	
	POPR (12)		
	BFDIV (266)	Same as with BFDIV subroutine	
	ARCTAN (954)	Same as with ARCTAN subroutine	
	BFADD (263)	Same as with BFADD subroutine	
	LOADA (27)		
	BFMUL (251)	Same as with BFMUL subroutine	
	BFSUB (270)	Same as with BFSUB subroutine	
	BFCOMP (6)		
	SQR (1075)	Same as with SQR subroutine	

Table 1-15 SQR and POWER Subroutine Configurations

Subroutine Name \ Level	1	2	3	
SQR POWER (1075)	PUSHR (12)			
	POPR (12)			
	STORE0 (21)			
	STORE1 (21)			
	LOADA0 (24)			
	LOADO1 (11)			
	BFABS (4)			
	BFSIGN (15)			
	BFINT (130)	BFLT (14)		
		BFIX2 (108)	BFCOM1 (21)	
			BFRSH (27)	
	BFSUB (270	Same as with BFSUB subroutine		
	LNX (880)	Same as with LNX subroutine		
	EXP (927)	Same as with EXP subroutine		

Table 1-16 TRACA Subroutine Configurations

Subroutine Name \ Level	1	
TRACA (1120)	PUSHF (39)	
	POPR (12)	
	LOADA0 (27)	
	LOADA1 (27)	
	STORE1 (21)	
	STORE2 (21)	
	SIN (1047)	Same as with SIN subroutine
	COS (1076)	Same as with COS subroutine
	BFMUL (251)	Same as with BFMUL subroutine

Table 1-17 TRACB Subroutine Configurations

Subroutine Name \ Level	1	
TRACB (1186)	STORE1 (21)	
	STORE2 (21)	
	LOADA0 (27)	
	LOADA1 (27)	
	LOADA2 (27)	
	LOADA (21)	
	LOADO1 (14)	
	PUSHF (39)	
	POPR (12)	
	BFSIGN (15)	
	BFCOMP (6)	
	BFDIV (266)	Same as with BFDIV subroutine
	BFMUL (251)	Same as with BFMUL subroutine
	BFADD (263)	Same as with BFADD subroutine
	ARCTAN (954)	Same as with ARCTAN subroutine
	SQR (1075)	Same as with SQR subroutine

Table 1-18 BFINP Subroutine Configurations

Subroutine Name \ Level	1	
BFINP (848)	STORE1 (21)	
	CHARD (9)	
	ADADJ (9)	
	BFADD (263)	Same as with BFADD subroutine
	BFMUL (251)	Same as with BFMUL subroutine
	BFDIV (266)	Same as with BFDIV subroutine
	BFLT (14)	

Table 1-19 BFOUT Subroutine Configurations

Subroutine Name \ Level	1	2
BFOUT (940)	STORE1 (21)	
	LOADA0 (27)	
	LOADA1 (27)	
	PUSHF (39)	
	POPR (12)	
	BFADD (263)	Same as with BFADD subroutine
	BFSUB (270)	Same as with BFSUB subroutine
	BFMUL (251)	Same as with BFMUL subroutine
	BFDIV (266)	Same as with BFDIV subroutine
	BFABS (4)	
	BFLT (14)	
	BFIX0 (108)	BFCOM1 (21)
		BFRSH (27)

1.5 Common Subroutines

Common subroutines used for the operation subroutines are described in Table 1-20.

The number of bytes (decimal) used for each subroutine is indicated in parentheses below each subroutine name.

Table 1-20 List of Common Subroutines

Subroutine Name	Processing
POLYN1 POLYN2 POLYN3 (612)	The result of operation with the operand square substituted in polynomial (5.2) multiplied by the original operand (refer to equation 5.1). The result is stored into the B.F.P.ACC. The table address at which the number of polynomial terms and each term coefficient have been stored is set into the HL pair register. Polynomial = $X(C_0 + C_1X^2 + C_2X^4 + C_3X^6 \ldots) \ldots$ (5.1)
POLY (566)	The operand polynomial (5.2) is calculted and the result is stored into the B.F.P.ACC. The table address at which the number of polynomial terms and each term coefficient have been stored is set into the HL pair register. Polynomial = $C_0 + C_1X + C_2X^2 + \ldots$ (5.2)
BFNOR (27)	The operand fixed point parts (WS1, WS2, WS3 and WS4) are normalized. Because the normalization causes them to be left-shifted, the exponential part (ACCE) is also adjusted.
BFROND (39)	The operand fixed point parts (WS1, WS2, WS3 and WS4) are half-adjusted with respect to the MSB of WS4. The results (WS1, WS2 and WS3) are transferred to the B.F.P.ACC fixed point parts (ACC1, ACC2 and ACC3).
BFCOM1 (21)	The operand fixed point parts (WS1, WS2, WS3 and WS4) are represented as two's complements and the B.F.P.ACC sign (ACCS) is reversed.
BFCOMP (6)	The B.F.P.ACC sign (ACCS) is reversed.
BFSIGN (15)	The B.F.P.ACC sign is checked and the resulting information is input into the accumulator and FSN. FSN, accumulator = { 0 ; B.F.P.ACC = 0 1 ; B.F.P.ACC = 0 (Positive) 0FFH ; B.F.P.ACC = 0 (Negative)
BFSTR1 (14)	The operand is transferred to the 5-byte consecutive area indicated by the HL pair register. (HL, HL + 1, HL + 2, HL + 3, HL + 4) ← Operand
BFSTRS (9)	The operand fixed point parts (WS1, WS2 and WS3) are transferred to the 3-byte consecutive area indicated by the HL pair register. (HL, HL + 1, HL + 2) ← Fixed point parts (WS1, WS2, WS3)
BFABS (4)	The B.F.P.ACC absolute value is set.
BFRSH0 BFRSH (27)	The fixed point parts (WS1, WS2 and WS3) are right-shifted by an accumulator serving as a shift counter and are stored into (WS1, WS2, WS3 and WS4). "BFRSH0" is set to 8 for the accumulator.

Table 1-20 List of Common Subroutines (cont'd)

Subroutine Name	Processing
BFRSH2 BFRSH4 (13)	The fixed point parts (WS1, WS2 and WS3) including the CY flag are right-shifted by 1 bit and the results are stored into (WS1, WS2, WS3 and WS4).
BFLSH1 (9)	A counter is set into the B register from the area indicated by the HL pair register and each 1 byte, including the CY flag, is left-shifted by 1 bit to the upper bytes.
BFIX0 BFIX2 (108)	The binary floating point numbers (ACCE, ACCS and ACC1 to ACC3) are converted into binary fixed point numbers (WSE and WS1 to WS4). The fixed point position is set into the accumulator. Since the fixed point part has 32 bits, the decimal point position can be set up to 32. Conversion example: If the fixed point position (accumulator) is set to 8 when data (80H, 80H, 82H, 00H, 00H; 0.5078125) is stored in the B.F.P.ACC, and 0.10000010.0 ◄——► 0;0.5078125. 8 16 bits Because the decimal point is right-shifted by 8-bit as shown below, the fixed point part must be right-shifted by 8 bits. If the B.F.P.ACC value is negative, it is represented as a two's complement. If the B.F.P.ACC value is 0 (WS1 to WS4), it is 0-cleared.
BFLT (14)	The binary fixed point numbers (WSE and WS1 to WS4) are converted into binary floating point numbers (ACCE, ACCS and ACC1 to ACC3). The exponential part (ACCE) is set by adding a bias of 80H to WSE. The sign part (ACCS) is positive, and the fixed point parts (ACC1 to ACC3) are normalized and half-adjusted.
BFINTO (429) BFINT (130)	The integer part of the B.F.P.ACC value is stored into the B.F.P.ACC. This subroutine processing is described below: (a) If the B.F.P.ACC value is negative, 1 is subtracted because the X register part is I when $I \leqq X < I + 1$ (where I = integer). (b) The fixed point position is set to 32 so that the decimal point is positioned to the right of the fixed point part, the binary floating point numbers (ACCE, ACCS and ACC1 to ACC3) are converted into binary fixed point numbers (WSE and WS1 to WS3) and only the integer part is left in the fixed point part. (c) The binary fixed point numbers are converted into B.F.P.ACC values.
BFRD (48)	An exponential part operation (addition) is carried out for multiply/division with the B.F.P.ACC operand. The resultant sign is set from each sign and the fixed point part of the operand becomes unsigned.
STORE0 STORE1 STORE2 (21)	The B.F.P.ACC value is transferred to the 4-byte consecutive memory indicated by the HL pair register.
LOADA0 LOADA1 LOADA2 (27) LOADA (21)	The 4-byte consecutive memory data is transferred to the B.F.P.ACC.

Table 1-20 List of Common Subroutines (cont'd)

Subroutine Name	Processing
LOADO1 LOADO2 (14) LOADO3 (13)	The 4-byte consecutive memory data indicated by the HL pair register is transferred to the operand.
PUSHF (39)	The B.F.P.ACC value is first transferred to the operand and is then stored into the stack.
PUSHR (12)	The operand is stored into the stack.
POPR (12)	The 4-byte consecutive data in the stack is recovered into the operand.

Chapter 2 Applications

This chapter describes a program in which an "equation" is input by the μCOM-87AD functional package from a console connected with the μCOM-87AD serial interface and the result is displayed on the console.

A description is also made of a program for an application of the μCOM-87AD functional package in which an "equation" is input from a console connected to the μCOM-87AD with a serial interface and the result is displayed on the console.

2.1 System Configuration

Figure 2-1 shows a system configuration for data transfer by the μCOM-87AD connected to the console with a serial interface in the asynchronous mode.

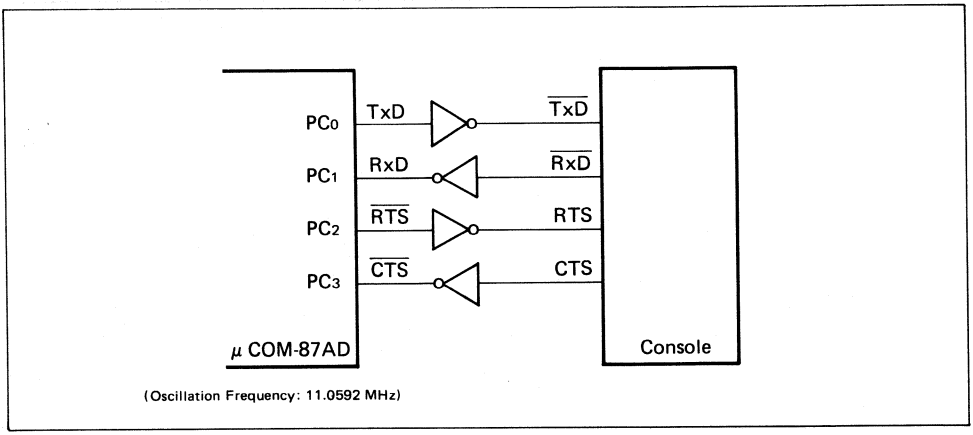


Figure 2-1 System Configuration

The system format is:

(1) Start bit : 1 bit
(2) Data length : 8 bits
(3) Parity : None
(4) Stop bit : 2 bits
(5) Baud rate : 4800 bps

Four lines are necessary for serial data transfer, two are serial data input/output lines, TxD and RxD, and the other two are control lines, RTS and CTS. RTS is a data request signal from the μCOM-87AD and is always active (at the low level) for this system. When the μCOM-87AD transmits data, the CTS signal from the console is checked.

2.2 Equation Evaluation

This program uses the "character string" input from the console as an <equation>. This section describes the concept of the <equation>.

In this section three terms are defined as follows:

(1) <Factor> : Numeral, function or equation in parentheses
(2) <Term> : Value obtained by multiplying or dividing a <factor>
(3) <Equation> : Addition or subtraction for <terms>

As shown in figure 2-2, an <equation> is in hierarchical configuration. Thus, when an <equation> is called, <terms> and <factors> are called in sequence.

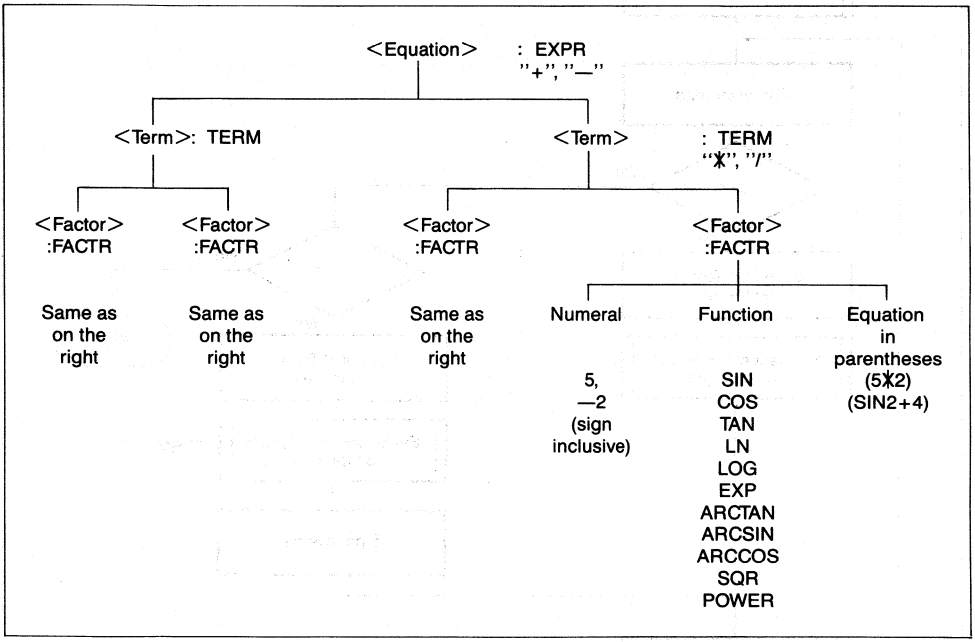


Figure 2-2 Hierarchical Configuration of <Equation>

Figure 2-3 shows the <equation> evaluation flowchart.

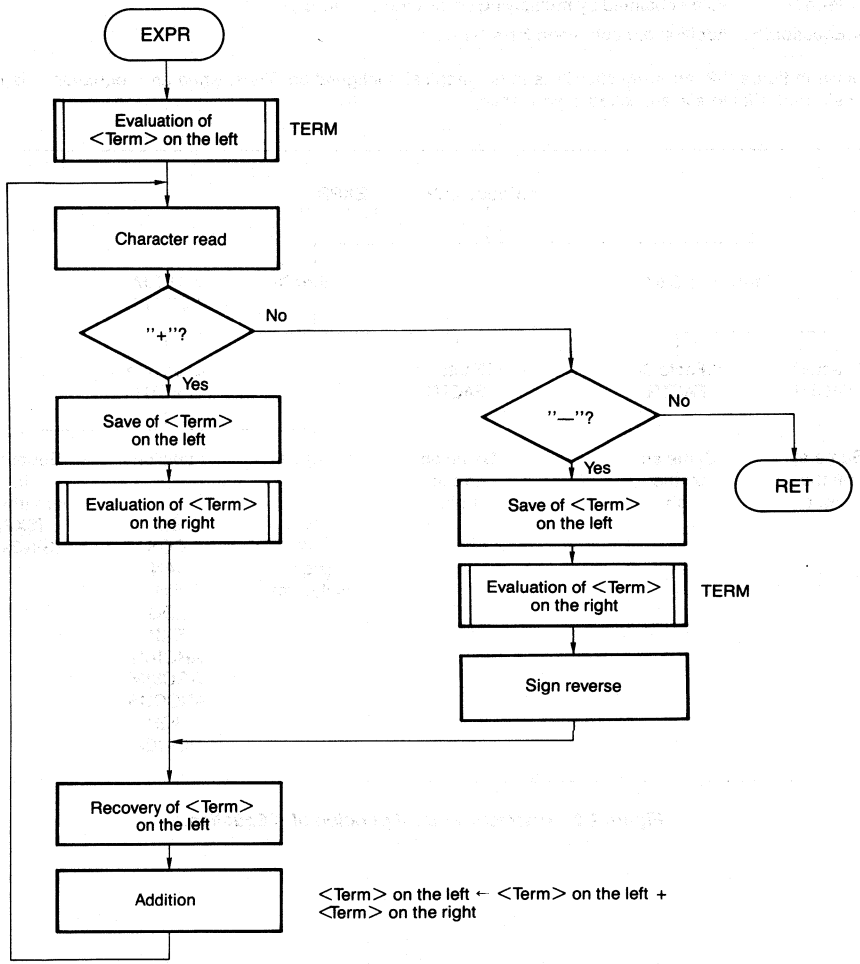
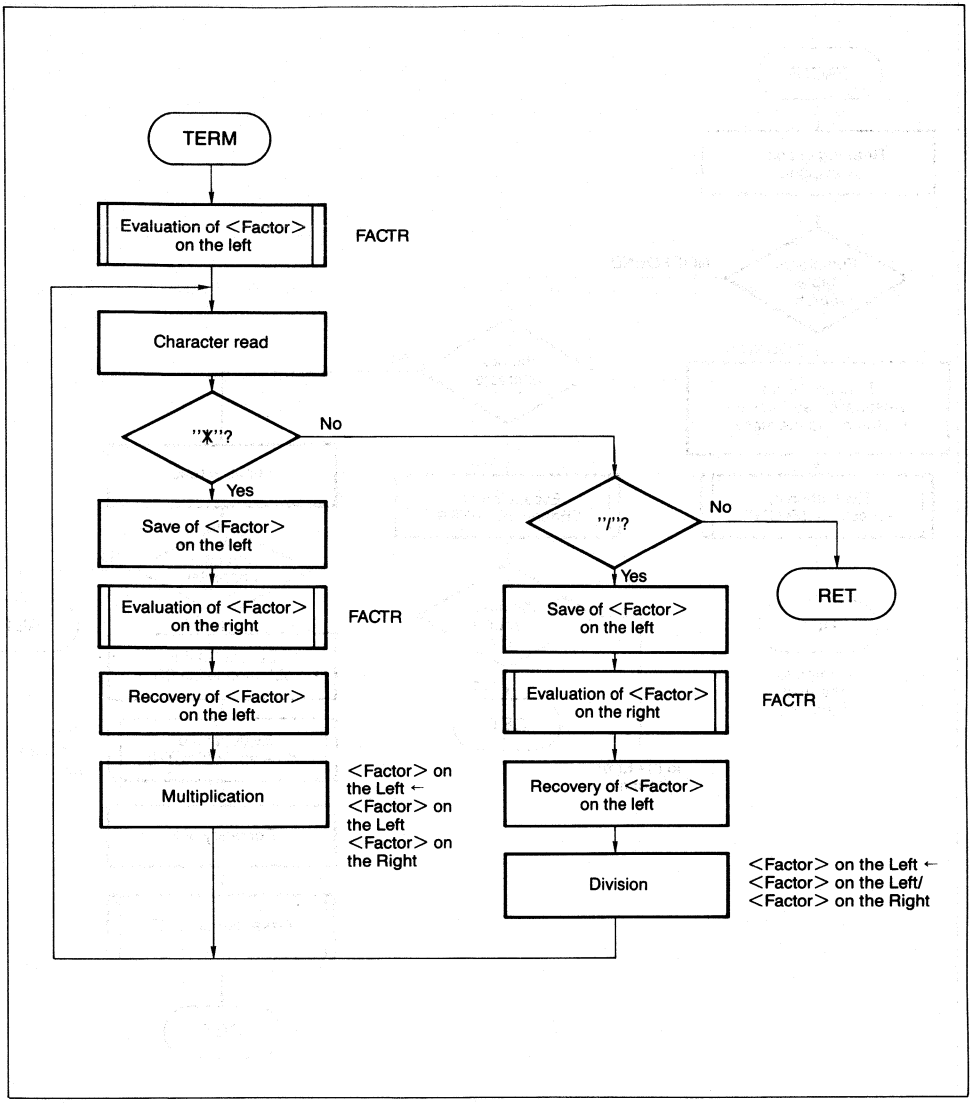
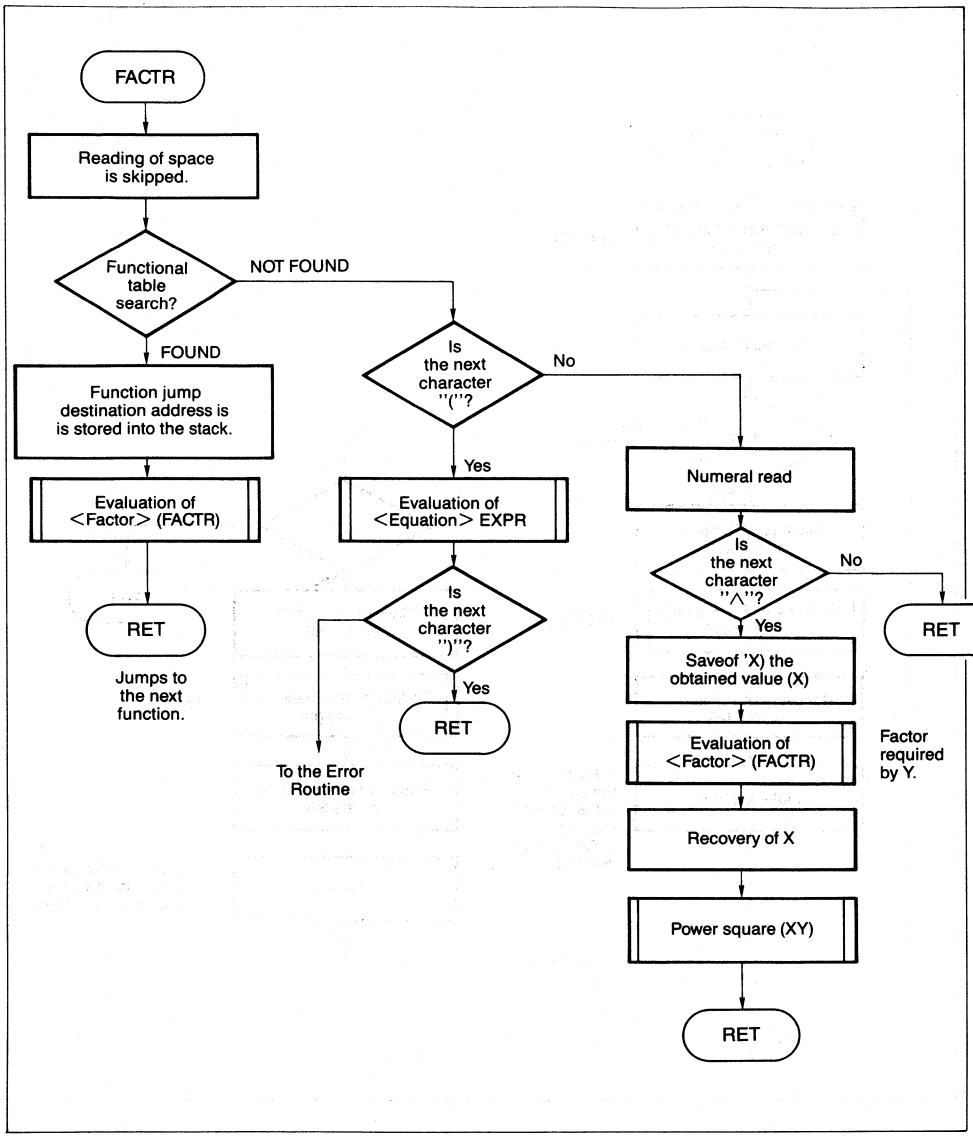


Figure 2-3 <Equation> Evaluation


TERM
Evaluation of <Factor> on the left
FACTR
Character read
"*"?
No
Yes
Save of <Factor> on the left
Evaluation of <Factor> on the right
FACTR
Recovery of <Factor> on the left
Multiplication
<Factor> on the Left ← <Factor> on the Left <Factor> on the Right
"/"?
No
Yes
RET
Save of <Factor> on the left
Evaluation of <Factor> on the right
FACTR
Recovery of <Factor> on the left
Division
<Factor> on the Left ← <Factor> on the Left/ <Factor> on the Right



In <factor> processing, the power square of two <factors> is calculated and the result is treated as a <factor> once again. If there are parentheses, a value in them is treated as an <equation> and the result of the equation is processed as a <factor>. Thus, the nesting of the parentheses is also enabled.

The following is an example of <equation> evaluation:

Example: ∘ 2 * 3 + 412 * 10

```
— — — — —          ... <Factor>
——— ———            ... <Term>
———————            ... <Equation>
```

∘ (10 + 8.5) * 2 + 9

```
— — — — —          ... <Factor>
———                ... <Factor>
——————— —          ... <Term>
———————            ... <Equation>
```

∘ 10 * — 2 + — 2 * — 3

```
— — — — — —        ... <Factor>
——— ———            ... <Term>
———————            ... <Equation>
```

∘ SIN1 * 2

```
——— —              ... <Factor>
———                ... <Term>
———                ... <Equation>
```

∘ 2 ∧ 3 + 4

```
— — —              ... <Factor>
———                ... <Factor>
——— —              ... <Term>
———                ... <Equation>
```

∘ 2 ∧ (3 + 4) * 2

```
— ——— —            ... <Factor>
———                ... <Factor>
———————            ... <Term>
———————            ... <Equation>
```

Through the use of the <equation> evaluation method described in this section, the program capacity can be decreased to a relatively small value. However, the processing speed is decreased because the stack is frequently used. Furthermore, extra care should be taken to ensure the stack nesting level does not exceed the allowable stack range.

2.3 Variables

This section describes the variables used for this program. In this program the variables and stack areas are set in the internal RFAM area (FF00H to FFFFH) of the μCOM-87AD.

Variable Name	Address	Description
B.F.P.ACC	FF00H to FF04H	• ACCE: Exponential part • ACCS: Sign of fixed point part sign • ACC1: 1st fixed point part • ACC2: 2nd fixed point part • ACC3: 3rd fixed point part
SF	FF05H	• Addition/subtract subroutine used
OPERAND	FF06H to FF0AH	• WSE: Exponential part • WS1: Sign of fixed point part + 1st fixed point part • WS2: 2nd fixed point part • WS3: 3rd fixed point part
WORK X REGISTER	FF0BH to FF0EH	B.F.P.ACC and operand save area • WXE, WX1, WX2, WX3
WORK Y REGISTER	FF0FH to FF12H	B.F.P.ACC and operand save area • WYE, WY1, WY2, WY3
WORK D REGISTER	FF13H to FF16H	BFDX subroutine operation area • WD1, WD2, WD3, (Quotient) WD4, WD5, WD5 (Remainder)
WORK A REGISTER	FF19H to FF1BH	BFMX subroutine operation area • WA1, WA2, WA3

(cont'd)

Variable Name	Address	Description
WORK REGISTER	FF1CH	• POWERK: Used for POWER subroutine. Least significant digit of INT(Y) is stored.
	FFIDH	• LNWORK: Used for LNX subroutine. X exponent — 80H is stored.
	FFIEH	• CMULT0: Used for BFMX subroutine. Digit-number (3-digit) counter.
	FFIFH	• CMULT1: Used for MULT subroutine. Byte-number (3-byte) counter.
	FF20H	• CSIN: Used for SIN subroutine. Scaling work.
	FF21H	• TMPWSL: Temporary save of WS4
	FF22H	• FASIN: Used for ARCSIN subroutine. X sign is stored (if overflow).
FLAG OF SIGN	FF23H	• FSN: B.F.P.ACC sign is stored. 0: B.F.P.ACC = 0 1: B.F.P.ACC > 0 0FFH: B.F.P.ACC < 0
BFINP & BFOUT WORK	FF24H FF25H	• ADDRS: Character string store address
	FF26H	• TMP1: Temporary 1
	FF27H	• TMP2: Temporary 2
	FF28H	• TMP3: Temporary 3
	FF29H	• TMP4: Temporary 4
OUT AREA	FF2AH	• OUTADR: Output character string store area
CHARACTER STORE AREA	FF37H FF87H	• CHARST: Input character store area (80 characters max.)
STACK AREA	FF88H FFFFH	• Stack area

2.4 Program

In this program, data continues to be read in the character store area (80 bytes from CHARST) until "=" is input from the console. The read data is evaluated as an <equation> described in section 2.2 and is consequantly output into the console.

However, because two operated values must be output for coordinate conversion (in the TRACA or TRACB subroutine), <equation> evaluation must be called once again after the previous <equation> evaluation and each coordinate conversion subroutine must be called using the values of the two <equations>.

The <equation> evaluation is started by a currently indicated character string pointer and terminates if an irregular character is found. After that, the value obtained through the <equation> evaluation is output onto the console and the next character is checked to be "=". If it is not "=", "?" is output onto the console as a format error. Therefore, if a format error occurs, the <equation> value evaluated until then is output.

Figure 2-4 shows a general flowchart of this program.

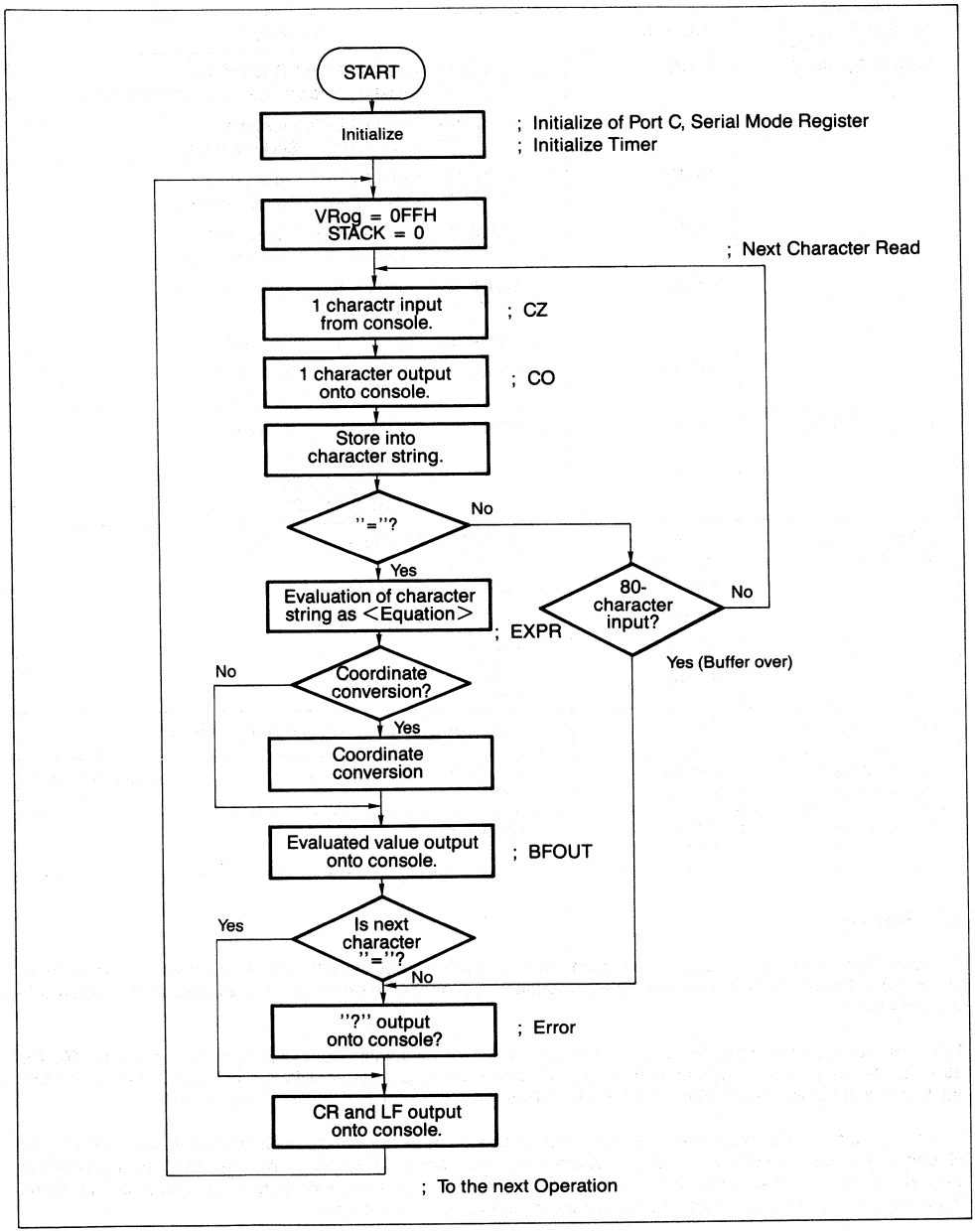
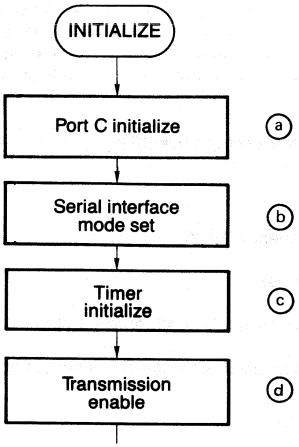


Figure 2-4 General Flowchart

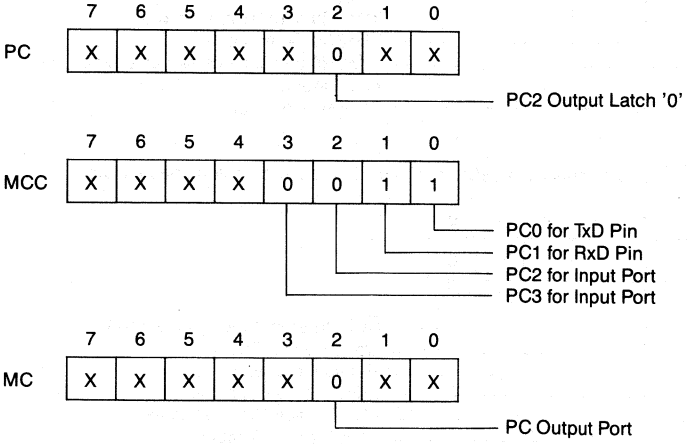
Next, the μCOM-87AD serial data transmission/reception method is described.

The serial mode register, timer, port C, etc. must be initialized in advance so that the μCOM-87AD can transmit or receive serial data.

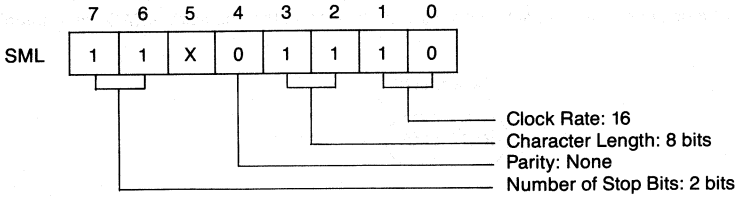
Flowchart:



(a) In port C; PC_0, PC_1, PC_2, and PC_3 serve as TxD pin, RxD pin, output port ($\overline{RTS}$ output) and input port ($\overline{CTS}$ input) respectively.



(b) Parameters for serial data transmission and reception are set into the serial mode register.



(c) The TO output of timer is used as a serial clock ($\overline{SCK}$).

$$C = \frac{f}{0.24 \times N \times B}$$

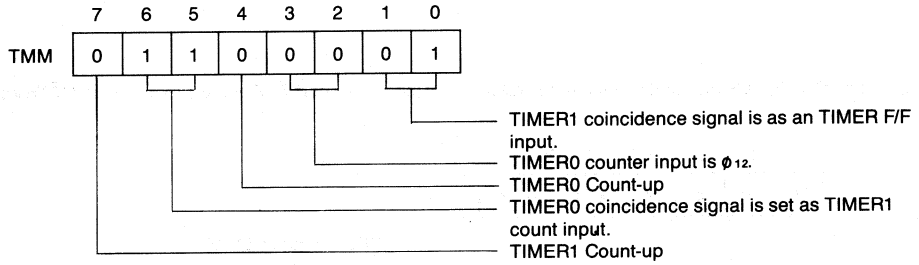
where f: Clock oscillation frequency (MHz)
N: Clock rate
B: Data transfer rate (kbps)

Because f = 11.0592 MHz, N = 16 and B = 48 for this system, C = 6.

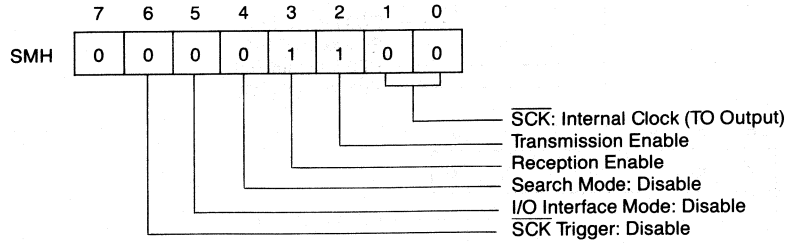
Thus, TIMER0 and TIMER1 are used with cascade connection, and TM0 and TM1 are set to 6 and 1 respectively.

TM0 = 6
TM1 = 1

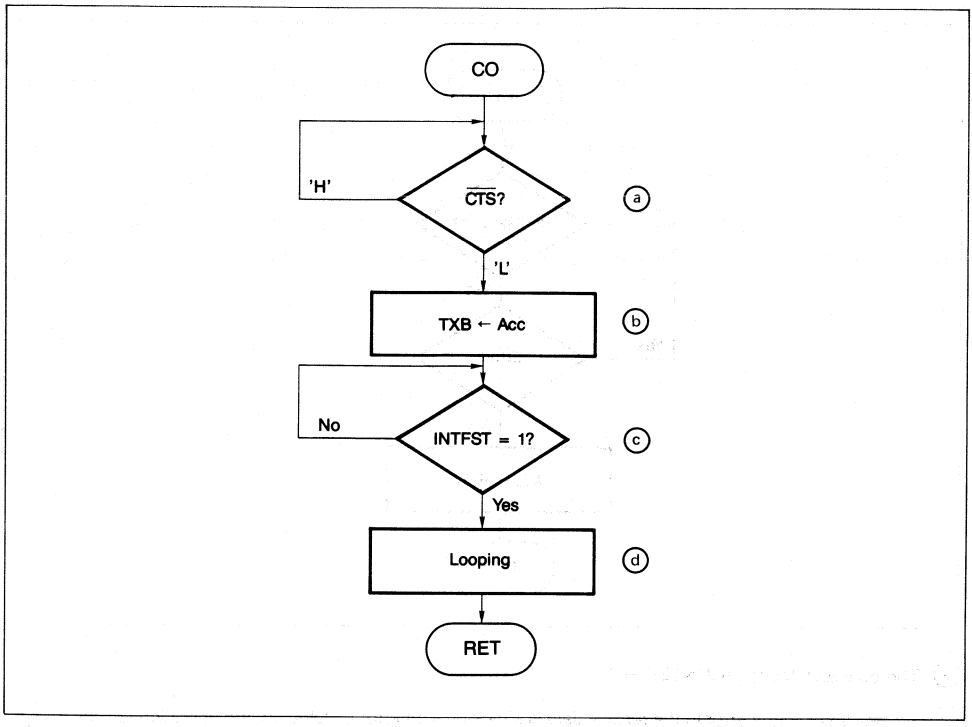
The timer register (TMM) is set as follows:



(d) The timer TO output is used as a serial clock (SCK) and is set in the transmission enable state.



Next, the subroutine for 1 byte transmission of the accumulator data (ACC) is described.



(a) The operation loops until the transfer request signal (CTS) is received at the reception end.

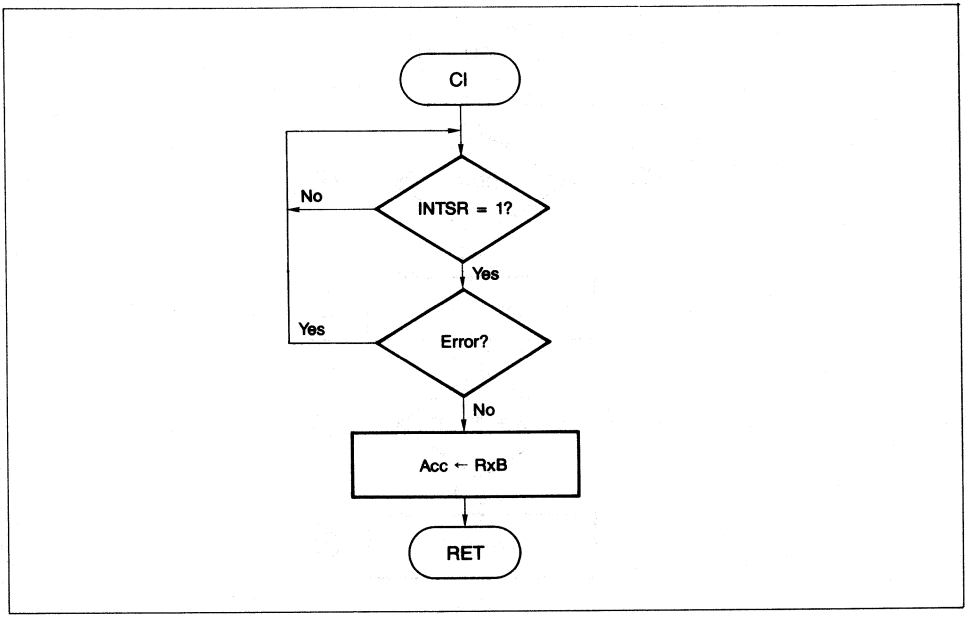
(b) The ACC data is transferred to the transmission buffer.

(c) The operation loops until the transmission buffer becomes empty.

(d) The operation further loops until the serial register becomes empty.

Through operations (c) and (d), both the transmission and serial registers are empty whenever the C0 subroutine is called. In this state the $\overline{CTS}$ signal can be checked and the transmitted data is no longer cancelled at the reception end.

Data reception is carried out as follows:



(a) The operation loops until INTSR = 1.

(b) Wheter there is any error in the receive data is checked. If an error is detected, the operation returns to (a).

(c) The received data is transferred to ACC.

Chapter 3 Execution Results

This chapter describes the μCOM-87AD functional package execution results from the programs described in chapter 2.

For functional packages, except for addition, subtraction, multiply and division, the execution results are compared to the operation results by the PC-8001.

The accuracy of five to six effective digits has been achieved for all functional packages.

3.1 BFADD Subroutine

```
1+0= 1.000000                          ; 0-check
0+1= 1.000000

1E38+1= 1.000000E+38                   ; Near max.
1E38+9E38=?
1E38+1E38=?
0.9E38+0.01E38= 9.100004E+37

9.9E-39+0.1E-39= 9.900000E-39          ; Near min.
9.9E-39+0.2E-39= 9.900000E-39
9.9E-39+1E-39= 9.900000E-39
9.9E-39+1E39=?
9.9E-39+1E30= 1.000000E+30

1345+98765= 100110.0                   ; Random
455675211368643+214447796= 4.556755E+14
1.3E20+4E-30= 1.300000E+20
223+0.11111111= 223.1111
5465678627R45TFG

134465463+4784876876685= 0.000000      ?
3246754564+233= 3.246755E+09

1E-30+1E-10= 1.000000E-10              ;
122344.E-10+345667E-9= 3.579014E-04
1232343.6787654E-27+0= 1.232344E-21
98765E-25+111111111E-3= 111111.1
```

3.2 BFSUB Subroutine

```
1-0= 1.000000                                   ; 0-check
0-1=-1.000000

2133E-29-12.45E-10=-1.245000E-09                ; Near max.
23352E-9-0.999999E-20= 2.335200E-05
3763467E-30-12E-1=-1.200000

1E-39-8E-39=-8.000004E-39                       ; Near min.
9.9E-39-0.1E-19=-1.000000E-20
9.9E-39-0.1E-39= 9.900000E-39
9.9E-39-1.1E-39= 9.900000E-39
9.9E-39-2=-2.000000
9.9E-38-2.2E-38= 0.000000     ?
9.9E-39-2.2E-36=-2.190100E-36

2.2E-38= 2.200000E-38                           ; Random
9.9E-38-2.2E-38= 7.700001E-38
2.2E-39= 0.000000
12356565-45876= 1.231069E+07
123456789-123456789= 0.000000
125654-988656=-863002.2
2356555656-787788778=-5.521332E+08
1E20-23454.3344E10= 9.999979E+19
```

3.3 BFMUL Subroutine

```
1*0= 0.000000                                   ; 0-check
0*1= 0.000000

9.9E37*0.1= 9.900001E+36                        ; Near max.
1E38*0.1= 1.000000E+37
1E38*0.99999999=?
1E38*0.9= 9.000002E+37
12E37*2=?

1E-38*2= 2.000001E-38                           ; Near min.
9.9E-39*2= 1.980000E-38
9.9E-39*1E-1= 0.000000
9.9E-39*1E2= 9.900000E-37

6543665534*12344= 8.077501E+13                  ; Ramdon
123456789*123456789= 1.524158E+16
1E30*1E-39= 0.000000
1E30*1E-10= 1.000000E+20
1.234E20*234= 2.887560E+22

1E-20*100= 1.000000E-18
1.234E-30*1.2E-5= 1.480801E-35
12344E-35*99= 1.222056E-29
1.23456R= 1.234560     ?
1.2345E-30*2E-7= 2.469000E-37
```

3.4 BFDIV Subroutine

```
0/1= 0.000000                          ; 0-check
1/0=?

1E38/2= 5.000001E+37                   ; Near max.
1E38/0.1=?
1E38/1E-10=?
1E38/1E10= 1.000000E+28

1E-38/1E-10= 1.000000E-28              ; Near min.
1E38-= 1.000000E+38
1E-38/2= 0.000000
9.9E-39/9.9E-2= 1.000000E-37

13346546534/4675765= 2854.410          ; Ramdon
123456789/123456789= 1.000000
987654321/123456789= 8.000000
12/21= .5714286
1E10/2.3E20= 4.347826E-11
1.23E10/-1.2E10=-1.025000

1.23E-20/.99E-10= 1.242425E-10
456E-10/32432E-9= 1.406019E-03
122212E-10/123E-2= 9.935935E-06
```

3.5 COS Subroutine

μCOM-87AD	PC-8001
COS0= 1.000000	cos (0)= 1
COS3.141592653=-1.000000	cos (3.14159)=-1
COS1.570796326= 0.000000	cos (1.5708)= 0
COS500=-.8838571	cos (500)=-.883857
COS-10000=-.9523750	cos (-10000)=-.952141
COS-999999= .1950903	cos (-999999)= .19509
COS8.377580409=-.4999996	cos (8.37758)=-.5
COS15707.96326= .9999988	cos (15708)= .999999
COS-9424.77796= .9999998	cos (-9424.78)= 1
COS162.3156204= .5000035	cos (162.316)= ,499993

- Comment

The larger a numeral becomes, tghe larger an error becomes.

Example: COS-10000 = —0.9523750 . . . —0.85282 (Real value)

This is because the integer part of a value obtained by dividing a numeral by 2π is discarded and the decimal point part is used. As a result, the larger a numeral becomes, the smaller the effective numeral of the decimal point part becomes.

This is also the same with the SIN and TAN subroutines.

3.6 SIN Subroutine

μCOM-87AD	PC-8001
SIN0= 0.000000	sin (0)= 0
SIN0.2= .1986692	sin (.2)= .198669
SIN0.5= .4794254	sin (.5)= .479426
SIN0.6= .5646424	sin (.6)= .564643
SIN0.8= .7173561	sin (.8)= .717356
SIN1.1= .8912075	sin (1.1)= .891207
SIN1.45= .9927130	sin (1.45)= .992713
SIN1.88= .9525762	sin (1.88)= .952576
SIN2.22= .7965655	sin (2.22)= .796566
SIN2.77= .3631000	sin (2.77)= .3631
SIN-2.77=-.3631000	sin (-2.77)=-.3631
SIN3.111= 3.058796E-02	sin (3.111)= .030588
SIN3.333=-.1902402	sin (3.333)=-.19024
SIN-3.333= .1902402	sin (-3.333)= .19024
SIN3.141592653= 1.872535E-07	sin (3.14159)= 0
SIN6.283185307= 1.498028E-06	sin (6.28319)= 0
SIN9.42477796=-7.490141E-07	sin (9.42478)= 0
SIN1.570796326= 1.000000	sin (1.5798)= .999959
SIN10.99557428=-1.000000	sin (10.9956)=-1
SIN-32.98672286=-1.000000	sin (-32.9867)=-1
SIN33.3333333= .9405295	sin (33.3333)= .940531
SIN111.11=-.9144961	sin (111.11)=-.914496
SIN31415.92653=-3.067957E-03	sin (31415.9)=-6.13589E-03
SIN157079.6326=-1.227154E-02	sin (157080)=-.0122715
SIN5.55555555=-.6651012	sin (5.55556)=-.665102
SIN77E-1= .9881681	sin (7.7)= .988168
SIN100=-.5063675	sin (100)=-.506368
SIN500=-.4677570	sin (500)=-.467757
SIN1000= .8268607	sin (1000)= .826361
SIN10000=-.3056596	sin (10000)=-.30566
SIN50000=-.9998306	sin (50000)=-.999331
SIN90000=-.3426607	sin (90000)=-.342661
SIN999999=-.9807852	sin (999999)=-.980785
SIN9999999= 1.000000	sin (1E+07)= 1
SIN-50000= .9998306	sin (-50000)= .999831
SIN-10000= .3056596	sin (-10000)= .30566
SIN-159.435827=-.7071153	sin (-159.436)=-.707115
SIN46.60029102= .5000035	sin (46.6003)= .500001
SIN-8.377580409=-.8660253	sin (-8.37758)=-.866025

3.7 TAN Subroutine

μCOM-87AD	PC-8001
TAN0= 0.000000	tan (0)= 0
TAN-1.047197551=-1.732051	tan (-1.0472)=-1.73205
TAN2.09438=-1.732110	tan (2.09438)=-1.73211
TAN-2.0943951:= 1.732050	tan (-2.0944)= 1.73205
TAN2.1=-1.709847	tan (2.1)=-1.70985
TAN2.094395102=-1.732050	tan (2.0944)=-1.73205
TAN.523598775= .5773501	tan (.523599)= .57735
TAN10471.45191= .5756471	tan (10471.5)= .577436
TAN157080.1562= .5541788	tan (157080)= .570502
TAN1000= 1.470218	tan (1000)= 1.47022
TAN10000= .3209446	tan (10000)= .320945
TAN999999=-5.027339	tan (999999)=-5.02734
TAN9999999=?	tan (1E+07)=
TAN500= .5292224	tan (500)= .529222

3.8 LN Subroutine

μCOM-87AD	PC-8001
LN1E36= 82.89307	ln (1E+36)= 82.8931
LN1.6E30= 69.54757	ln (1.6E+30)= 69.5476
LN2.7182919= 1.000004	ln (2.71829)= 1
LN98765.4321= 11.50050	ln (98765.4)= 11.5005
LN12345.6789= 9.421061	ln (12345.7)= 9.42106
LN3.3333333= 1.203973	ln (3.33333)= 1.20397
LN1= 1.966953E-06	ln (1.96695E-06)=-13.139
LN0.000000001=-20.72326	ln (1E-09)=-20.7233
LN0.9999999E-30=-69.07755	ln (1E-30)=-69.0776
LN2.22E-35=-79.79298	ln (2.22E-35)=-79.793
LN9.11E-40=?	ln (0)=
LN4.539992976E-5=-10.00000	ln (4.53999E-05)=-10
LN1318815734= 21.00000	ln (1.31882E+09)= 21
LN20.08553692= 3.000000	ln (20.0855)= 3
LN2.590448618= .9518313	ln (2.59045)= .951831
LN59874.14171= 11.00000	ln (59874.1)= 11
LN0=?	
LN-0.00000001=?	

3.9 LOG Subroutine

μCOM-87AD	PC-8001
LOG0=?	log (0)=
LOG-0.0000000001=?	log (-1E-10)=
LOG1= 8.542369E-07	log (1)=
LOG0.1111111E-20=-20.95424	log (1.11111E-21)=-20.9542
LOG0.3333333E-30=-30.47712	log (3.33333E-31)=-30.4771
LOG0.7777777E-40=?	log (0)=
LOG0.9999999E-36=-36.00000	log (1E-36)=-36
LOG987654321= 8.994606	log (9.87654E+08)= 8.99461
LOG1234567890= 9.091516	
LOG9.999999999= 1.000000	log (10)= 1
LOG66.666666666= 1.823909	log (66.6667)= 1.82391
LOG0.4444444444=-.3521825	log (.444445)=-.352183
LOG10= 1.000000	log (10)= 1
LOG1000= 3.000000	log (1000)= 3
LOG100000= 5.000000	log (100000)= 5
LOG1E30= 30.00000	log (1E+30)= 30
LOG1E37= 37.00001	log (1E+37)= 37
LOG1E36= 36.00001	log (1E+36)= 36
LOG1E38= 38.00000	log (1E+38)= 38

3.10 EXP Subroutine

μCOM-87AD	PC-8001
EXP80= 5.540605E+34	exp (80)= 5.54061E+34
EXP85= 8.222998E+36	exp (85)= 8.22302E+36
EXP88=?	exp (88)=
EXP63= 2.293768E+27	exp (63)= 2.29378E+27
EXP60= 1.142008E+26	exp (60)= 1.14201E+26
EXP52= 3.831011E+22	exp (52)= 3.831E+22
EXP40= 2.353852E+17	exp (40)= 2.35385E+17
EXP20= 4.851631E+08	exp (20)= 4.85165E+08
EXP11= 59873.83	exp (11)= 59874.1
EXP10= 22026.47	exp (10)= 22026.5
EXP5= 148.4132	exp (5)= 148.413
EXP1= 2.718282	exp (1)= 2.71828
EXP0= 1.000000	exp (0)= 1
EXP0.3333333= 1.395612	exp (.333333)= 1.39561
EXP0.987654321= 2.684930	exp (.987654)= 2.68483
EXP-60= 8.756509E-27	exp (-60)= 8.75651E-27
EXP-80= 1.804858E-35	exp (-80)= 1.80486E-35
EXP-87= 1.645807E-38	exp (-87)= 1.64581E-38
EXP-88= 0.000000	exp (-88)= 6.05462E-39
EXP-88.5= 0.000000	exp (-88.5)= 3.67231E-39
EXP1.609437912= 5.000001	exp (1.60944)= 5
EXP2.995732273= 20.00000	exp (2.99573)= 20
EXP-3.688879454= 2.499999E-02	exp (-3.68888)= .025
EXP4.465908118= 87.00001	exp (4.46591)= 87
EXP4.605170186= 99.99999	exp (4.60517)= 100
EXP198.022318=?	exp (198.022)=
EXP46.05170186= 1.000005E+20	exp (46.0517)= 9.99999E+19

3.11 SQR Subroutine

μCOM-87AD	PC-8001
SQR1= 1.000001	sqr (1)= 1
SQR4= 2.000003	sqr (4)= 2
SQR121= 11.00000	sqr (121)= 11
SQR11.11110888= 3.333327	sqr (11.1111)= 3.33333
SQR60.49381506= 7.777717	sqr (60.4938)= 7.77778
SQR30.86308025= 5.555501	sqr (30.8636)= 5.5555
SQR.3333333333= .5773519	sqr (.333333)= .57735
SQR.8888888888= .9428087	sqr (.888889)= .942809
SQR.00000000001= 3.162277E-06	sqr (1E-11)= 3.16228E-06
SQR0.99999E-40= 0.000000	sqr (0)= 0
SQR0.7777777E-35= 2.788863E-18	sqr (7.77778E-36)= 2.78887E-18
SQR0.3333333E-20= 5.773494E-11	sqr (3.33333E-21)= 5.77352E-11
SQR-0.000000001=?	sqr (-1E-08)=

3.12 POWER Subroutine

μCOM-87AD	PC-8001
0^0= 1.000000	0 ^ 0 = 1
0^1= 0.000000	0 ^ 1 = 0
1^0= 1.000000	1 ^ 0 = 1
1^1= 1.000002	1 ^ 1 = 1
1^-1= .9999975	1 ^-1 = 1
2^-2= .2499989	2 ^-2 = .25
-1^2.567=?	-1 ^ 2.567 =
-1^-4= .9999917	-1 ^-4 = 1
0^-9.8765=?	0 ^-9.8765 =
-3.456789^4= 142.7880	-3.45679 ^ 4 = 142.788
-9.99999999^0= 1.000000	-10 ^ 0 = 1
-543231^-99= 0.000000	-54321 ^-99 =0
-2.2222222222^-3=-9.112471E-02	-2.22222 ^-3 =-.091125
5.555555555^-1.111111111= .1487735	5.55556 ^-1.11111 = .148773
9.87654321^1.23456789= 16.90081	9.87654 ^ 1.23457 = 16.9008
-8^100=?	-8 ^ 100 =
8.88888888 10.00001= 3.079557E+09	8.88889 ^ 10 = 3.07953E+09
9^1.2345= 15.06642	9 ^ 1.2345 = 15.0665
2.19^-9.12= 7.855002E-04	2.19 ^-9.12 = 7.85505E-04
7.89^10.123= 1.205343E+09	7.89 ^ 10.123 = 1.2053E+09
90.1^10.00001= 3.525869E+19	90.1 ^ 10 = 3.52588E+19
30^30=?	30 ^ 30 =
19.876^-19.876= 1.564628E-26	19.876 ^-19.876 = 1.56464E-26
9^9= 3.874231E+08	9 ^ 9 = 3.87421E+08
9.1^9.1= 5.336737E+08	9.1 ^ 9.1 = 5.33674E+08
9.999999^10.0000001= 1.000003E+10	10 ^ 10 = 1E+10

3.13 ARCTAN Subroutine

μCOM-87AD	PC-8001
ARCTAN0= 0.000000	arctan (0)= 0
ARCTAN1= .7854021	arctan (1)= .785398
ARCTAN1.732050807= 1.047198	arctan (1.73205)= 1.0472
ARCTAN-0.577350269=-.5235988	arctan (-.57735)=-.523599
ARCTAN10= 1.471128	arctan (10)= 1.47113
ARCTAN-10=-1.471128	arctan (-10)=-1.47113
ARCTAN1000= 1.569796	arctan (1000)= 1.5698
ARCTAN1000000= 1.570795	arctan (1E+06)= 1.5708
ARCTAN1000000000= 1.570796	arctan (1E+09)= 1.5708
ARCTAN9999999999= 1.570796	arctan (1E+10)= 1.5708
ARCTAN3.333333= 1.279340	arctan (3.33333)= 1.27934
ARCTAN-3.333333=-1.279340	arctan (-3.33333)=-1.27934

3.14 ARCSIN Subroutine

μCOM-87AD	PC-8001
ARCSIN1= 1.570796	
ARCSIN-1=-1.570796	
ARCSIN0= 0.000000	arcsin (0)= 0
ARCSIN-0.5=-.5235991	arcsin (-.5)=-.523599
ARCSIN0.707106781= .7853938	arcsin (.707107)= .785398
ARCSIN-0.866025403=-1.047193	arcsin (-.866025)=-1.0472
ARCSIN-1.00000001=-1.570796	
ARCSIN-1.000001=?	
ARCSIN1.0000001=?	
ARCSIN1.00000001= 1.570796	

3.15 ARCCOS Subroutine

μCOM-87AD	PC-8001
ARCCOS1= 0.000000	arccos (1)= 0
ARCCOS-1= 3.141593	arccos (-1)= 3.14159
ARCCOS0= 1.570796	
ARCCOS0.523598775= 1.019727	arccos (.523599)= 1.01973
ARCCOS0.866025403= .5236036	arccos (.866025)= .523599
ARCCOS-0.5= 2.094395	arccos (-.5)= 2.09439
ARCCOS-0.866025403= 2.617989	arccos (-.866025)= 2.61799
ARCCOS0.707106781= .7854026	arccos (.707107)= .785398
ARCCOS1.00000001= 0.000000	arccos (1)= 0
ARCCOS-0.1= 1.670964	arccos (-.1)= 1.67096
ARCCOS0.1= 1.470629	arccos (.1)= 1.47063
ARCCOS0.7777777= .6796740	arccos (.777778)= .679674
ARCCOS-0.99999999= 3.141593	arccos (-1)= 3.14159
ARCCOS4.0=?	
ARCCOS-1.00000001= 3.141593	arccos (-1)= 3.14159
ARCCOS0.3333333= 1.230960	arccos (.333333)= 1.23096
ARCCOS-0.88888888= 2.665711	arccos (-.888889)= 2.66571

3.16 TRACA Subroutine

```
1<-1.570796326= 0.000000      , -1.000000
1<0.523598775= .8660254       , .4999999
7<2.617993878=-6.062178       , 3.500001
99<-2.094395102=-49.50002     , -85.73652
8.8888888<2.35619449=-6.285396      , 6.285392
7.7777777<-57.52359877= 4.365061    , -6.437396
5.6666666<3145.519644=-4.008475     , -4.005402
2.8284217,-31416.71193= 31416.54    < -1.570706
2.888888888<-31416.71193= 2.036476  , -2.049010
-0.9999,UU= .9998995      < 3.141593    ?
-0.999<3.14= .9989987     , -1.591372E-03
1.777777,2.094395102= 2.747178      < .8669852
1.777777<2.094395102=-.8888888      , 1.539600
9.999999<-2.617993878=-8.660253     , -5.000000
4.444444<-3.141592653=-4.444444     , -8.322378E-07
99999999<-1.570796326= 0.000000     , -9.999999E+07
```

3.17 TRACB Subroutine

```
1,1= 1.414215      < .7854021
1,-1= 1.414215     < -.7854021
-1,1= 1.414215     < 2.356191
-1,-1= 1.414215    < -2.356191
7.777777,0.101010= 7.778374      < 1.298627E-02
-3.3333333,-0.785398163= 3.424603    < -2.910194
6.66666,-6.66666= 9.428085       < -.7854021
0.5,-0.866025403= .9999994       < -1.047198
-0.707106871,-0.707106781= 1.000002     < -2.356199
-8.8888888,-9.9999999= 13.37953      < -2.297437
-0.0001E-7.77.77777= 77.77783        < 1.570796
9.9E15,-2.22222222= 9.900027E+15 < -2.244668E-16
-6.66666E15,-0.012345E-15= 6.666662E+15 < -3.141593
1,-9= 9.055392       < -1.460139
12345.67898,0.987654321= 12345.68       < 8.000000E-05
0.5,0.866025403= .9999994        < 1.047198
-0.707106781,0.707106781= 1.000000       < 2.356191
-0.8660254,-0.5= .9999994        < -2.617994
-7.071067811,-7.07810= 10.00498      < -2.355694
11111111,111111111= 1.571339E+07 < .7854021
-999999999,-999999999= 1.414215E+09 < -2.356191
666666666,-666666666= 9.428083E+08 < -.7854021
```

Chapter 4 Source Lists

```
** UPD7811     SYMBOL LIST **          (1985.04.29  MON.    )  PAGE    1

SYMBOL  ADRS   SYMBOL  ADRS   SYMBOL  ADRS   SYMBOL  ADRS   SYMBOL  ADRS

ACC1    FF02   ACC2    FF03   ACC3    FF04   ACCE    FF00   ACCS    FF01
ACOS1   07FB   ACOS2   0802   ADADJ   0189   ADDRS   FF24   ARCCOS  07D6
ARCSIN  079B   ARCTAN  0765   ASIN1   07AA   ASIN2   07C3   ASIN3   07CD
ATAN1   0772   ATAN2   078B   ATNCON  0933   BF2DV   08D9   BF2PDV  08ED
BF2PI   08E9   BF4DV   08DD   BFABS   0A32   BFAD10  03FD   BFAD11  0403
BFAD2   03BA   BFAD3   03D3   BFAD4   03E4   BFAD9   03F7   BFADD   038A
BFADD0  038F   BFADD1  0392   BFADR   040E   BFADS   0417   BFCOM1  09F6
BFCOM2  09FD   BFCOMP  0A05   BFCOMX  0A0D   BFDIV   04A3   BFDIV0  04A8
BFDIV1  04AC   BFDV58  08CD   BFDX    04E2   BFDX1   04EC   BFDX3   050B
BFDX4   0512   BFDX5   051E   BFDX6   0524   BFDX7   053F   BFDX8   0549
BFDX9   053A   BFIN11  01D6   BFIN12  01DB   BFIN13  01E5   BFIN14  01E9
BFIN30  0250   BFIN31  022D   BFIN32  0231   BFIN33  024D   BFIN40  025D
BFIN41  0262   BFIN42  0272   BFIN43  0279   BFINP   01B6   BFINP1  01D3
BFINP2  01EF   BFINP3  0222   BFINP4  025A   BFINT   0AB6   BFINT0  0AAA
BFIX    0A64   BFIX0   0A5C   BFIX2   0A5E   BFIXED  0A89   BFIXEN  0A92
BFLGE2  08F1   BFLN10  08F9   BFLN2   08F5   BFLSH1  0A55   BFLT    0A97
BFMPI   08E5   BFMUL   0426   BFMUL0  042B   BFMUL1  042F   BFMULZ  0450
BFMX    0456   BFMX1   045E   BFMX2   046D   BFMX3   0478   BFN01   08D1
BFN1    08D5   BFNOR   09BC   BFNOR0  09C0   BFNOR1  09CC   BFOU00  029B
BFOU01  02A1   BFOU02  02AF   BFOU03  02D0   BFOU11  02E0   BFOU12  02EF
BFOU21  030F   BFOU22  0324   BFOU30  0347   BFOU31  034E   BFOU32  035C
BFOU33  0360   BFOU34  0370   BFOUT   028C   BFOUT1  02B6   BFOUT2  02FB
BFOUT3  0342   BFPI    08E1   BFRD    0ABF   BFRDZ   0AE9   BFRET   0B56
BFRETS  0B59   BFRND1  09DF   BFRNDR  09E6   BFROND  09D8   BFRS11  0A41
BFRSH   0A39   BFRSH0  0A37   BFRSH1  0A3E   BFRSH2  0A44   BFRSH3  0A4B
BFRSH4  0A46   BFSIGN  0A13   BFSTR1  0A23   BFSTR2  0A28   BFSUB   037D
BFSUB1  0381   BFTEN   08C9   CHARD   0282   CHARST  FF37   CI      0193
CMULT0  FF1E   CMULT1  FF1F   CO      019E   CO1     01A4   CO2     01AC
COS     0551   COS1    055F   CSIN    FF20   CTS     0008   ERROR   0096
EXACOP  03CA   EXP     0662   EXP0    06C0   EXP1    06C1   EXP2    06C4
EXPCON  0912   EXPR    00B9   EXPR1   00BC   EXPR2   00C9   EXPR3   00D6
FACT11  0125   FACT31  015F   FACTR   0118   FACTR1  0124   FACTR2  0141
FACTR3  0151   FACTR4  0162   FACTR5  016A   FASIN   FF22   FSN     FF23
FTABL   0B5E   K       0080   LNCON   0958   LNWORK  FF1D   LNX     05F1
LOADA   0B0F   LOADA0  0B06   LOADA1  0B09   LOADA2  0B0C   LOADO1  0B25
LOADO2  0B28   LOADO3  0B2B   LOG     0656   MAIN1   001D   MAIN2   0027
MAIN3   0035   MAIN31  0048   MAIN32  0050   MAIN33  0055   MAIN4   008D
MAIN40  0071   MAIN5   0096   MAIN6   009B   MULT    0483   MULT1   048E
MULT2   0495   OUT     00A7   OUT1    00B2   OUTADR  FF2A   POLY    0999
POLY1   09A1   POLYN1  097D   POLYN2  0980   POLYN3  0983   POLYX   0986
POPR    0B49   POWER   06D2   POWER1  06D5   POWER2  06EF   POWER3  071C
POWER4  0728   POWER5  0750   POWER6  0759   POWERK  FF1C   POWERZ  075F
PUSHF   0B39   PUSHR   0B3C   ROUTE   080C   SF      FF05   SIN     056C
SIN0    0579   SIN1    05AC   SINCOS  08FD   SINOVR  05C4   SQR     06CF
STACK   0000   STORE0  0AED   STORE1  0AF0   STORE2  0AF3   TAN     05CE
TANOVR  05EB   TERM    00E0   TERM1   00E4   TERM2   00EA   TERM3   00FA
TMP1    FF26   TMP2    FF27   TMP3    FF28   TMP4    FF29   TMPWSL  FF21
TRACA   081F   TRACA1  0822   TRACA2  084F   TRACB   0855   TRACB1  0858
TRACB2  08B1   TRACB3  0887   TRACB4  08C0   TRACB5  08C3   WA1     FF19
WA2     FF1A   WA3     FF1B   WD1     FF13   WD2     FF14   WD3     FF15
WD4     FF16   WD5     FF17   WD6     FF18   WS1     FF07   WS2     FF08
WS3     FF09   WS4     FF0A   WSE     FF06   WX1     FF0C   WX2     FF0D
WX3     FF0E   WXE     FF0B   WY1     FF10   WY2     FF11   WY3     FF12
WYE     FF0F
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE     1

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

     1                    ;
     2                    ;       ;       ;       ;       ;       ;       ;
     3                    ;***********************************************
     4                    ;*                                             *
     5                    ;*       UCOM87AD B.F.P.ARITHMETIC PACKAGE     *
     6                    ;*                    TEST PROGRAME            *
     7                    ;*                                             *
     8                    ;***********************************************
     9                    ;
    10                    ;
    11                    ;***********************************************
    12                    ;*       POWER ON INITIALIZE ROUTINE           *
    13                    ;***********************************************
    14                    ;
    15 0000 640200                MVI     PC,0H
    16                    ;
    17 0003 6903                  MVI     A,00000011B
    18 0005 4DD1                  MOV     MCC,A    ;SET MCC REG.
    19                    ;
    20 0007 69FB                  MVI     A,11111011B
    21 0009 4DD4                  MOV     MC,A     ;SET PC2 TO OUTPUT MODE
    22                    ;
    23 000B 69CE                  MVI     A,11001110B
    24 000D 4DCA                  MOV     SML,A    ;SET SML REG.
    25                                             ;CLOCK LATE = *16
    26                                             ;CHAR.LENGTH = 8 BITS
    27                                             ;PARITY DISEBLE
    28                                             ;STOP BITS = 2 BITS
    29                    ;
    30 000F 6906                  MVI     A,06H    ;BOW LATE = 4800
    31 0011 4DDA                  MOV     TM0,A
    32 0013 6901                  MVI     A,1
    33 0015 4DDB                  MOV     TM1,A
    34                    ;
    35 0017 648561                MVI     TMM,01100001B
    36                    ;
    37 001A 64810C                MVI     SMH,00001100B
    38                    ;
    39                    ;       ;       ;       ;       ;       ;       ;
    40                    ;***********************************************
    41                    ;*       MAIN ROUTINE                          *
    42                    ;***********************************************
    43                    ;
    44                    MAIN1:
    45 001D 68FF                  MVI     V,0FFH
    46 001F 040000                LXI     SP,STACK         ;SET STACK
    47                    ;
    48 0022 3437FF                LXI     H,CHARST         ;SET CHAR. START ADDR.
    49 0025 6B4F                  MVI     C,80-1   ;SET BUFFER SIZE
    50                    ;
    51                    MAIN2:
    52 0027 409301                CALL    CI       ;GET 1 CHAR. FROM CONSOLE
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE

              (                                        )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

    53                      ;
    54 002A 409E01              CALL    CO          ;PRINT IT
    55                      ;
    56 002D 3D                  STAX    H+          ;SET IT TO BUFFER
    57 002E 673D                NEI     A,'='
    58 0030 C4                  GJMP    MAIN3       ;IF CHAR.= '='
    59 0031 53                  DCR     C           ;DECREMENT BUFFER SIZE
    60 0032 F4                  GJMP    MAIN2       ;LOOP UNTILL '='
    61 0033 4E61                GJMP    ERROR       ;IF BUFFER OVER
    62                      ;
    63                  MAIN3:
    64 0035 3437FF              LXI     H,CHARST
    65 0038 703E24FF            SHLD    ADDRS
    66 003C 40B900              CALL    EXPR        ; GET ! / LEFT EXPR
    67                          ;
    68 003F 408202              CALL    CHARD
    69 0042 772C                EQI     A,','
    70 0044 C3                  GJMP    MAIN31
    71                          ;
    72 0045 693C                MVI     A,'<'
    73 0047 CD                  GJMP    MAIN33
    74                          ;
    75 0048 673C        MAIN31: NEI     A,'<'
    76 004A C5                  GJMP    MAIN32
    77                          ;
    78 004B 408901              CALL    ADADJ
    79 004E 4E3D                GJMP    MAIN4
    80                  MAIN32:
    81 0050 692C                MVI     A,','
    82 0052 341F08              LXI     H,TRACA
    83                  MAIN33:
    84 0055 345508              LXI     H,TRACB
    85 0058 B0                  PUSH    V
    86 0059 147100              LXI     B,MAIN40
    87 005C B1                  PUSH    B
    88 005D B3                  PUSH    H           ; PUSH JUMP ADDR
    89                          ;
    90 005E 40390B              CALL    PUSHF       ; SAVE LEFT EXPR
    91 0061 40B900              CALL    EXPR        ; GET ! / RIGHT EXPR
    92 0064 40490B              CALL    POPR        ; RESTORE LEFT EXPR
    93 0067 40F00A              CALL    STORE1      ; SAVE RIGHT EXPR
    94 006A 40060B              CALL    LOADA0      ; GET ! / LEFT EXPR TO B.F.P.ACC
    95                          ;
    96 006D 340BFF              LXI     H,WXE
    97 0070 B8                  RET                 ; JUMP TRACA OR TRACB
    98                          ;
    99 0071 4E23        MAIN40: GJMP    ERROR
   100                          ;
   101 0073 403C0B              CALL    PUSHR
   102 0076 40A700              CALL    OUT
   103 0079 40490B              CALL    POPR
   104 007C 40060B              CALL    LOADA0      ; GET !
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    3
                (                                                )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   105                              ;
   106 007F 6920                    MVI     A,' '
   107 0081 409E01                  CALL    CO
   108 0084 A0                      POP     V
   109 0085 409E01                  CALL    CO
   110 0088 6920                    MVI     A,' '
   111 008A 409E01                  CALL    CO
   112                              ;
   113 008D 40A700          MAIN4:  CALL    OUT
   114                      ;
   115 0090 408202                  CALL    CHARD   ;ICHAR.READ
   116 0093 673D                    NEI     A,'='
   117 0095 C5                      GJMP    MAIN6   ;IF '='
   118                      ;
   119                      MAIN5:
   120                      ERROR:
   121 0096 693F                    MVI     A,'?'
   122 0098 409E01                  CALL    CO      ;
   123                      ;
   124                      MAIN6:
   125 009B 690D                    MVI     A,0DH
   126 009D 409E01                  CALL    CO      ;PRINT CR
   127 00A0 690A                    MVI     A,0AH
   128 00A2 409E01                  CALL    CO      ;PRINT LF
   129                      ;
   130 00A5 4F76                    GJMP    MAIN1   ;LOOP NEXT ARITHMETIC
   131                      ;
   132                      ;  OUT  SUBROUTINE
   133                      ;
   134 00A7 342AFF          OUT:    LXI     H,OUTADR
   135 00AA 408C02                  CALL    BFOUT
   136                              ;
   137 00AD 342AFF                  LXI     H,OUTADR
   138 00B0 6B0C                    MVI     C,13-1
   139                      OUT1:
   140 00B2 2D                      LDAX    H+
   141 00B3 409E01                  CALL    CO
   142 00B6 53                      DCR     C
   143 00B7 FA                      GJMP    OUT1
   144 00B8 B8                      RET
   145                      ;
   146                              EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE     4
                (                                                )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

    147                    ;
    148                    ;================================================
    149                    ;=       EXPR  ROUTINE                         =
    150                    ;================================================
    151                    ;
    152                    EXPR:
    153 00B9 40E000                CALL     TERM     ;GET LEFT TERM
    154                    ;
    155                    EXPR1:
    156 00BC 408202                CALL     CHARD    ;GET CHAR.
    157 00BF 772B                  EQI      A,'+'
    158 00C1 C7                    GJMP     EXPR2    ;IF NOT PLUS
    159                    ;
    160                    ;       IF PLUS
    161                    ;
    162 00C2 40390B                CALL     PUSHF    ;PUSH LEFT TERM
    163 00C5 40E000                CALL     TERM     ;GET RIGHT TERM
    164 00C8 CD                    GJMP     EXPR3
    165                    ;
    166                    EXPR2:
    167 00C9 772D                  EQI      A,'-'
    168 00CB 4EBC                  GJMP     ADADJ    ;GOTO ADDR.ADJUST AND RETURN
    169                    ;
    170                    ;       IF MINUS
    171                    ;
    172 00CD 40390B                CALL     PUSHF    ;PUSH LEFT TERM
    173 00D0 40E000                CALL     TERM     ;GET RIGHT TERM
    174 00D3 40050A                CALL     BFCOMP   ;COMPRIMENT IT.S SIGN
    175                    ;
    176                    EXPR3:
    177 00D6 40490B                CALL     POPR     ;POP LEFT TERM
    178 00D9 409203                CALL     BFADD1   ;LEFT TERM + RIGHT TERM
    179 00DC 4FB8                  GJMP     ERROR    ;IF OVER FLOW
    180 00DE 4FDC                  GJMP     EXPR1    ;GOTO NEXT EXPR
    181                    ;
    182                    ;       ;        ;        ;       ;       ;
    183                    ;================================================
    184                    ;=       TERM ROUTINE                          =
    185                    ;================================================
    186                    ;
    187                    TERM:
    188 00E0 401801                CALL     FACTR    ;GET LEFT FACTOR
    189 00E3 00                    NOP
    190                    ;
    191                    TERM1:
    192 00E4 408202                CALL     CHARD    ;READ CHAR.
    193 00E7 772A                  EQI      A,'*'
    194 00E9 D0                    GJMP     TERM3    ;IF NOT '*'
    195                    ;
    196                    ;       IF MULTIPLICATION
    197                    ;
    198                    TERM2:
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    5

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   199 00EA 40390B              CALL     PUSHF    ;PUSH LEFT FACTOR
   200 00ED 401801              CALL     FACTR    ;GET RIGHT FACTOR
   201 00F0 00                  NOP
   202                          ;
   203 00F1 40490B              CALL     POPR     ;POP  LEFT FACTOR
   204 00F4 402F04              CALL     BFMUL1   ;LEFT FACTOR * RIGHT FACTOR
   205 00F7 4F9D                GJMP     ERROR    ;IF OVER FLOW
   206 00F9 EA                  GJMP     TERM1    ;GOTO NEXT CHARACTER
   207                  ;
   208                  TERM3:
   209 00FA 772F                EQI      A,'/'
   210 00FC 4E8B                GJMP     ADADJ    ;GOTO ADDR.ADJUST AND RETURN
   211                  ;
   212                  ;       IF DIVISION
   213                  ;
   214 00FE 40390B              CALL     PUSHF    ;PUSH LEFT FACTOR
   215 0101 401801              CALL     FACTR    ;GET RIGHT FACTOR
   216 0104 00                  NOP
   217                          ;
   218 0105 40F00A              CALL     STORE1   ;STORE RIGHT FACTOR
   219 0108 40490B              CALL     POPR
   220 010B 40060B              CALL     LOADA0   ;RESTOE BFP ACC
   221 010E 340BFF              LXI      H,WXE
   222 0111 40A304              CALL     BFDIV
   223 0114 4F80                GJMP     ERROR    ;IF OVER FLOW
   224 0116 4FCC                GJMP     TERM1    ;GOTO NEXT TERM
   225                  ;
   226                  ;       ;        ;        ;        ;        ;
   227                  ;=============================================
   228                  ;=      FACTOR  ROUTINE                      =
   229                  ;=============================================
   230                  ;
   231                  FACTR:
   232 0118 408202              CALL     CHARD    ;CHAR. READ
   233 011B 6720                NEI      A,' '
   234 011D FA                  GJMP     FACTR    ;READ UNTILL NOT SPACE
   235                  ;
   236 011E 408901              CALL     ADADJ    ;DECR. ADDRESS
   237                  ;
   238 0121 245E0B              LXI      D,FTABL
   239                  FACTR1:
   240 0124 B3                  PUSH     H
   241                  ;
   242                  FACT11:
   243 0125 408202              CALL     CHARD    ;CHAR.READ FROM THE STRING
   244 0128 1A                  MOV      B,A
   245 0129 2A                  LDAX     D
   246 012A 077F                ANI      A,7FH
   247                          ;
   248 012C 60FA                EQA      A,B
   249 012E D2                  GJMP     FACTR2   ;IF NOT MUCH
   250                  ;
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE     6

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   251 012F 2C                      LDAX    D+
   252 0130 4780                    ONI     A,80H
   253 0132 F2                      GJMP    FACT11   ;SEACH UNTILL TABLE END
   254                      ;
   255                      ;       IF FOUND FUNCTION
   256                      ;
   257 0133 A3                      POP     H        ;DUMMY POP
   258                              ;
   259 0134 2C                      LDAX    D+
   260 0135 1F                      MOV     L,A
   261 0136 2C                      LDAX    D+
   262 0137 1E                      MOV     H,A
   263 0138 145F01                  LXI     B,FACT31
   264 013B B1                      PUSH    B
   265 013C B3                      PUSH    H
   266                              ;
   267 013D 401801                  CALL    FACTR
   268 0140 B8                      RET              ; JUMP THE FUNCTION
   269                              ;
   270                      ;
   271                      FACTR2:
   272 0141 6980                    MVI     A,80H
   273 0143 70CC                    ONAX    D+
   274 0145 FB                      GJMP    FACTR2   ;LOOP UNTILL THE TABLE END
   275 0146 22                      INX     D
   276 0147 22                      INX     D        ;
   277 0148 A3                      POP     H        ;POP CHAR.STRING ADDR.
   278 0149 703E24FF                SHLD    ADDRS
   279 014D 2A                      LDAX    D        ;GET NEXT TABLE
   280 014E 51                      DCR     A
   281 014F 4FD3                    GJMP    FACTR1   ;SEACH NEXT FANCTION TABLE
   282                      ;                        ;IF END FUNCTION TABLE
   283                      ;
   284                      FACTR3:
   285 0151 408202                  CALL    CHARD    ;CHAR. READ FROM STRING
   286 0154 7728                    EQI     A,'('
   287 0156 CB                      GJMP    FACTR4
   288 0157 40B900                  CALL    EXPR
   289 015A 408202                  CALL    CHARD    ;CHAR.READ FROM STRING
   290 015D 7729                    EQI     A,')'
   291 015F 4F35            FACT31: GJMP    ERROR    ;IF NESTING ERROR
   292 0161 C8                      GJMP    FACTR5
   293                      ;
   294                      FACTR4:
   295 0162 408901                  CALL    ADADJ    ;DECR. STRING ADDR.
   296 0165 40B601                  CALL    BFINP
   297 0168 4F2C                    GJMP    ERROR    ;IF OVER FLOW
   298 016A 408202          FACTR5: CALL    CHARD    ;CHAR. READ FROM STRING
   299 016D 775E                    EQI     A,5EH    ; POWER FUNCTION ?
   300 016F D9                      GJMP    ADADJ    ;IF NOT POWER THEN
   301                      ;
   302                      ;       IF POWER FUNCTION
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE     7
                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   303                     ;
   304 0170 40390B               CALL    PUSHF   ;
   305 0173 401801               CALL    FACTR   ;
   306 0176 00                   NOP
   307                           ;
   308 0177 40F00A               CALL    STORE1  ;
   309 017A 40490B               CALL    POPR
   310 017D 40060B               CALL    LOADA0  ;RESTORE BFP ACC
   311 0180 340BFF               LXI     H,WXE
   312 0183 40D206               CALL    POWER   ;
   313 0186 4F0E                 GJMP    ERROR   ;IF OVER FLOW
   314 0188 B8                   RET
   315                     ;
   316                           EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE     8

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   317                    ;
   318                    ;=============================================
   319                    ;=       ADDRESS  ADJUST  ROUTINE             =
   320                    ;=============================================
   321                    ;
   322                    ADADJ:
   323 0189 703F24FF              LHLD    ADDRS
   324 018D 33                    DCX     H
   325 018E 703E24FF              SHLD    ADDRS
   326 0192 B8                    RET
   327                    ;
   328                    ;       ;       ;       ;       ;       ;       ;
   329                    ;=============================================
   330                    ;=       1 CHAR. INPUT FROM CONSOLE           =
   331                    ;=============================================
   332                    ;
   333                    CI:
   334 0193 4849                  SKIT    FSR
   335 0195 FD                    GJMP    CI      ;LOOP UNTILL FSR
   336                    ;
   337 0196 486B                  SKNIT   ER
   338 0198 FA                    GJMP    CI      ;RETRY IF ERROR
   339                    ;
   340 0199 4CD9                  MOV     A,RXB   ;GET 1 CHAR.
   341 019B 077F                  ANI     A,7FH
   342 019D B8                    RET
   343                    ;
   344                    ;       ;       ;       ;       ;       ;       ;
   345                    ;=============================================
   346                    ;=       1 CHAR. OUTPUT TO CONSOLE            =
   347                    ;=============================================
   348                    ;
   349                    CO:
   350 019E 645A08                OFFI    PC,CTS
   351 01A1 FC                    GJMP    CO      ;LOOP UNTILL /CTS = 0
   352                    ;
   353 01A2 4DD8                  MOV     TXB,A
   354                    ;
   355                    CO1:
   356 01A4 484A                  SKIT    FST
   357 01A6 FD                    GJMP    CO1     ;LOOP UNTILL FST = 1
   358                    ;
   359 01A7 B2                    PUSH    D
   360 01A8 B0                    PUSH    V
   361 01A9 242003                LXI     D,800
   362                    CO2:
   363 01AC 23                    DCX     D
   364 01AD 0C                    MOV     A,D
   365 01AE 609D                  ORA     A,E
   366 01B0 480C                  SK      Z
   367 01B2 F9                    GJMP    CO2
   368                    ;

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   369 01B3 A0                    POP     V
   370 01B4 A2                    POP     D
   371                    ;
   372 01B5 B8                    RET
   373                    ;
   374                            EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   10

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  375                     ;
  376                     ;*************************************************
  377                     ;*                                               *
  378                     ;*       BFINP  SUBROUTINE                       *
  379                     ;*                                               *
  380                     ;*        IN: HL REG <-- STRING START ADDR.      *
  381                     ;*                                               *
  382                     ;*        OUT:  BFP. ACC                         *
  383                     ;*              RETS:   NORMAL   CY=0            *
  384                     ;*                   RET:  OVERFLOW   CY=1       *
  385                     ;*                                               *
  386                     ;*************************************************
  387                     ;
  388                     BFINP:
  389 01B6 703E24FF               SHLD    ADDRS   ;SAVE STRING ADDR.
  390                     ;
  391 01BA 6900                   MVI     A,0
  392 01BC 6300                   STAW    ACCE MOD 100H   ;CLEAR BFP.ACC
  393 01BE 2426FF                 LXI     D,TMP1
  394 01C1 3C                     STAX    D+      ;TMP1 = 0
  395 01C2 3C                     STAX    D+      ;TMP2 = 0
  396 01C3 4A80                   MVIX    D,80H   ;TMP3 <-- 80H
  397                     ;
  398 01C5 408202                 CALL    CHARD   ;GET CHAR.TO ACC
  399 01C8 6720                   NEI     A,' '
  400 01CA C8                     GJMP    BFINP1  ;IF SPACE
  401 01CB 672B                   NEI     A,'+'
  402 01CD C5                     GJMP    BFINP1  ;IF PLUS
  403 01CE 772D                   EQI     A,'-'
  404 01D0 C5                     GJMP    BFIN11
  405 01D1 4A00                   MVIX    D,0     ;IF MINUS THEN TMP3 = 0
  406                     ;
  407                     BFINP1:
  408 01D3 408202                 CALL    CHARD   ;CHAR.READ
  409                     ;
  410 01D6 6A00           BFIN11: MVI     B,0     ;CLEAR EXP
  411                     ;
  412 01D8 772E                   EQI     A,'.'
  413 01DA CA                     GJMP    BFIN13  ;IF NOT DECIMAL POINT
  414                     ;
  415                     BFIN12:                 ;IF DECIMAL POINT FOUND
  416 01DB 749027                 XRAW    TMP2 MOD 100H
  417 01DE 6327                   STAW    TMP2 MOD 100H
  418 01E0 480C                   SK      Z
  419 01E2 F0                     GJMP    BFINP1  ;GOTO NEXT STRING
  420 01E3 4E75                   GJMP    BFINP4  ;IF DUBLE DECIMAL POINT
  421                     ;
  422                     BFIN13:
  423 01E5 6745                   NEI     A,'E'
  424 01E7 4E39                   GJMP    BFINP3  ;IF EXP
  425                     ;
  426                     BFIN14:
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    11

                (                                    )

E STNO ADRS OBJECT   M   SOURCE STATEMENT

   427 01E9 6630             SUI     A,'0'
   428 01EB 370A             LTI     A,10
   429 01ED 4E6B             GJMP    BFINP4  ;IF NOT DECIMAL DEGIT
   430                  ;
   431                  BFINP2:
   432 01EF 6329             STAW    TMP4 MOD 100H   ;SAVE DIGIT
   433 01F1 34C908           LXI     H,BFTEN
   434 01F4 402604           CALL    BFMUL   ;PRE.RESULT * 10
   435 01F7 54560B           GJMP    BFRET   ;IF OVER FLOW
   436 01FA 40F00A           CALL    STORE1  ;STORE PRE.RESULT
   437 01FD 3406FF           LXI     H,WSE
   438 0200 6908             MVI     A,8
   439 0202 3D               STAX    H+      ;WSE = 8
   440 0203 0129             LDAW    TMP4 MOD 100H
   441 0205 3D               STAX    H+      ;WS1 = DIGIT
   442 0206 6900             MVI     A,0
   443 0208 3D               STAX    H+      ;WS2 = 0
   444 0209 3D               STAX    H+      ;WS3 = 0
   445 020A 3D               STAX    H+      ;WS4 = 0
   446 020B 40970A           CALL    BFLT    ;GET NEW DIGIT TO BFP.ACC
   447 020E 00               NOP
   448 020F 340BFF           LXI     H,WXE
   449 0212 408A03           CALL    BFADD   ;ADD PRE RESULT TO NEW DIGIT
   450 0215 54560B           GJMP    BFRET   ;IF OVER FLOW
   451 0218 652700           NEIW    TMP2 MOD 100H,0
   452 021B 4FB6             GJMP    BFINP1  ;GOTO NEXT STRING
   453 021D 3026             DCRW    TMP1 MOD 100H
   454 021F 00               NOP
   455 0220 4FB1             GJMP    BFINP1  ;GOTO NEXT STRING
   456                  ;
   457                  BFINP3:
   458 0222 6D2B             MVI     E,'+'
   459 0224 408202           CALL    CHARD   ;CHAR. READ
   460 0227 672D             NEI     A,'-'
   461 0229 C3               GJMP    BFIN31  ;IF MINUS
   462 022A 772B             EQI     A,'+'
   463 022C C4               GJMP    BFIN32  ;IF NOT PLUS
   464                  BFIN31:
   465 022D 1D               MOV     E,A     ;SEVE EXP SIGN
   466 022E 408202           CALL    CHARD   ;MORE 1 CHAR. READ
   467                  BFIN32:
   468 0231 6A00             MVI     B,0     ;CLEAR EXP DIGIT
   469 0233 6630             SUI     A,'0'   ;DIGIT IS 0 -- 9 ?
   470 0235 370A             LTI     A,10
   471 0237 4E21             GJMP    BFINP4  ;IF ILEAGAL CHAR.
   472                  ;
   473 0239 1A               MOV     B,A
   474 023A 408202           CALL    CHARD   ;GET LOW DIGIT
   475 023D 6630             SUI     A,'0'   ;IT IS 0 -- 9 ?
   476 023F 370A             LTI     A,10
   477 0241 CB               GJMP    BFIN33  ;IF ILLEGAL CHAR.
   478 0242 B0               PUSH    V       ; PUSH IT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   12
                (                                                )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   479 0243 690A                MVI     A,10
   480 0245 482E                MUL     B          ;B= B * 10
   481 0247 09                  MOV     A,EAL
   482 0248 1A                  MOV     B,A        ;B = B * 10
   483 0249 A0                  POP     V          ;POP LOW DIGIT
   484 024A 6042                ADD     B,A        ;B = B + ACC
   485 024C C3                  GJMP    BFIN30
   486                    ;
   487                    BFIN33:
   488 024D 408901              CALL    ADADJ      ;ADDRESS ADJUST
   489                    BFIN30:
   490 0250 747D2D              EQI     E,'-'
   491 0253 C9                  GJMP    BFIN40     ;IF NOT MINUS
   492 0254 6900                MVI     A,0        ;IF MINUS EXP SIGN
   493 0256 60E2                SUB     A,B
   494 0258 1A                  MOV     B,A        ;ADJUST EXP SIGN
   495 0259 C3                  GJMP    BFIN40
   496                    ;
   497                    BFINP4:
   498 025A 408901              CALL    ADADJ      ;ADDRESS ADJUST
   499                    BFIN40:
   500 025D 0128                LDAW    TMP3 MOD 100H
   501 025F 6301                STAW    ACCS MOD 100H    ;ACCS <-- TMP3
   502 0261 0A                  MOV     A,B              ;GET EXP SIGN
   503                    BFIN41:
   504 0262 74C026              ADDW    TMP1 MOD 100H
   505 0265 481C                SKN     Z
   506 0267 54590B              GJMP    BFRETS     ;IF END
   507                    ;
   508 026A 6326                STAW    TMP1 MOD 100H
   509 026C 34C908              LXI     H,BFTEN
   510 026F 4780                ONI     A,80H
   511 0271 C7                  GJMP    BFIN43
   512                    ;
   513                    BFIN42:
   514 0272 40A304              CALL    BFDIV
   515 0275 00                  NOP
   516 0276 6901                MVI     A,1        ;INCREMENT TMP1
   517 0278 E9                  GJMP    BFIN41
   518                    ;
   519                    BFIN43:
   520 0279 402604              CALL    BFMUL
   521 027C 54560B              GJMP    BFRET      ;IF OVER FLOW
   522 027F 69FF                MVI     A,0FFH     ;DECREMENT TMP1
   523 0281 E0                  GJMP    BFIN41
   524                    ;
   525                    ;       ;       ;       ;       ;       ;       ;
   526                    ;=============================================
   527                    ;=      STRING CHARACTER READ ROUTINE          =
   528                    ;=============================================
   529                    ;
   530                    CHARD:

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   531 0282 703F24FF            LHLD    ADDRS
   532 0286 2D                  LDAX    H+
   533 0287 703E24FF            SHLD    ADDRS
   534 028B B8                  RET
   535                    ;
   536                          EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   14

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

   537                    ;
   538                    ;*************************************************
   539                    ;*                                               *
   540                    ;*       BFOUT ROUTINE                           *
   541                    ;*                                               *
   542                    ;*              IN : STORE START ADDRESS         *
   543                    ;*                                               *
   544                    ;*              OUT : NONE                       *
   545                    ;*                                               *
   546                    ;*************************************************
   547                    ;
   548                    BFOUT:
   549 028C 6920                  MVI     A,' '
   550 028E 3D                    STAX    H+      ;SET PLUS SIGN
   551                    ;
   552 028F 750000                EQIW    ACCE MOD 100H,0
   553 0292 CE                    GJMP    BFOU01  ;IF BFP ACC <> 0
   554                    ;
   555 0293 6930                  MVI     A,'0'
   556 0295 3D                    STAX    H+
   557 0296 4B2E                  MVIX    H,'.'
   558 0298 32                    INX     H
   559 0299 6805                  MVI     C,6-1
   560                    BFOU00:
   561 029B 3D                    STAX    H+
   562 029C 53                    DCR     C
   563 029D FD                    GJMP    BFOU00
   564                    ;
   565 029E B3                    PUSH    H
   566 029F 4EA6                  GJMP    BFOU30
   567                    ;
   568                    BFOU01:
   569 02A1 6900                  MVI     A,0
   570 02A3 6326                  STAW    TMP1 MOD 100H
   571 02A5 6327                  STAW    TMP2 MOD 100H
   572 02A7 750100                EQIW    ACCS MOD 100H , 0
   573 02AA C4                    GJMP    BFOU02  ;IF BFP ACC > 0
   574                    ;
   575 02AB 692D                  MVI     A,'-'   ;IF BFP ACC < 0
   576 02AD 33                    DCX     H
   577 02AE 3D                    STAX    H+
   578                    ;
   579                    BFOU02:
   580 02AF B3                    PUSH    H       ;SAVE STRING ADDR.
   581 02B0 40320A                CALL    BFABS   ;ABSOLUTE BFP ACC
   582 02B3 40F00A                CALL    STORE1  ;SAVE BFP. ACC
   583                    ;
   584                    ;       SCALING 0.1 -- 1
   585                    ;
   586                    BFOUT1:
   587 02B6 350081                LTIW    ACCE MOD 100H,81H
   588 02B9 4E34                  GJMP    BFOU12          ;IF BFP ACC >= 1 GOTO DIV
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    15

                (                                                )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   589                      ;
   590 02BB 40390B               CALL    PUSHF   ;SAVE BFP ACC
   591 02BE 34D108               LXI     H,BFN01 ;SET 0.1
   592 02C1 407D03               CALL    BFSUB   ;BFP ACC - 0.1
   593 02C4 00                   NOP
   594 02C5 40490B               CALL    POPR    ;RESTORE BFP ACC TO OPERAND
   595 02C8 650000               NEIW    ACCE MOD 100H , 0
   596 02CB C4                   GJMP    BFOU03  ;IF BFP ACC = 0.1
   597 02CC 750180               EQIW    ACCS MOD 100H ,80H
   598 02CF D0                   GJMP    BFOU11  ;IF BFP ACC < 0.1 GOTO MULT
   599                      BFOU03:
   600 02D0 40060B               CALL    LOADA0  ;RESTORE BFP ACC
   601 02D3 34CD08               LXI     H,BFDV58
   602 02D6 408A03               CALL    BFADD   ;ROUNDINDG
   603 02D9 00                   NOP
   604 02DA 350081               LTIW    ACCE MOD 100H , 81H
   605 02DD 4FD7                 GJMP    BFOUT1  ;ONE MORE TRY SCALING
   606 02DF DB                   GJMP    BFOUT2
   607                      ;
   608                      BFOU11:                 ;IF 0.1 > BFP.ACC
   609 02E0 40060B               CALL    LOADA0  ;LOAD BFP. ACC
   610 02E3 34C908               LXI     H,BFTEN
   611 02E6 402604               CALL    BFMUL
   612 02E9 00                   NOP
   613 02EA 3027                 DCRW    TMP2 MOD 100H   ;TMP2 <-- TMP2 -1
   614 02EC 00                   NOP
   615 02ED 4FC7                 GJMP    BFOUT1
   616                      ;
   617                      BFOU12:                 ;IF 1 <= BFP. ACC
   618 02EF 34C908               LXI     H,BFTEN
   619 02F2 40A304               CALL    BFDIV
   620 02F5 00                   NOP
   621 02F6 2027                 INRW    TMP2 MOD 100H   ;TMP2 <-- TMP2+1
   622 02F8 00                   NOP
   623 02F9 4FBB                 GJMP    BFOUT1
   624                      ;
   625                      BFOUT2:
   626 02FB 0127                 LDAW    TMP2 MOD 100H
   627 02FD 1A                   MOV     B,A
   628 02FE 3708                 LTI     A,8
   629 0300 6A01                 MVI     B,1
   630 0302 60E2                 SUB     A,B     ;GET DECIMAL EXP.
   631 0304 6327                 STAW    TMP2 MOD 100H   ;STORE IT
   632 0306 6907                 MVI     A,7
   633 0308 60E2                 SUB     A,B     ;GET RIGHT DIGIT
   634 030A 6328                 STAW    TMP3 MOD 100H   ;STORE IT
   635 030C 0A                   MOV     A,B     ;GET LEFT DIGIT NUMBER
   636 030D 6326                 STAW    TMP1 MOD 100H   ;STORE IT
   637                      ;
   638                      BFOU21:
   639 030F 3026                 DCRW    TMP1 MOD 100H
   640 0311 D2                   GJMP    BFOU22
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   16

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

    641 0312 0128                LDAW    TMP3 MOD 100H
    642 0314 6326                STAW    TMP1 MOD 100H   ;TMP1 <-- TMP3
    643 0316 7128FF              MVIW    TMP3 MOD 100H ,0FFH ;TMP3 <-- 0FFH
    644 0319 6526FF              NEIW    TMP1 MOD 100H ,0FFH
    645 031C 4E24                GJMP    BFOUT3  ;IF MANTISSA END
    646 031E A3                  POP     H       ;GET STORE ADDRESS
    647 031F 692E                MVI     A,'.'
    648 0321 3D                  STAX    H+
    649 0322 B3                  PUSH    H
    650 0323 EB                  GJMP    BFOU21  ;LOOP UNTILL MANTISSA END
    651                  ;
    652                  ;       GET 1 DECIMAL DIGIT
    653                  ;
    654                  BFOU22:
    655 0324 34C908              LXI     H,BFTEN
    656 0327 402604              CALL    BFMUL
    657 032A 00                  NOP
    658 032B 405C0A              CALL    BFIX0   ;CONVERT FIXED POINT NUMBER
    659 032E 00                  NOP
    660 032F 0107                LDAW    WS1 MOD 100H    ;GET 1 DIGIT
    661 0331 4630                ADI     A,'0'
    662 0333 A3                  POP     H
    663 0334 3D                  STAX    H+      ;STORE DECIMAL DIGIT
    664 0335 B3                  PUSH    H
    665 0336 710608              MVIW    WSE MOD 100H , 8
    666 0339 710700              MVIW    WS1 MOD 100H , 0
    667 033C 40970A              CALL    BFLT    ;CONVERT BFP.NUMBER
    668 033F 00                  NOP
    669 0340 4FCD                GJMP    BFOU21  ;GOTO NEXT DIGIT
    670                  ;
    671                  BFOUT3:
    672 0342 0127                LDAW    TMP2 MOD 100H
    673 0344 7700                EQI     A,0
    674 0346 C7                  GJMP    BFOU31
    675                  BFOU30:
    676 0347 6920                MVI     A,' '
    677 0349 1A                  MOV     B,A
    678 034A 1B                  MOV     C,A
    679 034B 1C                  MOV     D,A
    680 034C 4E22                GJMP    BFOU34
    681                  ;
    682                  BFOU31:
    683 034E 6A2B                MVI     B,'+'
    684 0350 4780                ONI     A,80H
    685 0352 C9                  GJMP    BFOU32
    686 0353 6A2D                MVI     B,'-'
    687 0355 6900                MVI     A,0
    688 0357 74E027              SUBW    TMP2 MOD 100H
    689 035A 6327                STAW    TMP2 MOD 100H
    690                  BFOU32:
    691 035C 6BFF                MVI     C,0FFH
    692                  ;
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   17
                (                                                )
E STNO ADRS OBJECT    M    SOURCE STATEMENT

   693 035E 0127                LDAW    TMP2 MOD 100H
   694                  BFOU33:
   695 0360 1C                  MOV     D,A
   696 0361 43                  INR     C
   697 0362 00                  NOP
   698 0363 660A                SUI     A,10
   699 0365 480A                SK      CY
   700 0367 F8                  GJMP    BFOU33
   701 0368 744330              ADI     C,'0'   ;PLUS BIAS
   702 036B 744430              ADI     D,'0'
   703 036E 6945                MVI     A,'E'
   704                  BFOU34:
   705 0370 A3                  POP     H
   706 0371 3D                  STAX    H+      ;SET 'E'
   707 0372 0A                  MOV     A,B
   708 0373 3D                  STAX    H+      ;SET EXP.SIGN
   709 0374 0B                  MOV     A,C
   710 0375 3D                  STAX    H+      ;SET EXP. HIGH DIGIT
   711 0376 0C                  MOV     A,D
   712 0377 3D                  STAX    H+      ;SET EXP. LOW DIGIT
   713                  ;
   714 0378 40090B              CALL    LOADA1  ;RESTORE BFP. ACC
   715                  ;
   716 037B B8                  RET
   717                  ;
   718                          EJECT


** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   18
E STNO ADRS OBJECT    M    SOURCE STATEMENT

   719                  ;
   720                  ; ***********************************************
   721                  ; *                                             *
   722                  ; *    BINARY FLOATING POINT SUBTRACT SUBROUTINE *
   723                  ; *                                             *
   724                  ; *       B.F.P.ACC < B.F.P.ACC - OPERAND       *
   725                  ; *                                             *
   726                  ; *                                             *
   727                  ; *    INPUT:                                   *
   728                  ; *            V REG -- 0FFH                    *
   729                  ; *                                             *
   730                  ; *            BFSUB  - HL ( OPERAND TOP ADDR.) *
   731                  ; *                                             *
   732                  ; *    OUTPUT: RETS  -  NORMAL    CY=0          *
   733                  ; *            RET   -  OVERFLOW, CY =1         *
   734                  ; *                                             *
   735                  ; ***********************************************
   736                  ;
   737 037C 00                  NOP             ; DUMMY FOR RET
   738 037D 402B0B      BFSUB:  CALL    LOADO3  ; OPE <- (HL,---,HL+3)
   739                          ;
   740 0380 00                  NOP
   741 0381 6980        BFSUB1: MVI     A,80H   ; MASK TO INVERT
   742 0383 749007              XRAW    WS1 MOD 100H
   743 0386 6307                STAW    WS1 MOD 100H
   744                                          ; SIGN STORE
   745 0388 C9                  GJMP    BFADD1  ;FALL INTO BFADD SUBROUTINE
   746                  ;
   747                          EJECT
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   19
                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

   748                    ;
   749                    ; ************************************************
   750                    ; *                                              *
   751                    ; *   BINARY FLOATING POINT ADDITION SUBROUTINE   *
   752                    ; *                                              *
   753                    ; *    B.F.P.ACC <- B.F.P.ACC + OPERAND          *
   754                    ; *                                              *
   755                    ; *   INPUT:                                     *
   756                    ; *            V REG  --  0FFH                   *
   757                    ; *                                              *
   758                    ; *            BFADD    HL  ( OPERAND TOP ADDR)  *
   759                    ; *                                              *
   760                    ; *   OUTPUT: RETS  -  NORMAL     CY= 0          *
   761                    ; *            RET   -  OVERFLOW, CY= 1          *
   762                    ; *                                              *
   763                    ; ************************************************
   764                    ;
   765 0389 00                      NOP              ; DUMMY FOR RETS
   766 038A 402B0B      BFADD:      CALL     LOADO3  ; OPE <- (HL,HL+1,---,HL+3)
   767 038D C4                      GJMP     BFADD1
   768                              ;
   769 038E 00                      NOP
   770 038F 40490B      BFADD0:     CALL     POPR    ; OPE <- STACK
   771                              ;
   772 0392 650600      BFADD1:     NEIW     WSE MOD 100H.0
   773 0395 54590B                  GJMP     BFRETS  ; IF ADDEND=0
   774                    ;
   775                    ; OPE 1ST FRCTN !NON-SIGNED! DISPLAY
   776                    ;
   777 0398 2407FF                  LXI      D,WS1   ; DE= OPE 1ST FRCTN ADDR
   778 039B 2A                      LDAX     D
   779 039C 1A                      MOV      B,A
   780 039D 1780                    ORI      A,80H   ; OPE 1ST FRCTN !NON-SIGNED!
   781 039F 3E                      STAX     D-      ; NON-SIGN STORE
   782                              ;
   783 03A0 6092                    XRA      A,B
   784 03A2 3401FF                  LXI      H,ACCS
   785 03A5 7097                    XRAX     H-
   786                                       ; HL= BFP ACC EXP ADDR
   787 03A7 6305                    STAW     SF MOD 100H
   788                                       ; OPERATION FLAG SET
   789 03A9 2B                      LDAX     H
   790 03AA 6700                    NEI      A,0     ; JUDGE  BFP.ACC=0
   791 03AC 4E69                    GJMP     BFADS   ; IF AUGEND=0
   792                              ;
   793 03AE 050A00                  ANIW     WS4 MOD 100H.0
   794                                       ; 4TH FRCTN  INITIAL 0CLEAR
   795                    ;
   796                    ;  CHECK FOR / B.F.P.ACC < OPERAND  BY EXPONENT
   797                    ;
   798 03B1 70B2                    SUBNBX   D       ; DIFFERENCE OF EXPONENT
   799 03B3 C6                      GJMP     BFAD2   ; IF  ACCE < OPE.WSE
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   20
                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

   800                        ;
   801                        ; CHECK FOR / DIFFERENCE OF EXP. TOO LAGE
   802                        ;
   803 03B4 2719                      GTI     A,24+1
   804 03B6 DC                        GJMP    BFAD3   ; IF DIFFERENCE < 25
   805 03B7 54590B                    GJMP    BFRETS  ; RESULT = BFP.ACC
   806                        ;
   807                        ; CHECK FOR / BFP.ACC = NEGLIGIBLE
   808                        ;
   809 03BA 27E7              BFAD2:  GTI     A,100H-25
   810 03BC 4E59                      GJMP    BFADS   ; BFP.ACC NEGLIGIBLE !
   811                        ;
   812                        ; SET !SIGN OF RESULT!,!DIFER OF EXP!
   813                        ;
   814 03BE 483A                      NEGA            ; EXP COMPLEMENT
   815 03C0 B0                        PUSH    V       ; SAVE EXP
   816                                ;
   817 03C1 2C                        LDAX    D+
   818                                        ; DE= WS1 ADDR
   819 03C2 3D                        STAX    H+
   820                                        ; HL= ACCS ADDR
   821 03C3 0105                      LDAW    SF MOD 100H
   822 03C5 7093                      XRAX    H
   823 03C7 3D                        STAX    H+      ; RESULT.SIGN STORE
   824                                        ; HL = ACC1 ADDR
   825                        ;
   826                        ; EXCHANGE / BFP.ACC AND OPERAND /
   827                        ;
   828 03C8 6B02                      MVI     C,3-1
   829 03CA 2B                EXACOP: LDAX    H
   830 03CB 1A                        MOV     B,A
   831 03CC 2A                        LDAX    D
   832 03CD 3D                        STAX    H+
   833 03CE 0A                        MOV     A,B
   834 03CF 3C                        STAX    D+
   835                                ;
   836 03D0 53                        DCR     C
   837 03D1 F8                        GJMP    EXACOP
   838                        ;
   839                        ; (WS1,2,3,4) RIGHT SHIFT
   840                        ;               FOR EQUALLY OF EXP
   841                        ;
   842 03D2 A0                        POP     V
   843 03D3 40390A            BFAD3:  CALL    BFRSH
   844                        ;
   845                        ;  BRANCHING  OF OPERATION (+,-)
   846                        ;
   847 03D6 3404FF                    LXI     H,ACC3
   848 03D9 2409FF                    LXI     D,WS3
   849                                ;
   850 03DC 550580                    OFFIW   SF MOD 100H,80H
   851 03DF D7                        GJMP    BFAD9   ; IF !SUBTRACT!
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   21
                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   852                    ;
   853                    ;  ADDITION / OPE <- OPE + BFP.ACC
   854                    ;                    (WS1,2,3) + (ACC1,2,3)
   855                    ;
   856 03E0 482A                 CLC
   857 03E2 6B02                 MVI     C,3-1
   858 03E4 2F           BFAD4:  LDAX    H-
   859 03E5 70D2                 ADCX    D
   860 03E7 3E                   STAX    D-
   861                           ;
   862 03E8 53                   DCR     C
   863 03E9 FA                   GJMP    BFAD4
   864                           ;
   865 03EA 480A                 SK      CY      ; OVER FLOW ?
   866 03EC 4E20                 GJMP    BFADR   ; NO CARRY
   867                    ;
   868                    ; RIGHT SFIFT FOR / END AROUND CARRY
   869                    ;
   870 03EE 40460A               CALL    BFRSH4  ; (WS1,2,3,4) SHIFT
   871                           ;
   872 03F1 2000                 INRW    ACCE MOD 100H
   873                                   ; EXP INCREMENT
   874 03F3 DA                   GJMP    BFADR   ; GO TO ROUNDING
   875 03F4 54560B               GJMP    BFRET   ; IF OVER-FLOW
   876                    ;
   877                    ;              OPE. <-- BFP.ACC - OPE.
   878                    ;                       (ACC1,2,3)-(WS1,2,3,4)
   879                    ;
   880 03F7 22           BFAD9:  INX     D
   881 03F8 2E                   LDAX    D-
   882                                   ; DE= WS3 ADDR
   883 03F9 4835                 RAL                ; CARRY FROM MSB OF WS4
   884 03FB 6B02                 MVI     C,3-1
   885                           ;
   886 03FD 2F           BFAD10: LDAX    H-
   887 03FE 70F2                 SBBX    D
   888 0400 3E                   STAX    D-
   889                           ;
   890 0401 53                   DCR     C
   891 0402 FA                   GJMP    BFAD10
   892                           ;
   893 0403 481A         BFAD11: SKN     CY
   894 0405 40F609               CALL    BFCOM1  ; (WS1,2,3,4) COMPLEMENT
   895                    ;
   896                    ; NORMALIZE !
   897                    ;
   898 0408 40BC09               CALL    BFNOR
   899 040B 54590B               GJMP    BFRETS  ; IF ACCE=0
   900                    ;
   901                    ; ROUNDING !
   902                    ;
   903 040E 40D809       BFADR:  CALL    BFROND
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    22

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

   904 0411 54560B              GJMP     BFRET    ; IF OVERFLOW
   905 0414 54590B              GJMP     BFRETS
   906                    ;
   907                    ; TRANS / (ACCE,S,1,2,3) <- (WSE,A,1,2,3)
   908                    ;
   909 0417 0105          BFADS:  LDAW     SF MOD 100H
   910 0419 3401FF              LXI      H,ACCS
   911 041C 7097                XRAX     H-
   912                                   ; HL= ACCE ADDR
   913 041E 1A                  MOV      B,A
   914 041F 40230A              CALL     BFSTR1
   915 0422 54590B              GJMP     BFRETS
   916                          EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   23

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

   917                    ;
   918                    ; ************************************************
   919                    ; *                                              *
   920                    ; *    BINARY FLOATING POINT MULTIPLY SUBROUTINE *
   921                    ; *                                              *
   922                    ; *     B.F.P.ACC <- B.F.P.ACC * OPERAND         *
   923                    ; *                                              *
   924                    ; *    INPUT:                                    *
   925                    ; *                 V REG  --  0FFH              *
   926                    ; *                                              *
   927                    ; *           BFMUL    HL <- OPERAND TOP ADDR    *
   928                    ; *                                              *
   929                    ; *    OUTPUT: RETS  - NORMAL    CY=0            *
   930                    ; *            RET   - OVERFLOW, CY= 1           *
   931                    ; *                                              *
   932                    ; ************************************************
   933                    ;
   934 0425 00                      NOP             ; DUMMY FOR RETS
   935 0426 402B0B      BFMUL:  CALL    LOADO3  ; OPE <- (HL,HL+1,---,HL+3)
   936 0429 C5                      GJMP    BFMUL1
   937                              ;
   938 042A 00                      NOP
   939 042B 40490B      BFMUL0: CALL    POPR    ; OPE <- STACK
   940 042E 00                      NOP
   941 042F 4506FF      BFMUL1: ONIW    WSE MOD 100H.0FFH
   942 0432 DD                      GJMP    BFMULZ  ; IF OPERAND=0
   943                              ;
   944                    ;
   945                    ;  EXPONENT PART OPERATION
   946                    ;
   947 0433 40BF0A                  CALL    BFRD    ; OPERATION OF EXP
   948 0436 54560B                  GJMP    BFRET   ; IF OVERFLOW
   949                              ;
   950 0439 481C                    SKN     Z
   951 043B 54590B                  GJMP    BFRETS  ; IF UNDERFLOW OR RESULT=0
   952                              ;
   953                    ;
   954                    ;  (ACC1,2,3) * (WS1,2,3)
   955                    ;
   956 043E 405604                  CALL    BFMX
   957                    ;
   958                    ;  NORMALIZE
   959                    ;
   960 0441 40BC09                  CALL    BFNOR
   961 0444 54590B                  GJMP    BFRETS  ; RESULT= 0
   962                    ;
   963                    ;  ROUNDING
   964                    ;
   965 0447 40D809                  CALL    BFROND
   966 044A 54560B                  GJMP    BFRET   ; IF OVERFLOW
   967 044D 54590B                  GJMP    BFRETS  ; NORMAL RESULT
   968                    ;
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   24

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

   969                    ;  RESULT=0 /  PROCESS
   970                    ;
   971 0450 050000        BFMULZ: ANIW    ACCE MOD 100H,0
   972 0453 54590B                GJMP    BFRETS
   973                    ;
   974                    ;  MULTIPLY  (WS1,2,3) <- (WS1,2,3) * (ACC1,2,3)
   975                    ;
   976 0456 6B02          BFMX:   MVI     C,3-1
   977 0458 3407FF                LXI     H,WS1
   978 045B 2419FF                LXI     D,WA1
   979                            ;  (WA1,2,3) <- (WS1,2,3)
   980                            ;  (WS1,2,3)  0CLEAR
   981 045E 2B            BFMX1:  LDAX    H
   982 045F 1A                    MOV     B,A
   983 0460 6091                  XRA     A,A
   984 0462 3D                    STAX    H+
   985 0463 0A                    MOV     A,B
   986 0464 3C                    STAX    D+
   987                            ;
   988 0465 53                    DCR     C
   989 0466 F7                    GJMP    BFMX1
   990                    ;
   991                    ;  (WS1,2,3) * B.F.P.ACC -1BYTE / 3TH OPERATE
   992                    ;
   993 0467 711E02                MVIW    CMULT0 MOD 100H,3-1
   994 046A 3404FF                LXI     H,ACC3
   995 046D 2F            BFMX2:  LDAX    H-
   996 046E 1B                    MOV     C,A
   997                            ; C: B.F.P.ACC 1BYTE
   998 046F B3                    PUSH    H
   999 0470 407804                CALL    BFMX3
  1000 0473 A3                    POP     H
  1001                            ;
  1002 0474 301E                  DCRW    CMULT0 MOD 100H
  1003 0476 F6                    GJMP    BFMX2
  1004 0477 B8                    RET
  1005                            ;
  1006                   ;
  1007                   ;  (3BYTE/ WA1,2,3) * (1BYTE/ B.F.P.ACC)
  1008                   ;
  1009 0478 408304       BFMX3:  CALL    MULT
  1010 047B B0                   PUSH    V
  1011                   ;
  1012                   ;  RIGHT SHIFT / (WS2,3,4) <- (WS1,2,3)
  1013                   ;
  1014 047C 40370A               CALL    BFRSH0
  1015 047F A0                   POP     V
  1016 0480 6307                 STAW    WS1 MOD 100H
  1017 0482 B8                   RET
  1018                   ;
  1019                   ; MULTIPLY /  (3BYTE:WA1,2,3) * 1BYTE
  1020                   ;
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE   25

                (                                      )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1021 0483 6A00        MULT:   MVI     B,0      ; INITIAL 0CLEAR
  1022 0485 711F02              MVIW    CMULT1 MOD 100H,3-1
  1023                                  ; MULT COUNTER
  1024 0488 341BFF              LXI     H,WA3
  1025 048B 2409FF              LXI     D,WS3
  1026                          ;
  1027 048E 409504      MULT1:  CALL    MULT2
  1028                          ;
  1029 0491 301F                DCRW    CMULT1 MOD 100H
  1030 0493 FA                  GJMP    MULT1
  1031 0494 B8                  RET                ; MULTIPLY END !
  1032                          ;
  1033 0495 2F          MULT2:  LDAX    H-
  1034 0496 482F                MUL     C
  1035 0498 7042                EADD    EA,B
  1036                  ;
  1037 049A 2A                  LDAX    D
  1038 049B 7041                EADD    EA,A
  1039                  ;        .
  1040 049D 09                  MOV     A,EAL
  1041 049E 3E                  STAX    D-
  1042 049F 08                  MOV     A,EAH
  1043 04A0 1A                  MOV     B,A
  1044 04A1 B8                  RET
  1045                  ;
  1046                          EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   26

                (                                      )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1047                    ;
  1048                    ; ************************************************
  1049                    ; *                                              *
  1050                    ; *   BINARY FLOATING POINT DIVISION SUBROUTINE   *
  1051                    ; *                                              *
  1052                    ; *    B.F.P.ACC <- B.F.P.ACC / OPERAND          *
  1053                    ; *                                              *
  1054                    ; *                                              *
  1055                    ; *    INPUT:                                    *
  1056                    ; *              V REG  --  0FFH                 *
  1057                    ; *                                              *
  1058                    ; *              BFDIV    HL ( OPERAND TOP ADDR.)*
  1059                    ; *                                              *
  1060                    ; *    OUTPUT    RETS     NORMAL    CY=0         *
  1061                    ; *              RET      OVERFLOW, CY=1         *
  1062                    ; *                                              *
  1063                    ; ************************************************
  1064                    ;
  1065 04A2 00                       NOP
  1066 04A3 402B0B       BFDIV:      CALL    LOAD03
  1067 04A6 C5                       GJMP    BFDIV1
  1068                    ;
  1069 04A7 00                       NOP
  1070                   BFDIV0:
  1071 04A8 40490B                   CALL    POPR
  1072                    ;
  1073 04AB 00                       NOP
  1074 04AC 0106         BFDIV1:     LDAW    WSE MOD 100H
  1075 04AE 483A                     NEGA                ; OPE EXP PART COMPLEMENT
  1076 04B0 6700                     NEI     A,0
  1077 04B2 54560B                   GJMP    BFRET       ; IF DIVISOR=0
  1078                               ;
  1079 04B5 6306                     STAW    WSE MOD 100H
  1080                    ;
  1081                    ; EXPONENT PART OPERATION
  1082                    ;
  1083 04B7 40BF0A                   CALL    BFRD
  1084 04BA 54560B                   GJMP    BFRET       ; IF OVERFLOW
  1085                               ;
  1086 04BD 481C                     SKN     Z
  1087 04BF 54590B                   GJMP    BFRETS      ; IF RESULT=0 OR UNDERFLOW
  1088                               ;
  1089                    ;
  1090                    ; DIVISION / (ACC1,2,3) / (WS1,2,3)
  1091                    ;
  1092 04C2 710A00                   MVIW    WS4 MOD 100H,0
  1093 04C5 40E204                   CALL    BFDX
  1094 04C8 54560B                   GJMP    BFRET       ; IF OVERFLOW
  1095                    ;
  1096                    ; TRANS / (WS1,2,3,4) <- (WD1,2,3,4)
  1097                    ;
  1098 04CB 340AFF                   LXI     H,WS4
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   27

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1099 04CE 2416FF              LXI     D,WD4
  1100                          ;
  1101 04D1 2E                  LDAX    D-
  1102 04D2 3F                  STAX    H-        ; WS4 <- 0
  1103                                            ; STORE 4TH MANTISSA
  1104 04D3 2E                  LDAX    D-
  1105 04D4 3F                  STAX    H-        ; WS3 <- WD3
  1106 04D5 2E                  LDAX    D-
  1107 04D6 3F                  STAX    H-        ; WS2 <- WD2
  1108 04D7 2E                  LDAX    D-
  1109 04D8 3F                  STAX    H-        ; WS1 <- WD1
  1110                          ;
  1111                  ;
  1112                  ; ROUNDING !
  1113                  ;
  1114 04D9 40D809              CALL    BFROND
  1115 04DC 54560B              GJMP    BFRET     ; IF OVERFLOW
  1116 04DF 54590B              GJMP    BFRETS    ; NORMAL
  1117                  ;
  1118                  ;  DIVISION SUBROUTINE FOR MANTISSA
  1119                  ;
  1120                  ;          (WD1,2,3) <- (ACC1,2,3) / (WS1,2,3)
  1121                  ;
  1122                  ; B.F.P.ACC <- B.F.P.ACC - OPERAND
  1123                  ;
  1124 04E2 3404FF      BFDX:   LXI     H,ACC3
  1125 04E5 2409FF              LXI     D,WS3
  1126                  ;
  1127 04E8 482A                CLC
  1128 04EA 6B02                MVI     C,3-1
  1129 04EC 2B          BFDX1:  LDAX    H
  1130 04ED 70F6                SBBX    D-
  1131 04EF 3F                  STAX    H-
  1132                  ;
  1133 04F0 53                  DCR     C
  1134 04F1 FA                  GJMP    BFDX1
  1135                                            ; HL=ACC3,DE=WS3
  1136                  ;
  1137                  ; HALVES DIVISOR, AND STORE
  1138                  ;                         FOR NEXT BIT (QUOTIENT)
  1139                  ;
  1140 04F2 482B                STC               ;SET MSB OFF WS1
  1141 04F4 40460A              CALL    BFRSH4
  1142                  ;
  1143                  ; TRANS / (WD4,5,6) <- (ACC1,2,3)
  1144                  ;
  1145 04F7 3404FF              LXI     H,ACC3
  1146 04FA 2418FF              LXI     D,WD6
  1147                          ;
  1148 04FD 2F                  LDAX    H-
  1149 04FE 3E                  STAX    D-
  1150                          ;
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE   28

                (                                        )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1151 04FF 2F                      LDAX    H-
  1152 0500 3E                      STAX    D-
  1153                              ;
  1154 0501 2F                      LDAX    H-
  1155 0502 3E                      STAX    D-
  1156                    ;
  1157                    ;
  1158                    ; CHECK FOR NEXT OPERATION / (+,-)
  1159                    ;                       BY QUOTIENT.MSB
  1160                    ;
  1161 0503 5780                    OFFI    A,80H
  1162 0505 4E33                    GJMP    BFDX9   ; IF REMAINDER NEGATIVE
  1163                    ;
  1164                    ; ADJUST EXPONENT AND SET QUOTIENT.LSB
  1165                    ;
  1166 0507 2000                    INRW    ACCE MOD 100H
  1167 0509 C1                      GJMP    BFDX3
  1168                              ;
  1169 050A B8                      RET             ; OVERFLOW
  1170                              ;
  1171                  BFDX3:
  1172                                              ; DE= 1TH QUOTIENT
  1173 050B 6901                    MVI     A,1
  1174 050D 3E                      STAX    D-
  1175                                              ; DE= 2TH QUOTIENT
  1176 050E 6900                    MVI     A,0
  1177 0510 3E                      STAX    D-
  1178 0511 3A                      STAX    D       ; DE = 3TH QUOTIENT
  1179                    ;
  1180                    ;  SUBTRACT DIVISOR FROM REMAINDER
  1181                    ;                       IF IT IS POSSIBLE !
  1182                    ;
  1183                    ;    (WD4,5,6) <- (WD4,5,6) - (WS1,2,3,4)
  1184                    ;
  1185                  BFDX4:
  1186 0512 340AFF                  LXI     H,WS4
  1187 0515 2418FF                  LXI     D,WD6
  1188 0518 6900                    MVI     A,0
  1189 051A 70E7                    SUBX    H-
  1190                                              ;SET CARRY BY WS4
  1191 051C 6B02                    MVI     C,3-1
  1192                              ;
  1193 051E 2A          BFDX5:      LDAX    D
  1194 051F 70F7                    SBBX    H-
  1195 0521 3E                      STAX    D-
  1196                              ;
  1197 0522 53                      DCR     C
  1198 0523 FA                      GJMP    BFDX5
  1199                    ;
  1200                    ;  CHECK FOR / DIVISION SUBROUTINE END ?
  1201                    ;
  1202 0524 551380        BFDX6:    OFFIW   WD1 MOD 100H,80H
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   29

                (                                   )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1203 0527 B9                    RETS
  1204                                    ; IF QUOTIENT.MSB=1
  1205                    ;
  1206                    ;  (WD1,2,3,4,5,6) 6BYTE LEFT SHIFT BY BIT
  1207                    ;
  1208 0528 010A                  LDAW    WS4 MOD 100H
  1209 052A 4835                  RLL     A       ; CARRY FROM WS4.MSB
  1210                            ;
  1211 052C 6A05                  MVI     B,6-1
  1212 052E 2418FF                LXI     D,WD6
  1213 0531 40550A                CALL    BFLSH1
  1214                    ;
  1215                    ;  JUDGE  NEXT OPERATION / (+,-) BY WD3.LSB
  1216                    ;
  1217 0534 551501                OFFIW   WD3 MOD 100H,01H
  1218 0537 4FD9                  GJMP    BFDX4   ; IF REMAINDER: POSITIVE
  1219 0539 C5                    GJMP    BFDX7   ; IF REMAINDER: NEGATIVE
  1220                    ;
  1221                    ;  ADD DIVISOR / IF REMAINDER IS NEGATIVE
  1222                    ;
  1223                    ;     (WD4,5,6) <- (WD4,5,6) + (WS1,2,3)
  1224                    ;
  1225                    BFDX9:
  1226                                    ; QUOTIENT 0 SET
  1227 053A 6900                  MVI     A,0
  1228 053C 3E                    STAX    D-
  1229 053D 3E                    STAX    D-
  1230 053E 3A                    STAX    D
  1231                                    ; ADDITION
  1232 053F 3409FF        BFDX7:  LXI     H,WS3
  1233 0542 2418FF                LXI     D,WD6
  1234 0545 6B02                  MVI     C,3-1
  1235                    ;
  1236 0547 482A                  CLC
  1237 0549 2A            BFDX8:  LDAX    D
  1238 054A 70D7                  ADCX    H-
  1239 054C 3E                    STAX    D-
  1240                    ;
  1241 054D 53                    DCR     C
  1242 054E FA                    GJMP    BFDX8
  1243 054F 4FD3                  GJMP    BFDX6   ; JUMP TO / JUDGE DIV END
  1244                            ;
  1245                            EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   30

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1246                     ;
  1247                     ; ************************************************
  1248                     ; *                                              *
  1249                     ; *       COS (X) FUNCTION  SUBROUTINE           *
  1250                     ; *           INPUT:                             *
  1251                     ; *                    V REG  --  0FFH           *
  1252                     ; *           OUTPUT:  RETS -- NORMAL    CY=0     *
  1253                     ; *                    RET  -- OVERFLOW, CY=1    *
  1254                     ; *                                              *
  1255                     ; *                                              *
  1256                     ; ************************************************
  1257                     ;
  1258                     COS:
  1259 0551 40320A              CALL    BFABS
  1260 0554 40F00A              CALL    STORE1  ; WXE <- B.F.P.ACC
  1261 0557 34ED08              LXI     H,BF2PDV        ; PI/2 STORE TOP ADDR
  1262 055A 408A03              CALL    BFADD   ; X + PI/2
  1263 055D C1                  GJMP    COS1    ; IF OVERFLOW
  1264 055E CD                  GJMP    SIN
  1265                          ;
  1266 055F 40090B      COS1:   CALL    LOADA1  ; B.F.P.ACC <- WXE
  1267 0562 34ED08              LXI     H,BF2PDV
  1268 0565 407D03              CALL    BFSUB   ; X - PI/2
  1269 0568 00                  NOP             ; DUMMY FOR RETS
  1270 0569 40050A              CALL    BFCOMP  ; -(X-PI/2)
  1271                          EJECT
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   31

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1272                    ;
  1273                    ; ************************************************
  1274                    ; *                                              *
  1275                    ; *       SIN (X) FUNCTION  SUBROUTINE           *
  1276                    ; *           INPUT:                             *
  1277                    ; *                   V REG -- 0FFH              *
  1278                    ; *                                              *
  1279                    ; *           OUTPUT:  RETS --- NORMAL    CY=0     *
  1280                    ; *                    RET  --- OVERFLOW, CY=1   *
  1281                    ; *                                              *
  1282                    ; ************************************************
  1283                    ;
  1284 056C 40130A  SIN:     CALL    BFSIGN  ; CHECK SIGN
  1285 056F 77FF             EQI     A,0FFH
  1286 0571 C7               GJMP    SIN0
  1287                       ;
  1288 0572 40050A           CALL    BFCOMP  ; X <- (-X)
  1289 0575 340C0A           LXI     H,BFCOMX-1
  1290 0578 B3               PUSH    H
  1291                                       ; SIN (-X) = -SIN (X)
  1292 0579 34E908  SIN0:    LXI     H,BF2PI ; 2PI STORE TOP ADDR
  1293 057C 40A304           CALL    BFDIV   ; X/(2*PI)
  1294 057F 00               NOP
  1295 0580 40390B           CALL    PUSHF
  1296 0583 40B60A           CALL    BFINT
  1297                                       ; INTEGER(X/(2*PI))
  1298 0586 4E3C             GJMP    SINOVR  ; IF OVERFLOW
  1299                       ;
  1300 0588 40490B           CALL    POPR
  1301 058B 408103           CALL    BFSUB1  ; INT(X/(2*PI)) - X/(2*PI)
  1302 058E 00               NOP
  1303 058F 34DD08           LXI     H,BF4DV ; 1/4 STORE TOP ADDR
  1304 0592 408A03           CALL    BFADD   ; 1/4 - X
  1305 0595 00               NOP             ; DUMMY FOR RETS
  1306 0596 40130A           CALL    BFSIGN
  1307 0599 7120FF           MVIW    CSIN MOD 100H,0FFH
  1308                                       ; SIGN INVERT / JUDGE FLAG
  1309 059C 77FF             EQI     A,0FFH
  1310 059E CD               GJMP    SIN1    ; IF X > 1/4
  1311                       ;
  1312 059F 34D908           LXI     H,BF2DV ; 1/2 STORE TOP ADDR
  1313 05A2 408A03           CALL    BFADD   ; X <- X + 1/2
  1314 05A5 00               NOP
  1315 05A6 40130A           CALL    BFSIGN
  1316 05A9 712000           MVIW    CSIN MOD 100H,0
  1317                       ;
  1318                       ;   CSIN=0 : 0 < X < 1/4
  1319                       ;   CSIN=0FFH : X > 1/4
  1320                       ;
  1321 05AC 77FF     SIN1:   EQI     A,0FFH
  1322 05AE 40050A           CALL    BFCOMP
  1323                       ;
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE    32
                (                                                 )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1324 05B1 34DD08              LXI     H,BF4DV
  1325 05B4 408A03              CALL    BFADD   ; X <- X + 1/4
  1326 05B7 00                  NOP
  1327                          ;
  1328 05B8 652000              NEIW    CSIN MOD 100H,0
  1329 05BB 40050A              CALL    BFCOMP
  1330                          ;
  1331 05BE 40ED0A              CALL    STORE0  ; OPE <- X (B.F.P.ACC)
  1332 05C1 547D09              GJMP    POLYN1
  1333                  ;
  1334                  ;  SIN  FUNCTION  OVERFLOW PROCESS
  1335                  ;
  1336 05C4 40490B      SINOVR: CALL    POPR
  1337 05C7 6523FF              NEIW    FSN MOD 100H,0FFH
  1338 05CA A3                  POP     H
  1339 05CB 54560B              GJMP    BFRET
  1340                          ;
  1341                          EJECT


** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE    33
                (                                                 )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1342                  ;
  1343                  ; ************************************************
  1344                  ; *                                              *
  1345                  ; *      TAN (X)  FUNCTION  SUBROUTINE           *
  1346                  ; *                                              *
  1347                  ; *           TAN(X) = SIN(X) / COS(X)           *
  1348                  ; *                                              *
  1349                  ; *            INPUT:                            *
  1350                  ; *                    V REG :  0FFH             *
  1351                  ; *                                              *
  1352                  ; *            OUTPUT : RETS    NORMAL  CY=0     *
  1353                  ; *                     RET     OVERFLOW CY=1    *
  1354                  ; *                                              *
  1355                  ; *                                              *
  1356                  ; ************************************************
  1357                  ;
  1358 05CE 40390B      TAN:    CALL    PUSHF   ; STACK <- X(B.F.P.ACC)
  1359 05D1 405105              CALL    COS     ; COS (X)
  1360 05D4 D6                  GJMP    TANOVR  ; IF OVERFLOW
  1361                          ;
  1362 05D5 40F00A              CALL    STORE1  ; WXE <- COS (X)
  1363 05D8 40490B              CALL    POPR    ; B.F.P.ACC <- X(STACK)
  1364 05DB 40060B              CALL    LOADA0
  1365 05DE 40250B              CALL    LOADO1  ; STACK <- OPE(COS)
  1366 05E1 403C0B              CALL    PUSHR
  1367                          ;
  1368 05E4 406C05              CALL    SIN     ; SIN (X)
  1369 05E7 C3                  GJMP    TANOVR  ; IF OVERFLOW
  1370                          ;
  1371 05E8 54A804              GJMP    BFDIV0  ; TAN <- SIN/COS
  1372                          ;
  1373                          ;  OVER FLOW PROCESS
  1374                          ;
  1375 05EB 40490B      TANOVR: CALL    POPR
  1376 05EE 54560B              GJMP    BFRET
  1377                          EJECT
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   34

             (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1378                    ;
  1379                    ; ***********************************************
  1380                    ; *                                             *
  1381                    ; *     LN (X) FUNCTION  SUBROUTINE             *
  1382                    ; *        INPUT:                               *
  1383                    ; *                    V REG  --  0FFH          *
  1384                    ; *                                             *
  1385                    ; *        OUTPUT:     RETS - NORMAL    CY=0     *
  1386                    ; *                    RET -  OVERFLOW  CY=1     *
  1387                    ; *                                             *
  1388                    ; ***********************************************
  1389                    ;
  1390 05F1 40130A      LNX:    CALL    BFSIGN  ; CHECK X.SIGN
  1391 05F4 7701                EQI     A,1
  1392                                          ; SKIP, IF X > 0
  1393 05F6 54560B              GJMP    BFRET
  1394                          ;
  1395 05F9 0100                LDAW    ACCE MOD 100H
  1396 05FB 6680                SUI     A,80H
  1397 05FD 631D                STAW    LNWORK MOD 100H
  1398                          ;
  1399                          ; GET / POLYNOMIAL
  1400                          ;
  1401 05FF 710080              MVIW    ACCE MOD 100H,80H
  1402                                          ; EXPONENT CORRECT !
  1403 0602 34D508              LXI     H,BFN1
  1404 0605 408A03              CALL    BFADD   ; X + 1
  1405 0608 00                  NOP             ; DUMMY FOR RETS
  1406 0609 40ED0A              CALL    STORE0  ; OPE <- (X+1)
  1407                          ;
  1408 060C 34D508              LXI     H,BFN1
  1409 060F 400F0B              CALL    LOADA   ; B.F.P.ACC <- 1
  1410 0612 40AC04              CALL    BFDIV1  ;  1 / (X + 1)
  1411 0615 00                  NOP             ; DUMMY FOR RETS
  1412                          ;
  1413 0616 2000                INRW    ACCE MOD 100H
  1414 0618 00                  NOP             ;2 / (X+2)
  1415                          ;
  1416 0619 34D508              LXI     H,BFN1
  1417 061C 407D03              CALL    BFSUB   ; 2/(X+1) - 1
  1418 061F 00                  NOP
  1419 0620 40050A              CALL    BFCOMP  ; SIGN INVERT
  1420 0623 40ED0A              CALL    STORE0  ; OPE <- B.F.P.ACC
  1421                    ;
  1422                    ;  POLY / X*(1 + X**2/3 + X**4/5 + ----)
  1423                    ;
  1424 0626 408309              CALL    POLYN3
  1425 0629 00                  NOP
  1426 062A 2000                INRW    ACCE MOD 100H
  1427 062C 00                  NOP
  1428 062D 40390B              CALL    PUSHF
  1429                                          ; POLYNOMIAL STORE
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE   35
                (                                                 )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1430                     ;
  1431                     ;  K(COUNTER)*LN2 + POLY (X) / IXI<1
  1432                     ;
  1433 0630 2406FF              LXI     D,WSE
  1434 0633 6908                MVI     A,8       ; FRAC POINT ADDR
  1435 0635 3C                  STAX    D+
  1436                                    ; WS1 ADDR
  1437 0636 011D                LDAW    LNWORK MOD 100H
  1438 0638 5780                OFFI    A,80H     ; EXPONENT < 0
  1439 063A 483A                NEGA              ; IF < 0
  1440 063C 3C                  STAX    D+
  1441                                    ; WS2 ADDR
  1442 063D 6900                MVI     A,0       ; WS2,3,4 = 0
  1443 063F 3C                  STAX    D+
  1444 0640 3C                  STAX    D+
  1445 0641 3C                  STAX    D+
  1446                          ;
  1447 0642 40970A              CALL    BFLT      ; K -> B.F.P.ACC
  1448 0645 00                  NOP               ; DUMMY FOR RETS
  1449                          ;
  1450 0646 34F508              LXI     H,BFLN2
  1451 0649 402604              CALL    BFMUL     ; K * LN2
  1452 064C 00                  NOP               ; DUMMY FOR RETS
  1453
  1454 064D 551D80              OFFIW   LNWORK MOD 100H,80H
  1455                                    ; SUB,ADD ?
  1456 0650 40050A              CALL    BFCOMP    ; IF SUB  K*LN2
  1457 0653 548F03              GJMP    BFADD0    ; K*LN2 + POLYNOM
  1458                          ;
  1459                          EJECT


** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE   36
                (                                                 )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1460                     ;
  1461                     ; ***********************************************
  1462                     ; *                                             *
  1463                     ; *     LOG (X) FUNCTION SUBROUTINE             *
  1464                     ; *                                             *
  1465                     ; *             LOG (X) =LN (X) / LN (10)       *
  1466                     ; *                                             ;
  1467                     ; *        INPUT:                               *
  1468                     ; *                V REG. =  0FFH               *
  1469                     ; *                                             *
  1470                     ; *        OUTPUT:                              *
  1471                     ; *                RETS    NORMAL  CY=0         *
  1472                     ; *                RET     OVERFLOW CY=1        *
  1473                     ; *                                             *
  1474                     ; ***********************************************
  1475                     ;
  1476 0656 40F105         LOG:   CALL    LNX
  1477 0659 54560B                GJMP    BFRET     ; IF OVERFLOW
  1478                            ;
  1479 065C 34F908                LXI     H,BFLN10
  1480 065F 54A304                GJMP    BFDIV
  1481                                    ; LN(X) / LN(10)
  1482                            EJECT
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   37

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1483                    ;
  1484                    ; ***********************************************
  1485                    ; *                                             *
  1486                    ; *       EXP (X)   FUNCTION  SUBROUTINE        *
  1487                    ; *                                             *
  1488                    ; *           INPUT:                            *
  1489                    ; *                   V REG. = 0FFH             *
  1490                    ; *                                             *
  1491                    ; *           OUTPUT:                           *
  1492                    ; *                   RETS   NORMAL  CY=0       *
  1493                    ; *                   RET    OVERFLOW CY=1      *
  1494                    ; *                                             *
  1495                    ; ***********************************************
  1496                    ;
  1497                    EXP:
  1498 0662 40130A                  CALL    BFSIGN  ; CHECK X.SIGN
  1499 0665 B0                      PUSH    V       ; STORE X.SIGN
  1500                                              ; EXP(-X) = 1  / EXP(X)
  1501 0666 40320A                  CALL    BFABS   ; X <- -X
  1502 0669 40390B                  CALL    PUSHF   ; X STORE
  1503                              ;
  1504 066C 34F108                  LXI     H,BFLGE2
  1505                                              ; LOG(E) BASE 2 TABLE TOP ADDR
  1506 066F 402604                  CALL    BFMUL   ; X * LOG(E)
  1507 0672 4E4D                    GJMP    EXP1    ; IF OVERFLOW
  1508                              ;
  1509 0674 350088                  LTIW    ACCE MOD 100H,80H+8
  1510 0677 4E48                    GJMP    EXP1
  1511                                              ; IF OVERFLOW / > 2**127
  1512 0679 40AA0A                  CALL    BFINT0  ; INT( X * LOG(E))
  1513 067C 4E43                    GJMP    EXP1
  1514                                              ; IF NOT FIXED <- FLOAT
  1515 067E 0121                    LDAW    TMPWSL MOD 100H
  1516 0680 B0                      PUSH    V       ; STORE  INT(X * LOG(E))
  1517                              ;
  1518 0681 4680                    ADI     A,80H   ; ADD BIAS
  1519 0683 2602                    ADINC   A,80H-126
  1520 0685 4E39                    GJMP    EXP0    ; IF OVERFLOW / EXP > 126
  1521                              ;
  1522 0687 34F508                  LXI     H,BFLN2
  1523 068A 402604                  CALL    BFMUL   ; LN2 * INT(X*LOG(E))
  1524 068D 00                      NOP             ; DUMMY FOR RETS
  1525                              ;
  1526 068E A0                      POP     V
  1527 068F 40490B                  CALL    POPR    ; RESTORE.X
  1528 0692 B0                      PUSH    V
  1529 0693 408103                  CALL    BFSUB1  ; LN2*(INT(X*LOG(E))) - X
  1530 0696 00                      NOP
  1531 0697 40ED0A                  CALL    STORE0  ; OPE <- B.F.P.ACC
  1532                              ;
  1533                              ; POLY / 1 + X/1 + X**2/2*1 + X**3/3*2*1 + ---
  1534                              ;
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   38

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1535 069A 341209              LXI     H,EXPCON          ; EXP CONSTANT TOP ADDR
  1536 069D 409909              CALL    POLY
  1537                          ;
  1538                          ; * POLYNOMIAL END *
  1539                          ;
  1540 06A0 A0                  POP     V         ; RESTORE INT(X*LOG(E))
  1541 06A1 74C000              ADDW    ACCE MOD 100H
  1542 06A4 481A                SKN     CY
  1543 06A6 DD                  GJMP    EXP2      ; IF OVERFLOW
  1544                          ;
  1545 06A7 6300                STAW    ACCE MOD 100H
  1546                                            ; STORE B.F.P.ACC-EXP
  1547 06A9 A0                  POP     V         ; X.SIGN RESTORE
  1548 06AA 41                  INR     A
  1549 06AB 54590B              GJMP    BFRETS
  1550                          ;
  1551                          ;  EXP(-X) = 1 / EXP(X)
  1552                          ;
  1553 06AE 40ED0A              CALL    STORE0    ; OPE <- B.F.P.ACC
  1554 06B1 34D508              LXI     H,BFN1
  1555 06B4 400F0B              CALL    LOADA     ; B.F.P.ACC
  1556 06B7 40AC04              CALL    BFDIV1    ; 1 / EXP(X)
  1557 06BA 54560B              GJMP    BFRET     ; IF OVERFLOW
  1558 06BD 54590B              GJMP    BFRETS    ; IF NORMAL
  1559                          ;
  1560                          ;  OVER FLOW  RROCESS
  1561                          ;
  1562 06C0 A0          EXP0:   POP     V
  1563 06C1 40490B      EXP1:   CALL    POPR
  1564 06C4 A0          EXP2:   POP     V
  1565 06C5 41                  INR     A
  1566 06C6 54560B              GJMP    BFRET
  1567                          ;
  1568                          ;  IF X < 0 , EXP = 0
  1569                          ;
  1570 06C9 050000              ANIW    ACCE MOD 100H,0
  1571 06CC 54590B              GJMP    BFRETS
  1572                          ;
  1573                          EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   39
                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1574                    ;
  1575                    ; ***********************************************
  1576                    ; *                                             *
  1577                    ; *       POWER  FUNCTION  SUBROUTINE           *
  1578                    ; *                                             *
  1579                    ; *       INPUT:                                *
  1580                    ; *                 VREG. = 0FFH                *
  1581                    ; *                 X = B.F.P.ACC               *
  1582                    ; *                 Y = OPERAND                 *
  1583                    ; *                                             *
  1584                    ; *       OUTPUT: RETS    NORMAL     CY = 0     *
  1585                    ; *               RET     OVERFLOW , CY = 1     *
  1586                    ; *                                             *
  1587                    ; ***********************************************
  1588                            ;
  1589                            ;  SQR --- OPERAND <- 1/2
  1590                            ;
  1591 06CF 34D908       SQR:     LXI     H,BF2DV
  1592 06D2 402B0B       POWER:   CALL    LOADO3   ; OPE <- (HL,HL+1,HL+2,HL+3)
  1593                            ;
  1594                            ;  B.F.P.ACC <---> OPERAND
  1595                            ;
  1596 06D5 40F00A       POWER1:  CALL    STORE1   ; WXE <- B.F.P.ACC
  1597 06D8 40060B                CALL    LOADA0   ; B.F.P.ACC <- OPE
  1598 06DB 40250B                CALL    LOADO1   ; OPE <- WXE
  1599                    ;
  1600 06DE 40130A                CALL    BFSIGN   ; CHECK ! Y.SIGN
  1601 06E1 750600                EQIW    WSE MOD 100H,0
  1602 06E4 CA                    GJMP    POWER2
  1603                                            ; IF X= NOT.0
  1604 06E5 6700                  NEI     A,0
  1605 06E7 4F79                  GJMP    EXP
  1606                                            ; IF X=Y=0 ,'X**Y = 1
  1607 06E9 41                    INR     A
  1608 06EA 4E73                  GJMP    POWERZ   ; IF X=0,Y>0
  1609 06EC 54560B                GJMP    BFRET
  1610                                            ; IF X=0,Y<0
  1611 06EF 6700         POWER2:  NEI     A,0
  1612 06F1 4F6F                  GJMP    EXP
  1613                                            ; IF X= NOT.0 ,Y=0
  1614 06F3 0107                  LDAW    WS1 MOD 100H
  1615 06F5 4835                  RLL     A
  1616                                            ; CY = X.SIGN
  1617 06F7 403C0B                CALL    PUSHR
  1618 06FA 40F00A                CALL    STORE1   ; WXE <- Y(B.F.P.ACC)
  1619 06FD 40320A                CALL    BFABS
  1620 0700 40ED0A                CALL    STORE0
  1621                            ;
  1622 0703 6902                  MVI     A,2
  1623 0705 6321                  STAW    TMPWSL MOD 100H
  1624                                            ; INITIAL SSET
  1625 0707 480A                  SK      CY
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    40

              (                                        )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1626                                                ; CHECK X.SIGN
  1627 0709 D2                      GJMP    POWER3
  1628                              ;
  1629 070A 403C0B                  CALL    PUSHR
  1630 070D 40B60A                  CALL    BFINT     ; INTEGER  (Y)
  1631 0710 4E3E                    GJMP    POWER5    ; IF OVERFLOW
  1632                              ;
  1633 0712 40490B                  CALL    POPR
  1634 0715 408103                  CALL    BFSUB1    ; INTEGER (Y) -Y
  1635 0718 00                      NOP                ; DUMMY
  1636 0719 40130A                  CALL    BFSIGN    ; IF INT(Y) = Y ,A =0
  1637                                                ; SIGN  RESTORE
  1638                    ;
  1639                    ;  CHECK ! / INT (Y) = ODD
  1640                    ;
  1641 071C 40490B      POWER3: CALL    POPR
  1642 071F B0                      PUSH    V         ; SIGN --> STACK
  1643 0720 40060B                  CALL    LOADA0
  1644                              ;
  1645 0723 0121                    LDAW    TMPWSL MOD 100H
  1646 0725 631C                    STAW    POWERK MOD 100H   ; STORE "ODD" JUDGE AREA
  1647 0727 A0                      POP     V         ; SIGN RESTORE
  1648                    ;
  1649                    ;  X ** Y = EXP ( Y * LN(X) )
  1650                    ;
  1651                    POWER4:
  1652 0728 6700                    NEI     A,0
  1653 072A 40050A                  CALL    BFCOMP
  1654                              ;
  1655 072D 40250B                  CALL    LOADO1    ; OPE <- Y(WXE)
  1656 0730 403C0B                  CALL    PUSHR
  1657 0733 40F105                  CALL    LNX       ; LN (X)
  1658 0736 4E21                    GJMP    POWER6    ; IF OVERFLOW
  1659                              ;
  1660 0738 40490B                  CALL    POPR
  1661 073B 402F04                  CALL    BFMUL1    ; Y * LN (X)
  1662 073E 54560B                  GJMP    BFRET     ; IF OVERFLOW
  1663                              ;
  1664 0741 406206                  CALL    EXP       ; EXPONENT (Y*LN(X))
  1665 0744 54560B                  GJMP    BFRET     ; IF OVERFLOW
  1666                              ;
  1667 0747 551C01                  OFFIW   POWERK MOD 100H,00000001B
  1668 074A 40050A                  CALL    BFCOMP    ; INVERT SIGN
  1669 074D 54590B                  GJMP    BFRETS
  1670                    ;
  1671                    ;  OVER FLOW  PROCESS
  1672                    ;
  1673 0750 40490B      POWER5: CALL    POPR
  1674 0753 40490B                  CALL    POPR
  1675 0756 54560B                  GJMP    BFRET
  1676                              ;
  1677 0759 40490B      POWER6: CALL    POPR
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    41

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1678 075C 54560B              GJMP     BFRET
  1679                    ;
  1680                    ;   RESULT = 0 / PROCESS
  1681                    ;
  1682 075F 050000      POWERZ: ANIW     ACCE MOD 100H,0
  1683 0762 54590B              GJMP     BFRETS
  1684                          EJECT

** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    42

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1685                    ;
  1686                    ; ************************************************
  1687                    ; *                                              *
  1688                    ; *        ARCTAN (X)  FUNCTION  SUBROUTINE      *
  1689                    ; *                                              *
  1690                    ; *          INPUT:                              *
  1691                    ; *                  V REG. = 0FFH               *
  1692                    ; *                                              *
  1693                    ; *          OUTPUT:                             *
  1694                    ; *                  RETS:    NORMAL   CY=0      *
  1695                    ; *                  RET:     OVERFLOW CY=1      *
  1696                    ; *                                              *
  1697                    ; ************************************************
  1698                    ;
  1699                    ARCTAN:
  1700 0765 40130A              CALL     BFSIGN   ; CHECK XC.SIGN
  1701 0768 77FF                EQI      A,0FFH   ; IF X>0, X=0
  1702 076A C7                  GJMP     ATAN1    ; IF X>0,X=0
  1703                                             ; ARCTAN(-X) = -ARCTAN(X)
  1704 076B 340C0A              LXI      H,BFCOMX-1
  1705 076E B3                  PUSH     H
  1706 076F 40050A              CALL     BFCOMP
  1707                          ;
  1708 0772 250080      ATAN1:  GTIW     ACCE MOD 100H,80H
  1709 0775 D5                  GJMP     ATAN2    ; IF  IXI < 1
  1710                          ;
  1711 0776 40ED0A              CALL     STORE0
  1712 0779 34D508              LXI      H,BFN1
  1713 077C 400F0B              CALL     LOADA    ; B.F.P.ACC <- 1
  1714                          ;
  1715 077F 40AC04              CALL     BFDIV1   ; 1 / IXI
  1716 0782 00                  NOP
  1717 0783 340C0A              LXI      H,BFCOMX-1
  1718 0786 B3                  PUSH     H
  1719 0787 348003              LXI      H,BFSUB1-1
  1720 078A B3                  PUSH     H
  1721                                             ; PI/2 - ARCTAN(1/IXI)
  1722                    ;
  1723                    ;  POLY / C0 - C1*X**3/3 + C2*X**5/5 - -----
  1724                    ;
  1725 078B 40ED0A      ATAN2:  CALL     STORE0
  1726 078E 408009              CALL     POLYN2
  1727 0791 00                  NOP
  1728 0792 34ED08              LXI      H,BF2PDV
  1729 0795 402B0B              CALL     LOAD03
  1730                                             ; OPERAND <- PI/2
  1731 0798 54590B              GJMP     BFRETS
  1732                          EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   43
                (                                                )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1733                    ;
  1734                    ; ************************************************
  1735                    ; *                                              *
  1736                    ; *      ARCSIN (X)  FUNCTION  SUBROUTINE        *
  1737                    ; *                                              *
  1738                    ; *         INPUT:                               *
  1739                    ; *                  V REG. = 0FFH               *
  1740                    ; *                                              *
  1741                    ; *         OUTPUT:                              *
  1742                    ; *                RETS:   NORMAL   CY=0         *
  1743                    ; *                RET:    OVERFLOW CY=1         *
  1744                    ; *                                              *
  1745                    ; ************************************************
  1746                    ;
  1747                    ARCSIN:
  1748 079B 40130A                 CALL    BFSIGN  ; CHECK X.SIGN
  1749                              ;
  1750 079E 6322                   STAW    FASIN MOD 100H
  1751 07A0 77FF                   EQI     A,0FFH
  1752 07A2 C7                     GJMP    ASIN1
  1753                                             ; IF X < 0
  1754 07A3 340C0A                 LXI     H,BFCOMX-1
  1755 07A6 B3                     PUSH    H
  1756 07A7 40050A                 CALL    BFCOMP
  1757                              ;
  1758 07AA 40390B       ASIN1:    CALL    PUSHF   ; STACK,OPE <- X
  1759 07AD 400C08                 CALL    ROUTE
  1760 07B0 D2                     GJMP    ASIN2
  1761                              ;
  1762 07B1 40F00A                 CALL    STORE1
  1763 07B4 40490B                 CALL    POPR
  1764 07B7 40060B                 CALL    LOADA0
  1765 07BA 40250B                 CALL    LOAD01
  1766 07BD 40AC04                 CALL    BFDIV1  ; X / (1-X*X)**1/2
  1767 07C0 CC                     GJMP    ASIN3   ; IF OVERFLOW
  1768 07C1 4FA2                   GJMP    ARCTAN
  1769                              ;
  1770                              ; ARCSIN (X) = ARCTAN (X / (1-X*X)**1/2 )
  1771                              ;
  1772                    ;
  1773                    ;  ARCSIN  OVERFLOW
  1774                    ;
  1775 07C3 40490B       ASIN2:    CALL    POPR
  1776 07C6 6522FF                 NEIW    FASIN MOD 100H,0FFH
  1777                              ;
  1778 07C9 A3                     POP     H
  1779 07CA 54560B                 GJMP    BFRET
  1780                    ;
  1781                    ;    IF X = 1 , ARCSIN = PI/2
  1782                    ;
  1783 07CD 34ED08       ASIN3:    LXI     H,BF2PDV
  1784 07D0 400F0B                 CALL    LOADA

** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   44
                (                                                )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1785 07D3 54590B                 GJMP    BFRETS
  1786                              EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE    45

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1787                     ;
  1788                     ; ************************************************
  1789                     ; *                                              *
  1790                     ; *       ARCCOS (X)  FUNCTION  SUBROUTINE       *
  1791                     ; *                                              *
  1792                     ; *          INPUT:                              *
  1793                     ; *                 V REG. = 0FFH                *
  1794                     ; *                                              *
  1795                     ; *          OUTPUT:                             *
  1796                     ; *                   RETS:   NORMAL  CY=0       *
  1797                     ; *                   RET:    OVERFLOW CY=1      *
  1798                     ; *                                              *
  1799                     ; ************************************************
  1800                     ;
  1801                     ;    ARCCOS = ARCTAN ( ((1-X**2)**1/2)/X)   X>0
  1802                     ;    ARCCOS = PI + ARCTAN                   X<0
  1803                     ;    IF  X = 0 , ARCCOS = PI/2
  1804                     ;
  1805                     ARCCOS:
  1806 07D6 40130A                 CALL    BFSIGN   ; CHECK X.SIGN
  1807 07D9 B0                     PUSH    V        ; SIGN STORE
  1808                             ;
  1809 07DA 40390B                 CALL    PUSHF    ; STACK.OPERAND <- B.F.P.ACC
  1810 07DD 400C08                 CALL    ROUTE    ; ( 1-X**2) ** 1/2
  1811 07E0 DA                     GJMP    ACOS1    ; IF  OVERFLOW
  1812                             ;
  1813 07E1 40490B                 CALL    POPR     ; OPERAND <- X
  1814 07E4 40AC04                 CALL    BFDIV1   ; ( (1-X**2)**1/2 ) / X
  1815 07E7 DA                     GJMP    ACOS2    ; IF OVERFLOW
  1816                             ;
  1817 07E8 406507                 CALL    ARCTAN   ; ARCTAN( ((1-X**2)**1/2)/X )
  1818 07EB 00                     NOP              ; DUMMY FOR RETS
  1819                             ;
  1820 07EC A0                     POP     V        ; RESTORE X.SIGN
  1821 07ED 41                     INR     A
  1822 07EE 54590B                 GJMP    BFRETS   ; IF X > 0
  1823                             ;
  1824 07F1 34E108                 LXI     H,BFPI
  1825 07F4 408A03                 CALL    BFADD    ; PI + ARCTAN
  1826 07F7 00                     NOP              ; DUMMY FOR RETS
  1827 07F8 54590B                 GJMP    BFRETS
  1828                     ;
  1829                     ;  OVERFLOW  PROCESS
  1830                     ;
  1831                     ACOS1:
  1832 07FB 40490B                 CALL    POPR
  1833 07FE A0                     POP     V
  1834 07FF 54560B                 GJMP    BFRET
  1835                     ;
  1836                     ;  IF X = 0 , ARCCOS = PI/2
  1837                     ;
  1838                     ACOS2:
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    46

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  1839 0802 A0                    POP     V
  1840 0803 34ED08                LXI     H,BF2PDV
  1841 0806 400F0B                CALL    LOADA   ; B.F.P.ACC <- PI/2
  1842 0809 54590B                GJMP    BFRETS
  1843                    ;
  1844                    ;  GET / ( 1 - X*X ) ** 1/2
  1845                    ;
  1846 080C 402F04        ROUTE:  CALL    BFMUL1  ; X * X
  1847 080F B8                    RET             ; IF OVERFLOW
  1848                            ;
  1849 0810 34D508                LXI     H,BFN1
  1850 0813 407D03                CALL    BFSUB   ; X * X - 1
  1851 0816 B8                    RET             ; IF  OVERFLOW
  1852                            ;
  1853 0817 40050A                CALL    BFCOMP
  1854                            ;
  1855 081A 40CF06                CALL    SQR     ; ( 1 - X*X ) ** 1/2
  1856 081D B8                    RET
  1857 081E B9                    RETS
  1858                            ;
  1859                            EJECT
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   47

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1860                    ;
  1861                    ; ************************************************
  1862                    ; *                                              *
  1863                    ; *       POLAR ->  ORTHOGONAL -- COORDINATES    *
  1864                    ; *                                              *
  1865                    ; *                    X = R * COS(T)            *
  1866                    ; *                    Y = R * SIN(T)            *
  1867                    ; *        INPUT:                                *
  1868                    ; *                V REG. = 0FFH                 *
  1869                    ; *                BFP.ACC = R                   *
  1870                    ; *                HL = T START ADDR.            *
  1871                    ; *                                              *
  1872                    ; *        OUTPUT:                               *
  1873                    ; *                BFP.ACC = X                   *
  1874                    ; *                OPERAND = Y                   *
  1875                    ; *                                              *
  1876                    ; *                 RETS   NORMAL   CY=0         *
  1877                    ; *                 RET    OVERFLOW CY=1         *
  1878                    ; *                                              *
  1879                    ; ************************************************
  1880                    ;
  1881 081F 402B0B  TRACA:  CALL    LOADO3
  1882 0822 40F30A  TRACA1: CALL    STORE2
  1883                                       ; WYE <- R
  1884 0825 40060B          CALL    LOADA0   ; B.F.P.ACC <- T
  1885 0828 40F00A          CALL    STORE1   ; WXE <- T
  1886                      ;
  1887 082B 406C05          CALL    SIN      ; SIN (T)
  1888 082E 54560B          GJMP    BFRET    ; OVERFLOW
  1889                      ;
  1890 0831 340FFF          LXI     H,WYE
  1891 0834 402604          CALL    BFMUL    ; R * T
  1892 0837 00              NOP              ; DUMMY
  1893 0838 40390B          CALL    PUSHF
  1894 083B 40090B          CALL    LOADA1
  1895 083E 405105          CALL    COS      ; B.F.P.ACC <- COS (T)
  1896 0841 CD              GJMP    TRACA2
  1897                      ;
  1898 0842 340FFF          LXI     H,WYE
  1899 0845 402604          CALL    BFMUL    ; R * SIN (T)
  1900 0848 00              NOP
  1901 0849 40490B          CALL    POPR     ; OPE <- R * SIN(T)
  1902 084C 54590B          GJMP    BFRETS
  1903                    ;
  1904                    ;  OVER FLOW  PROCESS
  1905                    ;
  1906 084F 40490B  TRACA2: CALL    POPR
  1907 0852 54560B          GJMP    BFRET
  1908                      EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   48

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1909                    ;
  1910                    ; ************************************************
  1911                    ; *                                              *
  1912                    ; *       ORTHOGONAL -> POLAR  --  COORDINATES   *
  1913                    ; *                                              *
  1914                    ; *              R = ( X*X + Y*Y ) **1/2         *
  1915                    ; *              T = ARCTAN (Y/X)                *
  1916                    ; *                                              *
  1917                    ; *       INPUT:                                 *
  1918                    ; *              V REG. = 0FFH                   *
  1919                    ; *              BFP.ACC = X                     *
  1920                    ; *              HL = Y START ADDR.              *
  1921                    ; *                                              *
  1922                    ; *       OUTPUT:                                *
  1923                    ; *              BFP.ACC = R                     *
  1924                    ; *              OPERAND = T                     *
  1925                    ; *                                              *
  1926                    ; *              RETS:    NORMAL  CY=0           *
  1927                    ; *              RET:   OVERFLOW  CY=1           *
  1928                    ; *                                              *
  1929                    ; ************************************************
  1930                    ;
  1931                    ;  T = ARCTAN (Y/X)
  1932                    ;
  1933 0855 402B0B       TRACB:  CALL    LOADO3  ;OPE <-- Y
  1934                    ;
  1935                    ;  B.F.P.ACC  <-->  OPERAND
  1936                    ;
  1937 0858 40F00A       TRACB1: CALL    STORE1  ;SAVE X TO WXE
  1938 085B 40060B               CALL    LOADA0  ;GET Y TO BFP.ACC
  1939 085E 40F30A               CALL    STORE2  ;SAVE IT TO WYE
  1940                           ;
  1941 0861 40250B               CALL    LOADO1
  1942                           ;
  1943 0864 40130A               CALL    BFSIGN  ; Y.SIGN CHECK
  1944 0867 B0                   PUSH    V       ; SIGN RESTORE
  1945                           ;
  1946 0868 0107                 LDAW    WS1 MOD 100H
  1947 086A B0                   PUSH    V       ; MSB = SIGN
  1948                           ;
  1949 086B 40AC04               CALL    BFDIV1  ; Y / X
  1950 086E 4E41                 GJMP    TRACB2
  1951                                           ; IF Y/X TOO LAGE , ARCTAN <- PI/2
  1952 0870 406507               CALL    ARCTAN  ; ARCTAN (Y/X)
  1953 0873 00                   NOP
  1954                           ;
  1955 0874 A0                   POP     V
  1956 0875 4680                 ADI     A,80H   ; CY = X.SIGN
  1957 0877 A0                   POP     V
  1958                                           ; STORE Y.SIGN
  1959 0878 480A                 SK      CY
  1960 087A CC                   GJMP    TRACB3
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   49

                (                                        )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  1961                                                  ; IF 1TH,OR,4TH QUADRANT
  1962                  ;
  1963 087B 41                    INR     A
  1964 087C 34E108                LXI     H,BFPI  ;PI
  1965 087F 34E508                LXI     H,BFMPI ;-PI
  1966                  ;
  1967 0882 408A03                CALL    BFADD
  1968 0885 00                    NOP             ; DUMMY FOR RETS
  1969 0886 C0                    GJMP    TRACB3
  1970                            ;
  1971 0887 40390B      TRACB3:   CALL    PUSHF
  1972                  ;
  1973                  ;  R = ( X*X + Y*Y ) ** 1/2
  1974                  ;
  1975 088A 40090B                CALL    LOADA1
  1976 088D 340BFF                LXI     H,WXE
  1977 0890 402604                CALL    BFMUL   ;  X * X
  1978 0893 4E2E                  GJMP    TRACB5  ; OVERFLOW
  1979                            ;
  1980 0895 40390B                CALL    PUSHF   ; STACN <-  X*X
  1981 0898 400C0B                CALL    LOADA2
  1982 089B 340FFF                LXI     H,WYE
  1983 089E 402604                CALL    BFMUL   ; Y*Y
  1984 08A1 DE                    GJMP    TRACB4  ; IF OVERFLOW
  1985                            ;
  1986 08A2 40490B                CALL    POPR    ; OPE <- X*X(STACK)
  1987 08A5 409203                CALL    BFADD1  ; X*X + Y*Y
  1988 08A8 DA                    GJMP    TRACB5  ; OVERFLOW
  1989                            ;
  1990 08A9 40CF06                CALL    SQR
  1991 08AC D6                    GJMP    TRACB5  ; ( X*X + Y*Y ) ** 1/2
  1992                            ;
  1993 08AD 40490B                CALL    POPR    ; OPE <- T
  1994 08B0 B9                    RETS
  1995                  ;
  1996                  ; Y / X  --  TOO LAGE , ARCTAN (Y/X)  =  PI/2
  1997                  ;
  1998 08B1 A0          TRACB2:   POP     V
  1999 08B2 34ED08                LXI     H,BF2PDV
  2000 08B5 400F0B                CALL    LOADA
  2001                                            ; B.F.P.ACC <-  PI/2
  2002 08B8 A0                    POP     V
  2003 08B9 5780                  OFFI    A,80H
  2004                                            ; SKIP , IF Y > 0
  2005 08BB 40050A                CALL    BFCOMP
  2006                                            ; IF Y<0 , ARCTAN = - PI/2
  2007 08BE 4FC7                  GJMP    TRACB3
  2008                  ;
  2009                  ; OVFRFLOE  PROCESS
  2010                  ;
  2011 08C0 40490B      TRACB4:   CALL    POPR
  2012 08C3 40490B      TRACB5:   CALL    POPR
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   50

              (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2013 08C6 54560B                GJMP     BFRET
  2014                            EJECT

** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   51

              (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2015                        ;
  2016                        ; *******************************
  2017                        ; *   DATA  TABLE               *
  2018                        ; *******************************
  2019                        ;
  2020                        BFTEN:
  2021 08C9 84200000                  DB       84H,20H,0,0
  2022                        BFDV58:                 ;0.00000005
  2023 08CD 6856BFAD                  DB       68H,56H,0BFH,0ADH
  2024                        BFN01:
  2025 08D1 7D4CCCCC                  DB       7DH,4CH,0CCH,0CCH        ;0.1
  2026                        BFN1:
  2027 08D5 81000000                  DB       81H,00H,00H,00H          ;1
  2028                        BF2DV:
  2029 08D9 80000000                  DB       80H,00H,00H,00H       ; 1 / 2
  2030                        BF4DV:
  2031 08DD 7F000000                  DB       7FH,00H,00H,00H       ; 1 / 4
  2032                        BFPI:
  2033 08E1 82490FDB                  DB       82H,49H,0FH,0DBH      ; PI
  2034                        BFMPI:
  2035 08E5 82C90FDB                  DB       82H,0C9H,0FH,0DBH        ;-PI
  2036                        BF2PI:
  2037 08E9 83490FDB                  DB       83H,49H,0FH,0DBH      ; 2PI
  2038                        BF2PDV:
  2039 08ED 81490FDB                  DB       81H,49H,0FH,0DBH      ;  PI / 2
  2040                        BFLGE2:
  2041 08F1 8138AA3B                  DB       81H,38H,0AAH,3BH      ; LOG (E) BASE 2
  2042                        BFLN2:
  2043 08F5 80317218                  DB       80H,31H,72H,18H       ; LN2
  2044                        BFLN10:
  2045 08F9 82135D8E                  DB       82H,13H,5DH,8EH       ; LN (10)
  2046                        ;
  2047                        ; *******************************
  2048                        ; *   CONST  TABLE              *
  2049                        ; *******************************
  2050                        ;
  2051 08FD 04                SINCOS:  DB      5-1
  2052                        ;
  2053 08FE 861ED7BA                   DB      86H,1EH,0D7H,0BAH
  2054 0902 87992664                   DB      87H,99H,26H,64H
  2055 0906 87233458                   DB      87H,23H,34H,58H
  2056 090A 86A55DE0                   DB      86H,0A5H,5DH,0E0H
  2057 090E 83490FDA                   DB      83H,49H,0FH,0DAH
  2058                        ;
  2059 0912 07                EXPCON:  DB      8-1
  2060                        ;
  2061 0913 74942E40                   DB      74H,94H,2EH,40H
  2062 0917 772E4F70                   DB      77H,2EH,4FH,70H
  2063 091B 7A88026E                   DB      7AH,88H,02H,6EH
  2064 091F 7C2AA0E6                   DB      7CH,2AH,0A0H,0E6H
  2065 0923 7EAAAA50                   DB      7EH,0AAH,0AAH,50H
  2066 0927 7F7FFFFF                   DB      7FH,7FH,0FFH,0FFH
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE   52

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2067 092B 81800000          DB        81H,80H,00H,00H
  2068 092F 81000004          DB        81H,00H,00H,04H
  2069                ;
  2070 0933 08        ATNCON: DB        9-1
  2071                ;
  2072 0934 783BD74A          DB        78H,3BH,0D7H,4AH
  2073 0938 7B846E02          DB        7BH,84H,6EH,02H
  2074 093C 7C2FC1FE          DB        7CH,2FH,0C1H,0FEH
  2075 0940 7D9A3174          DB        7DH,9AH,31H,74H
  2076 0944 7D5A3B84          DB        7DH,5AH,3BH,84H
  2077 0948 7E917FC8          DB        7EH,91H,7FH,0C8H
  2078 094C 7E4CBBE4          DB        7EH,4CH,0BBH,0E4H
  2079 0950 7FAAAA6C          DB        7FH,0AAH,0AAH,6CH
  2080 0954 81000000          DB        81H,00H,00H,00H
  2081                ;
  2082 0958 08        LNCON:  DB        9-1
  2083                ;
  2084 0959 783BD74A          DB        78H,3BH,0D7H,4AH
  2085 095D 7B046E02          DB        7BH,04H,6EH,02H
  2086 0961 7C2FC1FE          DB        7CH,2FH,0C1H,0FEH
  2087 0965 7D1A3174          DB        7DH,1AH,31H,74H
  2088 0969 7D5A3B84          DB        7DH,5AH,3BH,84H
  2089 096D 7E117FC8          DB        7EH,11H,7FH,0C8H
  2090 0971 7E4CBBE4          DB        7EH,4CH,0BBH,0E4H
  2091 0975 7F2AAA6C          DB        7FH,2AH,0AAH,6CH
  2092 0979 81000000          DB        81H,00H,00H,00H
  2093                ;
  2094                        EJECT
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   53
                (                                                )
E STNO ADRS OBJECT    M   SOURCE STATEMENT

  2095                    ;
  2096                    ;
  2097                    ; ================================================
  2098                    ; =                                              =
  2099                    ; =                COMMON  SUBROUTINE            =
  2100                    ; =                                              =
  2101                    ; ================================================
  2102                    ;
  2103                    ; ================================================
  2104                    ; =                                              =
  2105                    ; =          POLYNOMIAL  SUBROUTINE              =
  2106                    ; =                                              =
  2107                    ; ================================================
  2108                    ;
  2109 097D 34FD08       POLYN1: LXI     H,SINCOS        ; SIN CONST TOP ADDR
  2110 0980 343309       POLYN2: LXI     H,ATNCON        ; ARCTAN
  2111 0983 345809       POLYN3: LXI     H,LNCON         ; LN
  2112                            ;
  2113 0986 403C0B       POLYX:  CALL    PUSHR
  2114 0989 242B04               LXI     D,BFMUL0
  2115 098C B2                   PUSH    D
  2116                           ;
  2117 098D B3                   PUSH    H
  2118 098E 40060B               CALL    LOADA0
  2119                           ;
  2120 0991 402F04               CALL    BFMUL1  ; X * X
  2121 0994 00                   NOP
  2122 0995 40ED0A               CALL    STORE0
  2123 0998 A3                   POP     H
  2124                    ;
  2125                    ;  POLYNOMIAL / C0 + C1*X + C2*X**2 + -----
  2126                    ;
  2127 0999 403C0B       POLY:   CALL    PUSHR
  2128 099C 2D                   LDAX    H+
  2129 099D B0                   PUSH    V
  2130 099E 400F0B               CALL    LOADA   ; B.F.P.ACC <- CONST
  2131                           ;
  2132 09A1 A0           POLY1:  POP     V
  2133 09A2 40490B               CALL    POPR
  2134 09A5 3601                 SUINB   A,1
  2135 09A7 B8                   RET
  2136                           ;
  2137 09A8 403C0B               CALL    PUSHR
  2138 09AB B0                   PUSH    V
  2139 09AC B3                   PUSH    H
  2140                           ;
  2141 09AD 402F04               CALL    BFMUL1  ; CONST * X
  2142 09B0 00                   NOP
  2143 09B1 A3                   POP     H
  2144 09B2 402B0B               CALL    LOAD03  ; OPE <- NEXT CONST
  2145 09B5 B3                   PUSH    H
  2146 09B6 409203               CALL    BFADD1  ; NEXT CONST + CONST * X(OPE)
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   54

             (                                          )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  2147 09B9 00                      NOP
  2148 09BA A3                      POP     H
  2149 09BB E5                      GJMP    POLY1
  2150                      ;
  2151                      ;
  2152                      ;     ++  (WS1,2,3,4)  NORMALIZE   ++
  2153                      BFNOR:
  2154 09BC 650000                  NEIW    ACCE MOD 100H , 0H
  2155 09BF B8                      RET                     ;IF BFP ACC = 0
  2156                      BFNOR0:
  2157 09C0 0107                    LDAW    WS1 MOD 100H
  2158 09C2 5780                    OFFI    A,10000000B
  2159 09C4 B9                      RETS
  2160                              ;
  2161                              ;  WS1,2,3,4  LEFT SHIFT
  2162                              ;
  2163 09C5 482A                    CLC
  2164 09C7 240AFF                  LXI     D,WS4
  2165 09CA 6B03                    MVI     C,4-1
  2166 09CC 2A              BFNOR1: LDAX    D
  2167 09CD 4835                    RAL
  2168 09CF 3E                      STAX    D-
  2169 09D0 53                      DCR     C
  2170 09D1 FA                      GJMP    BFNOR1
  2171                              ;
  2172 09D2 3000                    DCRW    ACCE MOD 100H
  2173 09D4 480C                    SK      Z
  2174 09D6 E9                      GJMP    BFNOR0
  2175 09D7 B8                      RET
  2176                      ;
  2177                      ;     ++  (WS1,2,3,4)  ROUNDING  ++
  2178                      ;
  2179 09D8 5F0A            BFROND: BIT     7,WS4 MOD 100H
  2180 09DA C4                      GJMP    BFRND1  ;  IF  WS4.MSB = 1
  2181                              ;
  2182 09DB 40E609                  CALL    BFRNDR
  2183 09DE B8                      RET
  2184                                              ; OVERFLOW
  2185 09DF 3402FF          BFRND1: LXI     H,ACC1
  2186 09E2 40280A                  CALL    BFSTR2
  2187                                              ; B.F.P.ACC <-  WS1,2,3
  2188 09E5 B9                      RETS
  2189                      ;
  2190                      ;     --  (WS1,2,3,4)  END-AROUND CARRY  --
  2191                      ;
  2192 09E6 2009            BFRNDR: INRW    WS3 MOD 100H
  2193 09E8 B9                      RETS
  2194                              ;
  2195 09E9 2008                    INRW    WS2 MOD 100H
  2196 09EB B9                      RETS
  2197                              ;
  2198 09EC 2007                    INRW    WS1 MOD 100H
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   55

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2199 09EE B9                    RETS
  2200                            ;
  2201 09EF 710780                MVIW    WS1 MOD 100H,80H
  2202 09F2 2000                  INRW    ACCE MOD 100H
  2203 09F4 B9                    RETS
  2204 09F5 B8                    RET
  2205                  ;
  2206                  ;     ++  WS1,2,3,4  COMPLEMENT  ++
  2207                  ;
  2208 09F6 482A        BFCOM1: CLC
  2209 09F8 6B03                  MVI     C,3
  2210 09FA 240AFF                LXI     D,WS4
  2211 09FD 6900        BFCOM2: MVI     A,0
  2212 09FF 70F2                  SBBX    D
  2213 0A01 3E                    STAX    D-
  2214 0A02 53                    DCR     C
  2215 0A03 F9                    GJMP    BFCOM2
  2216                  ;
  2217                  ;     ++  SIGN (ACCS) INVERT  ++
  2218                  ;
  2219 0A04 00                    NOP
  2220 0A05 0101        BFCOMP: LDAW    ACCS MOD 100H
  2221 0A07 1680                  XRI     A,80H
  2222 0A09 6301                  STAW    ACCS MOD 100H
  2223 0A0B B8                    RET
  2224                  ;
  2225                  ;     ++  SIGN INVERT  SUBROUTINE  ++
  2226                  ;
  2227 0A0C 00                    NOP
  2228 0A0D 40050A      BFCOMX: CALL    BFCOMP
  2229 0A10 54590B                GJMP    BFRETS
  2230                  ;
  2231                  ;     ++  CHECK, SIGN OF B.F.P.ACC  ++
  2232                  ;
  2233 0A13 0100        BFSIGN: LDAW    ACCE MOD 100H
  2234 0A15 6323                  STAW    FSN MOD 100H
  2235 0A17 6700                  NEI     A,0
  2236 0A19 B8                    RET
  2237                            ;
  2238 0A1A 5F01                  BIT     7,ACCS MOD 100H
  2239                                            ; B.F.P.ACC < 0
  2240 0A1C 69FF                  MVI     A,0FFH
  2241 0A1E 6901                  MVI     A,1
  2242 0A20 6323                  STAW    FSN MOD 100H
  2243 0A22 B8                    RET
  2244                  ;
  2245                  ;     ++  (HL,HL+1,HL+2,---,HL+4)  <-  OPERAND
  2246                  ;
  2247 0A23 0106        BFSTR1: LDAW    WSE MOD 100H
  2248 0A25 3D                    STAX    H+
  2249 0A26 0A                    MOV     A,B
  2250 0A27 3D                    STAX    H+
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   56
                (                                                )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2251 0A28 0107         BFSTR2: LDAW    WS1 MOD 100H
  2252 0A2A 3D                   STAX    H+
  2253 0A2B 0108                 LDAW    WS2 MOD 100H
  2254 0A2D 3D                   STAX    H+
  2255 0A2E 0109                 LDAW    WS3 MOD 100H
  2256 0A30 3D                   STAX    H+
  2257 0A31 B8                   RET
  2258                   ;
  2259                   ;     ++  B.F.P.ACC  : ABSOLUTE  ++
  2260                   ;
  2261 0A32 6980         BFABS:  MVI     A,80H
  2262 0A34 6301                 STAW    ACCS MOD 100H
  2263 0A36 B8                   RET
  2264                   ;
  2265                   ;     ++  (WS1,2,3,4)  RIGHT SHIFT  ++
  2266                   ;
  2267 0A37 6908         BFRSH0: MVI     A,8
  2268 0A39 050A00       BFRSH:  ANIW    WS4 MOD 100H,0
  2269 0A3C 1A                   MOV     B,A
  2270                                   ; B -- SHIFT COUNTER
  2271 0A3D C3                   GJMP    BFRS11
  2272                   ;
  2273                   BFRSH1:
  2274 0A3E 40440A               CALL    BFRSH2
  2275 0A41 52           BFRS11: DCR     B
  2276 0A42 FB                   GJMP    BFRSH1
  2277 0A43 B8                   RET
  2278                           ;
  2279 0A44 482A         BFRSH2: CLC
  2280 0A46 2407FF       BFRSH4: LXI     D,WS1
  2281 0A49 6B03                 MVI     C,3
  2282                           ;
  2283 0A4B 2A           BFRSH3: LDAX    D
  2284 0A4C 4831                 RLR     A
  2285 0A4E 3C                   STAX    D+
  2286 0A4F 53                   DCR     C
  2287 0A50 FA                   GJMP    BFRSH3
  2288 0A51 B8                   RET
  2289                   ;
  2290                   ;     ++  (WS1,2,3,4)  LEFT  SHIFT  ++
  2291                   ;
  2292                           ; B -- SHIFT COUNTER
  2293                   ;
  2294 0A52 240AFF               LXI     D,WS4
  2295 0A55 2A           BFLSH1: LDAX    D
  2296 0A56 4835                 RLL     A
  2297 0A58 3E                   STAX    D-
  2298                           ;
  2299 0A59 52                   DCR     B
  2300 0A5A FA                   GJMP    BFLSH1
  2301 0A5B B8                   RET
  2302                   ;
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   57

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2303                      ;     ++  BINARY FLOATING  ->  FIXED  ++
  2304                      ;         (ACC1,2,3)           (WS1,2,3,4)
  2305                      ;
  2306 0A5C 6908        BFIX0:  MVI     A,8
  2307 0A5E 6920        BFIX2:  MVI     A,32
  2308                          ;
  2309 0A60 2406FF              LXI     D,WSE
  2310 0A63 3C                  STAX    D+
  2311                                          ; DE= WS1 ADDR
  2312 0A64 3400FF      BFIX:   LXI     H,ACCE
  2313 0A67 650000              NEIW    ACCE MOD 100H,0
  2314 0A6A DE                  GJMP    BFIXED  ; IF B.F.P.ACC = 0
  2315                          ;
  2316 0A6B 467F                ADI     A,80H-1 ; APPLY BIAS = 1  FOR SIGN
  2317 0A6D 70B5                SUBNBX  H+      ; DIFFERENCE OF EXP
  2318 0A6F B8                  RET
  2319                          ;
  2320 0A70 371F                LTI     A,32-1
  2321 0A72 D6                  GJMP    BFIXED  ; IF B.F.P.ACC  TOO SMALL
  2322                          ;
  2323 0A73 4601                ADI     A,1
  2324 0A75 1A                  MOV     B,A     ; B - SHIFT COUNTER
  2325                                          ; DE =  WS1 ADDR
  2326 0A76 3402FF              LXI     H,ACC1
  2327 0A79 2D                  LDAX    H+
  2328 0A7A 3C                  STAX    D+
  2329 0A7B 2D                  LDAX    H+
  2330 0A7C 3C                  STAX    D+
  2331 0A7D 2B                  LDAX    H
  2332 0A7E 3A                  STAX    D
  2333                          ;
  2334 0A7F 0A                  MOV     A,B
  2335 0A80 40390A              CALL    BFRSH
  2336                          ;
  2337 0A83 5F01                BIT     7,ACCS MOD 100H
  2338                                          ; IF < 0, COMPLEMENT
  2339 0A85 40F609              CALL    BFCOM1
  2340 0A88 C9                  GJMP    BFIXEN
  2341                          ;
  2342 0A89 6900        BFIXED: MVI     A,0     ; WS1,2,3,4  0CLEAR
  2343 0A8B 3407FF              LXI     H,WS1
  2344                          ;
  2345 0A8E 3D                  STAX    H+
  2346 0A8F 3D                  STAX    H+
  2347 0A90 3D                  STAX    H+
  2348 0A91 3D                  STAX    H+
  2349                          ;
  2350 0A92 010A        BFIXEN: LDAW    WS4 MOD 100H
  2351 0A94 6321                STAW    TMPWSL MOD 100H
  2352 0A96 B9                  RETS
  2353                    ;
  2354                    ;     ++  BINARY FIXED  ->  FLOATING  ++
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   58

                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  2355                    ;       (WS1,2,3,4)          (ACC1,2,3)
  2356                    ;
  2357 0A97 2406FF      BFLT:   LXI     D,WSE
  2358 0A9A 2C                  LDAX    D+
  2359                                                 ; DE= WS1
  2360 0A9B 1680                XRI     A,80H
  2361                                                 ; APPLY EXP BIAS
  2362 0A9D 3400FF              LXI     H,ACCE
  2363 0AA0 3D                  STAX    H+
  2364                                                 ; HL= ACCS ADDR
  2365 0AA1 6980                MVI     A,80H
  2366 0AA3 3D                  STAX    H+
  2367                                                 ; HL= ACC1 ADDR
  2368 0AA4 2A                  LDAX    D
  2369 0AA5 4835                RLL     A                    ; CY FROM MSB
  2370 0AA7 540304              GJMP    BFAD11
  2371                                                 ; JUMP TO NORMAL,ROUNDING
  2372                    ;
  2373                    ;     ++  B.F.P.ACC  INTEGER PART  ++
  2374                    ;
  2375                    BFINT0:
  2376 0AAA 40130A              CALL    BFSIGN  ; GET X.SIGN
  2377 0AAD 41                  INR     A
  2378 0AAE C7                  GJMP    BFINT   ; NOT / X<0
  2379                          ;
  2380 0AAF 34D508              LXI     H,BFN1
  2381 0AB2 407D03              CALL    BFSUB   ; X-1
  2382 0AB5 B8                  RET             ; IF OVERFLOW
  2383                          ;
  2384 0AB6 405E0A      BFINT:  CALL    BFIX2
  2385 0AB9 B8                  RET             ; NOT DISPLAY
  2386 0ABA 40970A              CALL    BFLT
  2387 0ABD 00                  NOP
  2388 0ABE B9                  RETS
  2389                    ;
  2390                    ;     ++  EXPONENT  PART  OPERATION  ++
  2391                    ;                         MULTIPLY,DIVISION
  2392                    ;
  2393 0ABF 2406FF      BFRD:   LXI     D,WSE
  2394 0AC2 3400FF              LXI     H,ACCE
  2395                          ;
  2396 0AC5 2B                  LDAX    H
  2397 0AC6 6700                NEI     A,0
  2398 0AC8 4E8F                GJMP    BFRETS  ; IF B.F.P.ACC = 0
  2399                          ;
  2400 0ACA 70C2                ADDX    D
  2401 0ACC 3A                  STAX    D
  2402 0ACD 4831                RLR     A
  2403                          ;
  2404 0ACF 07C0                ANI     A,11000000B
  2405 0AD1 6700                NEI     A,00000000B
  2406 0AD3 D5                  GJMP    BFRDZ   ; IF UNDER FLOW
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    59

              (                                        )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2407                                ;
  2408 0AD4 67C0                      NEI     A,11000000B
  2409 0AD6 B8                        RET             ;IF OVER FLOW
  2410 0AD7 2C                        LDAX    D+
  2411 0AD8 1680                      XRI     A,80H
  2412 0ADA 3D                        STAX    H+
  2413                                ;
  2414 0ADB 481C                      SKN     Z
  2415 0ADD B9                        RETS            ;IF ZERO
  2416                                ;
  2417 0ADE 2B                        LDAX    H       ;GET ACCS
  2418 0ADF 7092                      XRAX    D       ;ACCS EXR WS1
  2419 0AE1 0780                      ANI     A,80H
  2420 0AE3 3B                        STAX    H       ; RESULT SIGN
  2421 0AE4 2A                        LDAX    D
  2422 0AE5 1780                      ORI     A,80H   ; WS1 -- NON SIGNED
  2423 0AE7 3A                        STAX    D
  2424 0AE8 B9                        RETS
  2425                                ;
  2426                                ;   IF  UNDER FLOW
  2427                                ;
  2428 0AE9 050000      BFRDZ:        ANIW    ACCE MOD 100H,0
  2429 0AEC B9                        RETS
  2430                  ;
  2431                  ;
  2432                  ;=============================================
  2433                  ;=      STORE BFP. ACCUMULATOR SUBROUTINE      =
  2434                  ;=============================================
  2435                  ;
  2436 0AED 3406FF      STORE0: LXI     H,WSE
  2437 0AF0 340BFF      STORE1: LXI     H,WXE
  2438 0AF3 340FFF      STORE2: LXI     H,WYE
  2439                  ;
  2440 0AF6 B2                        PUSH    D       ;PUSH DE REG.
  2441                  ;
  2442 0AF7 2400FF                    LXI     D,ACCE
  2443 0AFA 2C                        LDAX    D+
  2444 0AFB 3D                        STAX    H+      ;STORE ACCE
  2445 0AFC 2C                        LDAX    D+
  2446 0AFD 7094                      XRAX    D+
  2447 0AFF 3D                        STAX    H+      ;STORE ACCS+ACC1
  2448 0B00 2C                        LDAX    D+
  2449 0B01 3D                        STAX    H+      ;STORE ACC2
  2450 0B02 2C                        LDAX    D+
  2451 0B03 3D                        STAX    H+      ;STORE ACC3
  2452                  ;
  2453 0B04 A2                        POP     D       ;POP DE REG.
  2454                  ;
  2455 0B05 B8                        RET
  2456                  ;
  2457                  ;       ;       ;       ;       ;       ;       ;
  2458                  ;=============================================
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE    60

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2459                      ;=       LOAD BFP.ACCUMULATOR SUBROUTINE          =
  2460                      ;=================================================
  2461                      ;
  2462 0B06 3406FF          LOADA0: LXI      H,WSE
  2463 0B09 340BFF          LOADA1: LXI      H,WXE
  2464 0B0C 340FFF          LOADA2: LXI      H,WYE
  2465                      ;
  2466 0B0F B2              LOADA:  PUSH     D          ;PUSH DE REG.
  2467                      ;
  2468 0B10 2400FF                  LXI      D,ACCE
  2469                      ;
  2470 0B13 2D                      LDAX     H+
  2471 0B14 3C                      STAX     D+         ;LOAD ACCE
  2472 0B15 2B                      LDAX     H
  2473 0B16 1680                    XRI      A,80H
  2474 0B18 0780                    ANI      A,80H
  2475 0B1A 3C                      STAX     D+         ;LOAD ACCS
  2476 0B1B 2D                      LDAX     H+
  2477 0B1C 1780                    ORI      A,80H
  2478 0B1E 3C                      STAX     D+         ;LOAD ACC1
  2479 0B1F 2D                      LDAX     H+
  2480 0B20 3C                      STAX     D+         ;LOAD ACC2
  2481 0B21 2D                      LDAX     H+
  2482 0B22 3C                      STAX     D+         ;LOAD ACC3
  2483                      ;
  2484 0B23 A2                      POP      D          ;POP DE REG.
  2485                      ;
  2486 0B24 B8                      RET
  2487                      ;
  2488                      ;       ;        ;        ;       ;       ;       ;
  2489                      ;=================================================
  2490                      ;=       LOAD OPERAND SUBROUTINE                  =
  2491                      ;=================================================
  2492                      ;
  2493 0B25 340BFF          LOADO1: LXI      H,WXE
  2494 0B28 340FFF          LOADO2: LXI      H,WYE
  2495                      ;
  2496 0B2B B2              LOADO3: PUSH     D          ;PUSH DE REG.
  2497                      ;
  2498 0B2C 2406FF                  LXI      D,WSE
  2499                      ;
  2500 0B2F 2D                      LDAX     H+
  2501 0B30 3C                      STAX     D+         ;LOAD WSE
  2502 0B31 2D                      LDAX     H+
  2503 0B32 3C                      STAX     D+         ;LOAD WS1
  2504 0B33 2D                      LDAX     H+
  2505 0B34 3C                      STAX     D+         ;LOAD WS2
  2506 0B35 2D                      LDAX     H+
  2507 0B36 3C                      STAX     D+         ;LOAD WS3
  2508                      ;
  2509 0B37 A2                      POP      D          ;POP DE REG.
  2510                      ;
```

```
** UPD7811     ASSEMBLE LIST **          (1985.04.29  MON.      )  PAGE   61
                (                                    )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  2511 0B38 B8                      RET
  2512                      ;
  2513                      ;=============================================
  2514                      ;=        PUSH BFP.ACCUMULATOR                =
  2515                      ;=============================================
  2516                      ;
  2517                      PUSHF:
  2518 0B39 40ED0A                  CALL    STORE0
  2519                      ;
  2520                      ;       ;       ;       ;       ;       ;       ;
  2521                      ;=============================================
  2522                      ;=        PUSH OPERAND                        =
  2523                      ;=============================================
  2524                      ;
  2525                      PUSHR:
  2526 0B3C A1                      POP     B
  2527                      ;
  2528 0B3D 702F06FF                LDED    WSE
  2529 0B41 B2                      PUSH    D
  2530 0B42 702F08FF                LDED    WS2
  2531 0B46 B2                      PUSH    D
  2532                      ;
  2533 0B47 B1                      PUSH    B
  2534                      ;
  2535 0B48 B8                      RET
  2536                      ;
  2537                      ;
  2538                      ;=============================================
  2539                      ;=        POP  OPERAND                        =
  2540                      ;=============================================
  2541                      ;
  2542                      POPR:
  2543 0B49 A1                      POP     B
  2544                      ;
  2545 0B4A A2                      POP     D
  2546 0B4B 702E08FF                SDED    WS2
  2547 0B4F A2                      POP     D
  2548 0B50 702E06FF                SDED    WSE
  2549                      ;
  2550 0B54 B1                      PUSH    B
  2551 0B55 B8                      RET
  2552                      ;
  2553                      ;     ++  RET  PROCESS  ++
  2554                      ;
  2555 0B56 482B            BFRET:  STC
  2556 0B58 B8                      RET
  2557                      ;
  2558                      ;     ++  RETS  PROCESS  ++
  2559                      ;
  2560 0B59 750000          BFRETS: EQIW    ACCE MOD 100H,0
  2561 0B5C 00                      NOP
  2562 0B5D B9                      RETS
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   62

                (                                      )

E STNO ADRS OBJECT    M   SOURCE STATEMENT

  2563                      ;
  2564                      ;       ;       ;       ;       ;       ;       ;
  2565                      ;
  2566                      ;       FUNCTION  TABLE
  2567                      ;
  2568                      FTABL:
  2569 0B5E 5349CE                   DB      'S','I','N'+K
  2570 0B61 6C05                     DW      SIN
  2571                      ;
  2572 0B63 434FD3                   DB      'C','O','S'+K
  2573 0B66 5105                     DW      COS
  2574                      ;
  2575 0B68 5441CE                   DB      'T','A','N'+K
  2576 0B6B CE05                     DW      TAN
  2577                      ;
  2578 0B6D 4CCE                     DB      'L','N'+K
  2579 0B6F F105                     DW      LNX
  2580                      ;
  2581 0B71 4C4FC7                   DB      'L','O','G'+K
  2582 0B74 5606                     DW      LOG
  2583                      ;
  2584 0B76 4558D0                   DB      'E','X','P'+K
  2585 0B79 6206                     DW      EXP
  2586                      ;
  2587 0B7B 41524354                 DB      'A','R','C','T','A','N'+K
           41CE
  2588 0B81 6507                     DW      ARCTAN
  2589                      ;
  2590 0B83 41524353                 DB      'A','R','C','S','I','N'+K
           49CE
  2591 0B89 9B07                     DW      ARCSIN
  2592                      ;
  2593 0B8B 41524343                 DB      'A','R','C','C','O','S'+K
           4FD3
  2594 0B91 D607                     DW      ARCCOS
  2595                      ;
  2596 0B93 5351D2                   DB      'S','Q','R'+K
  2597 0B96 CF06                     DW      SQR
  2598                      ;
  2599 0B98 00                       DB      0         ;TABLE END
  2600                      ;
  2601           0000                STACK   EQU     0
  2602           0008                CTS     EQU     00001000B
  2603           0080                K       EQU     80H
  2604                      ;
  2605           FF00                ORG     0FF00H
  2606                      ;
  2607 FF00      0001                ACCE:   DS      1
  2608 FF01      0001                ACCS:   DS      1
  2609 FF02      0001                ACC1:   DS      1
  2610 FF03      0001                ACC2:   DS      1
  2611 FF04      0001                ACC3:   DS      1
```

```
** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   63

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

  2612                        ;
  2613 FF05      0001              SF:      DS       1
  2614                        ;
  2615 FF06      0001              WSE:     DS       1
  2616 FF07      0001              WS1:     DS       1
  2617 FF08      0001              WS2:     DS       1
  2618 FF09      0001              WS3:     DS       1
  2619 FF0A      0001              WS4:     DS       1
  2620                        ;
  2621                        ;
  2622 FF0B      0001              WXE:     DS       1
  2623 FF0C      0001              WX1:     DS       1
  2624 FF0D      0001              WX2:     DS       1
  2625 FF0E      0001              WX3:     DS       1
  2626                        ;
  2627 FF0F      0001              WYE:     DS       1
  2628 FF10      0001              WY1:     DS       1
  2629 FF11      0001              WY2:     DS       1
  2630 FF12      0001              WY3:     DS       1
  2631                        ;
  2632 FF13      0001              WD1:     DS       1
  2633 FF14      0001              WD2:     DS       1
  2634 FF15      0001              WD3:     DS       1
  2635 FF16      0001              WD4:     DS       1
  2636 FF17      0001              WD5:     DS       1
  2637 FF18      0001              WD6:     DS       1
  2638                        ;
  2639 FF19      0001              WA1:     DS       1
  2640 FF1A      0001              WA2:     DS       1
  2641 FF1B      0001              WA3:     DS       1
  2642                        ;
  2643 FF1C      0001              POWERK:  DS       1
  2644 FF1D      0001              LNWORK:  DS       1
  2645 FF1E      0001              CMULT0:  DS       1
  2646 FF1F      0001              CMULT1:  DS       1
  2647 FF20      0001              CSIN:    DS       1
  2648                        ;
  2649 FF21      0001              TMPWSL:  DS       1
  2650 FF22      0001              FASIN:   DS       1
  2651 FF23      0001              FSN:     DS       1
  2652                        ;
  2653 FF24      0002              ADDRS:   DS       2
  2654 FF26      0001              TMP1:    DS       1
  2655 FF27      0001              TMP2:    DS       1
  2656 FF28      0001              TMP3:    DS       1
  2657 FF29      0001              TMP4:    DS       1
  2658                        ;
  2659 FF2A      000D              OUTADR:  DS       13
  2660 FF37      0050              CHARST:  DS       80
  2661                        ;
  2662                              END

** UPD7811      ASSEMBLE LIST **          (1985.04.29  MON.     )  PAGE   64

                (                                    )

E STNO ADRS OBJECT    M    SOURCE STATEMENT

ERROR =     0
 (    0)
OPTIMIZATION COMPLETED.  TIMES =     8
```

```
** UPD7811      CROSS REFERENCE LIST ** (1985.04.29  MON.     )  PAGE    1

SYMBOL DEF. REF.  ( STATEMENT NUMBER )

ACC1   2609 2185  2326
ACC2   2610
ACC3   2611 0847  0994  1124  1145
ACCE   2607 0392  0552  0587  0595  0604  0872  0971  1166  1395  1401
            1413  1426  1509  1541  1545  1570  1682  1708  2154  2172
            2202  2233  2312  2313  2362  2394  2428  2442  2468  2560
ACCS   2608 0501  0572  0597  0784  0910  2220  2222  2238  2262  2337
ACOS1  1831 1811
ACOS2  1838 1815
ADADJ  0322 0078  0168  0210  0236  0295  0300  0488  0498
ADDRS  2653 0065  0278  0323  0325  0389  0531  0533
ARCCOS 1805 2594
ARCSIN 1747 2591
ARCTAN 1699 1768  1817  1952  2588
ASIN1  1758 1752
ASIN2  1775 1760
ASIN3  1783 1767
ATAN1  1708 1702
ATAN2  1725 1709
ATNCON 2070 2110
BF2DV  2028 1312  1591
BF2PDV 2038 1261  1267  1728  1783  1840  1999
BF2PI  2036 1292
BF4DV  2030 1303  1324
BFABS  2261 0581  1259  1501  1619
BFAD10 0886 0891
BFAD11 0893 2370
BFAD2  0809 0799
BFAD3  0843 0804
BFAD4  0858 0863
BFAD9  0880 0851
BFADD  0766 0449  0602  1262  1304  1313  1325  1404  1825  1967
BFADD0 0770 1457
BFADD1 0772 0178  0745  0767  1987  2146
BFADR  0903 0866  0874
BFADS  0909 0791  0810
BFCOM1 2208 0894  2339
BFCOM2 2211 2215
BFCOMP 2220 0174  1270  1288  1322  1329  1419  1456  1653  1668  1706
            1756  1853  2005  2228
BFCOMX 2228 1289  1704  1717  1754
BFDIV  1066 0222  0514  0619  1293  1480
BFDIV0 1070 1371
BFDIV1 1074 1067  1410  1556  1715  1766  1814  1949
BFDV58 2022 0601
BFDX   1124 1093
BFDX1  1129 1134
BFDX3  1171 1167
BFDX4  1185 1218
BFDX5  1193 1198
BFDX6  1202 1243
BFDX7  1232 1219
BFDX8  1237 1242
BFDX9  1225 1162
```

```
** UPD7811      CROSS REFERENCE LIST ** (1985.04.29  MON.     )  PAGE    2

SYMBOL DEF. REF.  ( STATEMENT NUMBER )

BFIN11 0410 0404
BFIN12 0415
BFIN13 0422 0413
BFIN14 0426
BFIN30 0489 0485
BFIN31 0464 0461
BFIN32 0467 0463
BFIN33 0487 0477
BFIN40 0499 0491  0495
BFIN41 0503 0517  0523
BFIN42 0513
BFIN43 0519 0511
BFINP  0388 0296
BFINP1 0407 0400  0402  0419  0452  0455
BFINP2 0431
BFINP3 0457 0424
BFINP4 0497 0420  0429  0471
BFINT  2384 1296  1630  2378
BFINT0 2375 1512
BFIX   2312
BFIX0  2306 0658
BFIX2  2307 2384
BFIXED 2342 2314  2321
BFIXEN 2350 2340
BFLGE2 2040 1504
BFLN10 2044 1479
BFLN2  2042 1450  1522
BFLSH1 2295 1213  2300
BFLT   2357 0446  0667  1447  2386
BFMPI  2034 1965
BFMUL  0935 0434  0520  0611  0656  1451  1506  1523  1891  1899  1977
            1983
BFMUL0 0939 2114
BFMUL1 0941 0204  0936  1661  1846  2120  2141
BFMULZ 0971 0942
BFMX   0976 0956
BFMX1  0981 0989
BFMX2  0995 1003
BFMX3  1009 0999
BFN01  2024 0591
BFN1   2026 1403  1408  1416  1554  1712  1849  2380
BFNOR  2153 0898  0960
BFNOR0 2156 2174
BFNOR1 2166 2170
BFOU00 0560 0563
BFOU01 0568 0553
BFOU02 0579 0573
BFOU03 0599 0596
BFOU11 0608 0598
BFOU12 0617 0588
BFOU21 0638 0650  0669
BFOU22 0654 0640
BFOU30 0675 0566
BFOU31 0682 0674
```

```
** UPD7811      CROSS REFERENCE LIST ** (1985.04.29  MON.      )  PAGE    3

SYMBOL DEF. REF.  ( STATEMENT NUMBER )

BFOU32 0690 0685
BFOU33 0694 0700
BFOU34 0704 0680
BFOUT  0548 0135
BFOUT1 0586 0605  0615  0623
BFOUT2 0625 0606
BFOUT3 0671 0645
BFPI   2032 1824  1964
BFRD   2393 0947  1083
BFRDZ  2428 2406
BFRET  2555 0435  0450  0521  0875  0904  0948  0966  1077  1084  1094
            1115  1339  1376  1393  1477  1557  1566  1609  1662  1665
            1675  1678  1779  1834  1888  1907  2013
BFRETS 2560 0506  0773  0805  0899  0905  0915  0951  0961  0967  0972
            1087  1116  1549  1558  1571  1669  1683  1731  1785  1822
            1827  1842  1902  2229  2398
BFRND1 2185 2180
BFRNDR 2192 2182
BFROND 2179 0903  0965  1114
BFRS11 2275 2271
BFRSH  2268 0843  2335
BFRSH0 2267 1014
BFRSH1 2273 2276
BFRSH2 2279 2274
BFRSH3 2283 2287
BFRSH4 2280 0870  1141
BFSIGN 2233 1284  1306  1315  1390  1498  1600  1636  1700  1748  1806
            1943  2376
BFSTR1 2247 0914
BFSTR2 2251 2186
BFSUB  0738 0592  1268  1417  1850  2381
BFSUB1 0741 1301  1529  1634  1719
BFTEN  2020 0433  0509  0610  0618  0655
CHARD  0530 0068  0115  0156  0192  0232  0243  0285  0289  0298  0398
            0408  0459  0466  0474
CHARST 2660 0048  0064
CI     0333 0052  0335  0338
CMULT0 2645 0993  1002
CMULT1 2646 1022  1029
CO     0349 0054  0107  0109  0111  0122  0126  0128  0141  0351
CO1    0355 0357
CO2    0362 0367
COS    1258 1359  1895  2573
COS1   1266 1263
CSIN   2647 1307  1316  1328
CTS    2602 0350
ERROR  0120 0061  0099  0179  0205  0223  0291  0297  0313
EXACOP 0829 0837
EXP    1497 1605  1612  1664  2585
EXP0   1562 1520
EXP1   1563 1507  1510  1513
EXP2   1564 1543
EXPCON 2059 1535
EXPR   0152 0066  0091  0288
```

```
** UPD7811      CROSS REFERENCE LIST ** (1985.04.29  MON.      )  PAGE    4

SYMBOL DEF. REF.  ( STATEMENT NUMBER )

EXPR1  0155 0180
EXPR2  0166 0158
EXPR3  0176 0164
FACT11 0242 0253
FACT31 0291 0263
FACTR  0231 0188  0200  0215  0234  0267  0305
FACTR1 0239 0281
FACTR2 0271 0249  0274
FACTR3 0284
FACTR4 0294 0287
FACTR5 0298 0292
FASIN  2650 1750  1776
FSN    2651 1337  2234  2242
FTABL  2568 0238
K      2603 2569  2572  2575  2578  2581  2584  2587  2590  2593  2596
LNCON  2082 2111
LNWORK 2644 1397  1437  1454
LNX    1390 1476  1657  2579
LOADA  2466 1409  1555  1713  1784  1841  2000  2130
LOADA0 2462 0094  0104  0220  0310  0600  0609  1364  1597  1643  1764
            1884  1938  2118
LOADA1 2463 0714  1266  1894  1975
LOADA2 2464 1981
LOADO1 2493 1365  1598  1655  1765  1941
LOADO2 2494
LOADO3 2496 0738  0766  0935  1066  1592  1729  1881  1933  2144
LOG    1476 2582
MAIN1  0044 0130
MAIN2  0051 0060
MAIN3  0063 0058
MAIN31 0075 0070
MAIN32 0080 0076
MAIN33 0083 0073
MAIN4  0113 0079
MAIN40 0099 0086
MAIN5  0119
MAIN6  0124 0117
MULT   1021 1009
MULT1  1027 1030
MULT2  1033 1027
OUT    0134 0102  0113
OUT1   0139 0143
OUTADR 2659 0134  0137
POLY   2127 1536
POLY1  2132 2149
POLYN1 2109 1332
POLYN2 2110 1726
POLYN3 2111 1424
POLYX  2113
POPR   2542 0092  0103  0177  0203  0219  0309  0594  0770  0939  1071
            1300  1336  1363  1375  1527  1563  1633  1641  1660  1673
            1674  1677  1763  1775  1813  1832  1901  1906  1986  1993
            2011  2012  2133
POWER  1592 0312
```

** UPD7811 CROSS REFERENCE LIST ** (1985.04.29 MON.) PAGE 5

SYMBOL DEF. REF. (STATEMENT NUMBER)

POWER1 1596
POWER2 1611 1602
POWER3 1641 1627
POWER4 1651
POWER5 1673 1631
POWER6 1677 1658
POWERK 2643 1646 1667
POWERZ 1682 1608
PUSHF 2517 0090 0162 0172 0199 0214 0304 0590 1295 1358 1428
1502 1758 1809 1893 1971 1980
PUSHR 2525 0101 1366 1617 1629 1656 2113 2127 2137
ROUTE 1846 1759 1810
SF 2613 0787 0821 0850 0909
SIN 1284 1264 1368 1887 2570
SIN0 1292 1286
SIN1 1321 1310
SINCOS 2051 2109
SINOVR 1336 1298
SQR 1591 1855 1990 2597
STACK 2601 0046
STORE0 2436 1331 1406 1420 1531 1553 1620 1711 1725 2122 2518
STORE1 2437 0093 0218 0308 0436 0582 1260 1362 1596 1618 1762
1885 1937
STORE2 2438 1882 1939
TAN 1358 2576
TANOVR 1375 1360 1369
TERM 0187 0153 0163 0173
TERM1 0191 0206 0224
TERM2 0198
TERM3 0208 0194
TMP1 2654 0393 0453 0504 0508 0570 0636 0639 0642 0644
TMP2 2655 0416 0417 0451 0571 0613 0621 0626 0631 0672 0688
0689 0693
TMP3 2656 0500 0634 0641 0643
TMP4 2657 0432 0440
TMPWSL 2649 1515 1623 1645 2351
TRACA 1881 0082
TRACA1 1882
TRACA2 1906 1896
TRACB 1933 0084
TRACB1 1937
TRACB2 1998 1950
TRACB3 1971 1960 1969 2007
TRACB4 2011 1984
TRACB5 2012 1978 1988 1991
WA1 2639 0978
WA2 2640
WA3 2641 1024
WD1 2632 1202
WD2 2633
WD3 2634 1217
WD4 2635 1099
WD5 2636
WD6 2637 1146 1187 1212 1233

```
** UPD7811      CROSS REFERENCE LIST ** (1985.04.29  MON.     )  PAGE    6

SYMBOL  DEF.  REF.   ( STATEMENT NUMBER )

WS1     2616  0660   0666   0742   0743   0777   0977   1016   1614   1946   2157
              2198   2201   2251   2280   2343
WS2     2617  2195   2253   2530   2546
WS3     2618  0848   1025   1125   1232   2192   2255
WS4     2619  0793   1092   1098   1186   1208   2164   2179   2210   2268   2294
              2350
WSE     2615  0437   0665   0772   0941   1074   1079   1433   1601   2247   2309
              2357   2393   2436   2462   2498   2528   2548
WX1     2623
WX2     2624
WX3     2625
WXE     2622  0096   0221   0311   0448   1976   2437   2463   2493
WY1     2628
WY2     2629
WY3     2630
WYE     2627  1890   1898   1982   2438   2464   2494
```

PART III
μCOM-87 Hardware Interface Examples

PREFACE

The μCOM-87 series are NEC's original 8-bit, single-chip microcomputers having a 16-bit ALU. This application note describes the procedure for operating the μCOM-87 series internal hardware and interfaces with the external hardware using execution examples.

Also read Application Note (I) for details of the fixed point four rule operations and Application Note (II) for details of the floating point operation programs.

Note: This application note is applicable to the following types of microcomputers:

μPD7811/7810/78PG11E, μPD7811H/7810H/78PG11E-15, μPD78C14/78C11/78C10/78CG14, μPD7809/7808/7807/78P09

Chapter 1 Keying-in Operations

1.1 Key Matrix Control

1.1.1 Keying-in Operations

(1) Keying-in

This section describes the control of a 64-key matrix as shown in figure 1-1. In this example, output ports, PB0 to PB7, generate key scan signals (active low) and input ports, PA0 to PA7, fetch key return signals.

A pull-up resistor is connected to the PA0 to PA7 key return signal input ports so that if no key is pressed a high-level key return signal is input. A diode is connected to the PB0 to PB7 key scan signal output ports so that if two or more keys are pressed simultaneously, the ports are not destroyed.

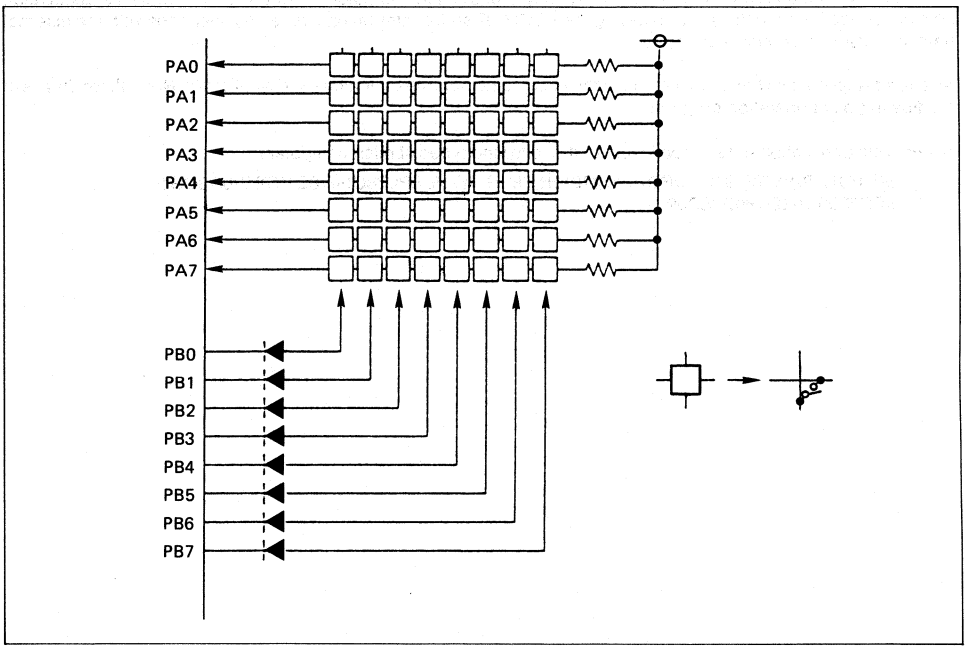


Figure 1-1 64(8 x 8)-Key Matrix

(2) Two-key pressing

The following two methods are available:

- Lock-out method: If two keys are pressed when no keys are held down, the two-key input is effective.
- Two-key roll-over method: If one key is pressed when another key is held down, the second key input is effective.

In both methods, key-input is invalidated if more than two keys are pressed simultaneously. Figure 1-2 shows key-in operations in the lock-out and two-key roll-over methods.

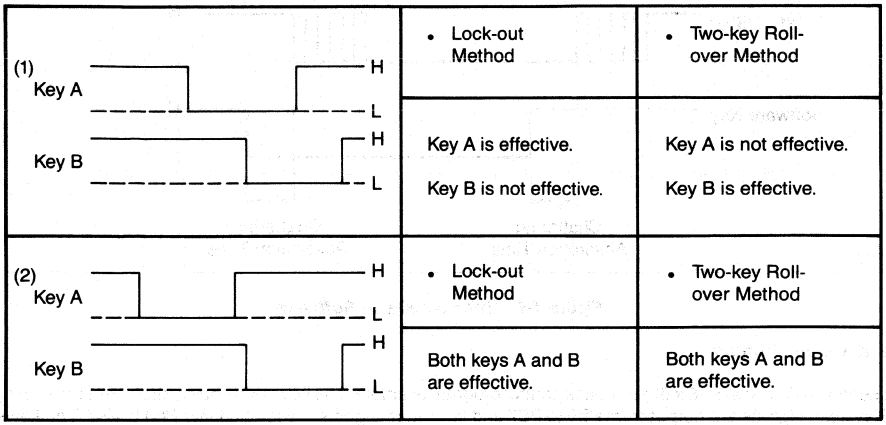


Figure 1-2 Keying-in Operations

(3) Key chattering

After a key is set to ON or OFF, an unstable state called "chattering" (ON-chattering or OFF-chattering) is generated until the corresponding key signal becomes stable at the high or low level. If the chattering floats between the two levels, the unit may malfunction. Therefore, unless the hardware is equipped with a chattering preventive circuit, the software must be able to prevent the key signal from chattering.

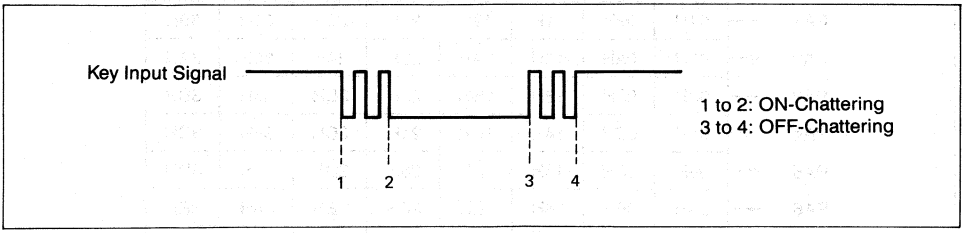


Figure 1-3 Key Input Signal Chattering

The software controls the chattering by acknowledging a key signal which has stabilized for a specified period of time. The time required for the key signal to be acknowledged is called "chattering absorption time".

When a program is created taking into account the chattering absorption time, the effective key signal in the software will be as shown in figure 1-4.

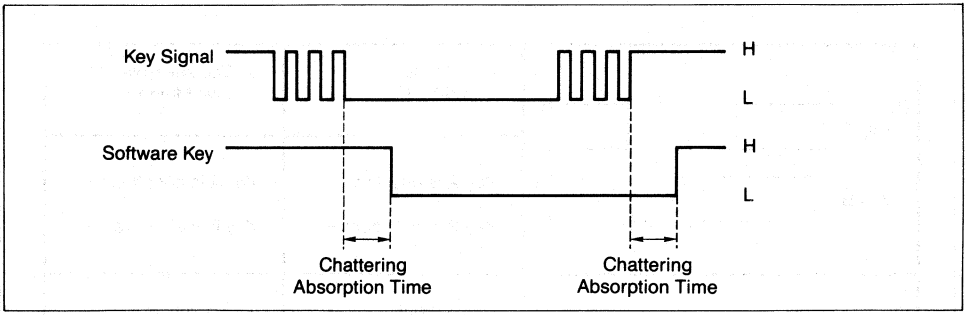


Figure 1-4 Effective Key in Software

1.1.2 Key Input Program

This section refers to a 64(8 x 8)-key matrix control program example. The key matrix has a circuit as shown in figure 1-1. Key scan signals are output from PB0 to PB7 and key return signals are fetched into PA0 to PA7. The key scan and key return signals are both processed at an active low level.

The lock-out keying-in method is used and any key input is invalidated if two or more keys are pressed simultaneously.

If the key statuses remain unchanged while the above routine is repeated several times, the chattering processing is effective. More precisely, the chattering processing is effective if the first read key status remains unchanged while the routine is executed at 5 msec intervals, the chattering absorption time is 20 to 25 msec.

The input key data is stored into the memory by 00H to 3FH key codes. Figure 1-5 shows the key codes and the corresponding key positions.

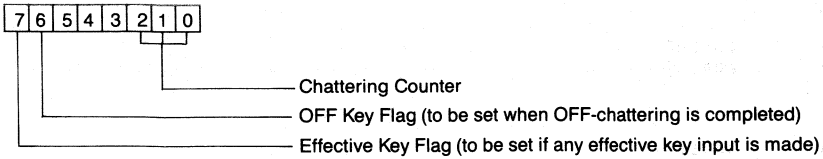
	PB0	PB1	PB2	PB3	PB4	PB5	PB6	PB7
PA0	00H	08H	10H	18H	20H	28H	30H	38H
PA1	01H	09H	11H	19H	21H	29H	31H	39H
PA2	02H	0AH	12H	1AH	22H	2AH	32H	3AH
PA3	03H	0BH	13H	1BH	23H	2BH	33H	3BH
PA4	04H	0CH	14H	1CH	24H	2CH	34H	3CH
PA5	05H	0DH	15H	1DH	25H	2DH	35H	3DH
PA6	06H	0EH	16H	1EH	26H	2EH	36H	3EH
PA7	07H	0FH	17H	1FH	27H	2FH	37H	3FH

Figure 1-5 Key Code and Key Relationships

When an effective key is input, the key effective flag is set. In the main processing operation, the key input can be judged by the key effective flag and the key code.

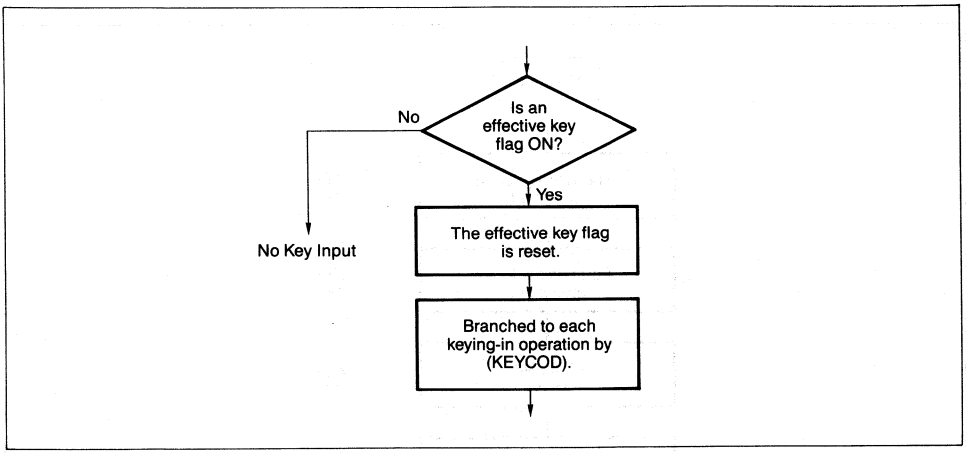
The following RAM and flag are used for keying-in operations:

- KEYCOD (1 byte): Area for storage of the input key codes
- KEYWRK (1 byte): Key flag and chattering counter



In the main processing operation effective key flags are checked to see whether or not a key input has been made. In this case, operations are carried out as follows:

- **Flowchart** (main processing)



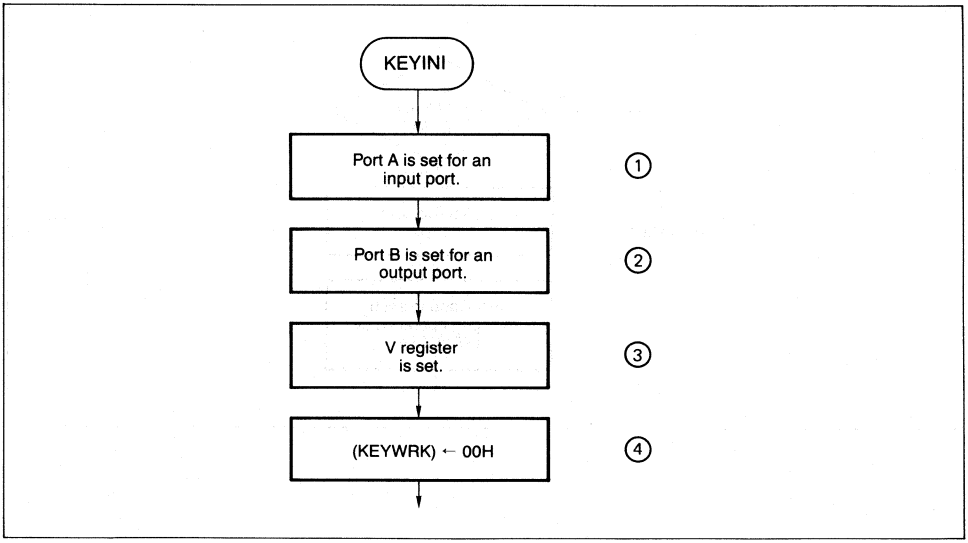
- **Program list** (main processing)

```
;********MAIN ROUTINE**************
;
MAIN00: ONIW    KEYWRK,100000003 ;CHECK! EFFECT KEY FLAG
        GJMP    NOKEY
        ANIW    KEYWRK,J11111113 ;RESET EFFECT KEY FLAG
        LDAW    KEYCOD           ;A<--KEY CODE
        ADD     A,A
        TABLE
        JB

        DW      CODE00
        DW      CODE01
        ;
        ;
        ;
        DW      CODE63
```

(1) Initialize

The initialize flow for keying-in operations is shown below:

- **Flowchart** (initialize)



① Port A is set for a key return signal input port.

② Port B is set for a key scan signal output port.

③ The V register is set at FFH.

④ KEYWRK is set to 00H and the chattering counter, the 0FF key flag, and the effective key flag are cleared.

- **Program list** (initialize)

```
;*******INITIALIZE ROUTINE***************
;
KEYINI: MVI     A,00000000B
        MOV     PB,A            ;PORT B LATCH<--00000000B
        MOV     MB,A            ;SET PORT B TO OUTPUT PORT
        MVI     A,11111111B
        MOV     MA,A            ;SET PORT A TO INPUT PORT
;
        ANIW    KEYWRK,00111000B ;INITIALIZE KEYWRK
                                ;       CHATTERING COUNTER<--000
                                ;       OFF KEY FLAG<--1
                                ;       EFFECT KEY FLAG<--0
        MVI     V,0FFH          ;
        EXA
        MVI     V,0FFH
```

(2) Keying-in operations

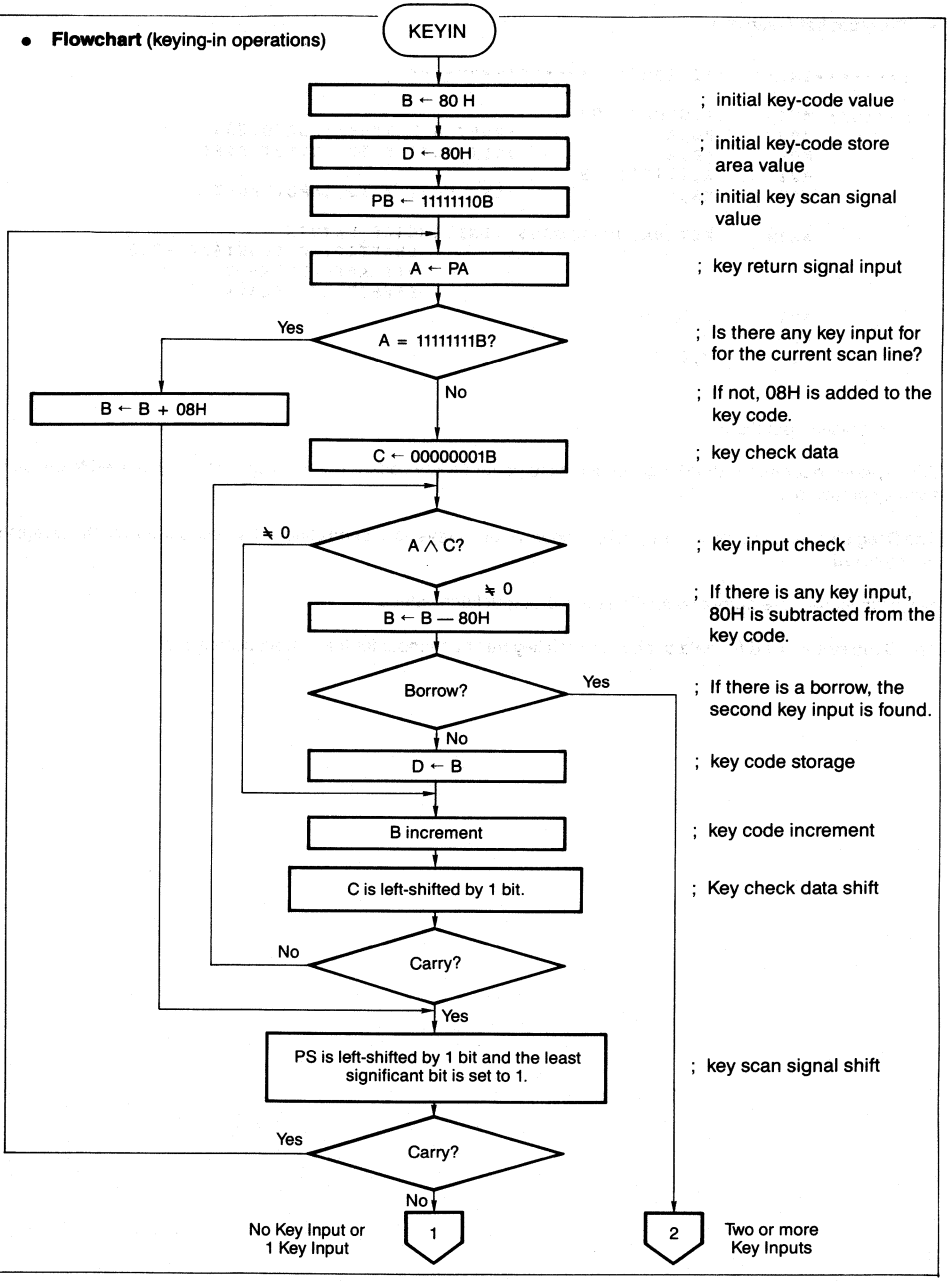
For keying-in operations, the B register serves to generate key codes. The most significant bit is used to check for two-key pressings.

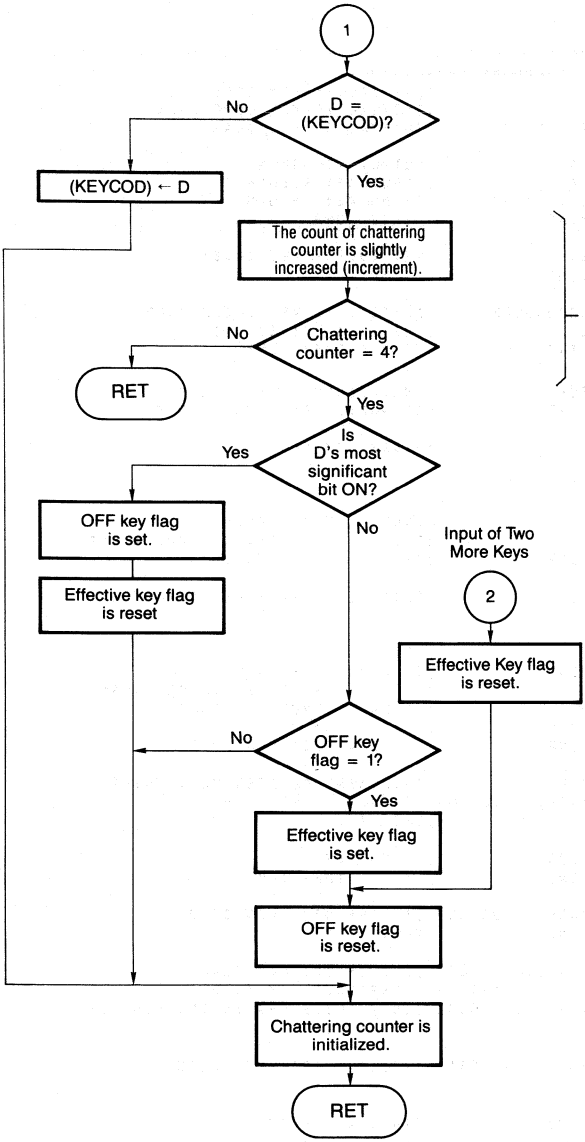
The D register serves to store the input key codes. After all keys have been checked, values stored into the D register are checked.

Key scan signals are output by shifting the PB output latch values.

The C register is used for data to check which key has been input for the current scan signal.

- **Flowchart** (keying-in operations)





; Compared to the previous key code.

; If different from previous key code, set new key-code.

; Chattering counting
The chattering processing operation completes if the counter value becomes 4.

; On key input if the most significant bit still remains.

; OFF key flag is set if OFF-chattering completes.

; Effective key input if OFF key flag is set.

; Chattering counter is set to 0.

- **Program list** (key input)

```
;*******KEY INPUT ROUTINE***************
;
;       INPUT:  -
;       OUTPUT: -
;       CHANGE: A,BC,D
;=======================================
KEYIN:  MVI     B,80H           ;INITIALIZE KEY CODE
        MVI     D,80H           ;INITIALIZE KEY CODE STORE
        MVI     A,11111110B     ;INITIALIZE KEY SCAN
;
KEYIN0: MOV     PB,A            ;OUTPUT KEY SCAN
        MOV     A,PA            ;INPUT KEY RETURN
        EQI     A,11111111B     ;IF NO KEY RETURN
        GJMP    KEYIN1          ; ELSE JUMP
;
        ADI     B,8             ;KEY CODE<--KEY CODE+8
        GJMP    KEYIN4          ; JUMP
;
KEYIN1: MVI     C,00000001B     ;INITIALIZE KEY CHECK CODE
KEYIN2: OFFA    A,C             ;CHECKING KEY IS ON?
        GJMP    KEYIN3          ;IF OFF THEN JUMP
;
        SUINB   B,80H           ;CHECK INPUT KEY NUMBER
        GJMP    KEYIN7          ;IF NUMBER IS 2 THEN JUMP
;
        MOV     A,B             ;STORE KEY CODE TO D
        MOV     D,A
        MOV     A,PA
;
KEYIN3: INR     B               ;INCREMENT KEY CODE
        SLLC    C               ;1 BIT LEFT SHIFT KEY CHECK CODE
        GJMP    KEYIN2          ;IF NO CARRY THEN JUMP
;
KEYIN4: MOV     A,PB            ;1BIT LEFT SHIFT KEY SCAN PORT
        SLL     A
        INR     A
        SKN     CY
        GJMP    KEYIN0          ;IF CARRY THEN JUMP
                                ;       ELSE FINISH KEY SCAN
        MOV     A,D             ;COMPARE INPUT KEY CODE
        EQAW    KEYCOD          ;       AND KEYCOD
        GJMP    KEYIN8          ;IF DISAGREE THEN JUMP
;
        INRW    KEYWRK          ;INCREMENT CHATTERING COUNTER
        ONIW    KEYWRK,00000100B ;IF NOT COUNT OVER
        RET                     ;THEN RET
;
        OFFI    A,10000000B     ;IF NO KEY
        GJMP    KEYIN9          ;       THEN JUMP
;
        ONIW    KEYWRK,01000000B ;IF OFF KEY FLAG IS OFF
        GJMP    KEYIN6          ;THEN JUMP
;
        ORIW    KEYWRK,10000000B ;SET EFFECT KEY FLAG
KEYIN5: ANIW    KEYWRK,10111111B ;RESET OFF KEY FLAG
KEYIN6: ANIW    KEYWRK,11111000B ;INITIALIZE CHATTERING COUNTER
        RET
;
KEYIN7: ANIW    KEYWRK,01111111B ;RESET EFFECT KEY FLAG
        GJMP    KEYIN5
;
KEYIN8: STAW    KEYCOD          ;STORE KEY CODE
        GJMP    KEYIN6
;
KEYIN9: ORIW    KEYWRK,01000000B ;SET OFF KEY FLAG
        ANIW    KEYWRK,01111111B ;RESET EFFECT KEY FLAG
        GJMP    KEYIN6
```

1.2 Combinations with Led Display

1.2.1 Display

The single-chip microcomputer controls display by limiting the number of ports so that dynamic display can be made.

Dynamic display is carried out to light up each display digit sequentially at the preset cycle. For each digit, the OFF time is longer than the ON time. The display ON/OFF cycle is set to a minimum of 50 to 100 Hz to prevent the display from flickering.

Figure 1-6 is a basic block diagram for dynamic display control of 8-digit, 7-segment display elements.

Segment signals generated from S0 to S7 are displayed at digits with active D0 to D7 digit signals.

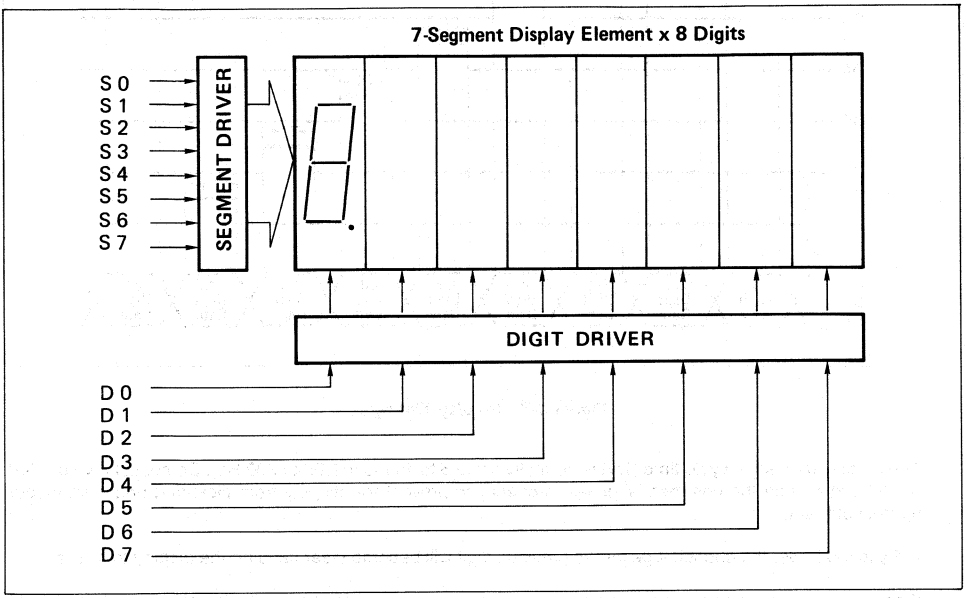


Figure 1-6 Basic Block Diagram of Dynamic Display

(a) Figure 1-7 shows the output timings for segment signals (S0 to S7) and digit signals (D0 to D7) for display processing operations. The digit signals are output sequentially from D0. Upon termination of D7 output, output from D0 is restarted. The segment signals are output for display data at digits where the digit signals are active.

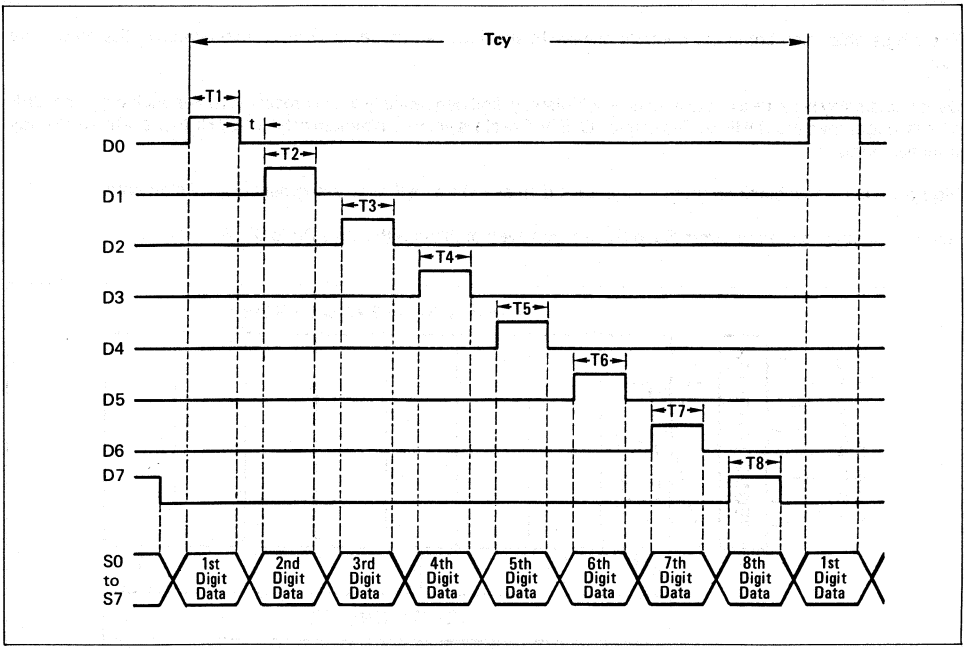


Figure 1-7 Display Timings

The display digit scan cycle time (Tcy) is normally set at 5 to 10 msec (100 to 200 Hz). Do not set a value (100 or 120 Hz) double the commercial power frequency to prevent the display from flickering under an indoor fluorescent lamp.

If Tcy is too large, the ghost image for the previous digit will become clear causing the display to flicker.

When switching the display from one digit to another, it is necessary to allow a display-off period (t) to prevent the previous digit segment output from being displayed for the next digit.

"t" should be determined in accordance with the time required for changing the outputs of the segment and digit signals (driver delay time plus software processing time).

If "t" is too large, the display-on time may be decreased and the display darkens. Thus, the display duty must be carefully set. It is given as

$$\text{Duty} = \frac{\text{Display-on time}}{\text{Disply-off time} + \text{Display-in time}} = \frac{Tn}{Tcy} \qquad \text{(where n = 1 to 8)}$$

(b) Next, the main processing and display processing timings are described.

As shown in figure 1-8, the display processing is carried out during the main processing.

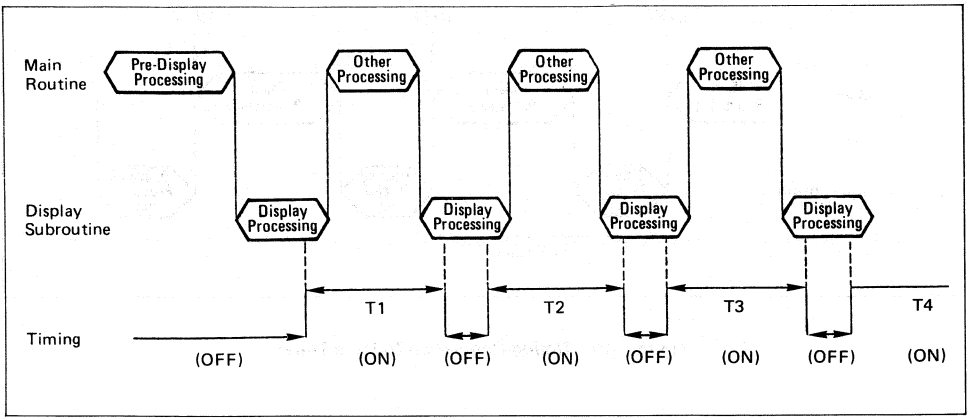


Figure 1-8 Main and Display Processing Timings

If the main processing is neither complicated nor requires many opertions, a method shown in figure 1-9 is available to generate the display processing timing.

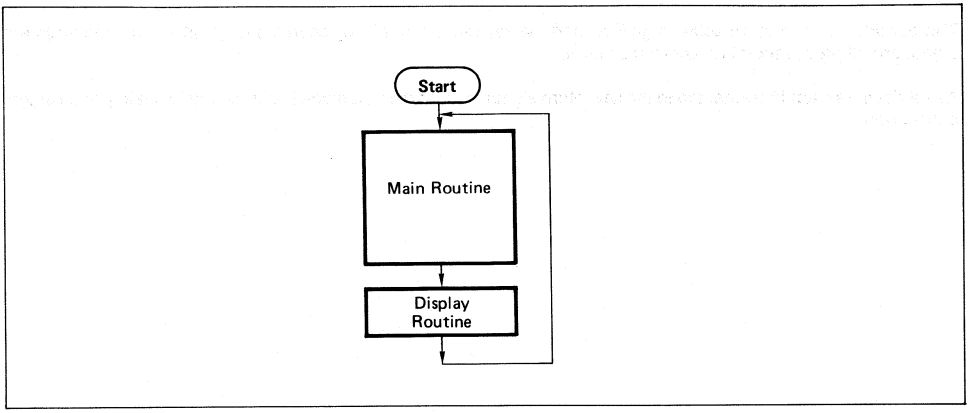


Figure 1-9 Assignment of Main and Display Routines

Another method for generating the display processing timing is available using a timer. This method enables the display processing to be executed at accurate cycles within a timer interruption. In this case the timer is operated as an interval timer, with the interval time (T) set as follows.

$$T = \frac{Tcy}{n}$$ (where n = number of digits displayed)

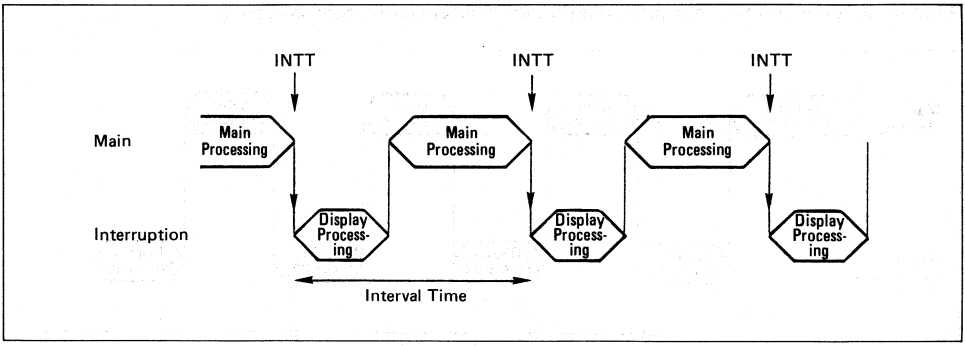


Figure 1-10 Timing Generation Using a Timer

1.2.2 Display/Keying-in Program

This section describes a program example for a 7-segment method LED 8 digits display and 40(8 x 5)-key matrix control.

Figure 1-11 shows its circuit diagram. PA0 to PA7 are used as LED segment signal output ports. The decoder output signal serves as an LED digit signal and a key scan signal. The decoder is controlled by PB0 to PB2. The key return signal is fetched from PB3 to PB7.

The decoder is used at an active high level and the key return signal, key scan signal (digit signal) and segment signal are all processed at an active high level.

A pull-down resistor is connected to the key return signal input port and a diode is connected to the key scan signal output port.

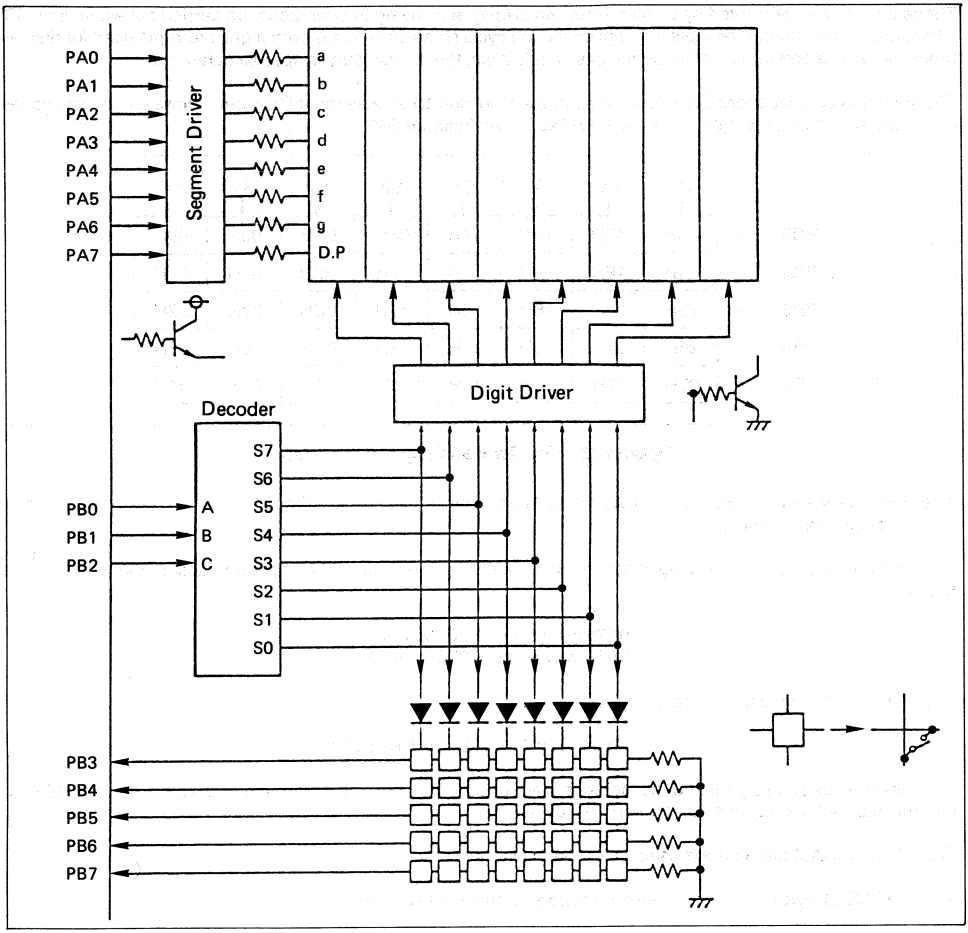


Figure 1-11 8 Digits of 7-segment LED and 40(8 x 5)-Key Matrix

Display and keying-in operations are carried out in time sharing and the timings are set as shown in figure 1-12.

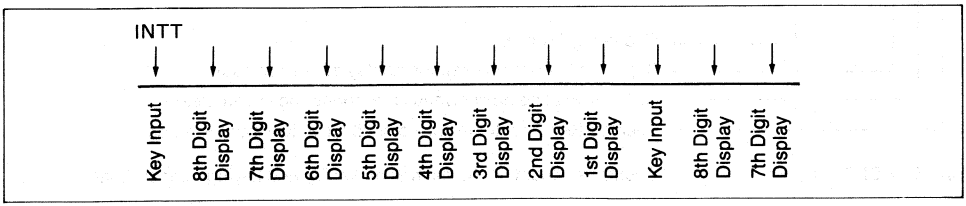


Figure 1-12 Display and Keying-in Timings

These timings are generated by a timer. Thus, the display and keying-in operations are carried out within an INTT interruption. Nine timer interrupts are applied for one cycle (scan cycle time), with eight interrupts used for display processing, and one for keying-in operations. In this case, the display duty is approximately 1/9.

The keying-in specifications are the same as those in section 1.1.2 „Key Input Program". However, the key codes which are stored into the memory are 0 to 27H as shown in figure 1-13.

	S7	S6	S5	S4	S3	S2	S1	S0
PB3	23H	1EH	19H	14H	0FH	0AH	05H	00H
PB4	24H	1FH	1AH	15H	10H	0BH	06H	01H
PB5	25H	20H	1BH	16H	11H	0CH	07H	02H
PB6	26H	21H	1CH	17H	12H	0DH	08H	03H
PB7	27H	22H	1DH	18H	13H	0EH	09H	04H

Figure 1-13 Key Code and Key Relationships

The timer interval time is set to 0.576 msec taking into account the scan cycle time for keying-in chattering processing and display operations.

Because one keying-in processing is carried out for nine timer interrupts, the chattering absorption time is set as follows:

0.576 ms x 9 x 4 = 20.736 ms (Max.)
0.576 ms x 9 x 5 = 25.92 ms (Min.)

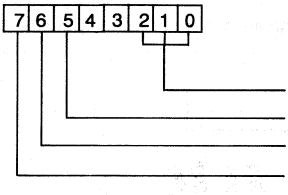
The display scan cycle time is set as follows:

0.576 ms x 9 = 5.184 ms (192.9 Hz)

Because the key scan signal and the digit signal are used in common, all segment outputs must be set to 0FF so that no incorrect display is made for keying-in operations.

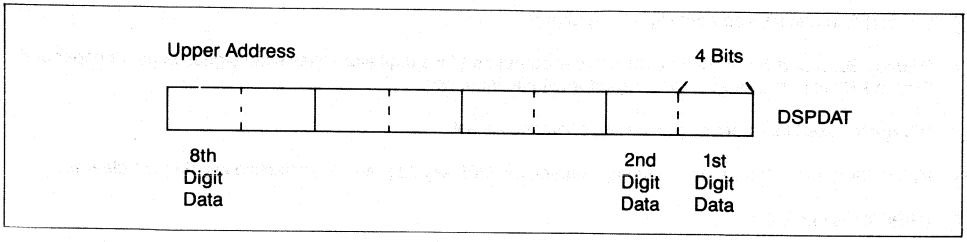
The following RAM and flag are used for keying-in operations:

- KEYCOD (1 byte): Area for storage of the input key codes
- KEYWRK (1 byte): Key flag and chattering counter



Chattering Counter
Suppress Flag (to be used for display zero suppress)
OFF Key Flag (to be set when OFF-chattering is completed)
Effective Key Flag (to be set if any effective key input is made)

- DSPCNT (1 byte): Counter for counting the timings of display and keying-in operations
- DSPDAT (4 bytes): Area for storing the display data. The 1st digit data has 4 bits (0 to FH).

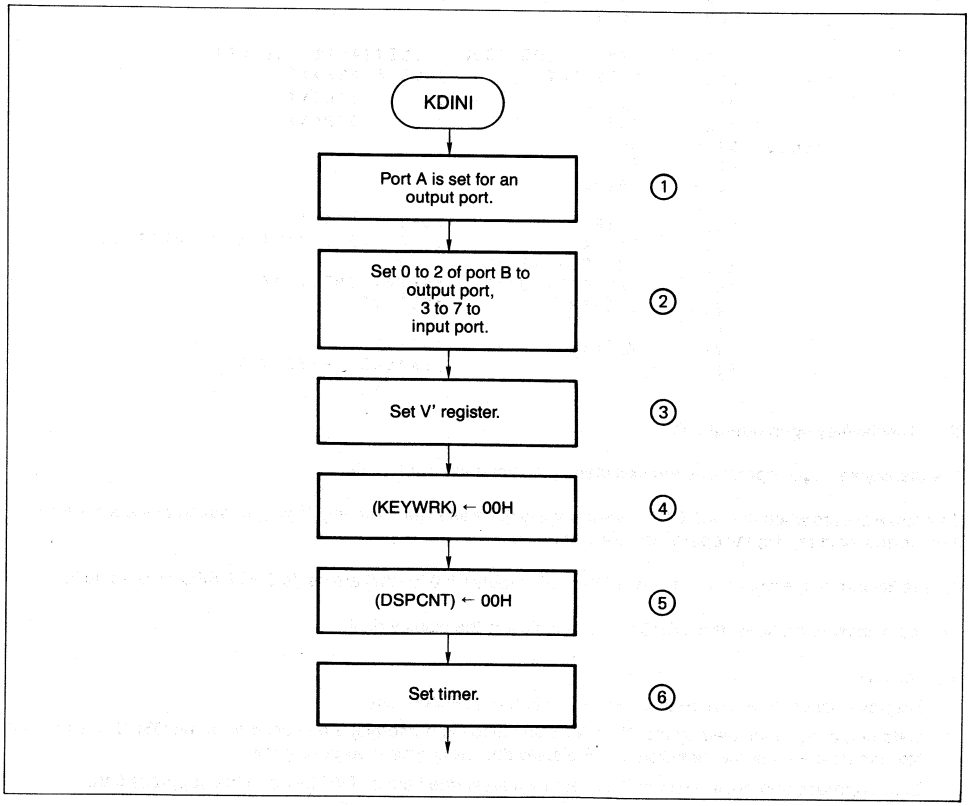


- DPOINT (1 byte): Area for indicating the decimal point position 0 to 7 are set.

(1) Initialize

The initialize flow for executing the display/key input program is shown below:

- **Flowchart** (initialize)



① Set port A to display segment signal output port.

② PB0 to PB2 of port B are set for a drive signal output port for a digit signal/key scan signal output decoder and PB3 to PB7 of port B are set for a key return signal input port.

③ V' register used for interrupt processing operations is set.

④ KEYWRK is set to 0 and the chattering counter, the OFF key flag, and the effective key flag are cleared.

⑤ DSPCNT is set to 0.

⑥ The timer operation is set and the timer interrupt is enabled.

- **Program list** (initialize)

```
;*******INNITIALIZE**********************
;
KDINI:  MVI     PA,00000000B    ;INITIALIZE PORT A
        MVI     A,00000000B
        MOV     MA,A
        MVI     A,11111000B     ;INITIALIZE PORT B
        MOV     MB,A
;-
        ANIW    KEYWRK,00011000B ;INITIALIZE KEYWRK
        LXI     H,DSPDAT        ;CLEAR DSPDAT
        MVI     A,0             ;       DPOINT
        MVI     B,5             ;       DSPCNT
KDINI0: STAX    H+
        DCR     B
        GJMP    KDINI0
;
        MVI     A,18            ;SET TIMER 1
        MOV     TM1,A           ; TO 0.576 MILI SEC INTERVAL
        ANI     TMM,00111111B
        ANI     MKL,11111011B   ;RESET INTT1 MASK
        MVI     V,0FFH          ;SET V'
        EXA
        MVI     V,0FFH          ;SEY V
        EI                      ;ENABLE INTERRUPT
```

(2) Display/keying-in operations

The display/keying-in operations are executed in the timer interrupt routine.

If a borrow occurs when the DSPCNT value is slightly decreased (decrement), keying-in operations are carried out. If no borrow occurs, display operations are carried out.

For the keying-in operations, all of segment signal outputs (PA0 to PA7) are set to 0FF. DSPCNT is set to 8.

For the display operations, the DSPCNT value indicates the display digit.

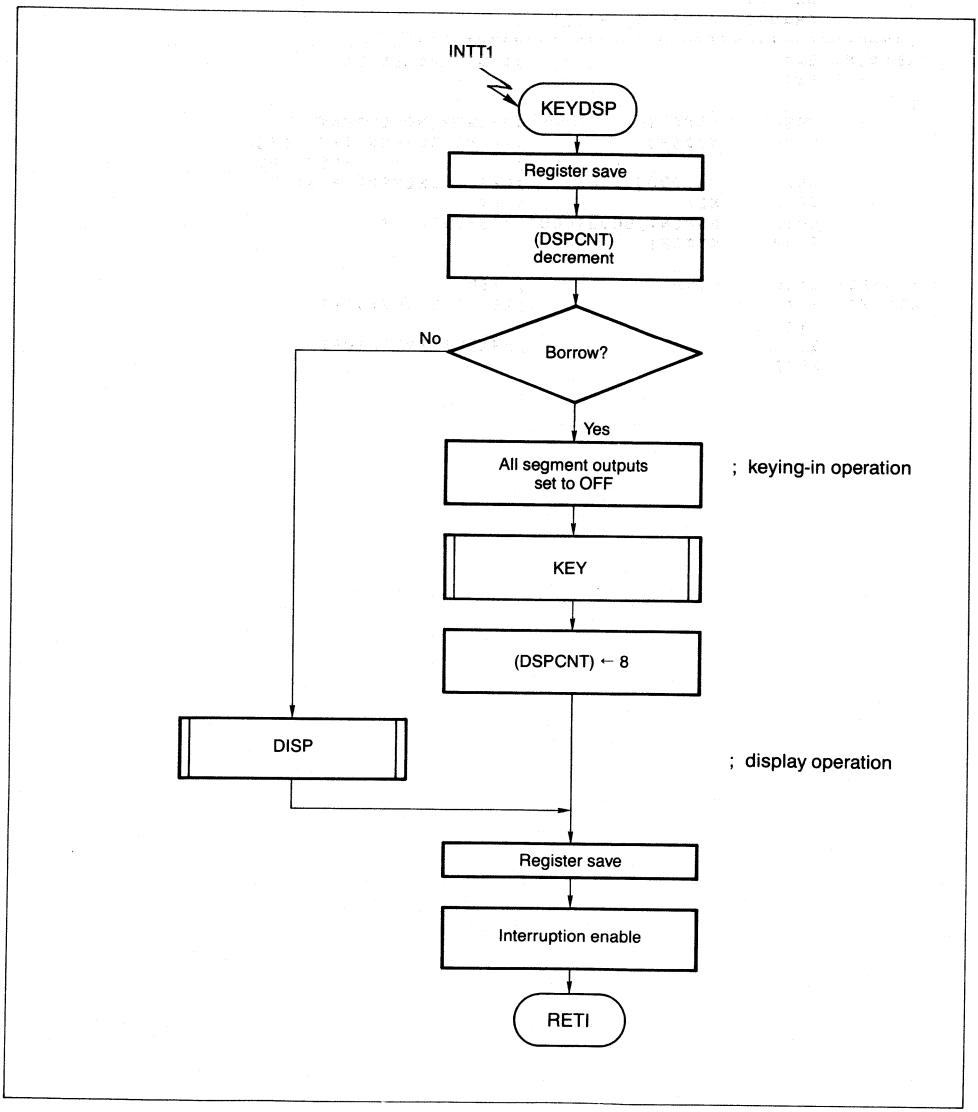
- Display

  Display operations are carried out according to the DSPCNT value.

  Data of the digit indicated by the DSPCNT value (0 to FH numerals) are fetched from the DSPDAT area. The fetched data are used to generate 0 to F display data as segment output signals.

  Zero-suppress and decimal point ON operations are carried out by DPCINT and the suppress flag.

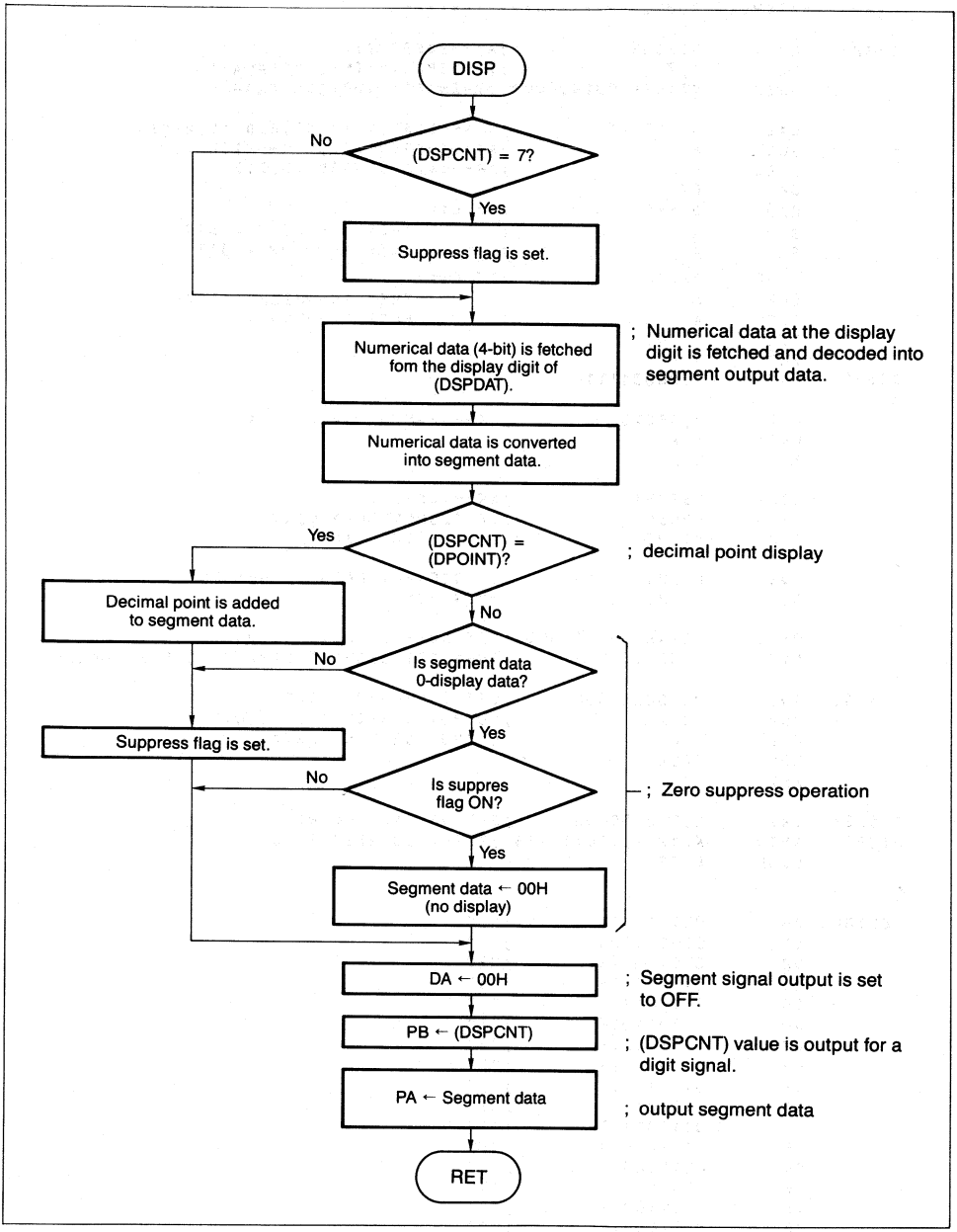
- Keying-in
  Same as with the keying-in operations described in section 1.1.2.
- **Flowchart** (display/keying-in)



- **Program list** (display/keying-in)

```
;*******KEY/DISPLAY ROUTINE***************
;       (INTTO ROUTINE)
;
;       INPUT:  (DSPCNT),(KEYWRK),(KEYCOD),
;               (DSPDAT),(DPOINT)
;       OUTPUT: -
;       CHANGE: A',B',C',H',L'
;=========================================
KEYDSP: EXX                         ;STORE REGISTER
        EXA
;
        DCRW    DSPCNT              ;DECREMENT DSPCNT
        GJMP    KYDSP0              ;IF NO BORROW THEN DISP
                                    ;               ELSE KEY
        ANI     PA,00000000B        ;OFF!! SEGMENT PORT
        CALL    KEY                 ;KEY
        ANIW    DSPCNT,00001000B    ;DSPCNT<--8
        GJMP    KYDSP1
;
KYDSP0: CALL    DISP                ;DISP
KYDSP1: EXA                         ;RESTORE REGISTER
        EXX
        EI                          ;ENABLE INTERRUPT
        RETI
```

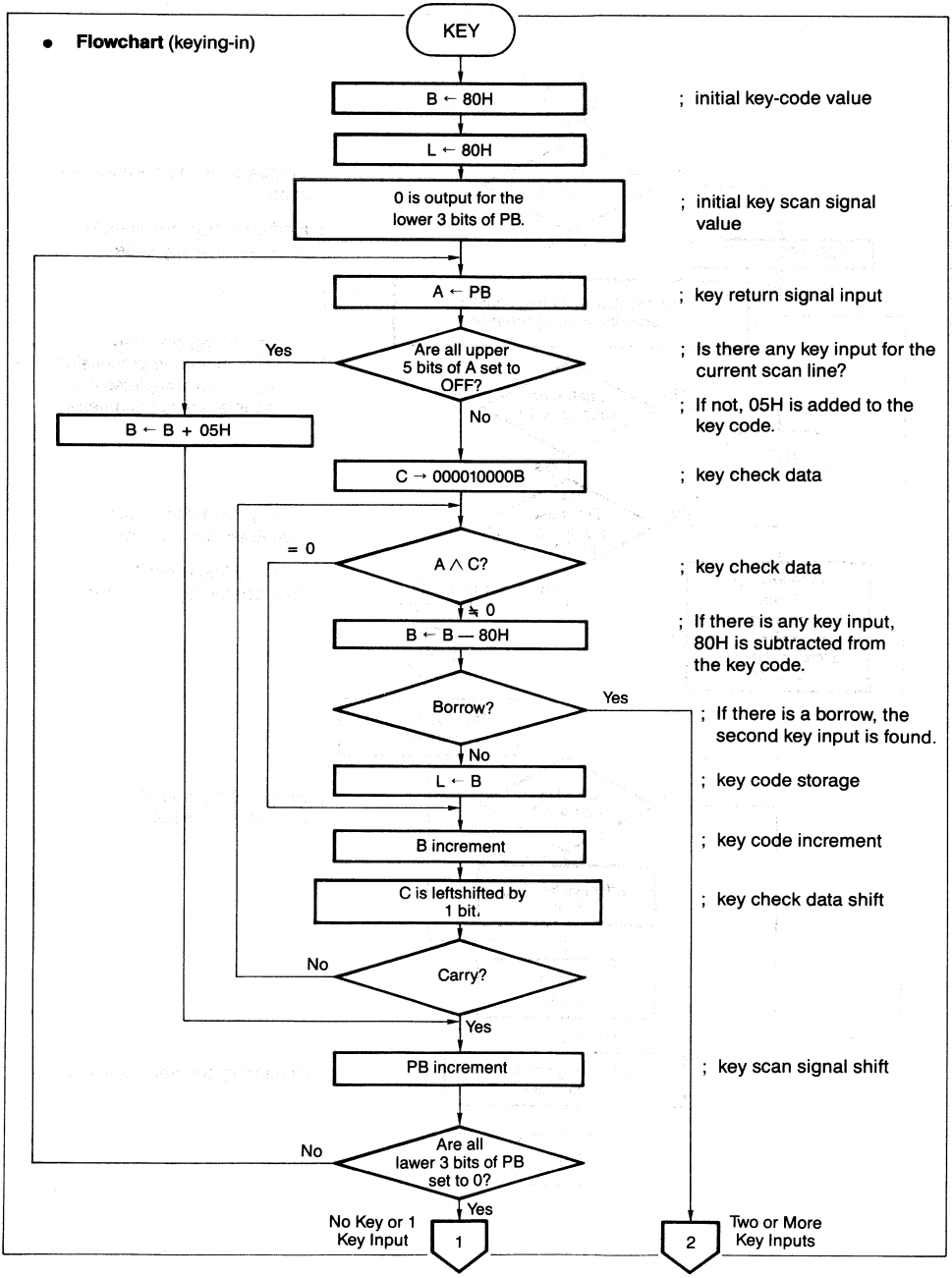
- **Flowchart** (display)

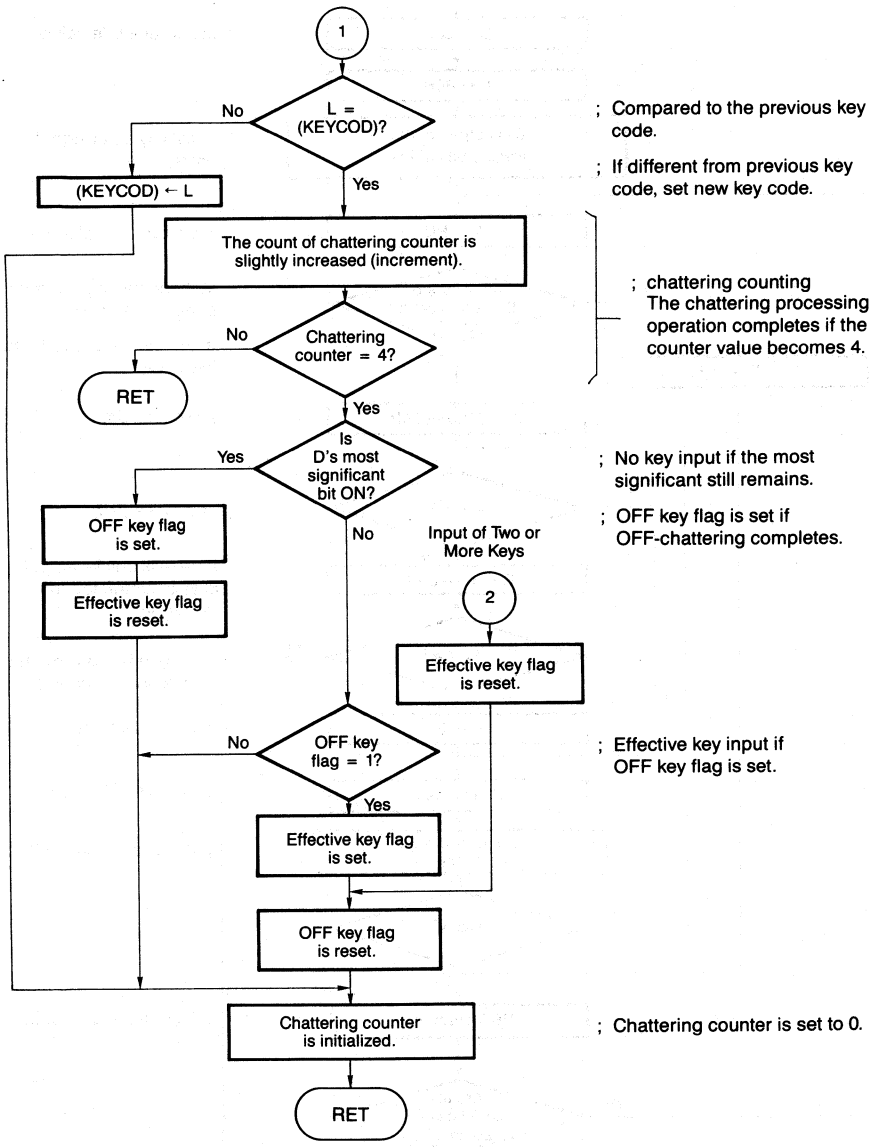


- **Program list** (display)

```
;*******DISPLAY ROUTINE*****************
;
DISP:   LDAW    DSPCNT          ;A<--(DSPCNT)
        NEI     A,7             ;IF DSPCNT=7(MSD DISPLAY)
        ORIW    KEYWRK,00100000B ;THEN SET SUPRESS FLAG
;
        LXI     H,DSPDAT        ;HL<--DISP DATA STORED ADDRESS
        SLR     A               ;A<--(DSPCNT)/2,CY<--HIGH/LOW
        LDAX    H+A             ;A<--DATA 2 WORD (3BIT)
        SK      CY
        GJMP    DISP0           ;IF CY=1
        SLR     A               ;       THEN A<--HIGH 4 BIT
        SLR     A               ;       ELSE A<--LOW 4 BIT
        GJMP    DISP0           ;IF CY=1
        SLR     A               ;       THEN A<--HIGH 4 BIT
        SLR     A               ;       ELSE A<--LOW 4 BIT
        SLR     A
        SLR     A
DISP0:  ANI     A,00001111B
;
        LXI     H,SEGTBL        ;L<--SEGMENT DATA BY A
        LDAX    H+A
        MOV     L,A
;
        LDAW    DSPCNT          ;A<--(DSPCNT)
        NEAW    DPOINT          ;IF (DSPCNT)=(DPOINT)
        GJMP    DISP2           ;       THEN JUMP
;
        EQI     L,01111111B     ;IF SEGMENT DATA IS NOT "0"
        GJMP    DISP3           ;THEN JUMP
;
        OFFIW   KEYWRK,00100000B ;IF SUPRESS FLAG IS ON
        MVI     L,00000000B     ;THEN SET SEGMENT DATA TO OFF(0 SUPRESS)
;
DISP1:  ANI     PA,00000000B    ;OFF!! SEGMENT PORT
        MOV     PB,A            ;OUTPUT (DSPCNT) TO DIGIT
        MOV     A,L             ;OUTPUT SEGMENT DATA
        MOV     PA,A
        RET
;
DISP2:  ORI     L,10000000B     ;SET DECIMAL POINT
DISP3:  ANIW    KEYWRK,11011111B ;RESET SUPRESS FLAG
        GJMP    DISP1
;
;
SEGTBL: DB      00111111B       ;"0"
        DB      00000110B       ;"1"
        DB      01011011B       ;"2"
        DB      01001111B       ;"3"
        DB      01100110B       ;"4"
        DB      01101101B       ;"5"
        DB      01111101B       ;"6"
        DB      00100111B       ;"7"
        DB      01111111B       ;"8"
        DB      01100111B       ;"9"
        DB      01110111B       ;"A"
        DB      01111100B       ;"B"
        DB      00111001B       ;"C"
        DB      01011110B       ;"D"
        DB      01111001B       ;"E"
        DB      01110001B       ;"F"
```

- **Flowchart** (keying-in)




1
No
L = (KEYCOD)?
Yes
(KEYCOD) ← L
The count of chattering counter is slightly increased (increment).
No
Chattering counter = 4?
RET
Yes
Yes
Is D's most significant bit ON?
OFF key flag is set.
Effective key flag is reset.
No
Input of Two or More Keys
2
Effective key flag is reset.
No
OFF key flag = 1?
Yes
Effective key flag is set.
OFF key flag is reset.
Chattering counter is initialized.
RET
; Compared to the previous key code.
; If different from previous key code, set new key code.
; chattering counting
The chattering processing operation completes if the counter value becomes 4.
; No key input if the most significant still remains.
; OFF key flag is set if OFF-chattering completes.
; Effective key input if OFF key flag is set.
; Chattering counter is set to 0.

- **Program list** (keying-in)

```
;********KEY INPUT ROUTINE********
;
KEY:    MVI     B,80H           ;INITIALIZE KEY CODE
        MVI     L,80H           ;INITIALIZE JEY CODE STORE
        ANI     PB,11111000B    ;INITIALIZE PORT B LATCH
;
KEY0:   MOV     A,PB            ;INPUT KEY RETURN
        OFFI    A,11111000B     ;IF KEY INPUT THEN JUMP
        GJMP    KEY1            ;          ELSE NO KEY
;
        ADI     B,5             ;KEY CODE<--KEY CODE+5
        GJMP    KEY4            ;JUMP
;
KEY1:   MVI     C,00001000B     ;INITIALIZE KEY CHECK
KEY2:   ONA     A,C             ;CHECK! KEY IS ON?
        GJMP    KEY3            ;IF OFF THEN JUMP
;
        SUINB   B,80H           ;CHECK INPUT KEY NUMBER
        GJMP    KEY7            ;IF NUMBER OVER 2 THEN JUMP
;
        MOV     A,B             ;STORE INPUT KEY CODE TO L
        MOV     L,A
        MOV     A,PA
;
KEY3:   INR     B               ;INCREMENT KEY CODE
        SLLC    C               ;1-BIT LEFT SHIFT KEY CHECK
        GJMP    KEY2            ;IF NO CARRY THEN JUMP
;
KEY4:   ADI     PB,1            ;KEY SCAN INCREMENT
        OFFI    PB,00001111B    ;IF FINISH ALL KEY CHECK
        GJMP    KEY0            ;ELSE JUMP
;
        MOV     A,L             ;COMPARE INPUT KEY CODE AND (KEYCOD)
        EQAW    KEYCOD
        GJMP    KEY8            ;IF DISAGREE THEN JUMP
;
        INRW    KEYWRK          ;INCREMENT CHATTERING COUNTER
        ONIW    KEYWRK,00000100B ;IF NOT COUNT OVER
        RET                     ;THEN RET
;
        OFFI    A,10000000B     ;IF NO KEY
        GJMP    KEY9            ;THEN JUMP
;
        ONIW    KEYWRK,01000000B ;IF OFF KEY FLAG IS OFF
        GJMP    KEY6            ;THEN JUMP
;
        ORIW    KEYWRK,10000000B ;SET EFFECT KEY FLAG
KEY5:   ANIW    KEYWRK,10111111B ;RESET OFF KEY FLAG
KEY6:   ANIW    KEYWRK,11111000B ;INITIALIZE CHATTERING COUNTER
        RET
;
KEY7:   ANIW    KEYWRK,01111111B ;RESET EFFECT KEY FLAG
        GJMP    KEY5
;
KEY8:   STAW    KEYCOD          ;STORE INPUT KEY CODE
        GJMP    KEY6
;
KEY9:   ORIW    KEYWRK,01000000B ;SET OFF KEY FLAG
        ANIW    KEYWRK,01111111B ;RESET EFFECT KEY FLAG
        GJMP    KEY6
```

Chapter 2 Interfaces with Display Control Elements

This chapter describes two interface examples with the LCD display element; the μPD7228, and the μPD72030.

2.1 μPD7228 Interface

The μPD7228 is an LCD controller/driver having interface functions with a dot-matrix format 8/16 time-sharing LCD and a microprocessor. The μPD7228 is configured with a single chip and can drive a dot-matrix LCD with 50 columns in 8 time-sharings or a dot-matrix LCD with 42 columns in 16 time-sharings.

If the μPD7228 is configured with multi-chips, it can easily drive a large-pattern LCD. The μPD7228 has an internal 5 x 7 dot-matrix character generator for ASCII/JIS specifications and can easily display alphamerics or Japanese katakana. An 8-bit serial interface, or a 4-bit parallel interface with a microprocessor, may be selected.

2.1.1 μPD7228 Connection Method

Figure 2-1 shows an LCD display system configuration example using the μPD7228. The system is connected to the master CPU with a serial interface (in the I/O interface mode) in synchronization with the SCK signal which has been synchronized with, and controlled for, the SCK clock. A total of four chips can be connected to the same serial interface using the chip address function of the μPD7228. In this case, chip selection is made with data transmitted in a serial mode.

Four 50 x 16 dot LCD panels can be displayed by using the μPD7228 in 16 time-sharing display and dividing the row signal output by two chips (R_1 to R_7 and R_8 to R_{16}).

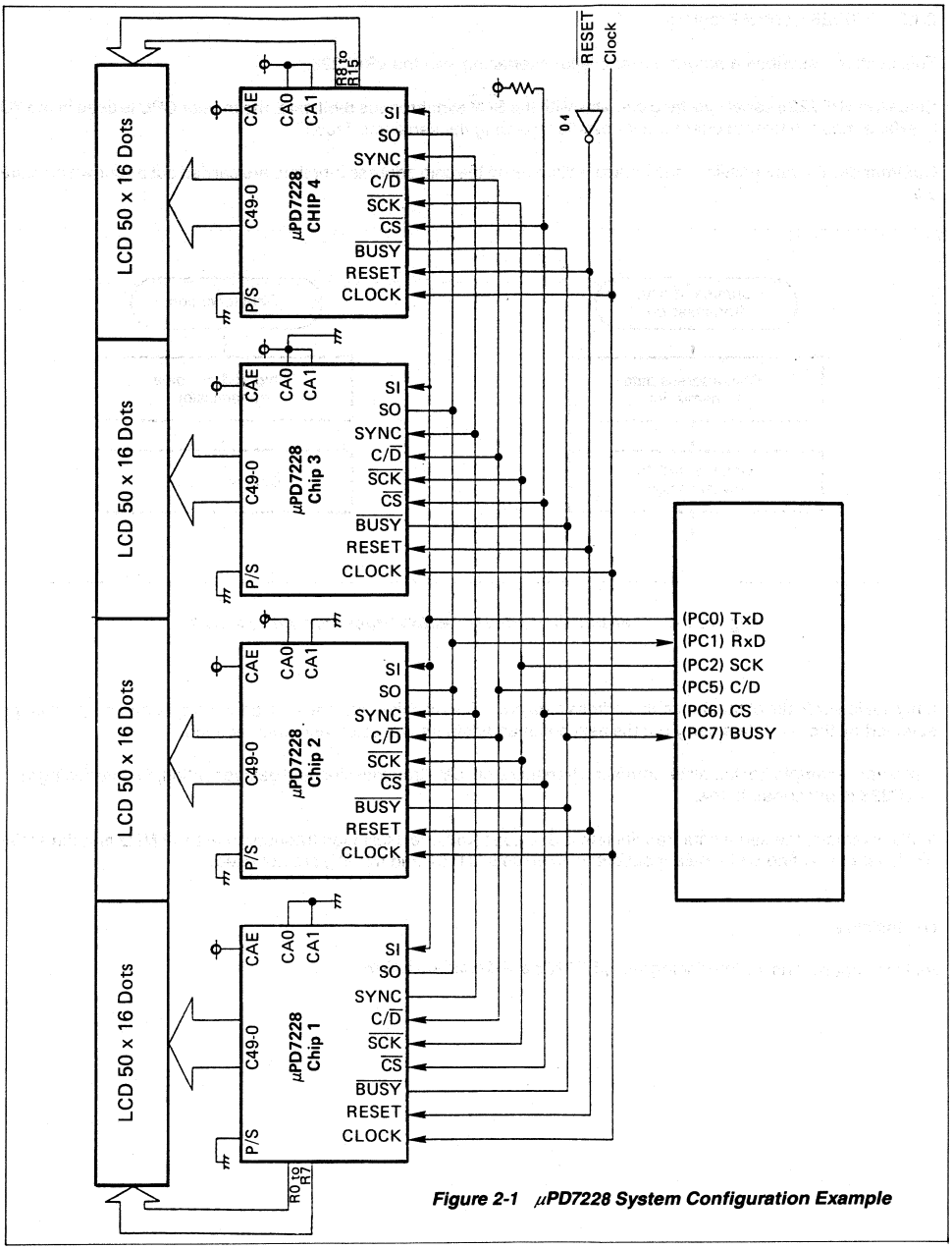


Figure 2-1 μPD7228 System Configuration Example

2.1.2 μPD7228 Control Program

This section describes a program example for interfacing with the μPD7228.

Since the μPD7228 serial interface operates with the $\overline{SCK}$ synchronous 8-bit data, the master CPU is used in the I/O interface mode. (Refer to chapter 3 for details regarding the serial interface.)

Command/data transmission, and data reception using the chip address function, are carried out as shown in figure 2-2.

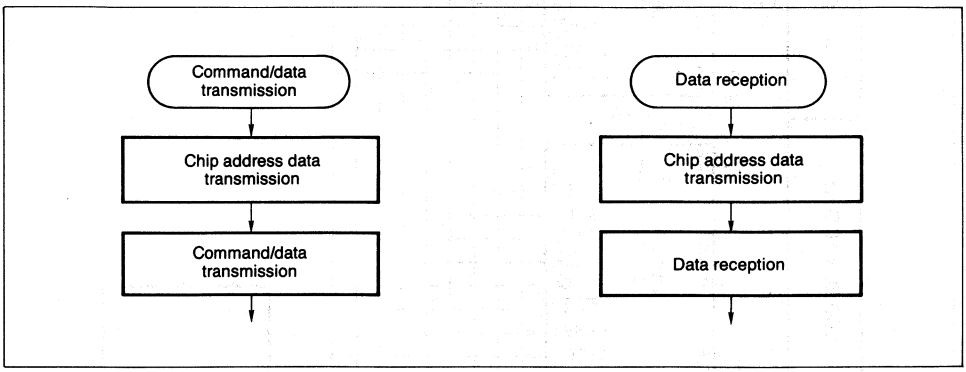


Figure 2-2 Transmission and Reception Procedure with μPD7228

Only the lower 2 bits of the 8-bit chip address data are valid. All chips read the chip address data and only the chips selected by the read data carry out the next command/data transmission and data reception.

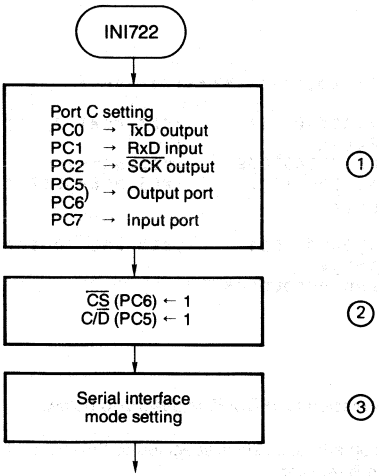
A program example (for initialize, command transmission, data transmission and data reception) for interfacing with μPD7228 is described below.

In this example, the serial data transfer rate is 500 kbps (with the oscillation frequency set to 12 MHz and the serial clock set to ϕ_{24}) and serial data input/output interrupts (INTST and INTSR) are not used.

(1) Initialize

Initialize operations for interfacing with μPD7228 are described below:

- **Flowchart** (initialize)



① Port C setting is carried out. PC0, PC1 and PC2 are set for the TxD, RxD and SCK pins, respectively. PC5 and PC6 are set for the output port and PC7 is set for the input port.

② CS (PC6) is set to a non-active mode and C/D (PC5) is set to command selection.

③ The serial interface is set. (For details of the serial interface, refer to section 3.2.1 "I/O Interface Mode Communications".)

- **Program list** (initialize)

```
;*******INITIALIZE ROUTINE***************
;
INI722: MVI     A,00000111B     ;SET PORT C  PC0=TXD,PC1=RXD,PC3=SCK
        MOV     MCC,A           ;
        ORI     PC,01100000B    ;PC5,6<--1
        MVI     A,10011111B     ;PC5,PC6=OUTPUT,PC7=INPUT
        MOV     MC,A
;
        MVI     A,00001100B     ;SET SML TO I/O INTERFACE MODE
        MOV     SML,A
        MVI     SMH,00100010B
```

(2) Command transmission

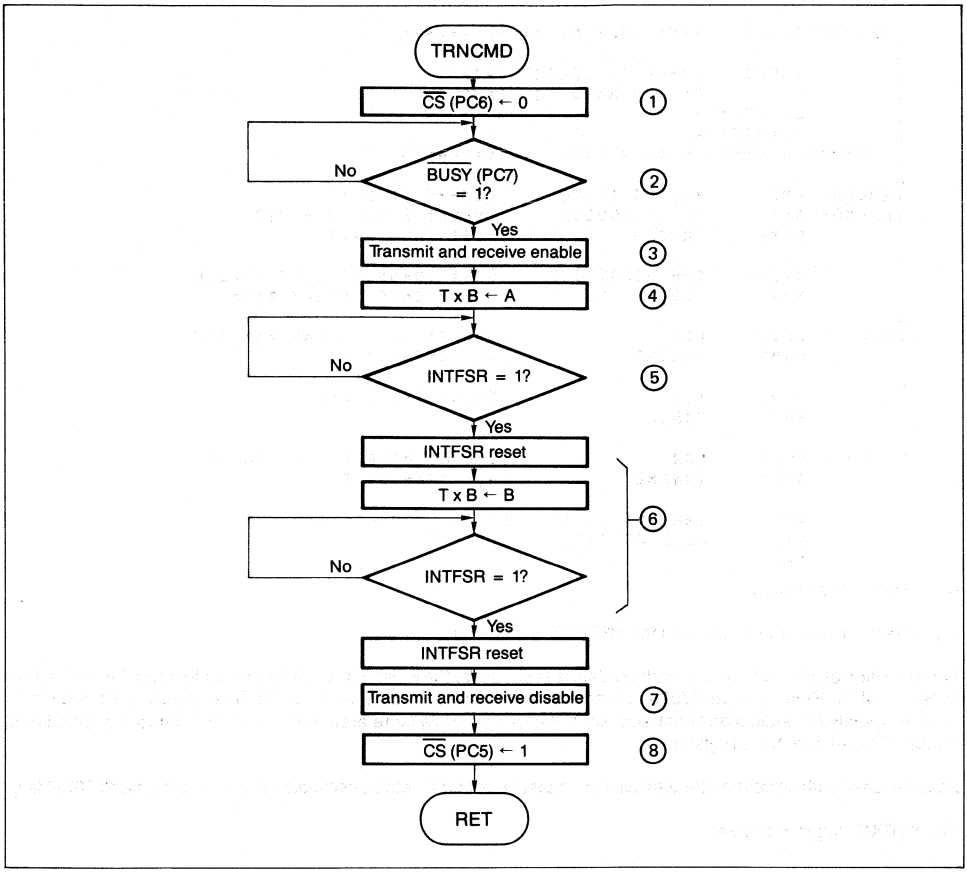
A program for command transmission to μPD7228 is described below.

For execution of this routine, chip address data is specified by the lower 2 bits of A (accumulator) and the transmit command code is stored in the B register.

Serial data transmission and reception are carried out simultaneously. Upon completion of data reception (with the INTFSR flag set), completion of data transmission is checked. Receive data is dummy data.

The INTFST flag remains set.

- **Flowchart** (command transmission)



① $\overline{CS}$ (PC6) is set in an active state.

② $\overline{BUSY}$ (PC7) of the μPD7228 is checked.

③ The serial interface is disabled for transmission and reception.

④ A data (chip address data) is transferred to the transmit buffer register (TxB). Because both transmission and reception are enabled, they start at the same time.

⑤ INTFSR is checked for the completion of a 1 byte transmission (completion of reception).

⑥ B register data (command code) is transmitted.

⑦ The serial interface is disabled for transmission and reception.

⑧ $\overline{CS}$ is set in a non-active state.

- **Program list** (command transmission)

```
;******COMMAND TRANS ROUTINE************
;
;       INPUT:  A<--CHIP SELECT DATA
;               B<--COMMAND TO TRANS
;       OUTPUT: -
;       CHANGE: A
;=======================================
;
TRNCMD: ANI     PC,10111111B    ;CS<--0 (C/D=1)
TRNCM0: ONI     PC,10000000B    ;CHECK! 7228 IS BUSY?
        GJMP    TRNCM0          ;YES! THEN WAIT
;
        ORI     SMH,00001100B   ;DATA TRANS,RECEIVE ENABLE
        MOV     TXB,A           ;TXB<--CHIP SELECT DATA
;
TRNCM1: SKIT    FSR             ;CHECK! DATA TRANS FINISH?
        GJMP    TRNCM1          ;NO! THEN WAIT
;
        MOV     A,B             ;TXB<--TRANS DATA
        MOV     TXB,A
;
TRNCM2: SKIT    FSR             ;CHECK! DATA TRANS FINISH?
        GJMP    TRNCM2          ;NO! THEN WAIT
;
        ANI     SMH,11110011B   ;DATA TRANS DISABLE
        ORI     PC,01000000B    ;CS<--1
        RET
```

(3) Data transmission

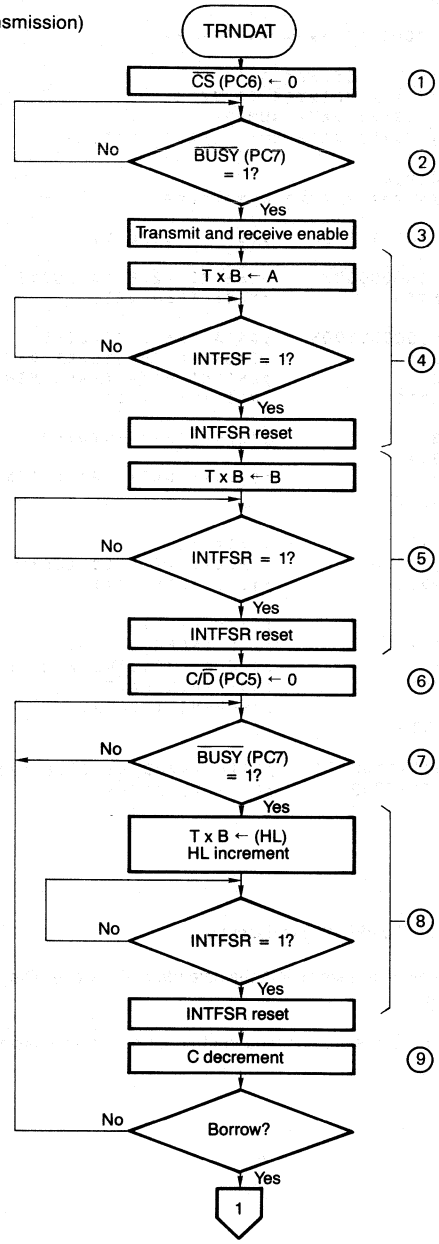
A program for data transmission to the μPD7228 is described.

For execution of this routine, chip address data is specified by the lower 2 bits of A (accumulator) and the codes (80H to B1H, C0H to F1H) of the μPD7228 data pointer load instruction are stored into the B register. The HL register is used to specify the least significant address of the transmit data store area. A value of "the number of data items minus 1" is set into the C register.

As is the case with command transmission, completion of serial data transmission is checked by the INTFSR flag.

The INTFST flag remains set.

- **Flowchart** (data transmission)



- **Program list** (data transmission)

```
;*******TRANS DATA ROUTINE***************
;
;       INPUT:  A<--CHIP SELECT DATA
;               B<--DATA POINTER TO 7228
;               C<--TRANS DATA NUMBER
;               HL<-TRANS DATA BUFF LEAST ADDRESS
;       OUTPUT: -
;       CHANGE: A,C,H,L
;=======================================
;
TRNDAT: ANI     PC,10111111B    ;CS<--0 (C/D=1)
;
TRNDT0: ONI     PC,10000000B    ;CHECK! 7228 IS BUSY?
        GJMP    TRNDT0          ;YES! THEN WAIT
;
        ORI     SMH,00001100B   ;DATA TRANS,RECEIVE ENABLE
        MOV     TXB,A           ;TXB<--CHIP SELECT DATA
TRNDT1: SKIT    FSR             ;CHECK! DATA TRANS FINISH?
        GJMP    TRNDT1          ;NO! THEN WAIT
;
        MOV     A,B             ;TXB<--DATA POINTER
        MOV     TXB,A
TRNDT2: SKIT    FSR             ;CHECK! DATA TRANS FINISH?
        GJMP    TRNDT2
;
        ANI     PC,11011111B    ;C/D<--0
TRNDT3: ONI     PC,10000000B    ;CHECK! 7228 IS BUSY?
        GJMP    TRNDT3          ;YES! THEN WAIT
;
        LDAX    H+              ;TXB<--(HL)
        MOV     TXB,A           ;AND INCREMENT HL
TRNDT4: SKIT    FSR             ;CHECK! RECEIVE FINISH?
        GJMP    TRNDT4          ;NO! THEN WAIT
        DCR     C               ;DECREMENT COUNTER
        GJMP    TRNDT3
;
        ANI     SMH,11110011B   ;DATA TRANS,RECEIVE DISABLE
        ORI     PC,01100000B    ;CS<--1,C/D<--1
        RET
```

(4) Data transmission

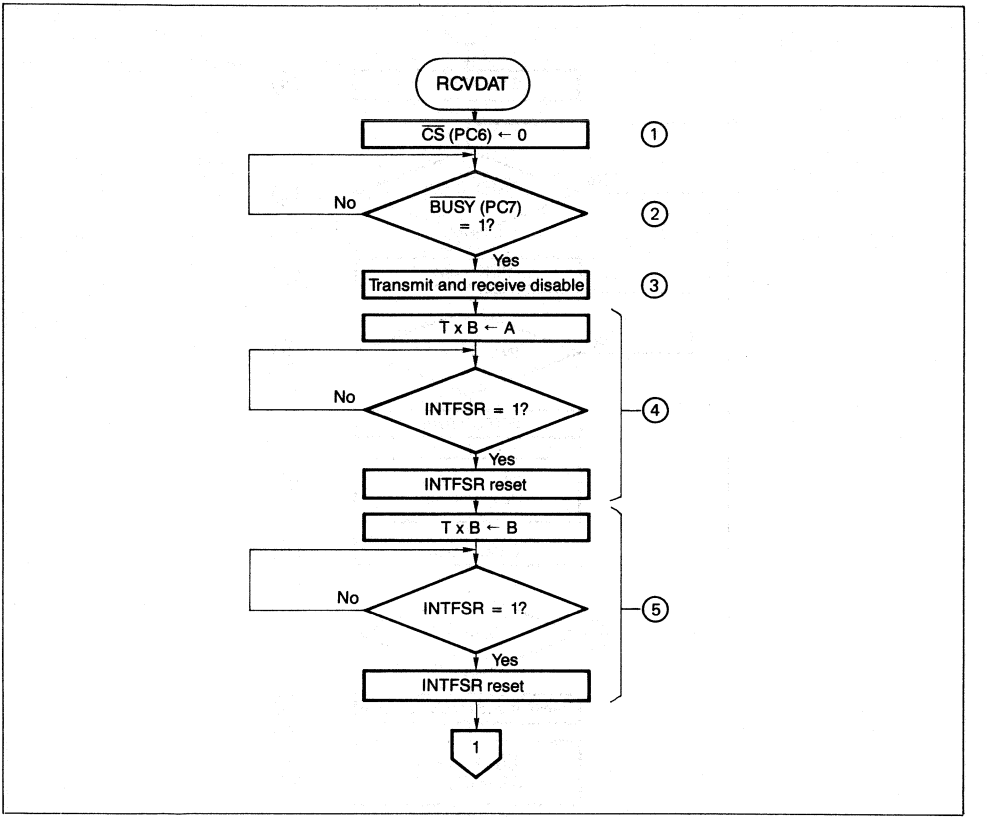
A program for data reception from the μPD7228 is described.

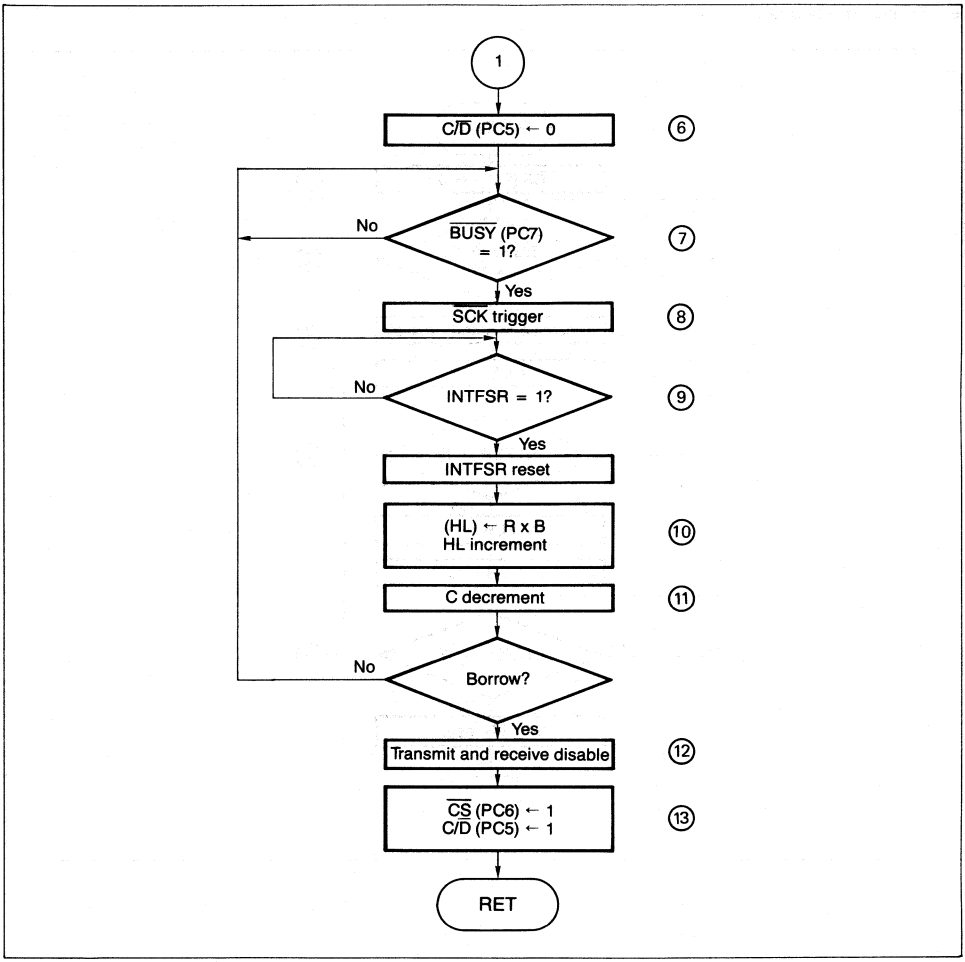
For execution of this routine, the register is set as with the data transmit routine. The received data is stored into the area indicated by the HL register.

As is the case with command transmission, completion of serial data transmission is checked by the INTFSR flag.

The INTFST flag remains set.

- **Flowchart** (data reception)





① $\overline{CS}$ (PC6) is set in an active state.

② $\overline{BUSY}$ (PC7) of the μPD7228 is checked.

③ The serial interface is disabled for transmission and reception.

④ 'A' data (chip address data) is transmitted.

⑤ 'B' register data (code of data pointer load instruction) is transmitted.

⑥ C/$\overline{D}$ (PC5) is specified for data selection.

⑦ $\overline{BUSY}$ of μPD7228 is checked.

⑧ $\overline{SCK}$ is generated for serial data reception.

⑨ Completion of data reception is checked.

⑩ The received data is stored into the area specified by the HL register and the HL register value is slightly increased (increment).

⑪ The value of C register (used as a counter) is decreased (decrement). Operations ⑦ through ⑩ are continued until a borrow occurs.

⑫ The serial interface is disabled for transmission and reception.

⑬ CS is set in a non-active state and C/$\overline{D}$ is set in a command select state.

- **Program list** (data transmission)

```
;********RECEIVE DATA ROUTINE*************
;
;       INPUT:  A<--CHIP SELECT DATA
;               B<--DATA POINTER TO 7228
;               C<--RECEIVE DATA NUMBER
;               HL<-DATA STORE BUFF LEAST ADDRESS
;       OUTPUT: -
;       CHANGE: A,C,H,L
;=======================================
;
RCVDAT: ANI     PC,10111111B    ;CS<--0 (C/D=1)
;
RCVDT0: ONI     PC,10000000B    ;CHECK! 7228 IS BUSY?
        GJMP    RCVDT0          ;YES! THEN WAIT
;
        ORI     SMH,00001100B   ;DATA TRANS,RECEIVE ENABLE
        MOV     TXB,A           ;TXB<--CHIP SELECT DATA
RCVDT1: SKIT    FSR             ;CHECK! DATA TRANS FINISH?
        GJMP    RCVDT1          ;NO! THEN WAIT
;
        MOV     A,B             ;TXB<--DATA POINTER
        MOV     TXB,A
RCVDT2: SKIT    FSR             ;CHECK! DATA TRANS FINISH?
        GJMP    RCVDT2
;
        ANI     PC,11011111B    ;C/D<--0
RCVDT3: ONI     PC,10000000B    ;CHECK! 7228 IS BUSY?
        GJMP    RCVDT3          ;YES! THEN WAIT
;
        ORI     SMH,01000000B   ;TRRIGER DATA RECEIVE
RCVDT4: SKIT    FSR             ;CHECK! RECEIVE FINISH?
        GJMP    RCVDT4          ;NO! THEN WAIT
;
        MOV     A,RXB           ;(HL)<--RECEIVE DATA
        STAX    H+              ;AND INCREMENT HL
        DCR     C               ;DECREMENT COUNTER
        GJMP    RCVDT3
        ANI     SMH,11110011B   ;DATA TRANS,RECEIVE DISABLE
        ORI     PC,01100000B    ;CS<--1,C/D<--1
        RET
```

2.2 Interfacing with the μPD72030

The μPD72030 is an LSI for display control of high-duty dot matrix LCD. A command from an 8-bit parallel bus interface enables characters and graphics to be displayed on the dot matrix LCD.

The μPD72030 can be directly connected to high withstand the voltage LCD driver μPD6307 or μPD6308. Display data can be transferred to the LCD driver at high speeds in 8-bit parallel.

A character generator with a maximum of 16 x 16 dot font can be externally mounted to the μPD72030 for kanji (Chinese character) display. The μPD72030 has an on-chip character generator with 64 alphanumerics or symbols (5 x 7 dots) in ASCII format.

2.2.1 μPD72030 Connection Method

Figure 2-3 shows a μPD72030 connection example. The μPD72030 is connected to the master CPU in 8-bit parallel.

Since μCOM87 series microcomputers have no I/O space, a memory mapped I/O is set for connection with the μPD72030. In the example in figure 2-3, the μPD72030 is mapped for addresses 2000H onwards. Thus, the external memory extension mode must be 16 Kbytes or more.

Since the μPD72030 generates the command acknowledge state from the $\overline{INT}$ pin, the $\overline{INT}$ pin is connected to the $\overline{INT2}$ external interrupt input pin. As a result, the μPD72030 command acknowledge is jedged by the $\overline{INT2}$ external interrupt.

Although DMA controller (μPD8257) can also be connected to the μPD72030, the controller is not used for this system (with $\overline{DACK}$ and $\overline{TC}$ fixed at the high level and DREQ opened).

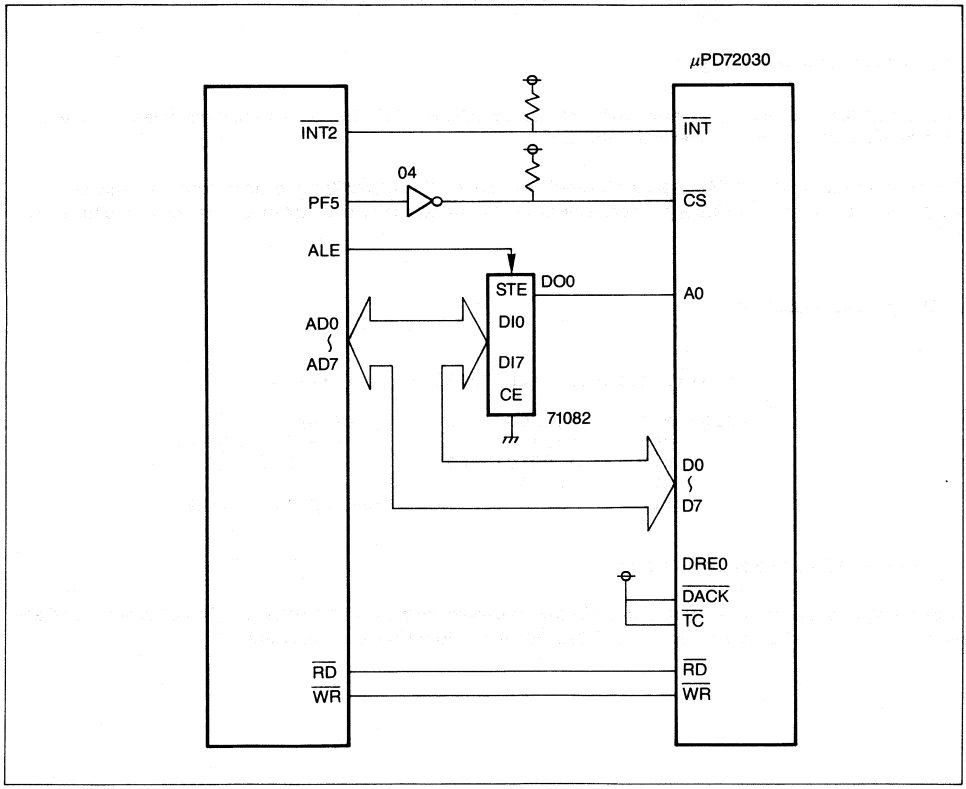


Figure 2-3 μPD72030 Connection Example

Table 2-1 shows the μPD7230 data bus operations.

Table 2-1 μPD72030 Data Bus Operations (1/2)

CS	DACK	A0	RD	WR	D0 to D7 Pin Status	Host CPU I/F Operations
1	1	x	x	x	Z	No operation
x	x	x	1	1	Z	No operation

Statuses are 8-bit registers to indicate the μPD72030 operations. When OBR (bit 0) of the status is "1", data can be read from the μPD72030. When IBR (bit 1) is "1", parameters and data can be written into the μPD72030.

2.2.2 μPD72030 Control Program

This section describes a program example for interfacing with the μPD72030 to carry out three types of operations, command/parameter transmission to the μPD72030.

Memory mapping register (MM) is set for initialization to connect the μPD72030, and mode control C register (MCC) is set to operate PC3 as $\overline{INT2}$ input pin. For initialization, the masked $\overline{INT2}$ pin is interrupt enabled by the EI instruction.

- **Program list** (initialize)

```
;*******INITIALIZE FOR 72030************
;
I720IN: MVI     A,00001110B     ;SET MM
        MOV     MM,A            ;16K BYTE,RAE<--1
        MVI     A,00001000B     ;SET PC3 TO INT2
        MOV     MCC,A
        EI                      ;INERRUPT ENABLE
```

(1) Command/parameter transmission

Command/parameter transmission to the μPD72030 is carried out with $\overline{INT2}$ interruption. Thus, the command and parameter values to be transmitted to the μPD72030 are prestored in the memory areas.

Figure 3-4 shows the memory configuration.

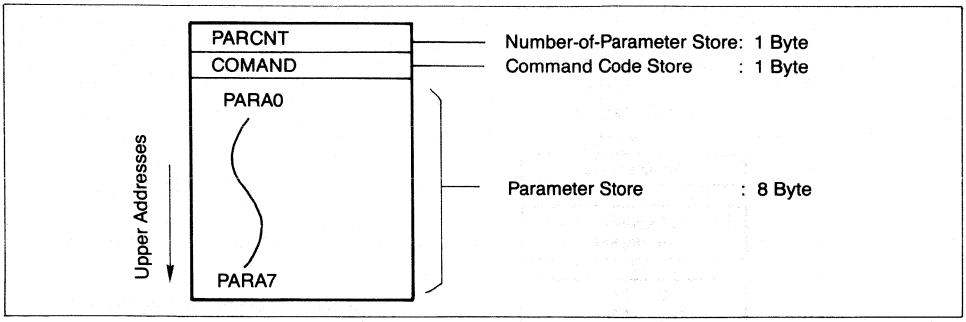
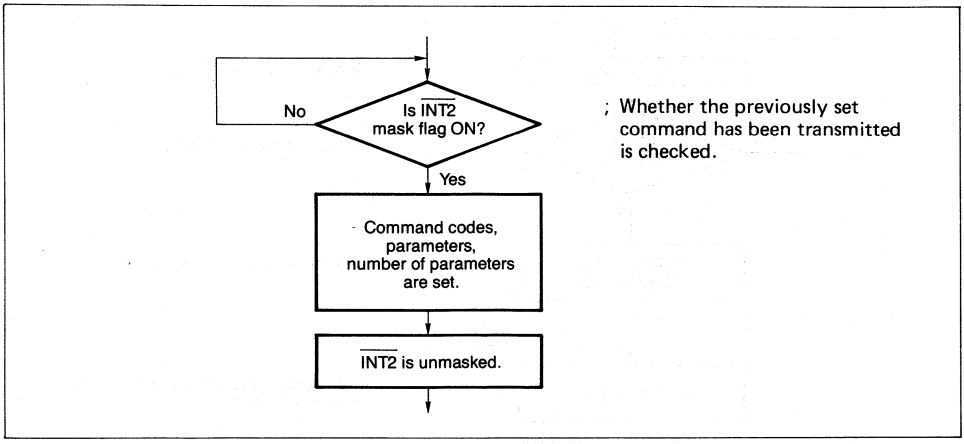


Figure 3-4 Memory Configuration

The commands to be transmitted to the μPD72030 have different parameters (0 to 8-byte) depending on the their types. Thus, the number of parameters for each command is set into PARCNT.

The following procedure is used to set the commands, parameters, and the number of parameters to the memory:

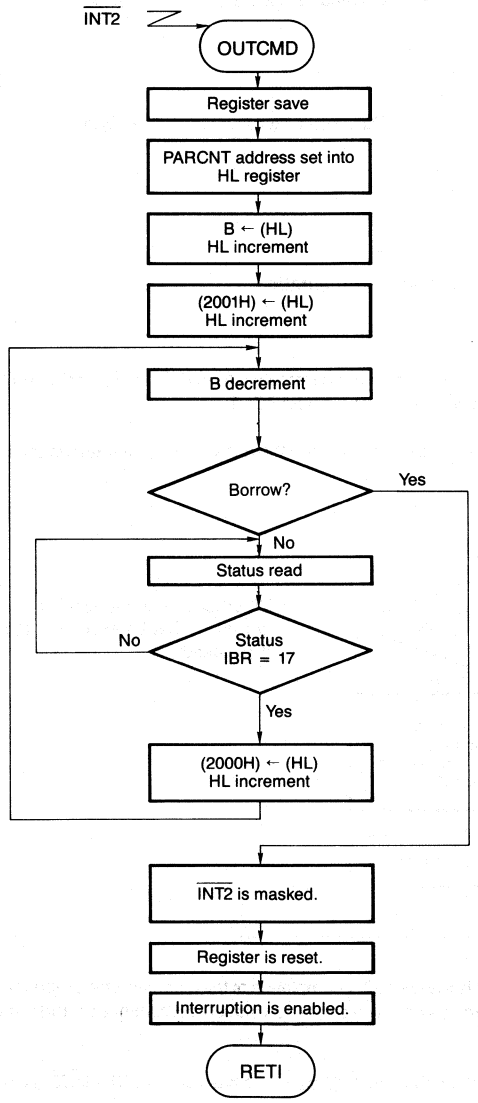


In the $\overline{INT2}$ interrupt processing, the $\overline{INT2}$ interrupt is masked after completion of the command/parameter transmission to the μPD72030. If the $\overline{INT2}$ interrupt mask flag has not been set, the previously set commands and parameters have not yet been transmitted.

Commands and parameters must be set after checking that the $\overline{INT2}$ mask flag has been set. The $\overline{INT2}$ is unmasked after the commands and parameters have been set.

The command/parameter transmit ($\overline{INT2}$ interrupt processing) flowchart is shown below:

- **Flowchart** (command/parameter transmission)



The commands and parameters are transmitted using the PARCNT value (number of parameters) as a counter. For parameter transmission, the IBR bit of the μPD72030 status is checked.

Upon completion of the command/parameter transmission, the $\overline{INT2}$ interrupt mask flag is set (to prevent interrupt processing from being executed before a new command/parameter is set).

- **Program list** (command/parameter transmission)

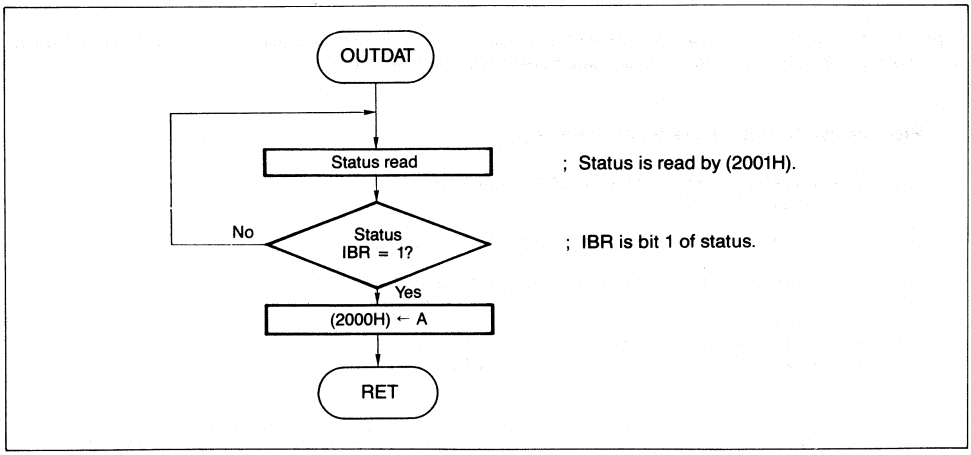
```
;*******OUTPUT COMMAND/PARAMETER ROUTINE*
;       (INT2 ROUTINE)
;
;       OUTPUT COMAND AND PARAMETER TO 72030
;
;       INPUT:  (PARCNT),(COMAND),(PARA0)-(PARA7)
;       OUTPUT: -
;       CHANGE: A',B',D',E',H',L'
;=======================================
OUTCM:  EXA                         ;STORE REGISTER
        EXX
;
        LXI     H,PARCNT            ;SET COMMAND/PARAMETER AREA ADDRESS
        LXI     D,2001H             ;SET ADDRESS COMMAND/STATUS TO 72030
        LDAX    H+                  ;SET COUNTER.
        MOV     B,A
;
        LDAX    H+                  ;OUTPUT COMMAND CODE
        STAX    D
;
OUTCM0: DCR     B                   ;DECREMENT COUNTER
        GJMP    OUTCM1
;
        ORI     MKL,00010000B       ;SET INT2 MASK
        EXX                         ;RESTORE REGISTER
        EXA
        EI
        RETI                        ;RETURN TO MAIN ROUTINE
;
OUTCM1: LDAX    D                   ;CHECK! STATUS
        ONI     A,00000010B         ;IBR=1?
        GJMP    OUTCM1              ;NO! THEN WAIT
;
        DCX     D                   ;ADJUST ADDRESS TO PARAMETER OUTPUT
        LDAX    H+                  ;OUTPUT PARAMETER
        STAX    D+                  ;AND SET ADDRESS TO STATUS INPUT
        GJMP    OUTCM0
```

(2) Data transmission

This is a subroutine for 1-byte data transmission to the μPD72030. The subroutine is executed by setting transmit data into 'A' (accumulator).

This routine is used to transmit data after the WRITE command has been transmitted.

- **Flowchart** (data transmission)



- **Program list** (data transmission)

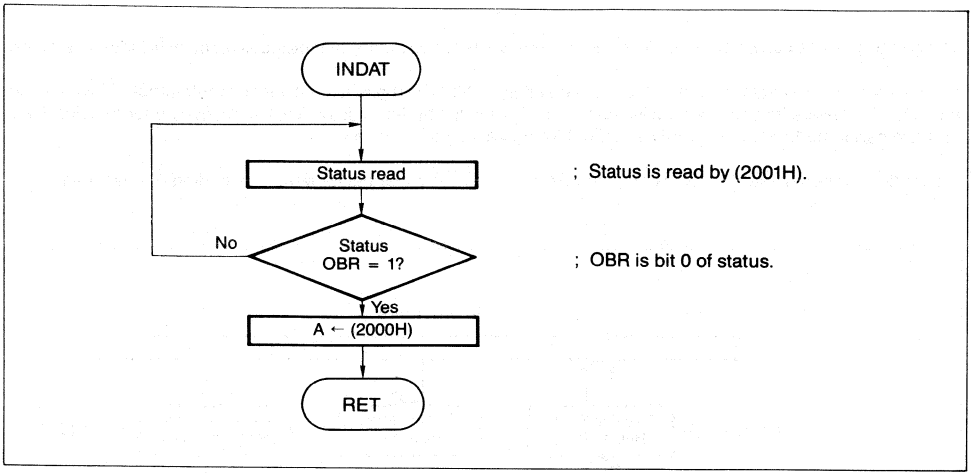
```
;*******OUTPUT 1BYTE DATA ROUTINE********
;
;       OUTPUT 1BYTE DATA TO 72030
;
;       INPUT:  A<--OUTPUT DATA
;       OUTPUT: -
;       CHANGE: C
;=======================================
OUTDAT: MOV     C,2001H         ;READ STATUS
        ONI     C,00000010B     ;CHECK! STATUS,IBR=1?
        GJMP    OUTDAT          ;NO! THEN WAIT
;
        MOV     2000H,A         ;OUTPUT 1BYTE DATA
        RET
```

(3) Data reception

This is a subroutine for 1-byte data reception from the μPD72030. The receive data is stored into 'A' (accumulator).

This routine is to receive data after the READ command has been transmitted.

- **Flowchart** (data transmission)



- **Program list** (data reception)

```
;********INPUT 1BYTE DATA ROUTINE*********
;
;       INPUT 1 BYTE DATA FROM 72030
;
;       INPUT:  -
;       OUTPUT: A<--1BYTE DATA
;       CHANGE: A
;=======================================
INDAT:  MOV     A,2001H         ;READ STATUS
        ONI     A,00000001B     ;CHECK! STATUS,OBR=1?
        GJMP    INDAT           ;NO! THEN WAIT
;
        MOV     A,2000H         ;INPUT 1BYTE DATA
        RET
```

Chapter 3 Serial Interface

The serial interface can be operated in 3 modes, the asynchronous mode, synchronous mode or I/O interface mode.

It consists of a serial data input (RxD) pin, a serial data output (TxD) pin, a serial clock input/output ($\overline{SCK}$) pin, two 8 bit serial registers for transmit/receive operations, a transmit buffer register (TxB), a receive buffer register (RxB) and two serial mode registers (SMH and SML) for operation specification.

Figure 3-1 is a serial interface block diagram. Refer to the user's manual for details regarding the hardware.

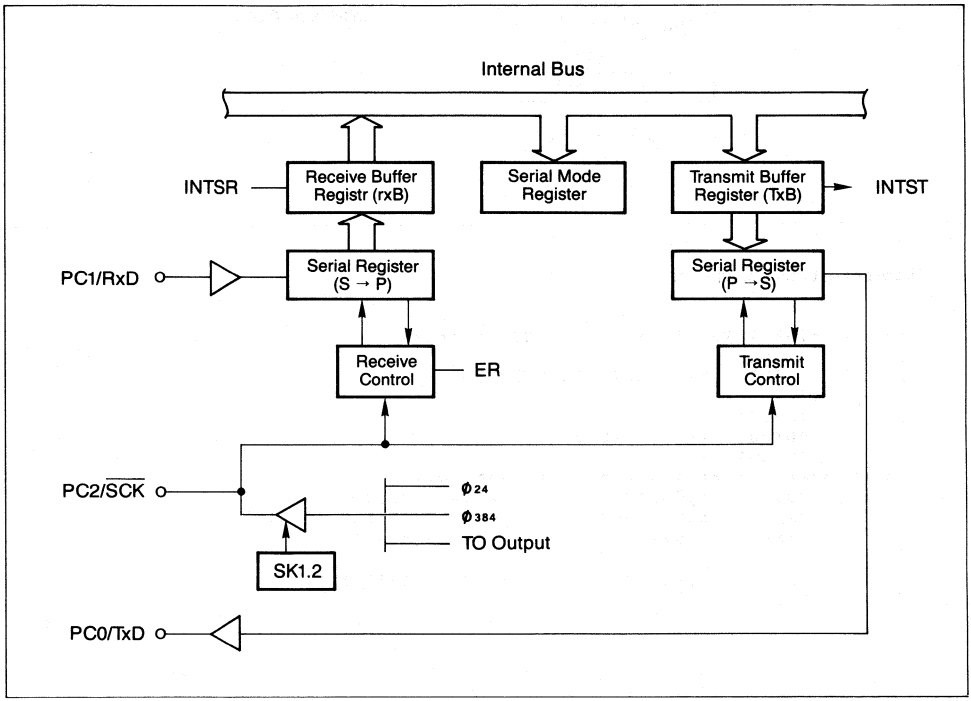


Figure 3-1 Serial Interface Block Diagram

Remarks: $\phi_{24} = f_{XTAL} \times \frac{1}{24}$

$\phi_{384} = f_{XTAL} \times \frac{1}{384}$

f_{XTAL}: Oscillation frequency (MHz)

3.1 Interfacing with the μPD71051(8251)

This section describes examples of serial data communications with the μPD71051(8251) using a serial interface.

The μPD71051(8251) is a serial control unit which can transmit data in 2 modes, the asynchronous or synchronous mode.

Examples of asynchronous and synchronous serial data communication are described in the following.

3.1.1 Asynchronous Mode Transmission

This section describes an example of serial data communication with the μPD71051(8251) in the asynchronous mode.

Figure 3-2 shows the connection with the μPD71051(8251). In addition to two serial data input/output lines (TxD and RxD), two control lines, $\overline{RTS}$ (PC7) and $\overline{CTS}$ (PC6), are used. $\overline{RTS}$ (PC7) is an output port for a data transmit enable signal to the μPD71051(8251). $\overline{CTS}$ (PC6) is an input port for a receive enable signal from the μPD71051(8251). Because $\overline{RTS}$ (PC7) is set for an input port after the resetting has been cancelled, it is set at a high level via a pull-up resistor.

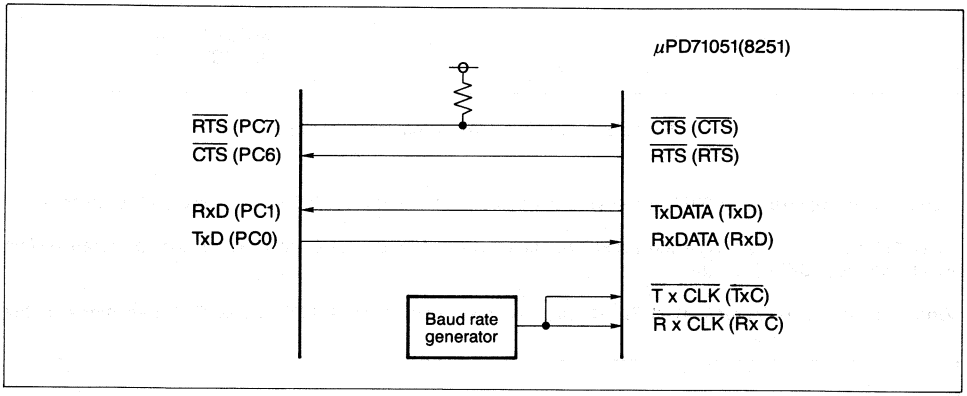


Figure 3-2 Connection Example with the μPD71051(8251) (Asynchronous Mode)

The data communication format is set as follows:

- Data transfer rate — 9600 bps
- Serial Clock — Internal clock (TO output)
- Clock rate — x 16
- Oscillation frequency — 11.0592 MHz
- Data format — Character 8 bits
  - Stop bit 2 bits
  - Even parity
- Full-duplex communications

For full-duplex data transfer operations, the interrupt (INTSR) routine is used to receive data, and the received data is stored into a buffer of the memory. Data transmission is carried out by testing the interrupt request flag (INTFST).

A buffer of 16-byte length FIFO (first in first out) is used to store the received data. The received data which has been stored into FIFO by a subroutine is read out.

(1) Buffer operations to store the received data are as follows:

The buffer is made up of a 16-byte, length data store area, and two 1-byte counters; BUFFST and BUFEND. BUFFST indicates the location where the first data written into FIFO. BUFEND indicates the next data write location.

Data read/write addresses are determined by the first FIFO address and BUFFST and BUFFEND values.

After initialization, both BUFFST and BUFEND are set at 00H (first FIFO position).

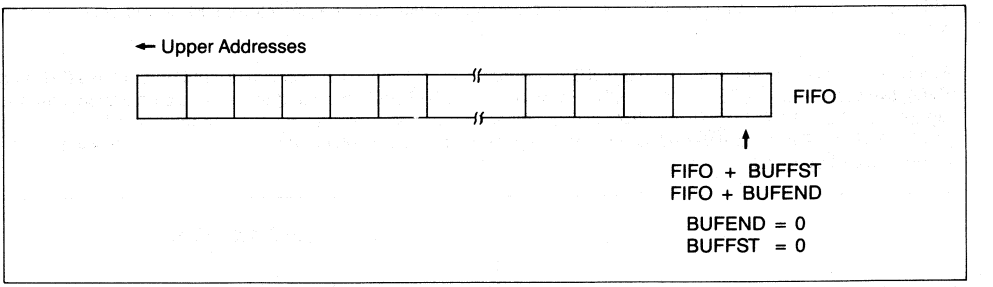


Figure 3-3 Buffer Status (after Initialization)

1-byte data is written into the FIFO address indicated by BUFEND and the BUFEND value is increased (increment).

Since FIFO is of an endless type, the first FIFO address is reset if the BUFEND value becomes greater than the final FIFO address (BUFEND > 15).

When the increased (increment) BUFEND value becomes equal to the BUFFST value, FIFO becomes data-full.

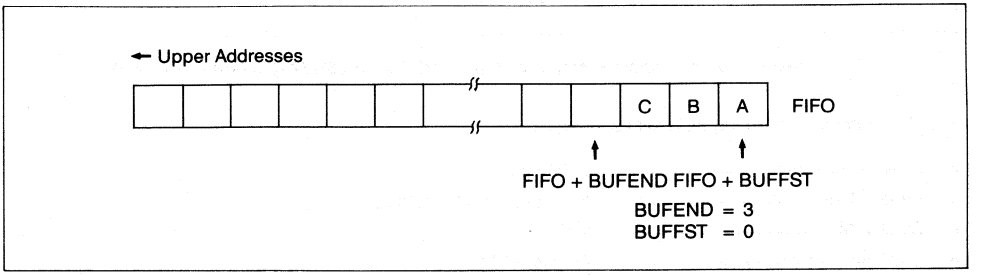


Figure 3-4 Buffer Status (3-Byte Write after Initialization)

Prior to reading data from FIFO, the FIFO is checked for previously stored data. If the BUFFST and BUFEND values coincide with each other, and data receive is enabled, then no data has been prestored in the buffer.

If there is any read data, data stored at the address indicated by the BUFFST value is read and the BUFFST value is increased (increment). If the BUFFST value becomes greater than the final FIFO address (BUFFST > 15), the first FIFO address is set.

When 1-byte data is read from FIFO, a 1-byte empty area is added to the buffer.

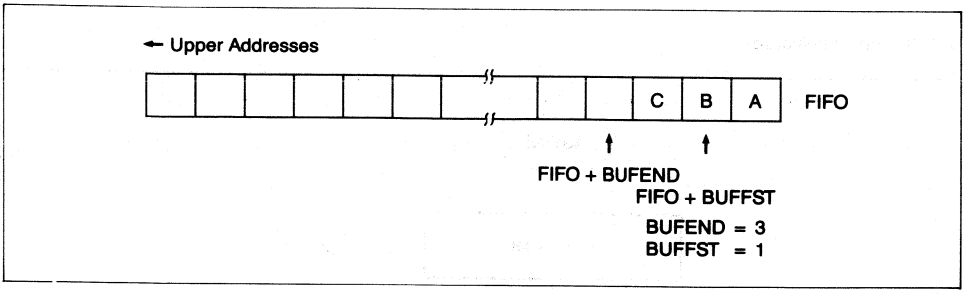


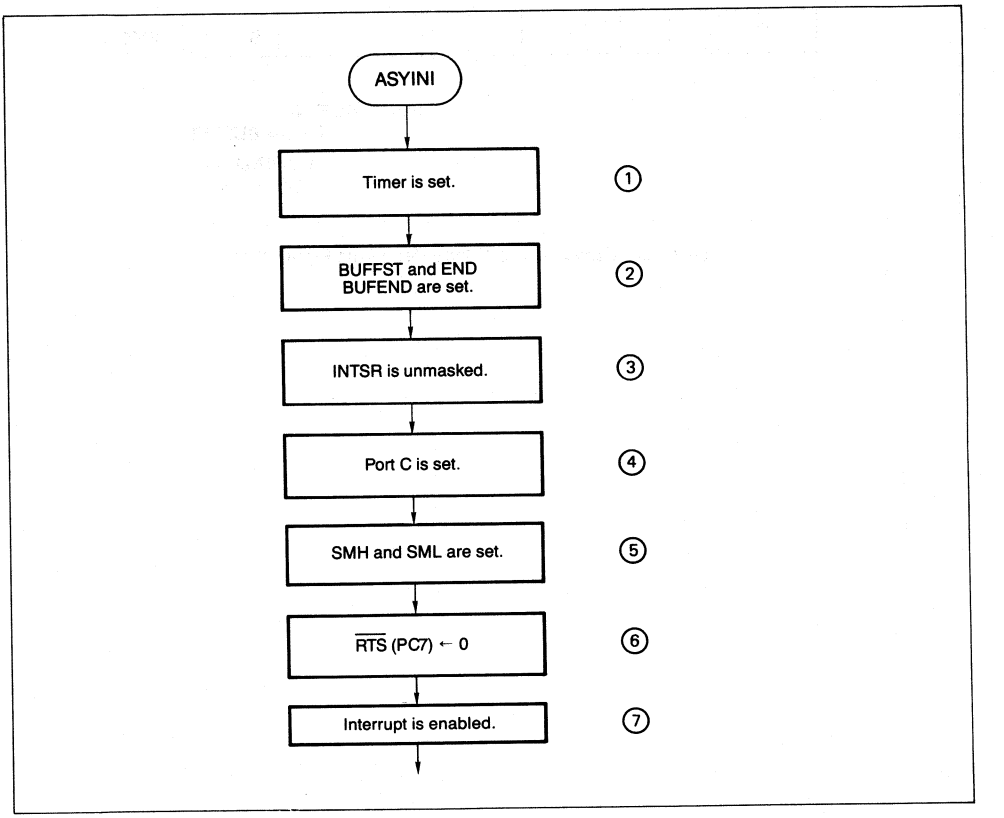
Figure 3-5 Buffer Status (1-byte Read from Figure 6-4)

(2) Description of programs for initialize, data transmission/reception and received data read.

(a) Initialize

Initialize for serial data communications in the asynchronous mode is described below:

- **Flowchart** (initialize)



① The timer operation is set to generate a serial clock ($\overline{SCK}$).
When using a TO output for $\overline{SCK}$, the timer count value is given as follows:

- When timer clock count = ϕ_{12}: $C = \dfrac{f_{2XTAL}}{2 \times 12 \times N \times B}$
- When timer clock count = ϕ_{384}: $C = \dfrac{f_{XTAL}}{2 \times 384 \times n \times B}$

f_{XTAL}: Oscillation frequency
N : Clock rate
B : Data transfer rate
C : Timer count value

If ocsillation frequency = 11.0592 MHz, clock rate = x 16, data transfer rate = 9600 bps and timer count clock = ϕ_{12}, then the time count value is 3.

② The data receive buffer is initialized and the first FIFO position (00H) is set for BUFFST and BUFEND.

③ Interruption is set and INTSR is unmasked. INTST remains masked.

④ Port C is set. PC0 and PC1 are set for TxD and RxD respectively. PC6 and PC7 are set for an input/output pin. After that, the output latch of PC7 is set to "1", and PC7 and PC6 are set for an output port and an input port respectively.

⑤ The serial mode registers (SMH and SML) are as shown in Figure 3-6 to specify the serial interface operation.

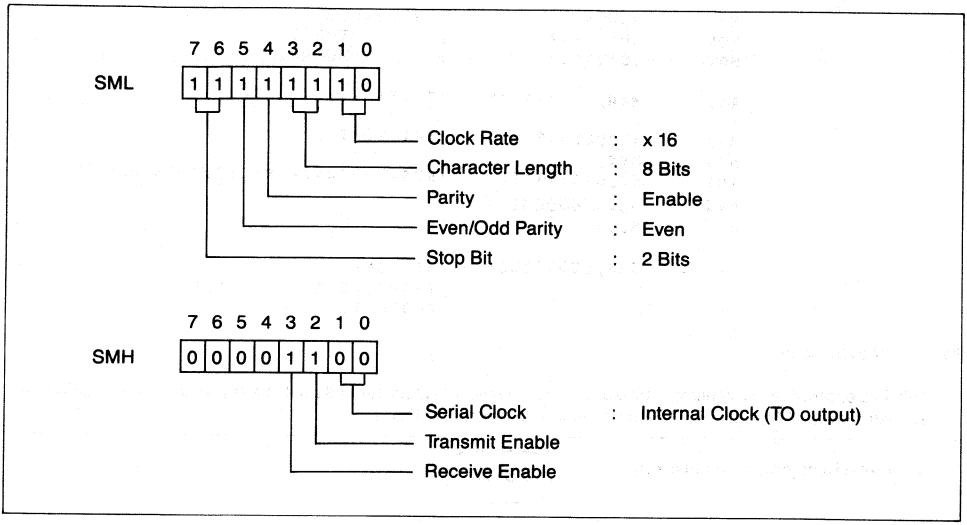


Figure 3-6 SMH and SML Settings (Asynchronous Mode)

⑥ $\overline{RTS}$ (PC7) is set to "0" and receive enable is displayed on the μPD71051(8251).

⑦ Interruption is enabled by the EI instruction.

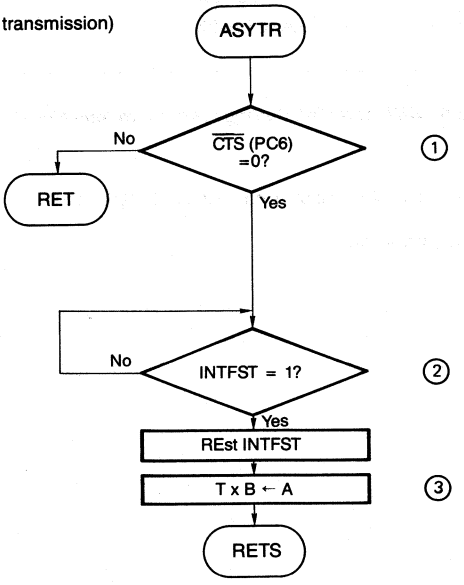
- **Program list** (initialize)

```
;********INITIALIZE**********************
ASYINI: MVI     A,11111110B     ;SET SML
        MOV     SML,A           ;DATA 8BIT,EVEN PARITY ENABLE
                                ;2 STOP BIT
                                ;ASYNCHRONOUS,*16
;
        MVI     A,3             ;SET TIMER0
        MOV     TM0,A
        ANI     TMM,11100000B
;
        MVI     A,00H           ;INITIALIZE
        MOV     BUFFST,A        ;       BUFFST
        MOV     BUFEND,A        ;       BUFEND
;
        ANI     MKH,11111101B   ;RESET INTSR MASK
;
        MVI     A,00000111B     ;SET PORT C
        MOV     MCC,A
        ANI     PC,01111111B    ;RTS<--0(DATA RECEIVE ENABLE)
        MVI     A,01000000B
        MOV     MC,A
;
        MVI     SMH,00011000B   ;SET SMH
                                ;TRANS,RECEIVE ENABLE
                                ;CLOCK TO
        EI
```

(b) Data transmission

This is a subroutine to transmit 1 byte of 'A' (accumulator) data as serial data. Data is transmitted by testing the interrupt request flag (INTFST) without using the interrupt (INTST).

- **Flowchart** (data transmission)



① Ability to transmit data is checked by $\overline{CTS}$ (PC6). If the μPD71051(8251) does not acknowledge data transmission, the transmission is terminated by RET.

② Ability to write data into the transmit buffer register (TxB) is checked by testing the INTFST flag.

③ The 'A' data is transferred to TxB. After completion of the 'A' data transfer, the transmission is terminated by RETS.

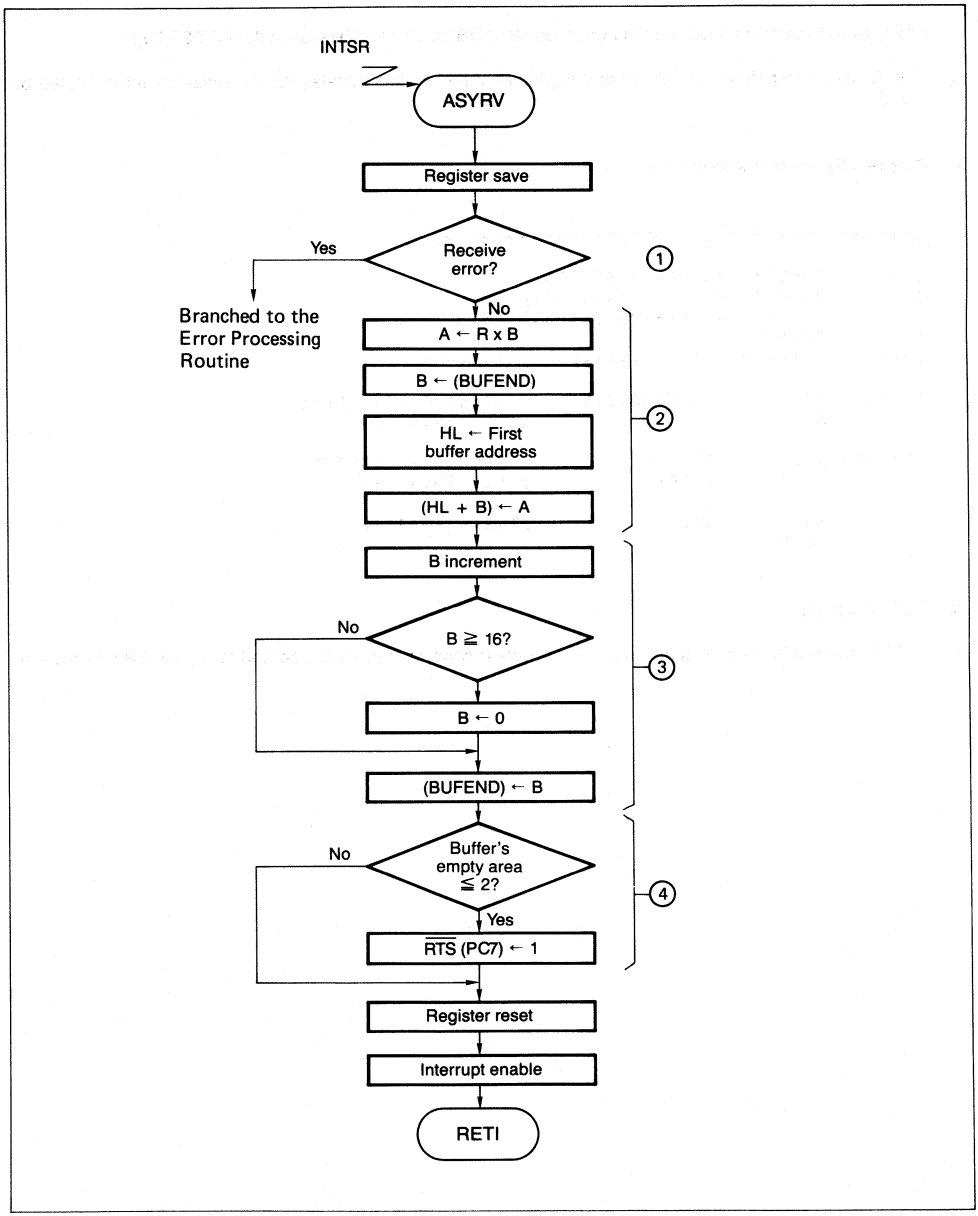
- **Program list** (data transmission)

```
;
;*******DATA TRANS ROUTINE***************
;
;         INPUT:   A<--DATA FOR TRANS
;         OUTPUT:  RETS<--TRANS DATA
;                  RET<---NO TRANS(71051 IS' BUSY)
;         CHANGE:  -
;=======================================
;
ASYTR:.   OFFI     PC,01000003      ;CHECK! 71051 READY?
          RET                       ; NO! THEN RET
;
ASYTR1:   SKIT     FST              ;CHECK! TXB EMPTY?
          JMP      ASYTR1           ; NO! THEN WAIT
;
          MOV      TXB,A            ;TXB<--A(DATA)
          RETS
```

(c) Data reception

INTSR interrupt is used to receive data. Thus, data receive operations are carried out by INTSR interruption.

- **Flowchart** (data reception)



① A check is made for serial data reception error. If any error is detected, the operation is branched to the error processing operation.

② The received data is read from the receive buffer register (RxB) and is stored into the buffer.

③ The buffer's empty area is checked. If the empty area is 2 bytes or less, $\overline{RTS}$ (PC7) is set to a non-active state to disable data transmission from the μPD71051(8251).

If $\overline{RTS}$ (PC7) is set to a non-active state, then the μPD71051(8251) completes the transmission of data currently being transmitted and the μPD71051(8251) data set in the transmit data buffer, and then is disabled for data transmission. Because a maximum of 2-byte data may be received aftr $\overline{RTS}$ (PC7) has been set to a non-active state, $\overline{RTS}$ (PC7) becomes non-active if the buffer's empty area is 2 bytes or less.

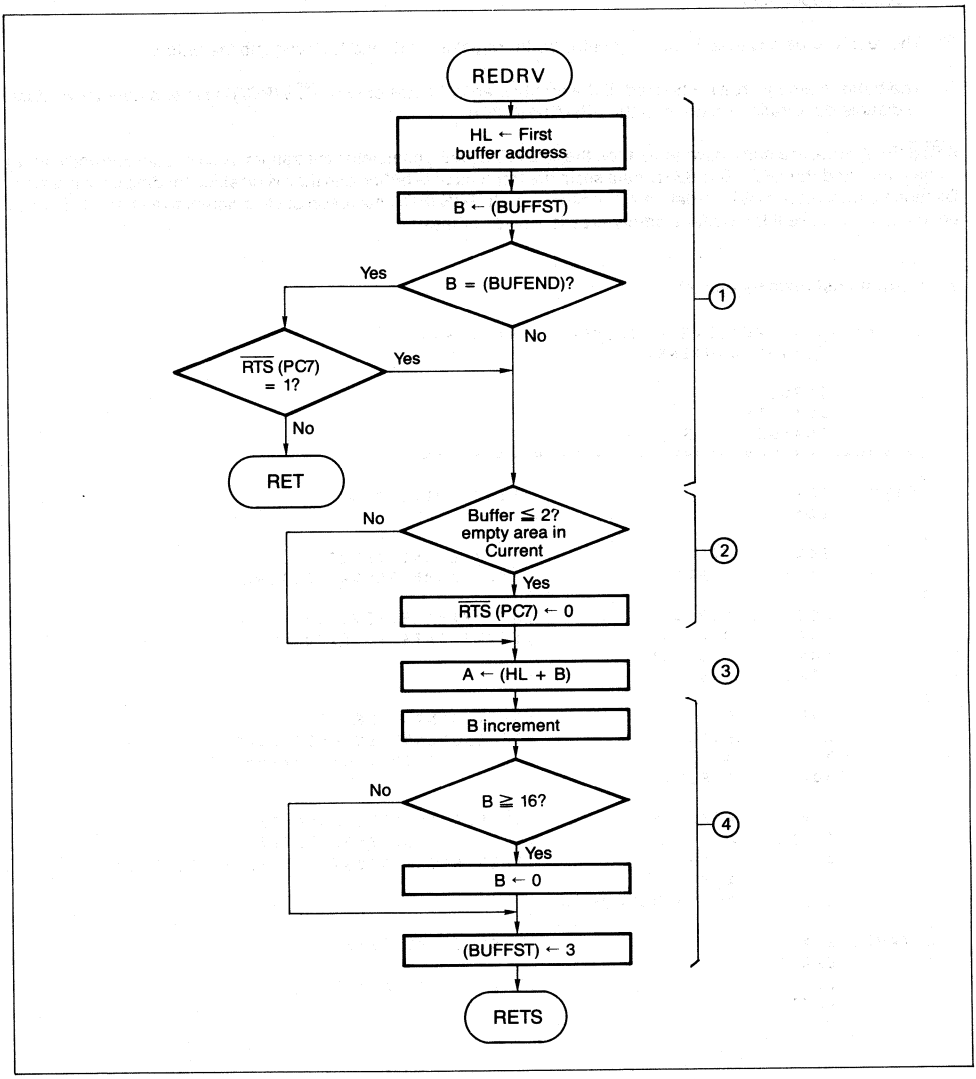
- **Program list** (data reception)

```
;*******DATA RECEIVE ROUTINE************
;       (INTSR ROUTINE)
;
;       INPUT:  -
;       OUTPUT: -
;       CHANGE: A',B',H'L'
;=======================================
;
ASYRV:  EXX                         ;STORE REGISTER
        EXA
;
        SKNIT   ER                  ;IF RECEIVE ERROR
        GJMP    ERROR               ;THEN JUMP ERROR ROUTINE
;
        MOV     A,RXB               ;A<--RXB(RECEIVE DATA)
        LXI     H,FIFO              ;STORE DATA TO FIFO
        MOV     B,BUFEND
        STAX    H+B
;
        INR     B                   ;INCREMENT (BUFEND)
        LTI     B,16                ;IF (BUFEND) IS OVER FIFO
        MVI     B,0                 ;       THEN (BUFEND)<--00H
        MOV     BUFEND,B
;
        MOV     A,BUFFST            ;IF FIFO'S FREE AREA IS
        SUBNB   A,B                 ;       LESS THAN 2BYTE
        ADI     A,16                ;THEN RTS<--1(DATA RECEIVE DISABLE)
        GTI     A,2
        ORI     PC,100000003
;
ASYRVO: EXX                         ;RESTOER REGISTER
        EXA
        EI                          ;INTERRUPT ENABLE
        RETI
```

(d) Received data read

This is a subroutine to read the received data from the buffer. Since the buffer is in FIFO format, the first-received data is first read. The read data is stored into 'A' (accumulator).

- **Flowchart** (received data read)



① Whether there is any read data in the buffer is checked. When BUFFST = BUFEND and $\overline{RTS}$ (PC7) are active, then the buffer is empty. If there is no read data, then the read operation is terminated by RET.

② The buffer checked for any empty area. If the empty area becomes 3 bytes or more after data has been read from the buffer, $\overline{RTS}$ (PC7) is set to an active state.

③ Data is read from the buffer and is stored into 'A'.

④ The BUFFST value is updated.

Upon completion of data read, the operation is terminated by RETS.

- **Program list** (received data read)

```
;********READ RECEIVE DATA ROUTINE********
;
;       INPUT:  -
;       OUTPUT: RET     FIFO IS EMPTY
;               RETS    A<--DATA
;       CHANGE: A,BC,HL
;
;=======================================
REDRV:  LXI     H,FIFO          ;HL<--FIFO LEAST ADDRESS
        MOV     A,BUFFST
        MOV     B,A
        MOV     C,BUFEND
        EQA     A,C             ;IF (BUFFST)=(BUFEND)
        GJMP    REDRV0          ;       .AND.
        ONI     PC,10000000B    ;RTS=0(DATA RECEIVE ENABLE)
        RET                     ;THEN FIFO IS EMPTY
;
REDRV0: SUBNB   A,C             ;IF NOW FIFO HAVE FREE AREA
        ADI     A,16            ;       THAN 2BYTE
        LTI     A,2             ;THEN RTS<--0(DATA RECEIVE ENABLE)
        ANI     PC,01111111B
;
        LDAX    H+3             ;GET 1BYTE DATA FROM FIFO
        INR     B               ;AND INCREMENT (BUFFST)
        LTI     B,16
        MVI     B,0
        MOV     BUFFST,B
        RETS
```

3.1.2 Synchronous Mode Communications

This section describes an example of serial data communications with the μPD71051(8251) in the synchronous mode.

Figure 3-7 shows the connection with the μPD71051(8251). Serial data input/output lines (TxD and RxD), a serial clock supply line ($\overline{SCK}$), control lines, $\overline{RTS}$)PC7) and $\overline{CTS}$ (PC6), for a total of 5 lines, are used. $\overline{RTS}$ (PC7) is a transmit enable signal input port for the μPD71051. Because $\overline{RTS}$ (PC7) is set for an input port after the resetting has been cancelled, it is set at a high level via a pull-up resistor.

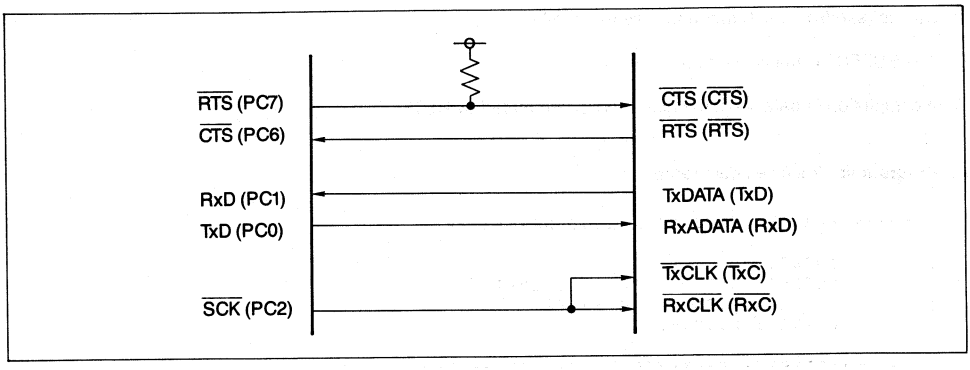


Figure 3-7 Connection Example with the μPD71051(8251) (Synchronous Mode)

The data communication format is set as follows:

- Data transfer rate — 28.8 kbps
- Serial clock — Internal clock (ϕ384); Output to SCK pin
- Oscillation frequency — 11.0593 MHz
- Data format — Character 8-bit; Non-parity
- Half-duplex communications

1 to 256-bytes of data are transmitted or received in a single communication. A 1-byte character (16H in this program) for synchronization at the beginning of data is followed by 1 byte for the number of transfer data items followed by transfer data. The value for the number of transfer data items is given as "the number of data items which are transferred minus 1".

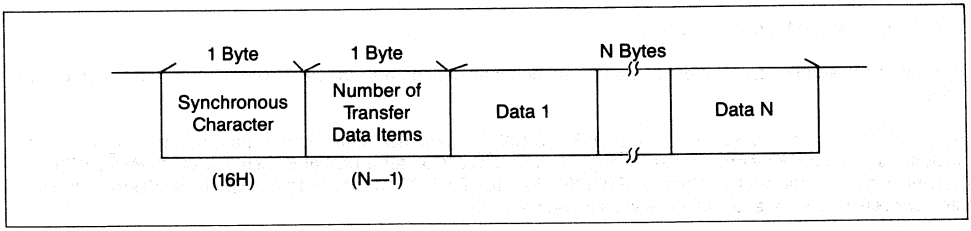


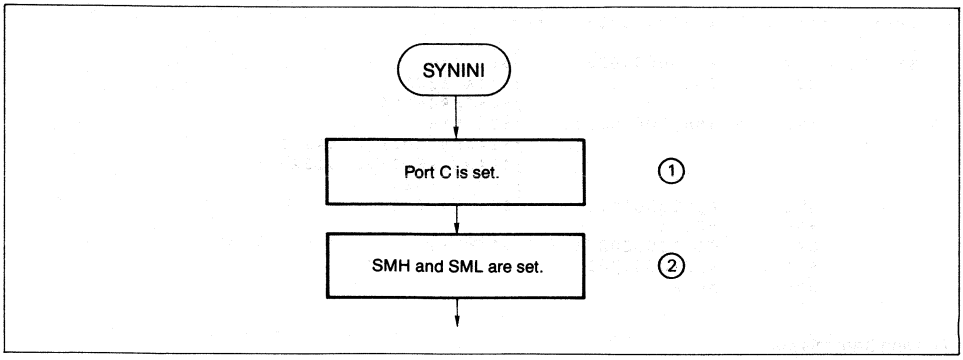
Figure 3-8 Transfer Data Format

Initialize, data transmit, and data receive programs are described below.

(1) Initialize

Initialize for serial data communications in the synchronous mode is carried out as follows:

- **Flowchart** (initilaize)



① Port C is set. PC0 and PC1 are set for TxD and RxD respectively. PC2 is set for SCK nd PC6 and PC7 are set for an input/output pin. After that, the PC7 output latch is set to "1" and PC7 and PC6 are set for an output port and an input port respectively.

② The serial mode registers (SMH and SML) ae set as shown in figure 3-9.

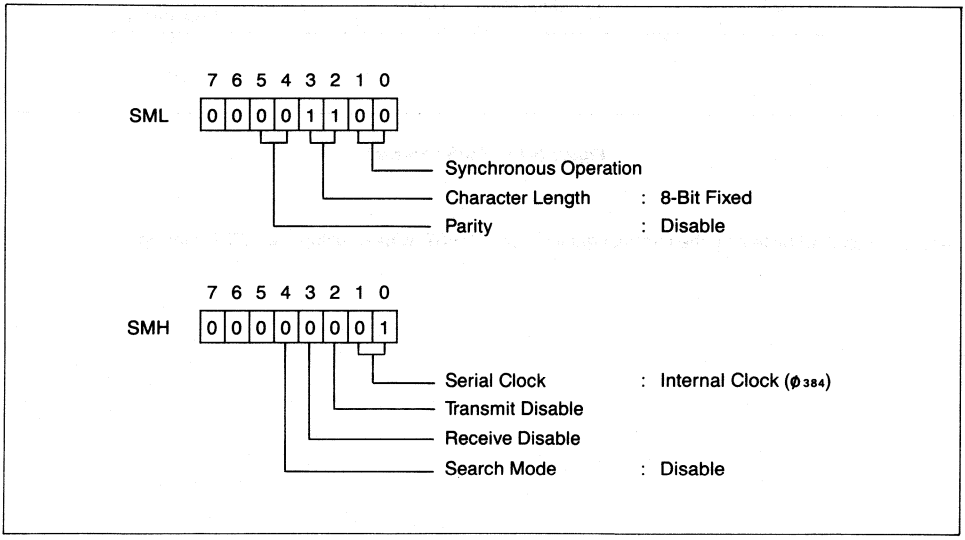


Figure 3-9 SMH and SML Settings (Synchronous Mode)

Since none of INTST and INTSR interrupts are used, they remain masked.

- **Program list** (initialize)

```
;*******INITIALIZE ROUTINE*******
;
SYNINI: MVI     A,00001100B     ;SET SML
        MOV     SML,A           ;DATA 8BIT
                                ;SYNCHRONOUS
        MVI     SMH,00000001B   ;SET SMH
                                ;SEARCH MODE DISABLE
                                ;TRANS,RECEIVE DISABLE
                                ;CLOCK /384
        MVI     A,00000111B     ;SET PORT C
        MOV     MCC,A
        ORI     PC,10000000B    ;RTS<--1
        MVI     A,10000000B
        MOV     MC,A
```

(2) Data transmission

The data stored in the buffer is transmitted. This routine is executed by specifying the first buffer address with the HL register. The buffer is configured in the memory and has a format as shown in figure 3-10.

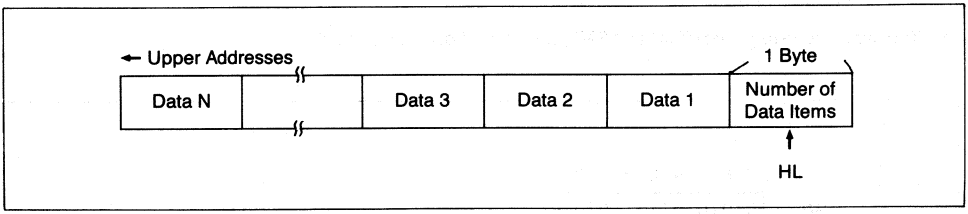
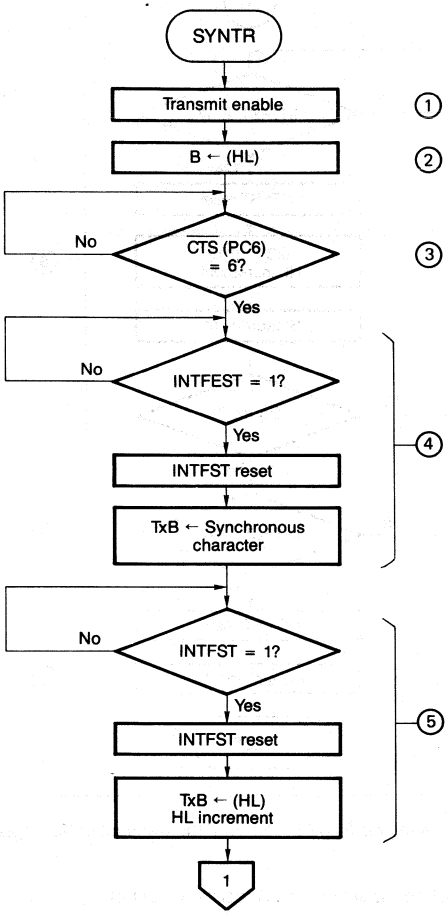
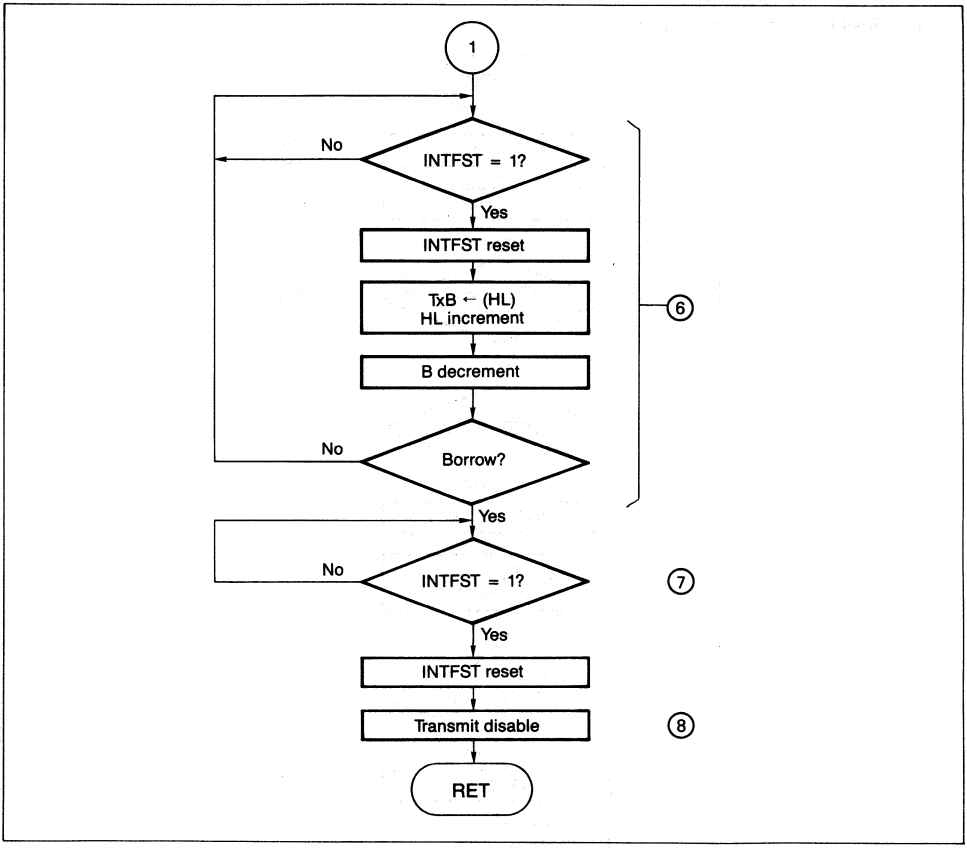


Figure 3-10 Buffer Format

Data is transmitted by testing the interrupt request flag (INTFST) without using the INTST interrupt.

- **Flowchart** (data transmission)





① Transmission is enabled by setting bis 2 (TxE) of SMH. If the transmit buffer register (TxB) is empty, then the INTFST flag is set.

② The number of transmit data items is set into the B register (counter).

③ The system waits for the μPD71051(8251) to be enabled for reception by $\overline{CTS}$ (PC6).

④ The synchronous character (16H) is transmitted.

⑤ The transmit data numerical value is transmitted.

⑥ Data is transmitted using the B register as a counter.

⑦ The system waits for INTFST set to check that data transmission has started after the final data was transferred from TxB to the transmit serial register.

⑧ TxE is reset and disabled for transmission. As a result, transmission is disabled after the currently transmitted data has been transmitted completely.

- **Program list** (data transmission)

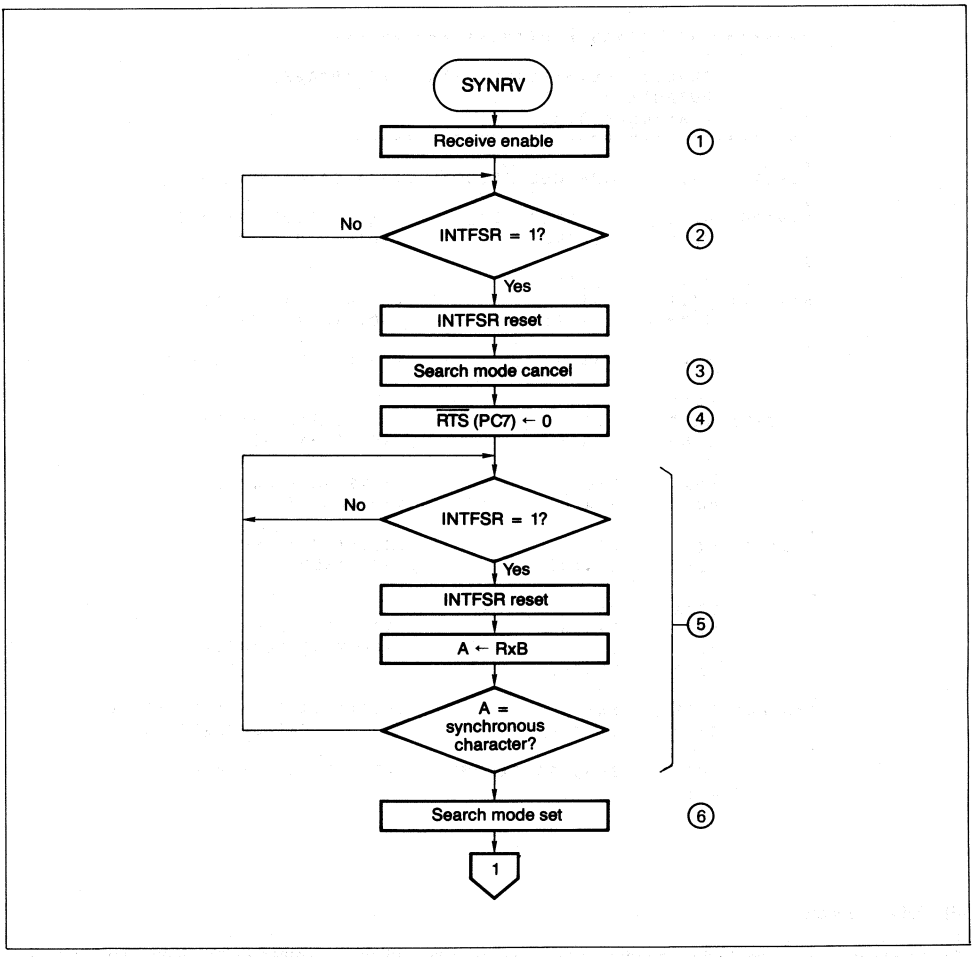
```
;*******DATA TRANS ROUTINE************
;
;       INPUT:  HL<--TRANS DATA BUFF ADDRESS
;       OUTPUT: -
;       CHANEGE:A,B,HL
;=======================================
;
SYNTR:  ORI     SMH,00000100B   ;DATA TRANS ENABLE
;
SYNTR0: OFFI    PC,01000000B    ;CHECK! 71051 READY?
        GJMP    SYNTR0          ; NO! THEN WAIT
;
        MVI     A,16H           ;
SYNTR1: SKIT    FST             ;CHECK! TXB EMPTY?
        GJMP    SYNTR1          ; NO! THEN WAIT
;
        MOV     TXB,A           ;TRANS 16H(SYNC CHARA)
;
        LDAX    H+              ;
SYNTR2: SKIT    FST             ;CHECK! TXB EMPTY?
        GJMP    SYNTR2          ; NO! THEN WAIT
;
        MOV     TXB,A           ;TRANS DATA NUMBER
        MOV     B,A             ;SET COUNTER
;
SYNTR3: SKIT    FST             ;CHECK! TXB EMPTY?
        GJMP    SYNTR3          ; NO! THEN WAIT
;
        LDAX    H+              ;
        MOV     TXB,A           ;TARNS 1BYTE DATA
        DCR     B               ;DECREMENT COUNTER
        GJMP    SYNTR3          ;
;
SYNTR4: SKIT    FST             ;CHECK! LAST DATA TRANS TO
        GJMP    SYNTR4          ;       SERIAL REGISTER?
;
        ANI     SMH,11111011B   ;DATA TRANS DISABLE
        RET
```

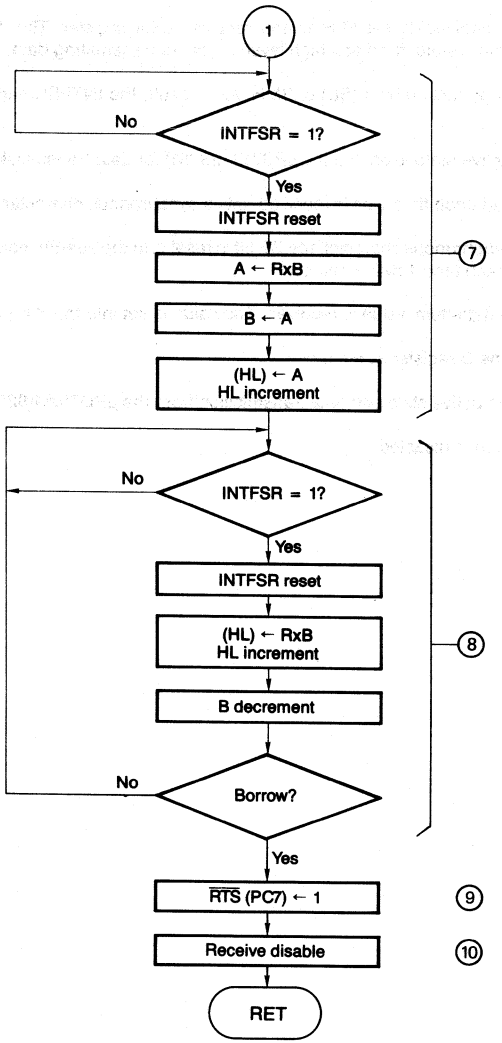
(3) Data reception

The received data is stored into the buffer of the memory. This routine is executed by specifying the first buffer address with the HL register. The data format stored in the buffer is the same as shown in figure 3-10.

Data is received by testing the interrupt request flag (INTFSR) without using the INTSR interrupt.

- **Flowchart** (data reception)




1
No
INTFSR = 1?
Yes
INTFSR reset
A ← RxB
B ← A
(HL) ← A
HL increment
7
No
INTFSR = 1?
Yes
INTFSR reset
(HL) ← RxB
HL increment
B decrement
8
No
Borrow?
Yes
RTS (PC7) ← 1
9
Receive disable
10
RET

① Reception is enabled by setting bit 3 (RxE) of SMH.

② 1-byte dummy data is received to set FFH to the receive serial register. The TxDATA (TXD) pin of the μPD71051(8251) is in a mark state (fixed at a high level) if it is not transmitting data.

③ The search mode is set by setting bit 4 (SE) of SMH. As a result, the INTFSR flag is set each time 1 bit is received.

④ $\overline{\text{RTS}}$ (AC7) is set in an active state to enable the μPD71051(8251) for data transmission.

⑤ Received data is checked each time 1 bit is received until a synchronous character (16H) is received.

⑥ Upon reception of the synchronous character, the SE bit is reset and the search mode is cancelled. After that, the INTFSR flag is set each time 1 byte is received.

⑦ The 'number-of-receive-data-item' value is received. The value is set into the B register for use as a counter.

⑧ Data is received using the B register as a counter.

⑨ $\overline{\text{RTS}}$)PC7) is set to a non-active state and data transmission from the μPD71051(8251) is disabled.

⑩ RxE is reset and reception is disabled.

- **Program list** (data reception)

```
;*******DATA RECEIVE ROUTINE************
;
;       INPUT:  DE<--RECEIVE DATA STORE BUFF ADDRESS
;       OUTPUT: -
;       CHANGE: A,B,D,E,H,L
;=======================================
;
SYNRV:  ORI     SMH,00001000B   ;DATA RECEIVE ENABLE
;
SYNRV0: SKIT    FSR             ;CHECK DATA INPUT
        GJMP    SYNRV0          ;       FOR STORE FFH TO SIFT REGISTER
;
        ORI     SMH,00010000B   ;SET SEARCH MODE
        ANI     PC,01111111B    ;RTS<--0(DATA RECEIVE ENABLE)
;
SYNRV1: SKIT    FSR             ;CHECK! 1BIT INPUT?
        GJMP    SYNRV1          ; NO! THEN WAIT
;
        MOV     A,RXB           ;
        EQI     A,16H           ;CHECK! INPUT DATA IS 16H(SYNC CHARA)
        GJMP    SYNRV1          ; NO! THEN JUMP
;
        ANI     SMH,11101111B   ;RESET SEARCH MODE
;
SYNRV2: SKIT    FSR             ;CHACK! DATA INPUT?
        GJMP    SYNRV2          ; NO! THEN WAIT
;
        MOV     A,RXB           ;INPUT DATA NUMBER
        MOV     B,A             ;SET COUNTER
        STAX    H+              ;STORE TIO BUFFER
;
SYNRV3: SKIT    FSR             ;CHECK! ADAT INPUT?
        GJMP    SYNRV3          ; NO! THEN WAIT
;
        MOV     A,RXB           ;STORE INPUT DATA
        STAX    H+              ;       TO BUFFER
        DCR     B               ;DECREMENT COUNTER
        GJMP    SYNRV3          ;
;
        ORI     PC,10000000B    ;RTS<--1(DATA RECEIVE DISABLE)
        ANI     SMH,11110111B   ;DATA RECEIVE DISABLE
        RET
```

3.2 Interfacing with the μPD7500

This section describes examples of serial data communications with a μPD7500 series 4-bit single-chip microcomputer.

The μPD7500 series serial interface has a $\overline{SCK}$ synchronous 8-bit shift register for synchronization with characters by eight serial clocks.

For serial data communications with a μPD7500 series microcomputer, the serial interface is used in the I/O interface mode.

3.2.1 I/O Interface Mode Communications

Figure 3-11 shows the connection for serial data communications with the μPD7500 series microcomputer.

Data input/output lines (TxD and RxD), a serial clock supply line ($\overline{SCK}$), and control lines (PC7 and PC6) are used. PC7 is connected to the external interrupt pin (INT0) of the μPD7500 series microcomputer and an interrupt is applied to PC7 for data communications. The $\overline{SCK}$ request signal from the μPD7500 series is input to PC6. Because PC7 is set for an input port after the resetting has been cancelled, it is set to a low level via a pull-up resistor.

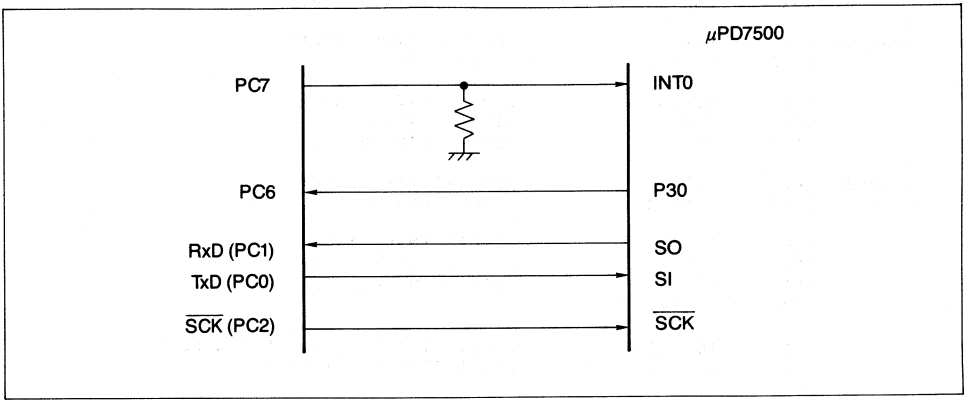


Figure 3-11 Connection Example with μPD7500 Series Microcomputer

The data communication format is set as follows:

- Data transfer rate ——— 153.6 kbps
- Serial clock ——— Internal clock (TO output)
  Output to $\overline{SCK}$ pin
- Oscillation frequency ——— 11.0592 MHz
- Data format ——— Character 8 bits

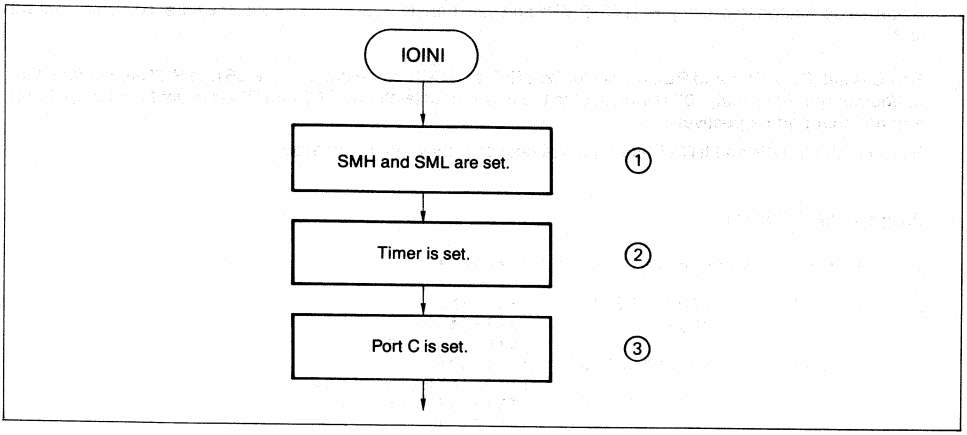
4-byte data is exchanged with the μPD7500 series microcomputer in a single communication.

Transmission and reception are performed simultaneously with eight serial clocks. Therefore, the μPD7500 series microcomputer must be ready for transmission and reception before clock supply to $\overline{SCK}$ is started. The PC6 line is used for control of preparations for transmission and reception.

(1) Initialize

Initialization is carried out for data communications with the μPD7500 series microcomputer in the I/O interface mode.

- **Flowchart** (initialize)



① The serial mode registers (SMH and SML) are set as shown in figure 3-12.

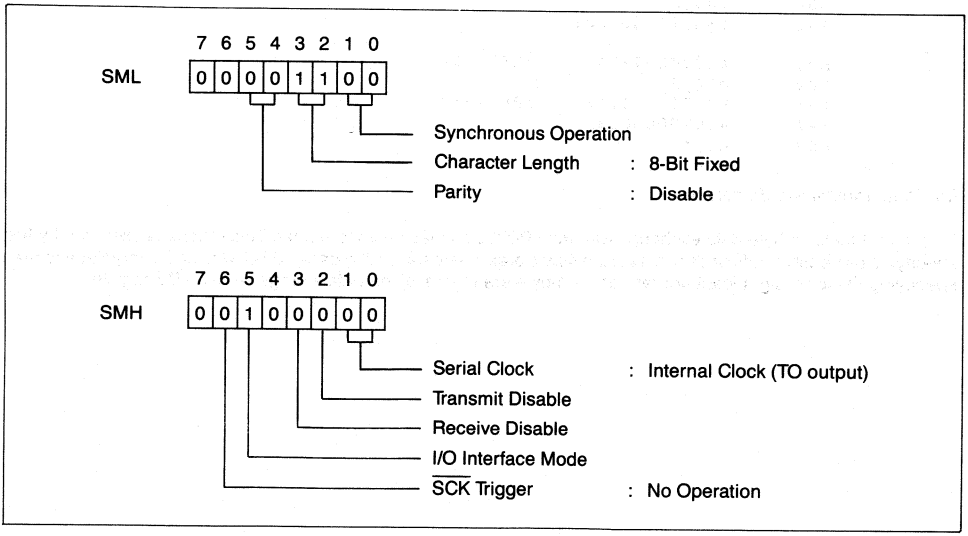


Figure 3-12 SMH and SML Settings (I/O Interface Mode)

② The timer is set to generate a serial clock ($\overline{SCK}$). When using ϕ_{12} for a timer count clock, the timer count value is given as follows:

$$C = \frac{f_{XTAL}}{2 \times 12 \times B}$$

f_{XTAL}: Oscillation frequency
B : Data transfer rate
C : Count value

Because oscillation frequency = 11.0592 MHz and data transfer rate = 153.6 kbps, the timer count value is set to 3.

③ Port C is set. PC0, PC1 and PC2 are set for TxD, RxD and $\overline{SCK}$ pin respectively. PC6 and PC7 are set for an input/output pin. After that, "0" is written onto the PC7 output latch and PC7 and PC6 are set for an output port and an input port respectively.

Because the INTSR and INTST interrupts are not used, they remain masked.

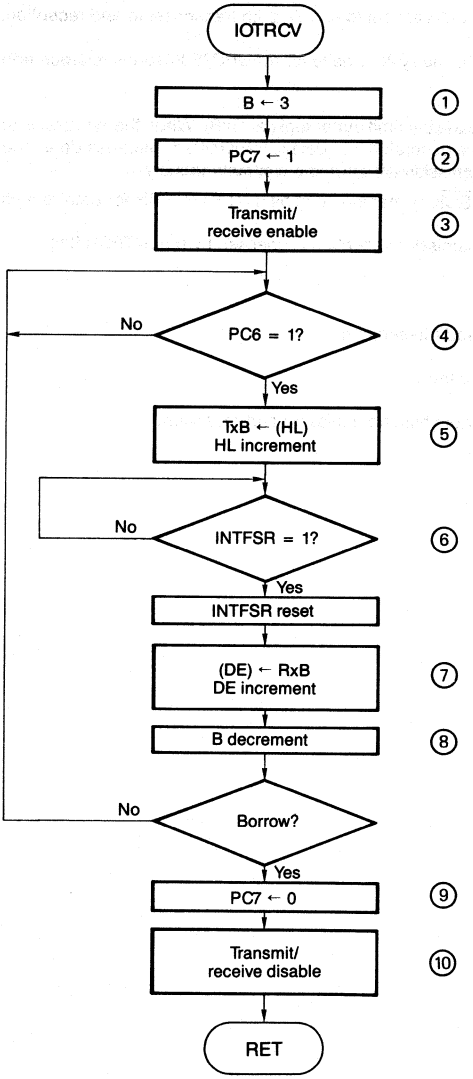
- **Program list** (initialize)

```
;********INITIALIZE ROUTINE***************
;
IOINI:  MVI     A,00001100B     ;SET SML
        MOV     SML,A           ;DATA 8BIT
                                ;SYNCHRONOUS
        MVI     SMH,00100000B   ;SET SMH
                                ;I/O-INTERFACE MODE
                                ;TRANS,RECEIVE ENABLE
                                ;CLOCK TO
;
        MVI     A,3             ;SET TIMER 0
        MOV     TM0,A
        ANI     TMM,11100000B
;
        MVI     A,00000111B     ;SET PORT C
        MOV     MCC,A
        ANI     PC,01111111B    ;PC7<--0
        MVI     A,01000000B
        MOV     MC,A
```

(2) Data transmission/reception

This is routine for 4-byte data exchange with the μPD7500 series microcomputer. This routine is executed by first specifying the least significant address of the 4-byte area where transmit data is stored with the HL register and then specifying the least significant address of the 4-byte area for received data storage with the DE register.

- **Flowchart** (data transmission/reception)



① The value (3) of "number of data items —1" is set into the B register.

② The PC7 output is set to a high level to request the μPD7500 series microcomputer for data communications. As a result, an external interrupt is generated in the μPD7500 series microcomputer.

③ Bit 2 (TxE) and bit 3 (RxE) of SMH are set to enable both transmission and reception.

④ The PC6 input is checked and the system waits for the μPD7500 series microcomputer to become ready for transmission and reception.

⑤ Transmit data is transferred to the transmit buffer register (TxB). When the TxB data is transferred to the transmit serial register, eight clocks are generated and data is transmitted. Because data receive has previously been enabled, reception and transmission are carried out simultaneously.

It is not necessary to trigger $\overline{SCK}$ by manipulating bit 6 (TSK) of SMH for data reception.

⑥ Completion of the 1-byte transmission/reception is checked by the INTFSR flag.

⑦ The received data is stored.

⑧ The counter value is decreased (decrement).

⑨ The PC7 output is set to a low level.

⑩ TxE and RxE are reset to disable both transmission and reception.

- **Program list** (data transmission/reception)

```
;*******DATA TRANS/RECEIVE ROUTINE*******
;
;       INPUT:  HL<--TRANS DATA FRONT ADDRESS
;       OUTPUT: -
;       CHANGE: A,B,HL,DE
;=======================================
;
IOTRCV: MVI     B,3             ;SET COUNTER
;
IOTRC0: ORI     PC,10000000B    ;PC7<--1(TRIGGER TO 75XX)

        ORI     SMH,00011000B   ;DATA TRANS,RECEIVE ENABLE
;
IOTRC1: SKIT    FST             ;CHECK! TXB EMPTY?
        GJMP    IOTRC1          ; NO! THEN WAIT
;
IOTRC2: ONI     PC,01000000B    ;CHECK! 75XX READY?
        GJMP    IOTRC2          ; NO! THEN WAIT
;
        LDAX    H+              ;SET TARNS DATA TO TXB
        MOV     TXB,A           ;AND INCREMENT HL
;
IOTRC3: SKIT    FSR             ;CHECK! RECEIVE(TRANS) FINISH?
        GJMP    IOTRC3          ; NO! THEN WAIT
;
        MOV     A,RXB           ;STORE RECEIVE DATA TO BUF
        STAX    D+              ;AND INCREMENT DE
        DCR     B               ;DECREMENT COUNTER
        GJMP    IOTRC1
;
        ANI     PC,01111111B    ;PC7<--0
        ANI     SMH,11110011B   ;DATA TRANS,RECEIVE DISABLE
        RET
```

Chapter 4 A/D Converter Functions

μCOM-87AD is equipped with a high-precision 8-bit A/D converter. Refer to the user's manual for details regarding the hardware. This chapter describes scan/select mode program examples.

4.1 Scan Mode Program Examples

As an analog/digital converter's scan mode program example, a description is made of the storage of AN0 to AN7 pin A/D converted values in memory areas 1000H to 103FH shown in figure 4-1.

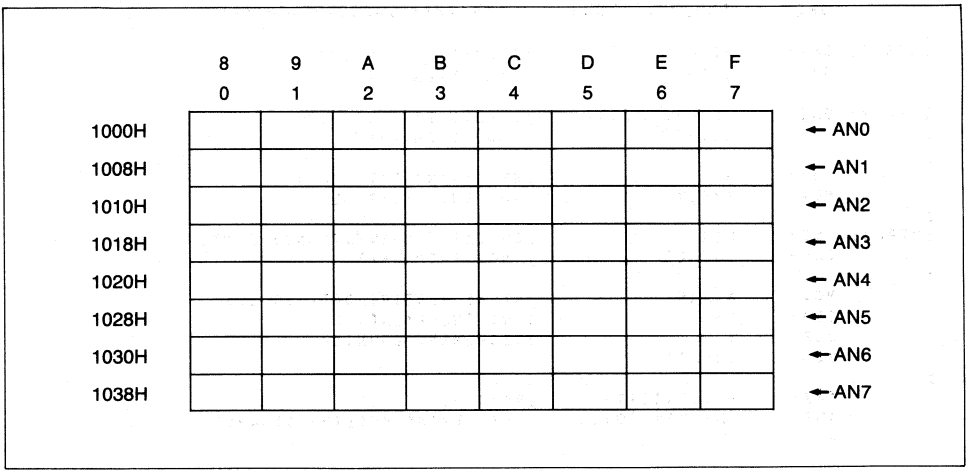


Figure 4-1 Memory Map

First, an A/D conversion of AN0 to AN3 pins is carried out four times and the AN0, AN1, AN2 and AN3 converted values are stored into the 1000H to 1003H, 1008H to 100BH, 1010H to 1013H and 1018H to 101BH areas respectively.

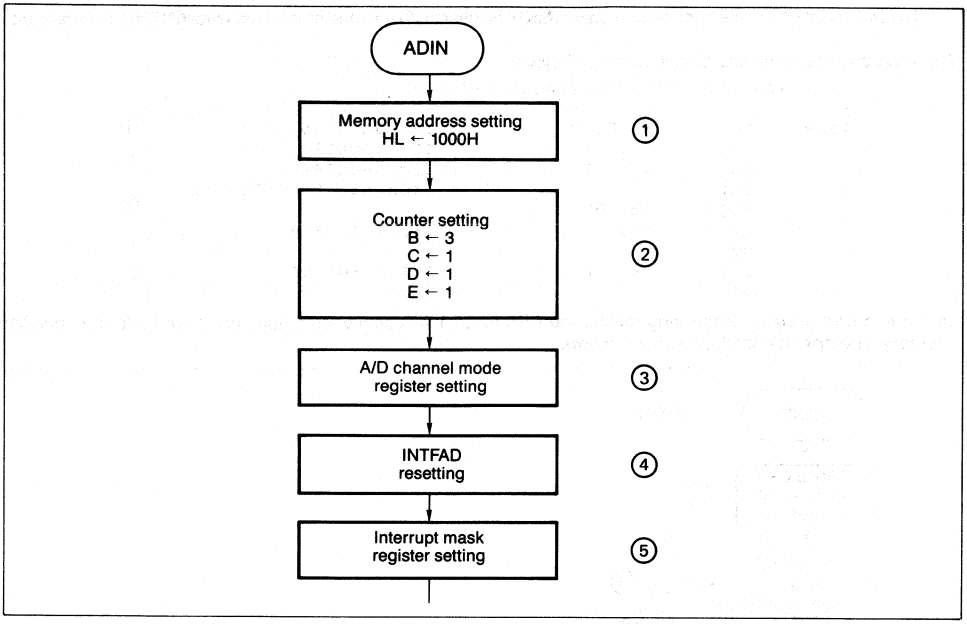
Next, an A/D conversion of AN4 to AN7 pins is carried out four times and the AN4, AN5, AN6 and AN7 converted values are stored into the 1020H to 1023H, 1028H to 102BH, 1030H to 1033H and 1038H to 103BH areas respectively.

After that, an A/D conversion of AN0to AN3 pins is carried out once again and the AN0, AN1, AN2 and AN3 converted values are stored into the 1004H to 1007H, 100CH to 100FH, 1014H to 1017H and 101CH to 101FH areas respectively.

Finally, an A/D conversion of AN4 to AN7 pins is carried out and the AN4, AN5, AN6 and AN7 converted values are stored into the 1024H to 1027H, 102CH to 102FH, 1034H to 1037H and 103CH to 103FH areas respectively.

A program example of repeating these operations is described.

First, the initialize operation flow is shown below.



① The memory address for storage of the A/D converted value is set at 1000H onwards of the HL pair register.

② The B, C, D, and E general-purpose registers are used as counters to store the A/D converted value into the specified memory. The B register is used to check that an A/D conversion of AN0 to AN3 or AN4 to AN7 has been carried out four times. Thus, the B registr is set to 03H.

The C, D, and E registers are used to store the A/D converted values into the respective memory areas and are set to 01H.

③ The A/D channel mode register is set to the scan mode and AN0 to AN3 serve as input pins.

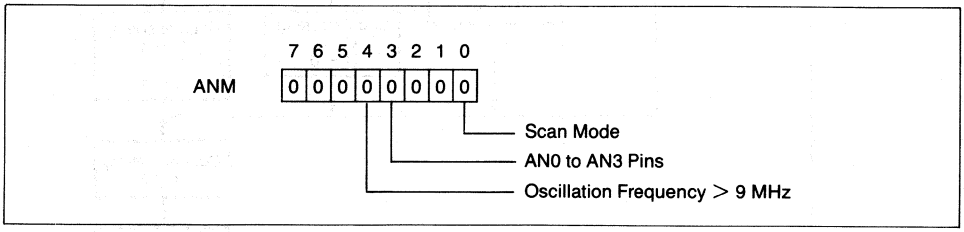


Figure 4-2 A/D Channel Mode Register Setting

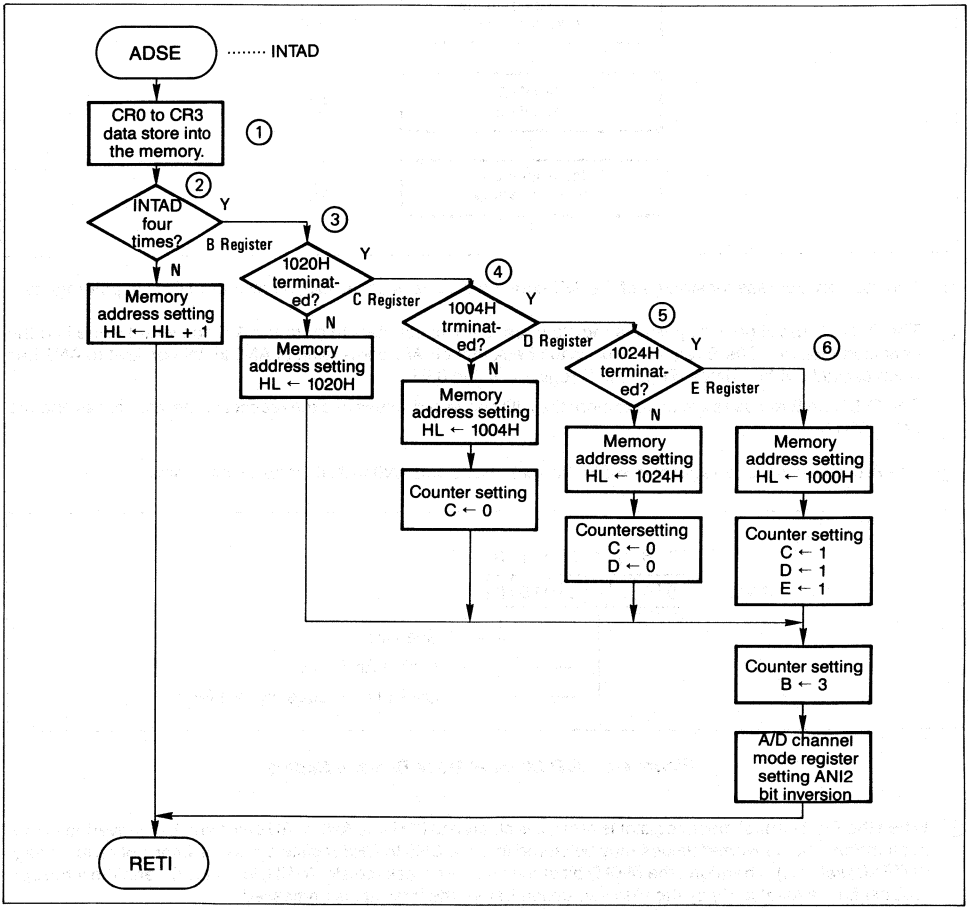
④ When the A/D channel mode register is reset, it is cleared to 00H and AN0 to AN3 pins are A/D converted in the scan mode. The converted values may be stored into the CR0 to CR3 registers with the interrupt request flag (INTFAD) set to (1). Therefore, the MKAD bit of the interrupt mask register (MKH) is set to "0" and the interrupt request flag is reset to (0) by the skip instruction before the interrupt is unmasked.

⑤ The MKAD bit of the interrupt mask register (MKH) is reset to (0) and the internal interrupt (INTAD) is unmasked.

The A/D converted initialize routine is shown below:

```
;*** A/D CONVERTER INITIALIZATION ***
;
ADIN:   LXI     H,1000H         ; SET DATA POINTER          ①
        LXI     B,0301H         ; SET COUNTER               }②
        LXI     D,0101H         ; SET COUNTER               }
        EXX                     ; EXCHANGE REGISTER SET     ③
        MVI     ANM,00H
        SKIT    FAD             ; RESET INTFAD              }④
        NOP                                                 }
        ANI     MKH,0FEH        ; INTAD ENABLE              ⑤
        EI
```

In the interrupt (INTAD) processing routine the CR0 to CR3 A/D converted values are stored into the specified memory. The operational flow is shown below:



① The data of CR0 to CR3 where AN0 to AN3 or AN4 to AN7 pin A/D converted values are stored into the specified memory.

② Internal interrupt (INTAD) is checked to have been generated four times. If it has been generated three times or less, the HL pair register value is increased by 1.

If four or more interrupts have been generated, the operation will jump to ③. The B register is a counter to check if an interrupt has been generated four times.

③ Because the A/D converted value has been stored into the memory blocks starting from 1000H (1000H to 1003H, 1008H to 100BH, 1010H to 1013H, and 1018H to 101BH), the first address (1020H) of the next block is set into the HL pair register and 03H is set into the B register. To change the input pin for A/D conversion, the AN12 bit of the A/D channel mode registr is inverted.

When the A/D converted value is stored into the memory blocks, starting from 1020H (1020H to 1023H, 1028H to 102BH, 1030H to 1033H, and 1028H to 102BH), the operation will jump to ④. The C register is a counter to check if the A/D converted value has been stored into the memory blocks starting from 1020H.

④ As the A/D converted value has been stored into the memory blocks starting from 1020H, the first address (1004H) of the next block, 03H, and 00H are stored into the HL pair register, the B register, and the C register respectively. To change the input pin for A/D conversion, the ANI2 bit of the A/D channel mode register is inverted.

When the A/D converted value is stored into the memory blocks starting from 1004H (1004H to 1007H, 100CH to 100FH, 1014H to 1017H, and 101CH to 101FH), the operation will jump to ⑤. The D register is a counter to check if the A/D converted value has been stored into the memory blocks starting from 1004H.

⑤ As the A/D converted value has been stored into the memory blocks starting from 1004H, the first address (1024H) of the next block, 03H, and 00H are stored into the HL pair register, the B register, and the C and D registers respectively. To change the input pin for A/D conversion, the ANI2 bis of the A/D channel mode register is inverted.

When the A/D converted value is stored into the memory blocks starting from 1024H (1024H to 1027H, 102CH to 102FH, 1024H to 1037H, and 103CH to 103FH), the operation will jump to ⑥. The E register is a counter to check if the A/D converted value has been stored into the memory blocks starting from 1004H.

⑥ The A/D converted value has been stored into the memory blocks 1000H to 103FH and those starting from 1024H onwards. Thus, initialization is carried out to restore the A/D converted value into the memory blocks starting from 1000H.

The interrupt processing routine is shown below. It is necessare to store the JMP ADSE instruction at the interrupt address (0020H) of INTAD.

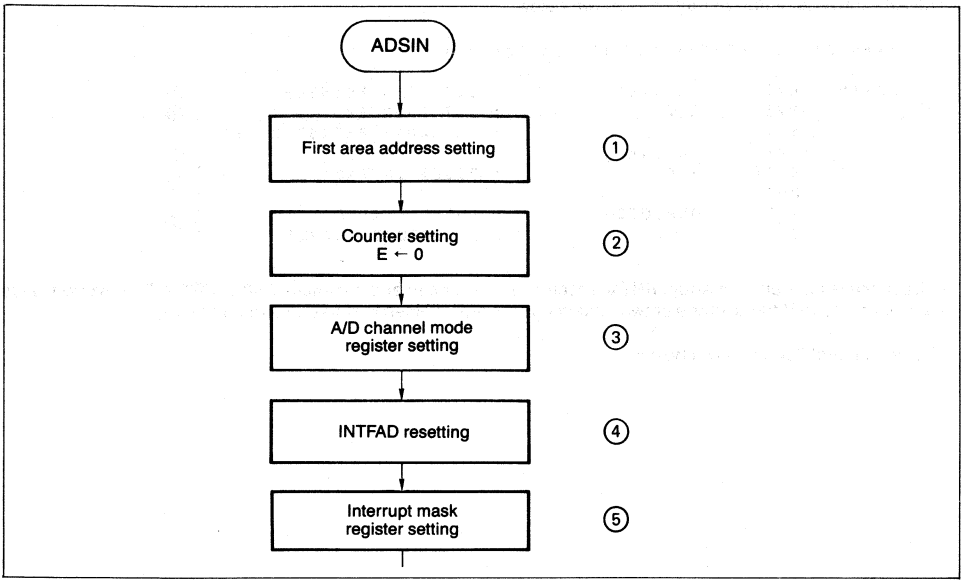
```
;*** A/D CONVERTER SERVICE ***
;
ADSE:   EXA                     ; SAVE ACCUMULATOR
        EXX                     ; SAVE REGISTER
        MOV     A,CR0
        STAX    H               ; STORE A/D CONVERSION DATA TO MEMORY
        MOV     A,CR1
        STAX    H+8H            ; STORE A/D CONVERSION DATA TO MEMORY
        MOV     A,CR2
        STAX    H+10H           ; STORE A/D CONVERSION DATA TO MEMORY
        MOV     A,CR3
        STAX    H+18H           ; STORE A/D CONVERSION DATA TO MEMORY
        DCR     B               ; DECREMENT COUNTER, SKIP IF BORROW
        JR      ARIN
        DCR     C
        JR      ARST0
        MOV     A,D
        DCR     A
        JR      ARST1
        MOV     A,E
        DCR     A
        JR      ARST2
        LXI     H,1000H         ; SET DATA POINTER
        LXI     D,0101H         ; SET COUNTER
        MVI     C,01H           ; SET COUNTER
        JR      RET1
ARIN:   INX     H               ; INCREMENT HL
        JR      RET2
ARST0:  LXI     H,1020H         ; SET DATA POINTER
        JR      RET1
ARST1:  LXI     H,1004H         ; SET DATA POINTER
        MOV     D,A
        JR      RET0
ARST2:  LXI     H,1024H         ; SET DATA POINTER
        MOV     E,A
        MVI     D,00H
RET0:   MVI     C,00H
RET1:   MVI     B,03H
        XRI     ANM,08H         ; INVERT ANI2 BIT
RET2:   EXX                     ; RECOVER REGISTER
        EXA                     ; RECOVER ACCUMULATOR
        EI                      ; ENABLE INTERRUPT
        RETI                    ; RETURN
```

① ② ③ ④ ⑤ ⑥

4.2 Select Mode Program Example

This section describes a program example for setting the A/D converter in the select mode, storing 256 (= 100H) average A/D converted values of the AN0 pin used as an analog input from a certain area, and then returning the pointer to the original area.

First, the initialize operation flow is shown below:



① The address of the A/D converted value store area is set by the HL pair register so that the converted value can be stored in the first area onwards.

② The converted value count flag (B register) is set so that the address is set in the first area once again when 256 or more converted values are stored in the first area onwards.

③ The A/D channel mode register is set to the select mode and only the AN0 pin is selected as an input pin.

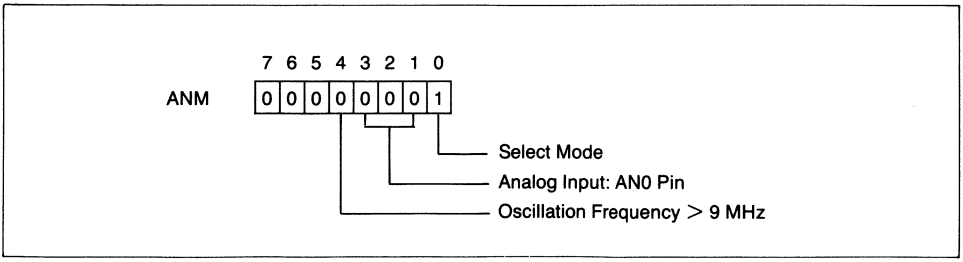


Figure 4-3 A/D Channel Mode Register Setting

④ When the A/D channel mode register is reset, it is cleared to 00H, and the AN0 to AN3 pins are A/D converted in the scan mode. The converted values may be stored into the CR0 to CR3 registers with the interrupt request flag (INTFAD) set to (1). Therefore, the MKD bit of the interrupt mask register (MKH) is set to "0" and the interrupt request flag is reset to (0) by the skip instruction before the interrupt is unmasked.

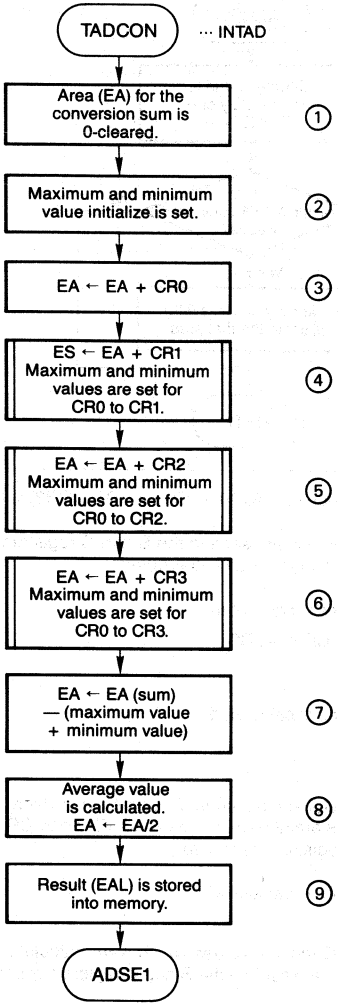
⑤ The MKAD bit of the interrupt mask register (MKH) is reset to (0) and the internal interrupt (INTAD) is unmasked.

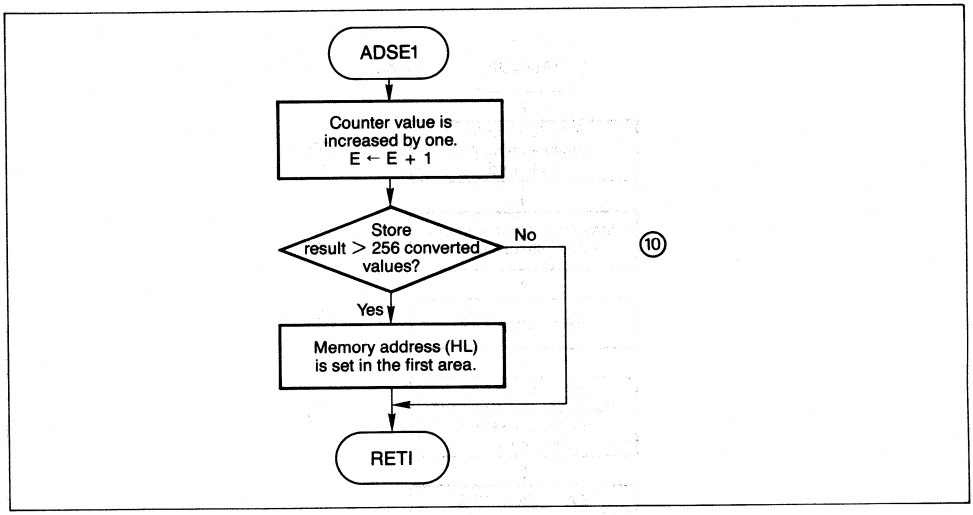
The A/D converter initialize routine is shown below:

```
;*** A/D CONVERTER INITIALIZATION ***
;
ADSIN:  LXI     H,STAD          ; SET DATA POINTER          ①
        MVI     E,0             ; SET COUNTER               ②
        EXX                     ; EXCHANGE REGISTER SET
        MVI     ANM,1H                                      ③
        SKIT    FAD             ; RESET INTFAD              }④
        NOP                                                 }
        ANI     MKH,0FEH        ; INTAD ENABLE              }⑤
        EI                      ; INTERRUPT ENABLE          }
```

If the processing routine interrupt (INTAD) occurs, the maximum and minimum CR0 to CR3 A/D converted values are invalidated and the average of two remaining converted values is stored into the memory.

The operational flow is shown below:


TADCON
··· INTAD
Area (EA) for the conversion sum is 0-cleared.
①
Maximum and minimum value initialize is set.
②
EA ← EA + CR0
③
ES ← EA + CR1
Maximum and minimum values are set for CR0 to CR1.
④
EA ← EA + CR2
Maximum and minimum values are set for CR0 to CR2.
⑤
EA ← EA + CR3
Maximum and minimum values are set for CR0 to CR3.
⑥
EA ← EA (sum)
— (maximum value + minimum value)
⑦
Average value is calculated.
EA ← EA/2
⑧
Result (EAL) is stored into memory.
⑨
ADSE1



① The EA register for the conversion sum (CR0 + CR1 +CR2 +CR3) is 0-cleared by initialization.

② The maximum and minimum values are set into the B and C registers respectively, and the CR0 converted value is loaded into the B and C registers by initialization.

③ The sum of the converted values is calculated.

EA ← (((EA + CR0) +CR1) +CR2) +CR3

"EA + CR0" is calculated.

④ The sum of the converted values is calculated.

EA ← EA + CR1

"EA + CR1" is calculated.

After that, the maximum B register and minimum C register which were set in ② are compared to CR1 and the maximum and minimum values are determined between CR0 and CR1. The maximum and minimum values are then set into the B and C registers respectively.

⑤ The sum of the converted values is calculated.

EA ← EA + CR2

After that, the B register and C register values which were calculated in ④ are compared to CR2 and the maximum and minimum values are set into the B and C registers respectively.

⑥ The sum of the converted values is calculated.

EA ← EA + CR3

After that, the B register and C register values which were calculated in ⑤ are compared to CR3 and the maximum and minimum values of the four converted values set into the B and C registers respectively.

⑦, ⑧ The B and C register values are subtracted from the sum of average values and an average of two intermediate values is calculated.

EA ← EA — B register — C register

EA ← EA/2

⑨ The result is stored into the memory.

⑩ The counter value is increased. If the result carries, more than 256 values have been stored. Thus, the memory address is set in the first area.

The interrupt processing routine is shown below.

It is necessary to set the JMP TADCON instruction at the interrupt address (20H) of INTAD.

```
;*** GET ARITHMETIC MEAN OF R0,1,2,3 ***
;
TADCON: EXA                         ; ACCUMULATOR RESTORE
        EXX                         ; REGISTER RESTORE
        LXI     EA,0                ; SUM, INITIAL 0 CLEAR          ①
        MOV     A,CR0
        MOV     B,A                 ; B REG <- INITIAL MAX          ②
        MOV     C,A                 ; C REG <- INITIAL MIN
        EADD    EA,A                ; SUM = SUM+CR0                 ③
;
;   GET CR0+CR1+CR2+CR3
;
        MOV     A,CR1
        CALL    GETM                ; 1ST GET / MAX,MIN             ④
        MOV     A,CR2
        CALL    GETM                ; 2ND GET / MAX,MIN             ⑤
        MOV     A,CR3
        CALL    GETM                ; 3RD GET / MAX,MIN             ⑥
;
;   B REG <- MAX / CREG <- MIN
;   EA <- CR0+CR1+CR2+CR3
;
        ESUB    EA,B                ; EA <- EA-MAX                  ⑦
        ESUB    EA,C                ; EA <- EA-MIN
        ;
        DSLR    EA                  ; RESULT <- RESULT/2            ⑧
        MOV     A,EAL
        STAX    H+                  ; RESULT STORE                  ⑨
        ADINC   E,1                 ; COUNTER INCREMENT             ⑩
        LXI     H,STAD
;
;   REGISTER STORE
;
        EXX                         ; REGISTER STORE
        EXA                         ; ACCUMULATOR STORE
        EI                          ; INTERRUPT ENABLE
        RETI                        ; RETURN
;
;   SUBROUTINE / GET MAX,MIN
;
GETM:   EADD    EA,A                ; TOTAL SUM <- SUM+CR0,1,2,3
        ;
        LTA     A,B                 ; CR0,1,2,3 > INITIAL MAX ?
        MOV     B,A
        ;
        LTA     C,A                 ; CR0,1,2,3 < INITIAL MIN ?
        MOV     C,A
        RET
```

Chapter 5 Interruption Control

5.1 Interruption

There are three external interrupt requests ($\overline{NMI}$, INT1 and $\overline{INT2}$), eight internal interrupt requests (INTT0, INTT1, INTE0, INTE1, INTEIN, INTAD, INTSR, and INTST), and a software interrupt instruction (SOFTI). Eleven types of interrupt requests except the SOFTI instruction are divided into six groups for 6-level priority.

Interrupt addresses for the 6-group interrupt requests and the SOFTI instruction are all fixed as described in Table 5-1.

Table 5-1 Priority and Interrupt Addresses (1/2)

Priority	Internal/ External	Interrupt Request		Interrupt Address	
				Decimal	Hexadecimal
1	External	$\overline{NMI}$	Rising edge (nonmaskable interrupt)	4	0004
2	Internal	INTT0	Coincidence signal from TIMER0	8	0008
		INTT1	Coincidence signal from TIMER1		
3	Extrnal	INT1	Rising edge	16	0010
		$\overline{INT2}$	Falling edge		
4	Internal	INTE0	Coincidence signal from the timer/event counter	24	0018
		INTE1	Coincidence signal from the timer/event counter		
5		INTEIN	Falling signal of CI pin or TO	32	0020
		INRAD	A/D converter interrupt		
6		INTSR	Serial receive interrupt	40	0028
		INTST	Serial transmit interrupt		
SOFTI instruction				96	0060

Figure 5-1 shows a general interrupt processing procedure.

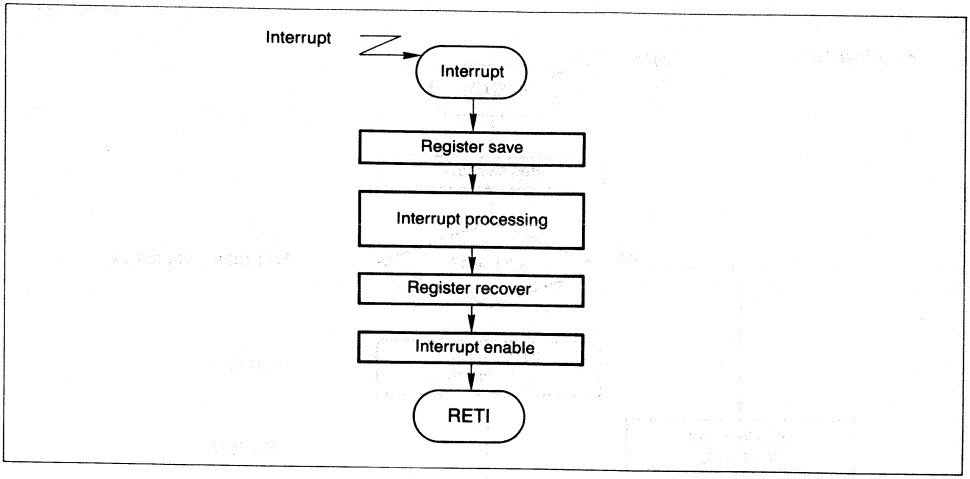


Figure 5-1 Interrupt Processing Procedure

For interrupt processing, register values used for the main processing must be saved to remain unchanged. For this purpose, ALT registers (EA', V', A', B', C', D', E', H', and L') are used. The MAIN registers (EA, V, A, B, C, D, E, H, and L) are replaced with ALT registers by EXA, EXX, and EXH instructions. Upon completion of the interrupt processing operations, the MAIN registers are replaced with the ALT registers once again.

When an interrupt is acknowledged and is branched to the interrupt address, the interrupt disable state (DI state) is automatically set by the hardware. Thus, when the interrupt processing operations are terminated, the interrupt is enabled by the EI instruction.

5.2 Selection from two Interrupts

Pairs of interrupts, INTT0 and INTT1, INT1 and $\overline{\text{INT2}}$, INTE0 and INTE1, INTEIN and INTAD, and INTSR and INTST, have the same interrupt address.

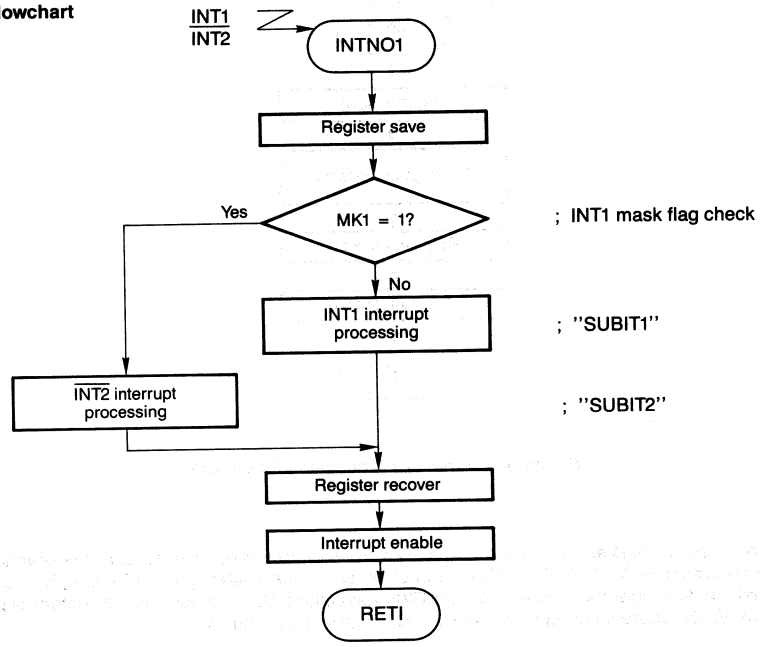
This section describes the processing of a pair of INT1 and $\overline{\text{INT2}}$ interrupts.

(1) Unmasking one of two interrupts

If an interrupt request is generated and acknowledged after one of two interrupts has been unmasked, the acknowledged interrupt request flag is reset and is branched to an interrupt address.

If the interrupt to be unmasked is fixed, only the other interrupt is used, the normal processing operation is carried out. If, however, the interrupt to be unmasked varies depending on the mode, and the respective processing operation varies, it is necessary to judge which interrupt processing operation should be carried, by checking the mask flag.

- **Flowchart**



- **Program List**

```
;
INTNO1: EXX                             ;STORE REGISTER
        EXA
        ONI     MKL,00001000B           ;CHECK! MK1
        GJMP    INTN11                  ;
        CALL    SUBIT2                  ;INT2 ROUTINE
        GJMP    INTN12
;
INTN11: CALL    SUBIT1                  ;INT1 ROUTINE
INTN12: EXA                             ;RESTORE REGISTER
        EXA
        EI                              ;INTERRUPT ENABLE
        RETI
```

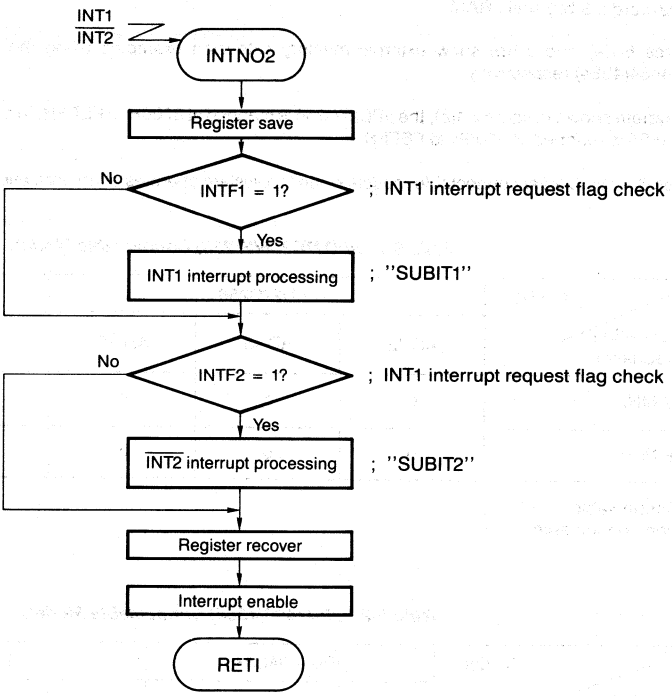
(2) Masking two interrupts

If an interrupt request is generated, acknowledged and branched to an interrupt address after both two interrupts have been unmasked, the interrupt request flag remains without being reset.

In this case, which interrupt has been generated is judged by testing the interrupt request flag by the skip instructions (SKIT and SKNIT). After the execution of the skip instructions, the tested interrupt request flag is reset.

The priority of the two interrupt processing operations is given to the interrupt request flag which was first confirmed by the skip instructions.

- **Flowchart**



- **Program list**

```
;
INTNO2: EXX                         ;STOER REGISTER
        EXA
        SKNIT   F1                  ;CHECK! INTF1
        CALL    SUBIT1              ;INT1 ROUTINE
        SKNIT   F2                  ;CHECK! INTF2
        CALL    SUBIT2              ;INT2 ROUTINE
        EXX                         ;RESTOER REGISTER
        EXA
        EI                          ;INTERRUPT ENABLE
        RETI
```

Chapter 6 Interfacing with Peripheral Elements

6.1 Memory Extension

This section describes connection examples for external extension of memories (ROM and RAM).

6.1.1 μPD27C256(27256) and μPD4464(4364) Connection Example

The μPD27C256(27256) is a 262,144-bit (32,768-word x 8-bit) EPROM. The μPD4464(4364) is a 65,536-bit (8,192-word x 8-bit) static RAM.

Figures 6-1(a) and 5-1(b) show external memory extension examples using the μPD27C256(27256) and the μPD4464(4364) respectively.

In a system shown in figure 6-1(a), the μPD27C256 is mapped at 8000H to FEFFH. In a system shown in figure 6.1(b), μPD4464 is mapped at 8000H to FEFFH.

Tables 6-1 and 6-2 list connectable models when the system operates at an oscillation frequency of 12 to 15 MHz.

Table 6-1 μPD27C256(27256) Connectable Models

Model / Oscillation Frequency	μPD27C256			μPD27256	
	AD-12	AD-15	AD-20	D	D-3
12 MHz	o	o	o	o	o
15 MHz	o	o	o	x	x

o: Connectable
x: Non-connectable

Table 6-2 μPD4464(4364) Connectable Models

Model / Oscillation Frequency	μPD4464C/G			μPD4364C(CX/G			
	-12	-15	-20	-10	-12	-15	-20*
12 MHz	o	o	o	o	o	o	o
15 MHz	o	o	o	o	o	o	o

*: Except μPD4364CX

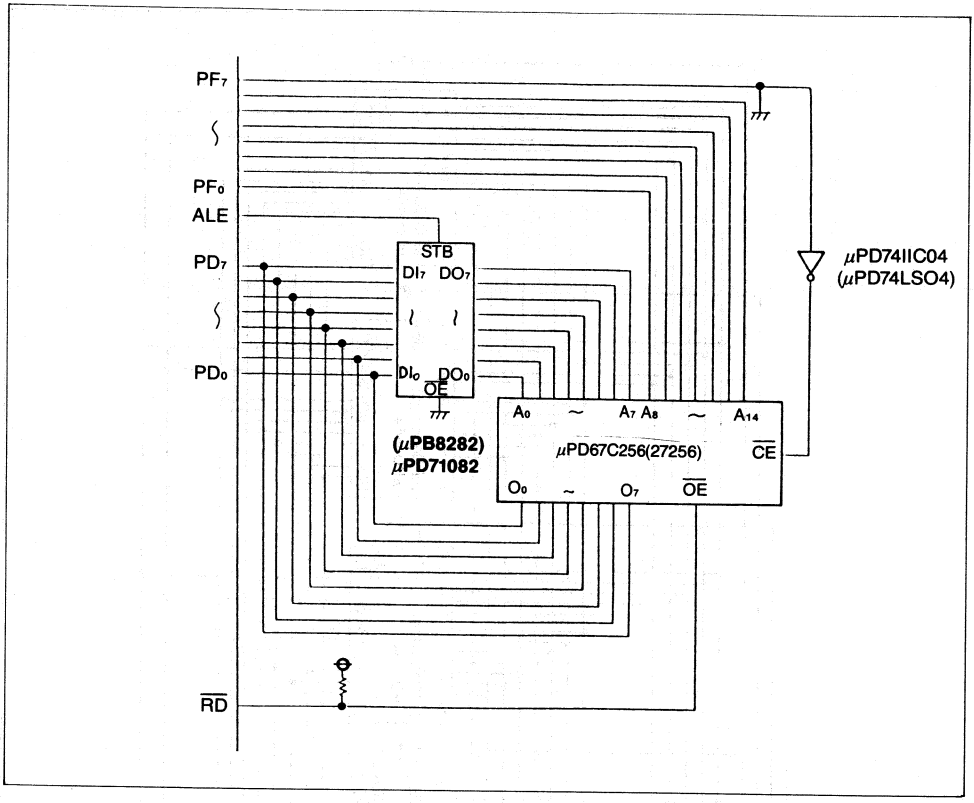


Figure 6-1(a) μPD27C256(27256) Connection Example

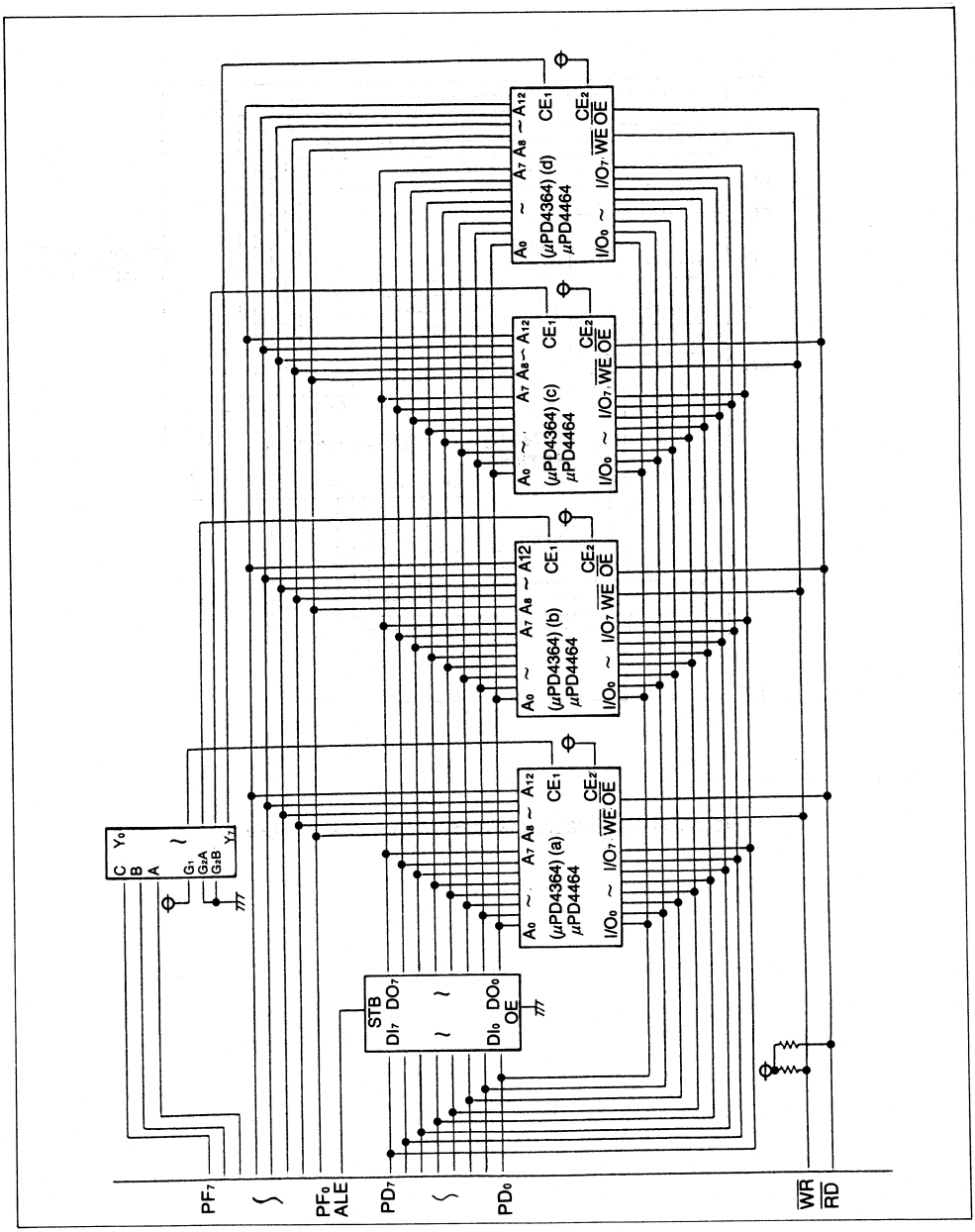


Figure 6-1(b) μPD4464(4364) Connection Example

6.1.2 μPD732008 Connection Example

The μPD732008 is a 2,097,152-bit mask programmable ROM. It has an on-chip address register and uses an address/data common bus configuration. The μPD732008 has a word format of 262,114-word x 8-bit, and is enabled for byte and nibble read.

The μPD732008 makes a ROM access via various on-chip registers instead of directly controlling memory addresses from outside. So although the μPD732008 cannot be used as a program memory for the CPU's real address space, it can be used as a device for storing a large amount of data.

Figure 6-2 shows a μPD732008 connection example.

The μPD732008 is mapped at address 2000 onwards. PF7 is an ordinary input/output port to receive the RDY (ready) signal transmitted from the μPD732008.

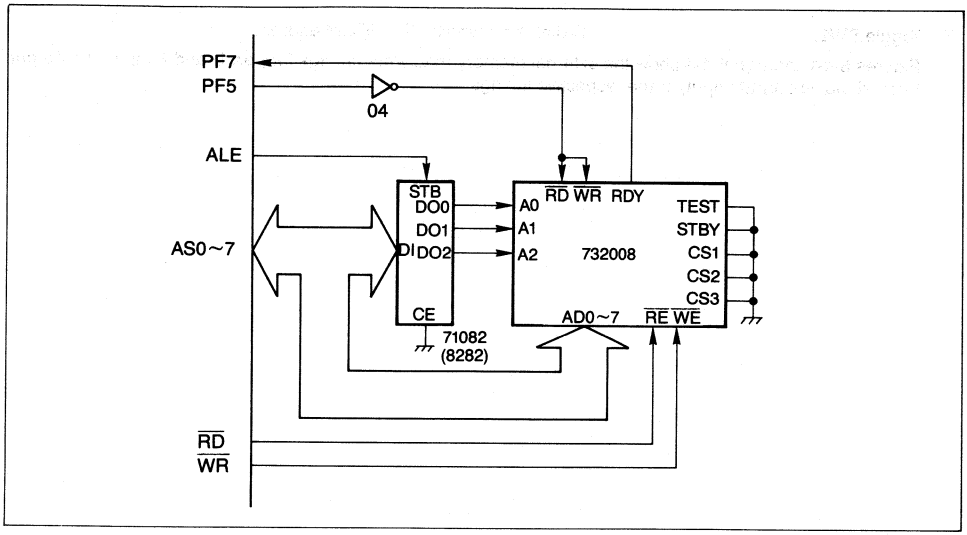


Figure 6-2 μPD732008 Connection Example

6.2 μPD78C14 Emulation Example

This section describes the μPD78C14 emulation which is carried out using the μPD78C10.

Figure 6-3 shows a reference example of an emulation circuit. This circuit consists of the following six blocks (as divided by dotted lines in the figure):

a) $\overline{IO}$/M signal decode block
b) Internal ROM block
c) Port D input/output block
d) Port F input/output block
e) Output port output enable signal generate block
f) Address data bus for external extension mode

The following four switches are used according to requirements:

① DIPSW1, DIPSW2: Input/output changeover switches for use with port F and port D serving as general-purpose input/output ports

ON : For use as an output port
OFF: For use as an input port

② Toggle SW1: Switch for adjustment of timings to validate the port output data in a high impedance state when port F and port D are used as output ports

Side A: For use with port F or Port D serving as an output port

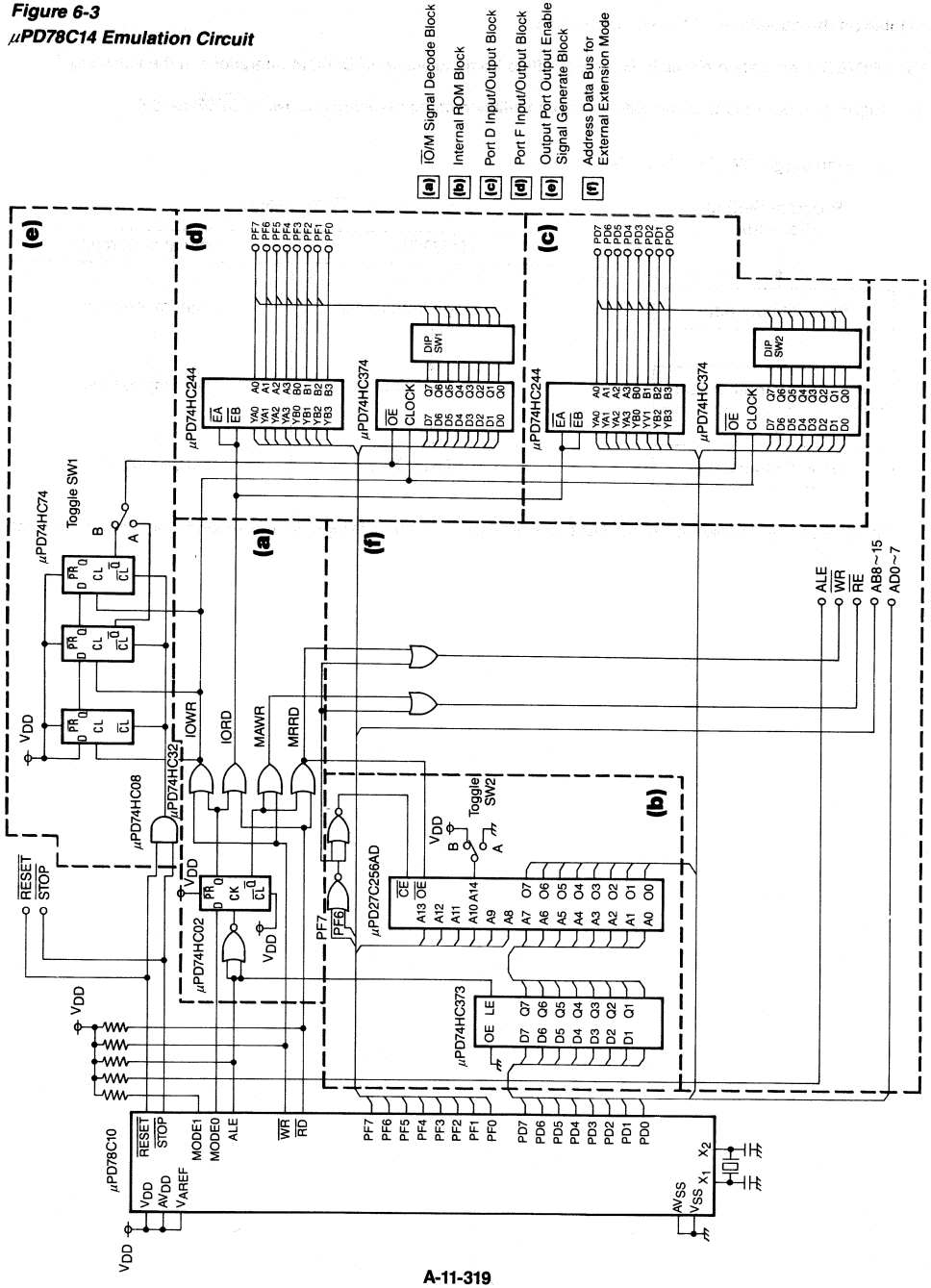
Side B: For use with both port F and port D serving as output ports

Port F can be switched for input or output for each bit. If even one bit is used as an output port, port F is set for an output port.

③ Toggle SW2: Switch for changing the PROM address

Figures 6-4-1 through 6-4-4 show the external memory read/write timings for port D and F input and output. Each signal is output (input) at the respective timings.

Figure 6-3
μPD78C14 Emulation Circuit

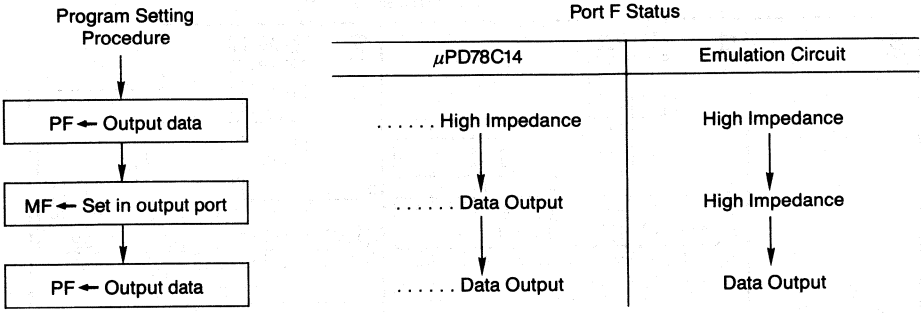


* Notes on the operations of the emulation circuit

The μPD78C14 emulation circuit in figure 6-3 differs from the actual μPD78C14 operations in the following:

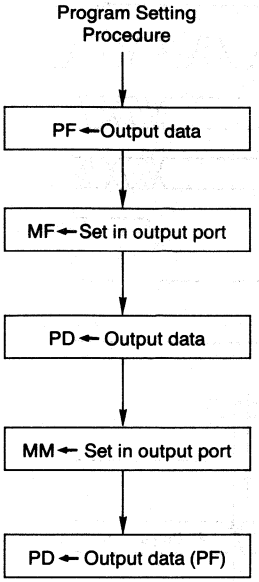
① Output port operations immediately after cancellation of the resetting (hardware $\overline{\text{STOP}}$ mode).

◎ When toggle SW1 is set to side A:



Therefore in the emulation circuit, data becomes valid only after it is input for the second time. This is the samt with port D.

ⓑ When toggle SW2 is set to side B:



Port F and D Statuses

μPD78C14	Emulation Circuit
Port F: High Impedance Port D: High Impedance	Port F: High Impedance Port D: High Impedance
↓	↓
Port F: Data Output Port D: High Impedance	
↓	
Port F: Data Output Port D: Data Output	Port F: Data Output Port D: Data Output

Therefore, in the emulation circuit, data becomes valid only after both port F and D have been set to the output mode.

② Port input and output cannot be chanced dynamically by the software.

③ Statues after resetting (hardware $\overline{\text{STOP}}$ mode)

Pin	μPD78C14	Emulation Circuit
ALE	High impedance	High level
$\overline{\text{RD}}$	High impedance	High level
$\overline{\text{WR}}$	High impedance	High level

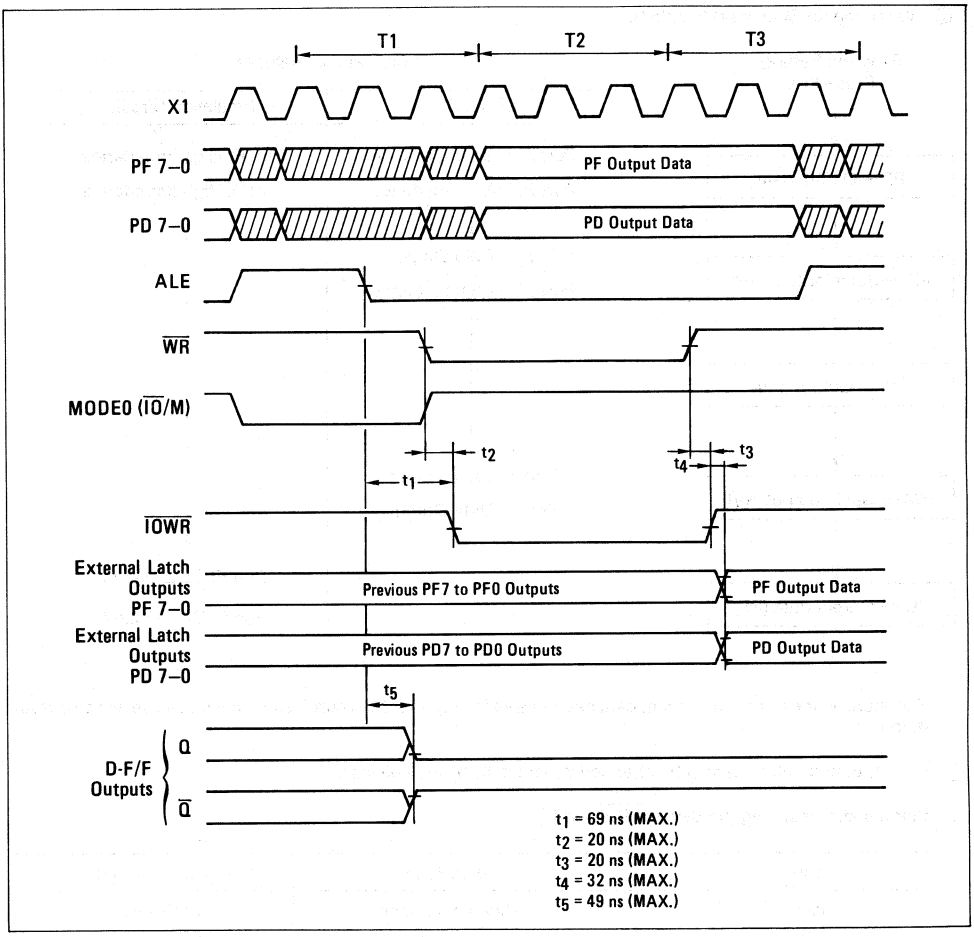


Figure 6-4-1 Port D and F output Timings (M4 Cycle for Port Output Instruction)

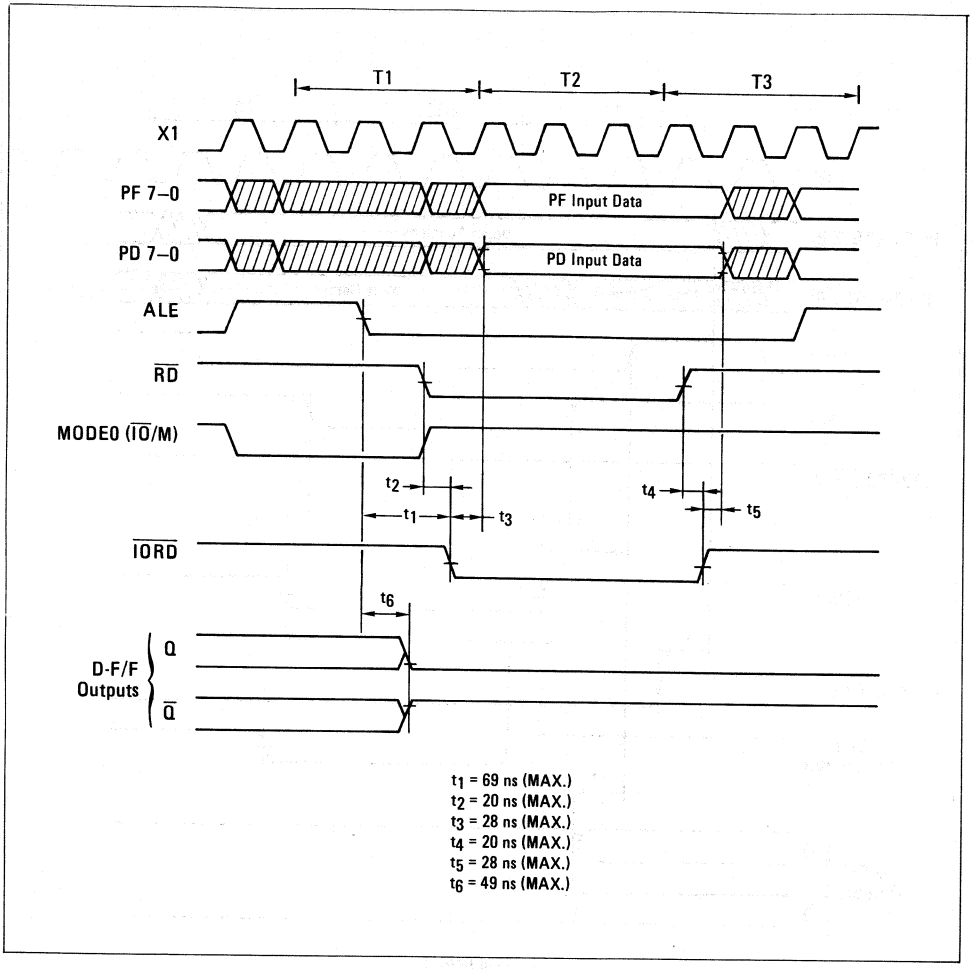


Figure 6-4-2 Port D and F Input Timings (M4 Cycle for Port Input Instruction)

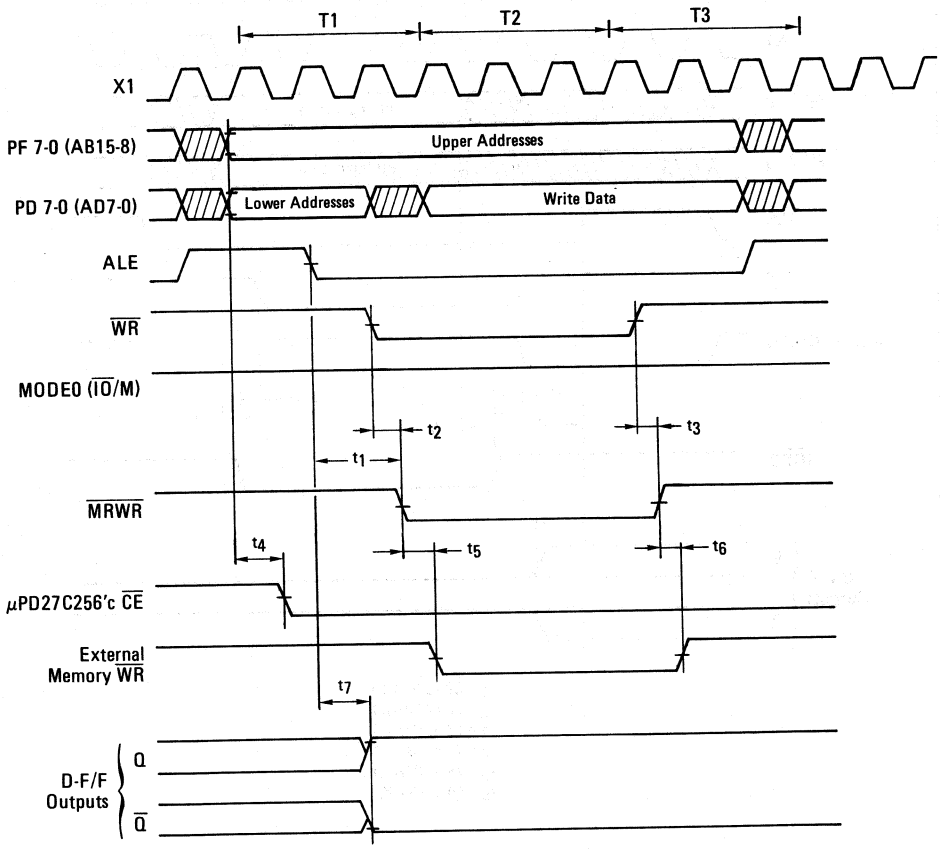


Figure 6-4-3 External Memory Write Timings

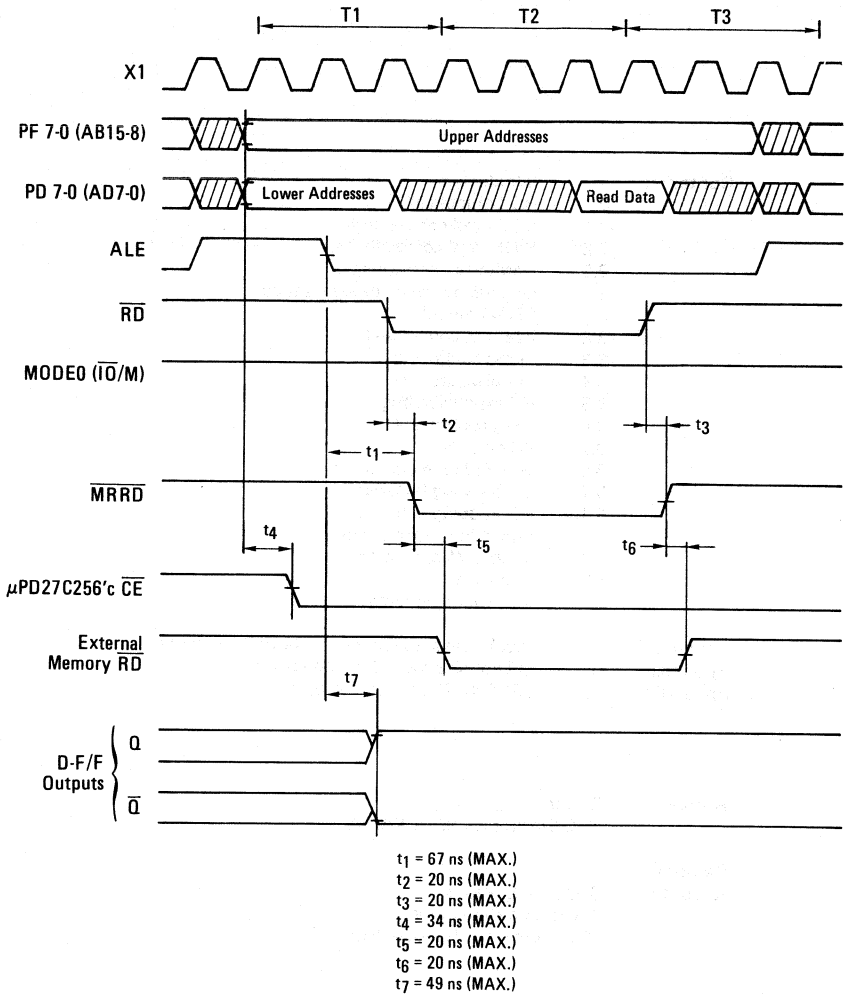


Figure 6-4-4 External Memory Read Timings

V40/V50 PC Compatibility

Contents:

Author: R. Naro
NEC Electronics NATICK/USA

Persons to contact: S. Gupta, B. Peters
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD70208/216 User's Manual
Data Book Microprocessors and Peripherals

Related Products

μPD70208 16 bit Microprocessor	CMOS
μPD70216 16 bit Microprocessor	CMOS

1. Introduction

For a variety of reasons, many new personal computer designs require compatibility with an earlier generation of personal computers, particularly the IBM PC. While these new machines can improve the price/performance ratio of a new PC design by employing the state-of-the-art V40/V50 microprocessors, software compatibility must be maintained to permit the use of the large installed base of application and system software.

The inability to run IBM PC application software has resulted in serious marketing consequences for non-PC-compatible machines and must be carefully considered prior to the introduction of a new personal computer. The μPD70208/216 (V40/V50) microprocessors present designers with the opportunity to design a low cost, low power PC-compatible. However, there are a number of design rules which must be observed to guarantee full IBM PC-compatibility.

Because the V40/V50 internal peripherals were designed to be compatible with existing discrete peripheral devices, hardware and software compatibility with most of the IBM PC motherboard peripherals is maintained. In fact, the only stumbling block to full IBM PC-compatibility is the V40/V50 DMA controller. The V40/V50 DMAU was designed to correct the problems associated with the previous generation of 8-bit DMA controllers like the μPD8237A. By providing a full 20-bit address, byte or word transfers and new operating modes, the DMAU enhances performance and minimizes external hardware. The V40/V50 DMAU is a functional superset of the μPD8237A DMA Controller but is not software compatible (but nearly so as this MicroNote will demonstrate).

A quick and easy way to guarantee PC-compatibility is by using an external μPD8237A DMA Controller. However, this approach is not cost effective in large volumes so other alternatives must be explored. This V40/V50 MicroNote assumes the addition of an external μPD8237A (and the associated logic) is unacceptable and examines an alternative approach. If a physical device such as the μPD8237A is not present in the system, a small amount of hardware and software can be employed to emulate the missing peripheral. This scheme has already been implemented in a number of PC-compatible designs and its compatibility has been verified by running hundreds of PC programs without failure.

Not all peripherals can be considered candidates for I/O emulation. For instance, of all the peripherals in the IBM PC, only the μPD765 (or the newer CMOS μPD72065) FDC (floppy disk controller) must be physically present. This requirement exists because some copy-protection schemes read data from the FDC without using DMA. These schemes require the processor to be able to read a new byte of data approximately every 13 μs (the rate each byte comes off the floppy disk). Because it is impossible to emulate a μPD765 in real-time on a V40/V50 (the interrupt and I/O emulation software overhead is just too great), the μPD765 FDC must be included in the design. None of the other IBM PC peripherals have such a strict real-time constraint and their operation can be emulated.

2. PC Compatibility

The first step is to define exactly what is PC-compatibility since there are multiple levels of PC-compatibility which can be implemented. Both application and system software written for the IBM PC interacts with the hardware at one, two, or three levels. Depending on the level of compatibility desired, the V40/V50 design can be tailored to meet the compatibility requirements at minimal cost.

2.1 DOS Level Compatibility

The first level is the PC-DOS (here in referred to as DOS) interface. Once DOS is brought up on a machine, it provides a consistent interface between application software and the BIOS software. Because these programs make no assumptions about the target hardware, software that only interacts with DOS can be run on any machine regardless of the underlying hrdware implementation. Software falling into this class would be the generic MS-DOS programs.

2.2 BIOS Level Compatibility

The next interface level is the BIOS (Basic Input-Output System) and is also easy to implement. If a design uses an IBM PC-compatible BIOS, software that restricts itself to DOS and BIOS system calls will be completely compatible. Like DOS, the BIOS hides the underlying hardware implementation from the application. The presence of a PC-compatible BIOS can usually be determined by observing if PC-DOS can boot on the machine.

A number of application programs bypass DOS and work directly with the BIOS. These applications do so to increase performance yet still exhibit a high degree of portability due to the length manufacturers go to keep their BIOS compatible with that of the IBM PC.

2.3 Hardware Compatibility

It is unfortunate but true that a great many programs bypass both the DOS interface and BIOS interface and work directly with physical I/O devices to either increase performance or to implement a software protection scheme. The PC BIOS is an incomplete implementation and is often unbearably slow, particularly in screen operations. As a result, many applications install their own screen drivers to bypass this problem and interact directly with the hardware.

While the application may restrict itself to DOS and BIOS calls, it is invariably true that copy protected software will attempt to verify a legitimate copy of the software by directly reading and writing physical I/O devices in the IBM PC. Thus it is at this level that the software will fail to run on a V40/V50 design relying on BIOS level compatibility.

Not all I/O port references cause problems. The video, hard and floppy diskports reside in expansion bus slots and are fully compatible. The problem typically lies in the implementation of the IBM PC motherboard peripherals and whether a V40 or V50 is used in the design. To identify solutions to these incompatibilities, one must examine the role of each peripheral on the motherboard. The peripherals in question are the interrupt controller, timer/counters and DMA controller.

3. General Hardware Design Issues

The V40/V50 make life much simpler for the designer of an IBM PC-compatible system. Because all of the LSI peripherals such as DMA controller, interrupt controller, refresh controller and timer/counters are placed on-chip with the CPU, the size and complexity of the design are greatly reduced. Furthermore, the glue logic associated with these functions is eleminated. Each of these hardware design areas are explored in further detail below.

3.1 DRAM Refresh

DRAM refresh in the IBM PC is accomplished by using timer/counter 1, DMA channel 0 and a large number of SSI devices. Because the V40/V50 contains a built-in refresh controller, the need to dedicate other peripherals to the memory refresh function is eliminated. However, two distinct areas of DRAM refresh must be considered since both the motherboard and the expansion chassis can contain dynamic memory devices.

The hardware design to refresh memory devices on the motherboard is simplified because only the RAS/CAS and address multiplexing logic must be added to interface the V40/V50 to low-cost DRAMs. A simpe RAS-only refresh of all motherboard memory devices is accomplished by using the REFRQ* output to disable CAS* and enable all RAS* outputs. On a motherboard with a single bank of 256K DRAMs, the hardware interface is further simplified since the need to enable all RAS* and disable all CAS* inputs is eliminated.

Memory in an expansion slot performs a refresh cycle whenever the motherboard drives the DACK0* expansion bus pin low. A V40/V50 design can accomplish this by simply routing the REFRQ* output to DACK0* expansion bus pin. This approach also has the advantage of leaving three DMA channels unused, the correct number for a PC-compatible design yet permits the use of the V40/V50 SCU as a communication channel or manufacturing diagnostic port if desired.

3.2 Interrupt Control Unit

The ICU (interrupt control unit) is a subset of the μPD71059 Interrupt Controller and is software compatible with the μPD8259A used in the IBM PC. All seven external interrupts (keyboard, floppy disk, hard disk, etc. . . .) are connected identically as in the PC. Interrupt input INT0 is hard-wired internally to timer/counter 0 but since this corresponds to the real-time clock interrupt, it is completely compatible.

The V40 can place the ICU at the same physical I/O address by programming the IULA to 20H and OPHA to 00H. Refer to the section on ICU/TCU emulation for making the V50 ICU PC-compatible.

3.3 Timer Control Unit

The TCU (timer control unit) is a subset of the μPD8253 timer/counter and is software compatible with the μPD8253 used in the IBM PC. It is made compatible by setting the TULA register to 40H and OPHA register to 00H. Refer to the section on ICU/TCU emulation for making the V50 TCU PC-compatible. The clock sources for each timer/counter should be selected from the TCLK input. This allows a PC-compatible clock to be used for each timer/counter.

In the IBM PC, the output of timer/counter 0 is used as a real-time clock and is connected to INT0. This connection is hard-wired internally in the V40/V50. The IBM PC uses timer/counter 1 is used as a refresh timer to generate memory refresh requests. Because the V40/V50 has an internal refresh controller, the need to utilize an timer/counter and DMA channel is eliminated. Thus timer/counter 1 is free to be used as the baud rate generator for the Serial Control Unit or as a general purpose output. Timer/counter 2 is the audio oscillator and drives the speaker. It is completely software compatible with the IBM PC.

3.4 Serial Control Unit

Because memory refresh is performed by the RCU rather than a DMA channel in a V40/V50 system, this leaves an additional DMA channel free or allows the use of the SCU (serial control unit). This allows a PC-compatible design to add a dedicated SCSI interface or use the SCU as serial channel for manufacturing diagnostics.

The IBM PC uses the 8250 serial controller which differs from the internal SCU in a number or ways, but a BIOS-compatible interface can be implemented using the SCU. This would allow a standard serial device to be attached with no additional cost. However, note that many communication programs directly drive the physical COM1/COM2 ports and are not software compatible with the SCU.

3.5 PC Expansion Bus

One other compatibility issue must also be addressed. The V40/V50 is fully compatible with the expansion bus in the IBM PC with the exception of the OSC signal. Since the clock generator has moved inside the V40/V50, designs must either ignore or implement this signal.

The option to ignore this signal is viable if no expansion bus is required (as in a laptop PC). Also, very few PC expansion bus cards actually use this signal (the exceptions being the color graphic adapter cards) so it can often be left out of the design and still work with the vast majority of peripheral boards.

The final option is to use the approach taken by IBM in the PC/AT. When IBM introduced the PC-AT, it provided the OSC signal for compatibility but restricted its use for synchronization only. This implies that boards designed to use both the PC and the PC/AT buses will remain compatible. In this case, a μPD71084 clock generator is used to generate the OSC signal and drive the V40/V50 if desired. A sample design using the μPD71084 clock generator is shown in figure 1. This approach has the advantage that the V40/V50 can use the OSC output of the μPD71084 as an external clock source. The V40/V50 divides this clock by two to obtain a 7.14 MHz CPU clock, or turbo mode. In order to emulate the 4.77 MHz IBM PC clock, the design can program the WCU (wait control unit) to insert three wait states in every bus cycle. Benchmarks run on this system show that the instruction throughput is equivalent to the stock IBM PC.

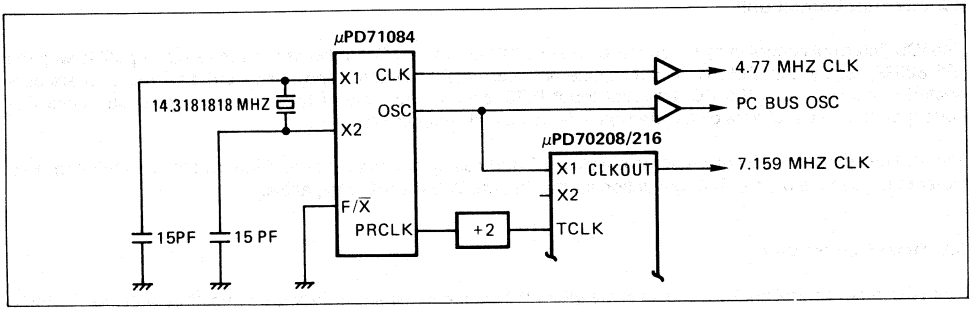


Figure 1. IBM PC Compatible Clock Generation

3.6 DMA Control Unit

The only real incompatibility is the V40/V50 DMA controller. However, the incompatibility is strictly a software problem and the functional and electrical characteristics of the V40/V50 DMAU are compatible with the μPD8237A.

DMA operations in the PC occur in two I/O address ranges. Addresses 00H—0FH re used by the μPD8237A while addresses 80H—83H are used by the 4 x 4 register file containing the upper 4 bits of the 20-bit address. Because the PC does not fully decode the motherboard peripheral I/O addresses, the I/O trapping ranges should be extended to 00H—1FH and 80H—9FH.

The way to deal with incompatibility of the internal DMA Unit is to use the fact that the V40/V50 will not begin the execution of the next instruction until the previous I/O bus cycle has completed. External hardware can exploit this by asserting NMI and disabling READY whenever an I/O reference occurs. The details of I/O emulation of the μPD8237A and other peripherals is described in the next section.

3.7 I/O Emulation

The remainder of this application note is devoted to a discussion of I/O emulation. I/O emulation requires that external logic intercept I/O bus cycles of interest and emulate the non-existent peripherals using the on-chip peripherals and a small amount of software. The μPD70208 (V40) is completely software compatible with the exception of the on-chip DMA controller. The μPD70216 (V50) requires some additional I/O emulation software to emulate the IBM PC μPD8253 and μPD8259A I/O port addresses. These differences are described in the next section.

3.8 V40/V50 Differences

Note that because of differences in bus structure, internal peripherals behave slightly differently in the V40 and V50. Three of the low address registers (SULA, IULA, TULA) in the V40 map the internal peripherals onto consecutive addresses (address lines A1—A0 select a register within a peripheral) while the V50 uses A0 as a bus select (upper/lower byte of the 16-bit data bus) and address lines A2—A1 contain the internal register address. This implies the V50 TCU and ICU cannot be mapped to the same I/O port addresses as used in the IBM PC. Designs using the μPD8086 or V30 could correct this problem by adding an external transceiver but because this solution is unavailable to V50 designs, the I/O emulation approach must be extended to these peripherals.

The decision to use I/O trapping for the translation of μPD8237A operations into DMAU operations can also be used to advantage to solve the ICU and TCU addressing problem. In this case, the external logic can be designed to also trap references to the IBM PC μPD8352 and μPD8259A devices. By locating the internal peripherals in an unused page in the I/O address space, the address translation software can provide complete software compatibility. The new physical I/O address can be computed by simply shifting left the lower two bits of the I/O port address and adding in the offset (the contents of the OPHA and TULA/IULA registers).

3.9 I/O Trapping Hardware

The way to deal with the incompatibility of the internal DMA controller or other internal peripherals is to use the fact that the V40/V50 will not begin the execution of the next instruction until the I/O bus cycle has completed. External hardware can exploit this by asserting NMI and disabling READY whenever a match occurs. READY must be negated for at least two clocks to meet the five clock NMI setup time and guarantee recognition. A timing diagram for the READY and NMI inputs is shown in figure 2.

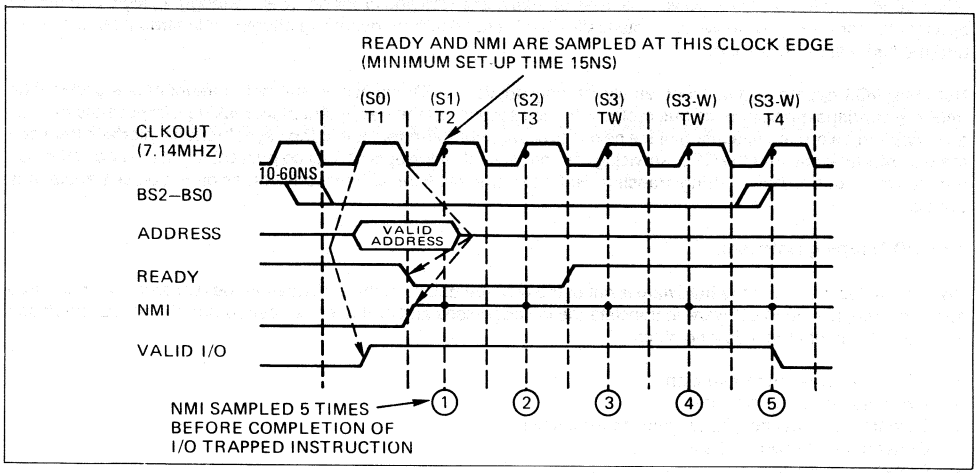


Figure 2. NMI, READY Timing

The I/O trapping hardware is implemented using a pair of low-cost PALs or a small amount of external logic. Note that the I/O trapping logic uses the bus status BS2—BS0 outputs to determine the start of a bus cycle and to distinguish DMA bus cycles (which assert the IORD/IOWR outputs and a 20-bit address) from ordinary I/O bus cycles. The BS2—BS0 outputs reflect the state of the memory strobes (MRD/MWR) and prevent DMA bus cycles from inadvertently causing an I/O trap. Figure 3 shows a sample PAL design for the I/O trapping locig.

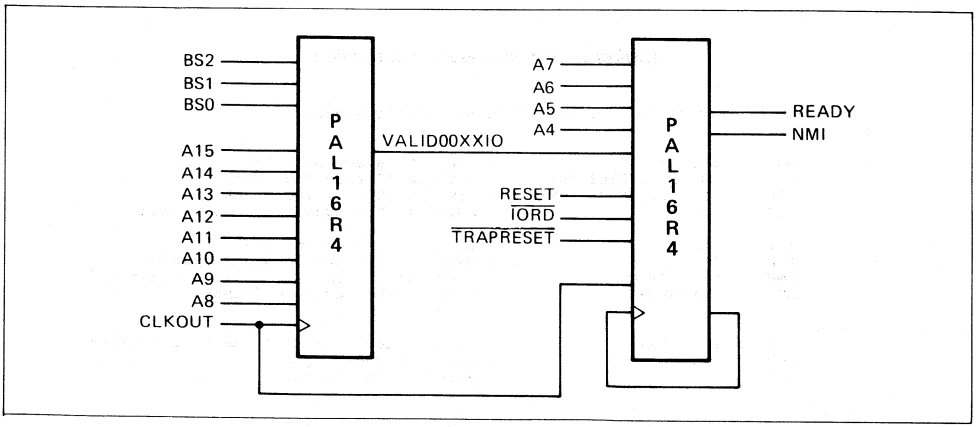


Figure 3. I/O Trapping Hardware Design

The location of the PS and PC for the NMI vector is also very important. Although a design can use the RAM-based NMI vector at address 0000H:0008H, its use is risky since a reliable design cannot risk software accidentally or intentionally modifying the NMI vector. A better solution is to keep a second vector to the I/O trap handler in ROM to be used in the servicing of I/O traps. Because the I/O trapping logic is implemented on the motherboard, the I/O trap signal can be used to select which NMI vector will be used in the servicing of non-maskable interrupts. In addition to being used as a non-maskable interrupt input, the trap output from the I/O trapping logic is used by the memory chip select logic to disable the NMI vector read from RAM and instead to enable the vector to be read from offset 0008H within the BIOS ROM. This frees up use of the NMI vector for hardware debugging logic and use by application software and prevents a potentially deadly situation from occurring should an I/O trap interrupt occur and the NMI vector be invalid.

Note that I/O trapping and software emulation only applies to I/O bus cycles and not to memory bus cycles. The V40/V50 overlaps memory write bus cycles with the execution of the next instruction, making it impossible to force an exception before the execution of the next instruction. Furthermore, due to the asynchronous interface between the bus interface unit and the CPU, servicing of non-maskable interrupts cannot be guaranteed at the end of the current bus cycle during block (string) transfers. Refer to Appendix A for NMI response times during block transfer bus cycles.

3.10 I/O Trapping Software

Now that a description of the hardware is out of the way, the details of the software can be explored. Once the trap has occurred, it is up to software to translate the I/O addresses and emulate the desired peripheral. I/O emulation is broken down into the following steps:

- I/O port address determination
- Operand size determination
- Data transfer direction (read/write) determination
- Convert the I/O port address
- Convert the I/O port data

On entry to the I/O trap handler, the stack contains the program counter and program segment values for the instruction immediately following the I/O instruction. Now there are two types of I/O instructions. The one byte form (opcodes ECH, EDH, EEH, EFH) specify the I/O port address in the DW register while the two byte form (opcodes E4H, E5H, E6H, E7H) specify a zero extended byte I/O port address in the instruction. Because there are no IBM PC I/O ports defined in the range E0H—EFH (they are reserved), it is simple for the software in listing 1 to determine the I/O port. Note that this code fragment assumes the registers have been previously saved upon entry into the trap handler.

Listing 1. I/O Port Address Extraction

```
;
; Assumes all registers used have been previously saved.
;

        MOV     BP, SP                  ; Copy the stack pointer
        MOV     IX, PC_OFFSET[BP]       ; Get the PC from the stack
        MOV     DS0, PS_OFFSET[BP]      ; Get the PS from the stack
        DEC     IX                      ; Point to the previous instruction

        MOV     DL, [IX]                ; Get an instruction byte
        AND     DL, 0F8H                ; Mask off the unneeded information
        CMP     DL, 0E8H                ; Check if variable or fixed addressing
        BE      VARIABLE                ; Process the variable format

FIXED:
        MOV     DL, [IX]                ; Get the lower byte of the port address
        MOV     DH, 0                   ; Zero the upper byte
        DEC     IX                      ; Point to the opcode

VARIABLE:
        MOV     BL, [IX]                ; Get the opcode in BL
        ...                             ; Opcode in BL, port address in DW
```

The next two steps go hand in hand. Once the I/O virtual port address has been located, the size (byte/word) and the direction (IN/OUT) are determined directly from the opcode byte. This information will be used later to re-execute the trapped instruction.

The next step involves converting the virtual I/O port address to a physical I/O port address. As shown in Listing 2, this can be accomplished by a simple computation of the V50 ICU and TCU or by a lookup table in the case of the μPD8237A or page register file.

Listing 2. I/O Port Address Translation

```
;
; Convert the μPD8253 register address to the equivalent TCU address
; The DW register is assumed to contain the μPD8253 address
; Only the base address must be changed for ICU emulation
;

        AND     DW, 11B                 ; Isolate the μPD8253 register address
        SHL     DW, 1                   ; Multiply by two for V50 addressing
        ADD     DW, TCU_BASE            ; Add in the base address of the TCU
        ...                             ; DW now contains the translated address
```

Note that in the case of an I/O write instruction, the AW/AL register contains the output data and no special adjustment is required. The case of an I/O read is a little trickier since upon exiting, the trap handler must setup the AW/AL register with the read data from the emulated instruction so thenext instruction is unaware that an I/O trap even took place.

4. Emulation Performance

What is the impact of I/O emulation on throughput, specifically on an I/O-bound program. Before one begins to worry about a performance degradation due to I/O trapping, keep in mind that the change in performance is the product of the change in execution time for the emulated instruction and the frequency of the trapped I/O instructions. Because the frequency of trapped I/O instructions is very low (on the order of one per 10^3—10^5 instructions executed) and the emulation is on the order of 10^2 slower, the actual change in throughput is insignificant and unobservable to the end-user.

4.1 μPD8237A Emulation

At this point, all of the software has been described except for the details of the μPD8237A emulation. This section will describe the internal organization of the μPD8237A and describe how the emultion software operates. A complete listing of the source code for the μPD8237A is provided in Appendix C.

In the V40/V50, the internal DMAU replaces both the μPD8237A and the 74LS670 4 x 4 register file. Thus accesses to both of these devices must be handled by the emulation software. Also, since the RCU is being used as the DRAM refresh controller and it is assumed that the SCU is also enabled, the emulation software must map references to μPD8237A channels [1, 2, 3] onto the DMAU channels [0, 1, 2] respectively.

The most significant difference is that many of the μPD8237A registers are accessed as a state machine and require two operations to the same address to complete an operation. Internally, the μPD8237A uses a First/Last F/F to maintain the current state and provides a command to clear the flip-flop to force a known state. By emulating the First/Last F/F in software, the μPD8237A emulation can determine which register to read or write. To avoid allocating memory for state information, the state of the Frist/Last F/F can be stored in an unused internal register, such as bit 3 in the DMA MASK register.

Let's examine each of the μPD8237A internal registers in detail.

Address/Count Registers

The μPD8237A address registers are similar to internal DMAU except that they are only 16-bit wide. Writes to the page register are mapped onto the upper four bits of the internal DMAU address register by the emulation software. The page register I/O port address can be used to determine which of the upper address registers must be selected.

The μPD8237A count register are similar to internal DMAU and differ only in the addresses assigned to the registers. No change to the count value is required. In the μPD8237A, writes to address/count registers update both base and current registers. The V40/V50 DMAU can be programmed to operate in this mode by clearing the BASE bit in the DCH register during the initialization of the DMAU.

Command Register

In the μPD8237A, the format of the Command register is identical to the DMAU DDC register which simplifies the emulation of this register. By initializing the upper byte of the DDC once during system initialization, only address translation is required to write the μPD8237A Command byte into the DMAU DDC register (lower byte).

Note that there is no need to emulate memory-memory transfers. The IBM PC uses DMA Channel 0 for memory refresh, prohibiting its use in memory-memory transfers. Also note that request/acknowledge polarities match those in the IBM PC and require no special handling.

Mode Register

The format of the μPD8237A Mode register is identical to the DMAU DMD register except that the lower two bits of the mode byte are used to select the channel in the μPD8237A. In this case the procedure is to use to lower two bits to select the proper channel in the DCH register, zero out the lower two bits in the mode byte and write the mode byte to the DMD register.

Request Register

The DMAU has no request register for software DMA requests. However, the request register is only used to initiate a memory-memory transfer and because the PC hardware restricts this mode, not PC software requires the existence and it can be safely ignored.

Mask Register

The mask registers in the μPD8237A and the DMAU differ slightly in operation. The DMAU mask register does not offer the capability to write a single mask bit, requiring that the software emulation convert μPD8237A bit mask commands to a read-modify-write sequence. The other mode (writing all mask bits simultaneously) is fully supported by a simple address translation.

Status Register

The status register is completely compatible and requires only address translation.

Temporary Register

No implementation required. This register is used during memory-memory transfers to hold the read data while performing the memory write bus cycle.

Software Commands

Three software commands are provided to clear the First/Last F/F, reset the μPD8237A and clear the mask register. These commands are all implemented with software routines.

5. Summary

The incompatibility of the V40/V50 DMAU and the μPD8237A does not prevent an IBM PC-compatible design from using the V40/V50. In fact, the V40/V50 provides the high performance, low power and high level of integration required of new PC-compatible designs yet maintains full compatibility with PC software with a minimum of external logic.

6. Acknowledgements

Many thanks to Jai Choi of JC Information Systems for providing the source code for the μPD8237A emulation and for testing out many of the concepts presented in this MicroNote. The JC Lips is one of the first IBM PC-compatibles designed using these techniques and is currently available. Contact JC Information Systems for complete information.

The following list of vendors also sell custom IC's designed for the V40/V50 or BIOS software that support the V40/V50 in IBM PC-compatible designs.

Customs IC's

VADEM has two custom IC's which implement the I/O trapping logic along with the remaining IBM PC motherboard logic. VADEM also has a custom BIOS which includes the I/O trapping software and has been tested to verify PC-compatibility.

Appendix A — Block Transfer Instruction Interrupt Latencies (String Bus Cycles)

Instruction	IX	IY	String Cycles
MOVBKW	Even	Even	2
	Odd	Odd	1
	Even	Odd	2
	Odd	Even	2
MOVBKB	—	—	2
CMPBKW	Even	Even	2
	Odd	Odd	1
	Even	Odd	2
	Odd	Even	2
CMPBKB	—	—	2
CMPMW		—	1
CMPMB		—	1
LDMW		—	1
LDMB		—	1
STMW		Even	3
		Odd	2
STMB		—	3

Appendix B — I/O Trapping Logic Documentation

A simple circuit which monitors accesses to the non-IBM PC-compatible peripherals can be constructed using two inexpensive PALs (figure B-2). The circuit takes advantage of the fact that the V40/V50 does not begin execution of the instruction following an I/O instruction until the I/O bus cycle has completed. To trap accesses to non-compatible peripherals it is necessary to monitor which I/O location is being accessed and guarantee that a NMI is recognized by the V40/V50 before the completion of the I/O bus cycle (NMI must be sampled by the rising edge of CLKOUT five consecutive times in order ot guarantee its recognition).

The two PALs contain the necessary logic to trap I/O accesses. The first PAL (IC1) monitors the bus status BS2—BS0 outputs to decode when an I/O bus cycle is taking place along with the upper byte of the I/O address bus (A15—A8). When an I/O access in the lower page of the I/O address space is detected, the VALID00XXIO signal will be asserted. The second PAL (IC2) monitors the VALID00XXIO signal and further decodes the address bus in order to determine accesses to the μPD8237A DMA Controller (000XH), DMA Page Register (008XH), μPD8253 Timer/Counter (004XH) and the μPD8259A Interrupt Controller (002XH). When an access to one of these peripherals takes place, the state machine within IC2 will leave state S0 (the idle state) and enter state S1.

Upon entering S1, the READY signal will be negated and the NMI signal will be asserted. The NMI signal from the PAL is connected to the NMI input of the V40/V50 and to the memory decoding logic. The decoding logic uses the NMI signal to force the NMI interrupt vector to be read from the BIOS ROM instead of from RAM. The PAL will continue to negate the READY signal in S2 causing two wait states to be asserted in the I/O bus cycle. Stetching the I/O bus cycle will guarantee that the NMI input is sampled for five consecutive CLKOUT cycles. This will guarantee that the V40/V50 recognizes the assertion of NMI before the completion of the I/O bus cycle.

The state machine will continue its state transitions until S3 is reached. Upon entering state S3, IC2 continuously samples the TRAPRESET input. In the meantime, the NMI output is then used to enable the ROM which contains the vector to the NMI interrupt handler for the peripheral emulation software. Once the I/O trap handler is entered, the NMI is cleared by simply asserting the TRAPRESET signal. This can be accomplished using a spare chip select or by using the unused terms in the I/O trap PALs.

Note that this scheme requires the BIOS ROM to be as fast as the interrupt vector table RAM. Because the V40/V50 believes the NMI vector access is occurring in the lower 1K of the physical address space, the contents of the LMW field in the WCY1 register is used to determine the number of wait states to be inserted into each bus cycle. This implies that the BIOS ROM read access time cannot be greater than the access time of the RAM. Otherwise, external READY logic must be employed to lengthen the interrupt vector bus cycles to meet the ROM read access times.

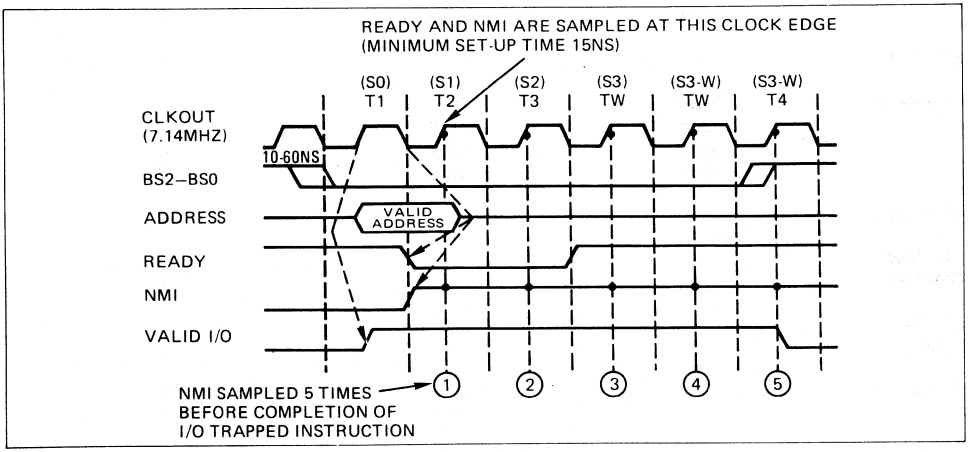


Figure B-1. Timing Waveforms

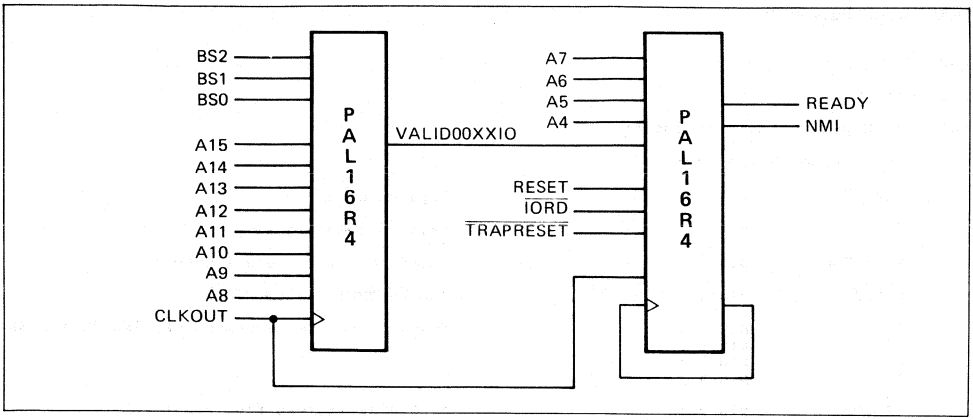


Figure B-2. Hardware Schematic

PAL1 Logic Equations

```
module bus_monitor

flag '-r2';

title 'bus activity monitor for IBM PC I/O trap logic'

        IC1 device 'P16R4' ;

"Pin connections

        CLKOUT      pin 1 ;                          "System clock

        BS2         pin 2 ;                          "Bus status lines
        BS1         pin 3 ;
        BS0         pin 4 ;

        A15         pin 5 ;                          "Upper I/O address
        A14         pin 6 ;
        A13         pin 7 ;
        A12         pin 8 ;
        A11         pin 9 ;
        A10         pin 12 ;
        A9          pin 13 ;
        A8          pin 18 ;

        OE          pin 11 ;                         "PAL output enable

        VALID       pin 17 ;

        CK, X, Z, H, L = .C., .X., .Z., 1, 0 ;

        BusCycle = [BS2, BS1, BS0] ;

        IORD = [L, L, H] ;                           "IORD bus status

        IOWR = [L, H, L] ;                           "IOWR bus status

        ADDRESS = [A15, A14, A13, A12, A11, A10, A9, A8] ;


equations

        VALID := !((!BS2 & !BS1 & BS0 & !A15 & !A14 & !A13 & !A12 & !A11 & !A10 & !A9 & !A8)
        +(!BS2 & BS1 & !BS0 & !A15 & !A14 & !A13 & !A12 & !A11 & !A10 & !A9 & !A8)) ;

end bus_monitor
```

PAL2 Logic Equations

```
module  iotrap

flag '-r2' ;

title   'IBM peripheral trap logic'

        IC2 device 'P16R4' ;

        CLKOUT          pin 1 ;                         "system clock

        IORD            pin 2 ;                         "system IORD signal

        TRAPRESET       pin 3 ;                         "used to clear the NMI output

        RESET           pin 4 ;                         "up to 8MHZ System Clock

        A7              pin 5 ;                         "used to decode addresses 0X, 2X, 4X and 8X
        A6              pin 6 ;
        A5              pin 7 ;
        A4              pin 8 ;

        VALID           pin 9 ;                         "I/O 00xx address from other PAL

        OE              pin 11 ;                        "OE connected to ground

        CLKIN           pin 12 ;

        INVCLKIN        pin 13 ;

        C0              pin 16 = 'feed_reg' ;

        READY           pin 15 = 'feed_reg' ;           "generate wait state for recognition of NMI

        NMI             pin 14 ;                        "non maskable interrupt

        NMIOUT          pin 19 ;                        "READY output to V40/V50.

        CK, X, Z, H, L = .C., .X., .Z., 1, 0 ;


"Output States
"Outputs are listed as follows: C0, READY, NMI

        S0 = ^b111 ;                                    "Idle State and Reset condition
        S1 = ^b000 ;                                    "Negate READY and assert NMI(C0)
        S2 = ^b100 ;                                    "READY negated, NMI asserted(C1)
        S3 = ^b010 ;                                    "Assert READY (Count2) two wait states inserted

"Modes

        Mode = [A7, A6, A5, A4, VALID, IORD, TRAPRESET, RESET] ;

        CLEARTRAP = [X, X, X, X, X, X, 0, 0] ;          "Switch back to normal NMI interrupt vector

        WAITING = [X, X, X, X, 0, X, 1, 0] ;            "Waiting for I/O command

        DONTCARE = [X, X, X, X, X, X, 1, X] ;

        CLEAR = [X, X, X, X, X, X, X, 1] ;              "Reset Condition

        TRAPICU2X = [0, 0, 1, 0, 0, X, 1, 0] ;          "Trap reads to peripherals
        TRAPTCU4X = [0, 1, 0, 0, 0, X, 1, 0] ;
        TRAPDMA0X = [0, 0, 0, 0, 0, X, 1, 0] ;
        TRAPREG8X = [1, 0, 0, 0, 0, X, 1, 0] ;

        NOVALIDIO = [X, X, X, X, 1, X, 1, 0] ;          "No trapping if I/O read or write is not
                                                        "valid
        NOTRAP1X = [0, 0, 0, 1, 0, X, 1, 0] ;           "Locations not to trap
        NOTRAP3X = [0, 0, 1, 1, 0, X, 1, 0] ;
        NOTRAP5X = [0, 1, 0, 1, 0, X, 1, 0] ;
        NOTRAP6X = [0, 1, 1, 0, 0, X, 1, 0] ;
        NOTRAP7X = [0, 1, 1, 1, 0, X, 1, 0] ;
        NOTRAP9X = [1, 0, 0, 1, 0, X, 1, 0] ;
        NOTRAPAX = [1, 0, 1, 0, 0, X, 1, 0] ;
```

```
        NOTRAPBX = [1, 0, 1, 1, 0, X, 1, 0] ;
        NOTRAPCX = [1, 1, 0, 0, 0, X, 1, 0] ;
        NOTRAPDX = [1, 1, 0, 1, 0, X, 1, 0] ;
        NOTRAPEX = [1, 1, 1, 0, 0, X, 1, 0] ;
        NOTRAPFX = [1, 1, 1, 1, 0, X, 1, 0] ;


" Outputs are listed as follows  C0, READY, NMI, MASK

equations

        ENABLE NMIOUT = H ;

        NMIOUT = !NMI ;

        [C0, READY, NMI]:= (Mode == CLEAR)&[1, 1, 1] ;

        ENABLE INVCLKIN = H ;

        INVCLKIN = !CLKIN ;


state_diagram [C0, READY, NMI]

        State S0:       case (Mode == TRAPICU2X) : S1 ;
                        (Mode == TRAPTCU4X) : S1 ;
                        (Mode == TRAPDMA0X) : S1 ;
                        (Mode == TRAPREG8X) : S1 ;

                        (Mode == NOVALIDIO) : S0 ;

                        (Mode == NOTRAP1X) : S0 ;
                        (Mode == NOTRAP3X) : S0 ;
                        (Mode == NOTRAP5X) : S0 ;
                        (Mode == NOTRAP6X) : S0 ;
                        (Mode == NOTRAP7X) : S0 ;
                        (Mode == NOTRAP9X) : S0 ;
                        (Mode == NOTRAPAX) : S0 ;
                        (Mode == NOTRAPBX) : S0 ;
                        (Mode == NOTRAPCX) : S0 ;
                        (Mode == NOTRAPDX) : S0 ;
                        (Mode == NOTRAPEX) : S0 ;
                        (Mode == NOTRAPFX) : S0 ;
                        endcase ;

        State S1 : goto S2;

        State S2 : goto S3;

        State S3 : if Mode == CLEARTRAP then S0 else S3;

end iotrap
```

Appendix C — μPD8237A Emulation Software

```
Microsoft (R) Macro Assembler  Version 4.00              1/1/80 00:46:13

JC LIPS ROM BIOS (c) 1986, JC Information Systems Corp.    Page     1-1
8237 DMA to V40 DMA Converter


                        C  .list
                        C  ;       |-------------------------------------------------|
                        C  ;       |                                                 |
                        C  ;       |      8237 DMA TO V40 DMA CONVERSION ROUTINE     |
                        C  ;       |                                                 |
                        C  ;       |  DESCRIPTION:                                   |
                        C  ;       |      CONVERTS 8237 DMA I/O PROGRAM TO V40 DMA   |
                        C  ;       |      I/O THRU NMI(NON-MASKABLE INTERRUPT) TRAP. |
                        C  ;       |  ENTRY:                                         |
                        C  ;       |      I/O INSTRUCTION TO PORT 00 - 0F (8237 DMA  |
                        C  ;       |      PORT ADDRESS) AND 80 - 83 (DMA PAGE ADDRESS)|
                        C  ;       |      WILL TRIGGER NMI UPON THE COMPLETION.      |
                        C  ;       |      FOLLOWING ROUTINE TRANSLATES THE TRAPPED   |
                        C  ;       |      I/O INSTRUCITONS INTO V40 DMA MODE.        |
                        C  ;       |  TRAPPED INSTRUCTION TYPES:                     |
                        C  ;       |      IN      al,DX                              |
                        C  ;       |      OUT     DX,al                              |
                        C  ;       |      IN      al,pa                              |
                        C  ;       |      OUT     pa,al                              |
                        C  ;       |      (DX register = 0 - f, 80 - 83)             |
                        C  ;       |      (pa | port address = 0 - f, 80 - 83        |
                        C  ;       |                                                 |
                        C  ;       |  CONDITIONS:                                    |
                        C  ;       |      STACK POINTER, DX AND AX REGISTERS SHOULD  |
                        C  ;       |      BE PRESERVED AS IT IS UPON ENTRY. OTHER    |
                        C  ;       |      HARDWARES WHICH REQUIRE NMI TRAP HANDLING  |
                        C  ;       |      SHOULD RETURN THE CONTROL BACK TO ORIGINAL |
                        C  ;       |      NMI TRAP ROUTINE WITHOUT DISTURBING THE    |
                        C  ;       |      INFORMAITON PRIOR TO NMI IF NMI WAS NOT    |
                        C  ;       |      FROM THEIR SOURCE.                         |
                        C  ;       |                                                 |
                        C  ;       |-------------------------------------------------|
                        C
C120                    C  DMA_NMI_TRAP_HANDLER:
C120  1E                C          PUSH    DS                      ; SAVE REGISTERS
C121  53                C          PUSH    BX
C122  51                C          PUSH    CX
C123  52                C          PUSH    DX
C124  8B DC             C          MOV     BX,SP                   ; GET STACK POINTER
C126  36: 8B 47 0C      C          MOV     AX,SS:12[BX]            ; GET SAVED CODE SEGMENT ADDRESS
C12A  8E D8             C          MOV     DS,AX
C12C  36: 8B 4F 08      C          MOV     CX,SS:8[BX]             ; GET SAVED AX INTO CX
C130  36: 8B 5F 0A      C          MOV     BX,SS:10[BX]            ; GET SAVED INSTRUCTION POINTER
C134  8B 5F FE          C          MOV     BX,-2[BX]               ; GET INSTRUCTION
C137  B8 0030           C          MOV     AX,30H                  ; SET TO STACK SEGMENT
C13A  8E D8             C          MOV     DS,AX                   ; SET DATA SEGMENT
C13C  80 FF EE          C          CMP     BH,0EEH                 ; OUT DX,AL ?
C13F  74 19             C          JZ      DMA_OUT                 ; JUMP IF YES
C141  80 FF EC          C          CMP     BH,0ECH                 ; IN AL,DX ?
C144  74 11             C          JZ      DMA_IN_TRAP             ; JUMP IF YES
C146  32 F6             C          XOR     DH,DH
```

```
Microsoft (R) Macro Assembler  Version 4.00              1/1/80 00:46:13

JC LIPS ROM BIOS (c) 1986, JC Information Systems Corp.     Page     1-2
8237 DMA to V40 DMA Converter


C148  8A D7                 C           MOV     DL,BH
C14A  80 FB E6              C           CMP     BL,0E6H             ; OUT DMA,AL ?
C14D  74 08                 C           JZ      DMA_OUT             ; JUMP IF YES
C14F  80 FB E4              C           CMP     BL,0E4H             ; IN AL,DMA ?
C152  74 03                 C           JZ      DMA_IN_TRAP         ; JUMP IF YES
C154  E9 C27F R             C           JMP     NMI_EXIT            ; EXIT IF NOT DMA NMI
C157                        C  DMA_IN_TRAP:
C157  E9 C227 R             C           JMP     DMA_IN
                            C
```

```
Microsoft (R) Macro Assembler  Version 4.00              1/1/80 00:46:13

JC LIPS ROM BIOS (c) 1986, JC Information Systems Corp.     Page     1-3
8237 DMA to V40 DMA Converter


                        C          page
                        C  ;-----------------------------------------------------
                        C  ; INPUT                                              .
                        C  ;       (DX) 8237 ADDRESS                            .
                        C  ;       (AL) 8237 OUTPUT COMMAND                     .
                        C  ;-----------------------------------------------------
C15A                    C  DMA_OUT:                         ; 8237 OUT TRAP ROUTINE
C15A  83 FA 10          C          CMP     DX,10H           ; CHECK UPPER LIMIT
C15D  72 03             C          JB      DOUT0            ; JUMP IF 8237 REGISTERS
C15F  E9 C1E6 R         C          JMP     DOUTP            ; CHECK IF PAGE REGISTERS
C162  8B DA             C  DOUT0:  MOV     BX,DX            ; SET POINTER
C164  03 DA             C          ADD     BX,DX
C166  2E: FF A7 C207 R  C          JMP     WORD PTR CS:[BX+OFFSET DOUT_TABLE]
                        C  ;
C168                    C  DOUT2:                           ; 8237 ADDRESS 2
C168  B0 01             C          MOV     AL,1             ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C16D  EB 12 90          C          JMP     DOUT62           ; GO TO COMMON ROUTINE
                        C  ;
C170                    C  DOUT3:                           ; 8237 ADDRESS 3
C170  B0 01             C          MOV     AL,1             ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C172  EB 14 90          C          JMP     DOUT72           ; GO TO COMMON ROUTINE
                        C  ;
C175                    C  DOUT4:                           ; 8237 ADDRESS 4
C175  B0 02             C          MOV     AL,2             ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C177  EB 08 90          C          JMP     DOUT62           ; GO TO COMMON ROUTINE
                        C  ;
C17A                    C  DOUT5:                           ; 8237 ADDRESS 5
C17A  B0 02             C          MOV     AL,2             ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C17C  EB 0A 90          C          JMP     DOUT72           ; GO TO COMMON ROUTINE
                        C  ;
C17F                    C  DOUT6:                           ; 8237 ADDRESS 6
C17F  B0 03             C          MOV     AL,3             ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C181  B2 14             C  DOUT62: MOV     DL,DMAU+4        ; POINT TO HIGH OR LOW
C183  EB 05 90          C          JMP     DOUT78           ; GO TO COMMON ROUTINE
                        C  ;
C186                    C  DOUT7:                           ; 8237 ADDRESS 7
C186  B0 03             C          MOV     AL,3             ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C188  B2 12             C  DOUT72: MOV     DL,DMAU+2        ; POINT TO HIGH OR LOW
C18A  E6 11             C  DOUT78: OUT     DMAU+1,AL
C18C  33 DB             C          XOR     BX,BX
C18E  02 17             C          ADD     DL,[BX]          ; GET F/F
C190  32 F6             C          XOR     DH,DH
C192  8A C1             C          MOV     AL,CL            ; ADDRESS OR COUNT
C194  EE                C          OUT     DX,AL
C195  80 37 01          C          XOR     BYTE PTR [BX],1  ; CHANGE F/F
C198  E9 C27F R         C          JMP     NMI_EXIT
                        C  ;
C19B  8A C1             C  DOUT8:  MOV     AL,CL            ; GET OUTPUT COMMAND
C19D  E6 18             C          OUT     DMAU+8,AL        ; OUTPUT COMMAND REGISTER
C19F  E9 C27F R         C          JMP     NMI_EXIT
                        C  ;
```

```
Microsoft (R) Macro Assembler  Version 4.00                1/1/80 00:46:13

JC LIPS ROM BIOS (c) 1986, JC Information Systems Corp.     Page     1-4
8237 DMA to V40 DMA Converter


C1A2  8A C1                C  DOUTA:  MOV     AL,CL           ; COPY MASK COMMAND
C1A4  80 E1 03             C         AND     CL,3
C1A7  B4 01                C         MOV     AH,1            ; POSITION BIT
C1A9  D2 E4                C         SHL     AH,CL           ; SHIFT TO LEFT
C1AB  A8 04                C         TEST    AL,00000100B    ; TEST SET/CLR BIT
C1AD  E4 1F                C         IN      AL,DMAU+0FH     ; GET MASK BITS
C1AF  74 07                C         JZ      DOUTA1          ; JUMP IF CLEAR
C1B1  0A C4                C         OR      AL,AH           ; SET MASK BIT
C1B3  E6 1F                C         OUT     DMAU+0FH,AL
C1B5  E9 C27F R            C         JMP     NMI_EXIT
C1B8  F6 D4                C  DOUTA1: NOT     AH              ; COMPLEMENT AH
C1BA  22 C4                C         AND     AL,AH           ; CLAR MASK BIT
C1BC  E6 1F                C         OUT     DMAU+0FH,AL
C1BE  E9 C27F R            C         JMP     NMI_EXIT
                           C  ;
C1C1  8A C1                C  DOUTB:  MOV     AL,CL           ; COPY MODE REGISTER
C1C3  24 03                C         AND     AL,3            ; MASK CHANNEL SELECT BITS
C1C5  E6 11                C         OUT     DMAU+1,AL       ; SELECT CHANNEL
C1C7  8A C1                C         MOV     AL,CL           ; GET BACK MODE COMMAND
C1C9  E6 1A                C         OUT     DMAU+0AH,AL     ; ISSUE MODE COMMAND
C1CB  E9 C27F R            C         JMP     NMI_EXIT
                           C  ;
C1CE  33 DB                C  DOUTC:  XOR     BX,BX           ; CLEAR F/F AT ADDRESS 0
C1D0  C6 07 00             C         MOV     BYTE PTR [BX],0
C1D3  E9 C27F R            C         JMP     NMI_EXIT
                           C  ;
C1D6  B0 01                C  DOUTD:  MOV     AL,1            ; MASTER RESET
C1D8  E6 10                C         OUT     DMAU+0,AL
C1DA  E9 C27F R            C         JMP     NMI_EXIT
                           C  ;
C1DD  32 C9                C  DOUTE:  XOR     CL,CL           ; CLAR ALL MASKS
C1DF  8A C1                C  DOUTF:  MOV     AL,CL           ; GET MASK BITS
C1E1  E6 1F                C         OUT     DMAU+0FH,AL
C1E3  E9 C27F R            C         JMP     NMI_EXIT
                           C  ;
C1E6  B0 02                C  DOUTP:  MOV     AL,2            ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C1E8  81 FA 0081           C         CMP     DX,81H          ; PAGE REGISTER FOR DMA CHANNEL 2
C1EC  74 10                C         JZ      DOUTP8
C1EE  B0 03                C         MOV     AL,3            ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C1F0  81 FA 0082           C         CMP     DX,82H          ; PAGE REGISTER FOR DMA CHANNEL 3
C1F4  74 08                C         JZ      DOUTP8
C1F6  B0 01                C         MOV     AL,1            ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C1F8  81 FA 0083           C         CMP     DX,83H          ; PAGE REGISTER FOR DMA CHANNEL 1
C1FC  75 06                C         JNZ     DOUTPX
C1FE  E6 11                C  DOUTP8: OUT     DMAU+1,AL
C200  8A C1                C         MOV     AL,CL           ; GET PAGE ADDRESS
C202  E6 16                C         OUT     DMAU+6,AL       ; SET DMA PAGE
C204  EB 79 90             C  DOUTPX: JMP     NMI_EXIT
                           C  ;
C207                       C  DOUT_TABLE      LABEL   WORD
C207  C27F R               C         DW      OFFSET NMI_EXIT ; REG 0
```

```
Microsoft (R) Macro Assembler  Version 4.00                1/1/80 00:46:13

JC LIPS ROM BIOS (c) 1986, JC Information Systems Corp.     Page     1-5
8237 DMA to V40 DMA Converter

C209  C27F R                    C       DW      OFFSET NMI_EXIT ; REG 1
C20B  C168 R                    C       DW      OFFSET DOUT2    ; REG 2
C20D  C170 R                    C       DW      OFFSET DOUT3    ; REG 3
C20F  C175 R                    C       DW      OFFSET DOUT4    ; REG 4
C211  C17A R                    C       DW      OFFSET DOUT5    ; REG 5
C213  C17F R                    C       DW      OFFSET DOUT6    ; REG 6
C215  C186 R                    C       DW      OFFSET DOUT7    ; REG 7
C217  C198 R                    C       DW      OFFSET DOUT8    ; REG 8
C219  C27F R                    C       DW      OFFSET NMI_EXIT ; REG 9
C21B  C1A2 R                    C       DW      OFFSET DOUTA    ; REG A
C21D  C1C1 R                    C       DW      OFFSET DOUTB    ; REG B
C21F  C1CE R                    C       DW      OFFSET DOUTC    ; REG C
C221  C1D6 R                    C       DW      OFFSET DOUTD    ; REG D
C223  C1DD R                    C       DW      OFFSET DOUTE    ; REG E
C225  C1DF R                    C       DW      OFFSET DOUTF    ; REG F
                                C
```

```
Microsoft (R) Macro Assembler  Version 4.00              1/1/80 00:46:13

JC LIPS ROM BIOS (c) 1986, JC Information Systems Corp.     Page     1-6
8237 DMA to V40 DMA Converter


                        C          page
                        C  ;-------------------------------------------------------
                        C  ; INPUT                                                .
                        C  ;       (DX) 8237 ADDRESS                              .
                        C  ; OUTPUT                                               .
                        C  ;       (AL) 8237 INPUT DATA                           .
                        C  ;-------------------------------------------------------
C227                    C  DMA_IN:                          ; 8237 IN TRAP ROUTINE
C227  83 FA 09          C          CMP     DX,9             ; CHECK UPPER LIMIT
C22A  73 53             C          JNB     NMI_EXIT
C22C  8B DA             C          MOV     BX,DX            ; SET POINTER
C22E  03 DA             C          ADD     BX,DX
C230  2E: FF A7 C260 R  C          JMP     WORD PTR CS:[BX+OFFSET DIN_TABLE]
                        C  ;
C235                    C  DIN2:                            ; 8237 ADDRESS 2
C235  B0 01             C          MOV     AL,1             ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C237  EB 12 90          C          JMP     DIN62            ; GO TO COMMON ROUTINE
                        C  ;
C23A                    C  DIN3:                            ; 8237 ADDRESS 3
C23A  B0 01             C          MOV     AL,1             ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C23C  EB 14 90          C          JMP     DIN72            ; GO TO COMMON ROUTINE
                        C  ;
C23F                    C  DIN4:                            ; 8237 ADDRESS 4
C23F  B0 02             C          MOV     AL,2             ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C241  EB 08 90          C          JMP     DIN62            ; GO TO COMMON ROUTINE
                        C  ;
C244                    C  DIN5:                            ; 8237 ADDRESS 5
C244  B0 02             C          MOV     AL,2             ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C246  EB 0A 90          C          JMP     DIN72            ; GO TO COMMON ROUTINE
                        C  ;
C249                    C  DIN6:                            ; 8237 ADDRESS 6
C249  B0 03             C          MOV     AL,3             ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C24B  B2 14             C  DIN62:  MOV     DL,DMAU+4        ; POINT TO HIGH OR LOW
C24D  EB 05 90          C          JMP     DIN78            ; GO TO COMMON ROUTINE
                        C  ;
C250                    C  DIN7:                            ; 8237 ADDRESS 7
C250  B0 03             C          MOV     AL,3             ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C252  B2 12             C  DIN72:  MOV     DL,DMAU+2        ; POINT TO HIGH OR LOW
C254  E6 11             C  DIN78:  OUT     DMAU+1,AL        ; SELECT CHANNEL
C256  33 DB             C          XOR     BX,BX
C258  02 17             C          ADD     DL,[BX]          ; GET-F/F
C25A  32 F6             C          XOR     DH,DH            ; MAKE UP 16 BIT ADDRESS
C25C  80 37 01          C          XOR     BYTE PTR [BX],1  ; CHANGE F/F
C25F  EC                C          IN      AL,DX            ; GET CONTENTS OF POINTED REGISTER
                        C  ;
C260  8B DC             C  DINEX:  MOV     BX,SP            ; REPLACE SAVED AL
C262  36: 88 47 08      C          MOV     SS:8[BX],AL
C266  EB 17 90          C          JMP     NMI_EXIT
                        C  ;
C269  E4 1B             C  DIN8:   IN      AL,DMAU+0BH      ; 8237 ADDRESS 8
C26B  EB F3             C          JMP     DINEX
```

```
Microsoft (R) Macro Assembler  Version 4.00                1/1/80 00:46:13

JC LIPS ROM BIOS (c) 1986, JC Information Systems Corp.     Page     1-7
8237 DMA to V40 DMA Converter


                        C  ;
C260                    C  DIN_TABLE        LABEL   WORD
C260  C27F R            C          DW       OFFSET NMI_EXIT ; REG 0
C26F  C27F R            C          DW       OFFSET NMI_EXIT ; REG 1
C271  C235 R            C          DW       OFFSET DIN2     ; REG 2
C273  C23A R            C          DW       OFFSET DIN3     ; REG 3
C275  C23F R            C          DW       OFFSET DIN4     ; REG 4
C277  C244 R            C          DW       OFFSET DIN5     ; REG 5
C279  C249 R            C          DW       OFFSET DIN6     ; REG 6
C27B  C250 R            C          DW       OFFSET DIN7     ; REG 7
C27D  C269 R            C          DW       OFFSET DIN8     ; REG 8
                        C
C27F                    C  NMI_EXIT:
C27F  5A                C          POP      DX                      ; RESTORE ALL REGISTERS
C280  59                C          POP      CX
C281  5B                C          POP      BX
C282  1F                C          POP      DS
C283  58                C          POP      AX                      ; RESTORE ORIG CONTENTS OF AX
C284  CF                C          IRET
                        C  NMI_INT ENDP
```

```
Microsoft (R) Macro Assembler  Version 4.00                1/1/80 00:46:13

JC LIPS ROM BIOS (c) 1986, JC Information Systems Corp.     Page     1-8
8237 DMA to V40 DMA Converter

                              C          page
 FFE1                           code     ends
bios.ASM(23) : error 102: Segment near (or at) 64K limit

   8990 Source  Lines
   9002 Total   Lines
    859 Symbols

  20178 Bytes symbol space free

      1 Warning Errors
      0 Severe  Errors
```

8237 DMA to V40 DMA Converter

```
                        C  .list
                        C  ;       |-------------------------------------------------|
                        C  ;       |                                                 |
                        C  ;       |      8237 DMA TO V40 DMA CONVERSION ROUTINE     |
                        C  ;       |                                                 |
                        C  ;       |  DESCRIPTION:                                   |
                        C  ;       |       CONVERTS 8237 DMA I/O PROGRAM TO V40 DMA  |
                        C  ;       |       I/O THRU NMI(NON-MASKABLE INTERRUPT) TRAP.|
                        C  ;       |  ENTRY:                                         |
                        C  ;       |       I/O INSTRUCTION TO PORT 00 - 0F (8237 DMA |
                        C  ;       |       PORT ADDRESS) AND 80 - 83 (DMA PAGE ADDRESS)|
                        C  ;       |       WILL TRIGGER NMI UPON THE COMPLETION.     |
                        C  ;       |       FOLLOWING ROUTINE TRANSLATES THE TRAPPED  |
                        C  ;       |       I/O INSTRUCITONS INTO V40 DMA MODE.       |
                        C  ;       |  TRAPPED INSTRUCTION TYPES:                     |
                        C  ;       |       IN      al,DX                             |
                        C  ;       |       OUT     DX,al                             |
                        C  ;       |       IN      al,pa                             |
                        C  ;       |       OUT     pa,al                             |
                        C  ;       |       (DX register = 0 - f, 80 - 83)            |
                        C  ;       |       (pa | port address = 0 - f, 80 - 83       |
                        C  ;       |                                                 |
                        C  ;       |  CONDITIONS:                                    |
                        C  ;       |       STACK POINTER, DX AND AX REGISTERS SHOULD |
                        C  ;       |       BE PRESERVED AS IT IS UPON ENTRY. OTHER   |
                        C  ;       |       HARDWARES WHICH REQUIRE NMI TRAP HANDLING |
                        C  ;       |       SHOULD RETURN THE CONTROL BACK TO ORIGINAL|
                        C  ;       |       NMI TRAP ROUTINE WITHOUT DISTURBING THE   |
                        C  ;       |       INFORMAITON PRIOR TO NMI IF NMI WAS NOT   |
                        C  ;       |       FROM THEIR SOURCE.                        |
                        C  ;       |                                                 |
                        C  ;       |-------------------------------------------------|
                        C
C120                    C  DMA_NMI_TRAP_HANDLER:
C120  1E                C          PUSH    DS                  ; SAVE REGISTERS
C121  53                C          PUSH    BX
C122  51                C          PUSH    CX
C123  52                C          PUSH    DX
C124  8B DC             C          MOV     BX,SP               ; GET STACK POINTER
C126  36: 8B 47 0C      C          MOV     AX,SS:12[BX]        ; GET SAVED CODE SEGMENT ADDRESS
C12A  8E D8             C          MOV     DS,AX
C12C  36: 8B 4F 08      C          MOV     CX,SS:8[BX]         ; GET SAVED AX INTO CX
C130  36: 8B 5F 0A      C          MOV     BX,SS:10[BX]        ; GET SAVED INSTRUCTION POINTER
C134  8B 5F FE          C          MOV     BX,-2[BX]           ; GET INSTRUCTION
C137  B8 0030           C          MOV     AX,30H              ; SET TO STACK SEGMENT
C13A  8E D8             C          MOV     DS,AX               ; SET DATA SEGMENT
C13C  80 FF EE          C          CMP     BH,0EEH             ; OUT DX,AL ?
C13F  74 19             C          JZ      DMA_OUT             ; JUMP IF YES
C141  80 FF EC          C          CMP     BH,0ECH             ; IN AL,DX ?
```

8237 DMA to V40 DMA Converter

```
C144  74 11              C          JZ      DMA_IN_TRAP          ; JUMP IF YES
C146  32 F6              C          XOR     DH,DH
C148  8A D7              C          MOV     DL,BH
C14A  80 FB E6           C          CMP     BL,0E6H              ; OUT DMA,AL ?
C14D  74 0B              C          JZ      DMA_OUT              ; JUMP IF YES
C14F  80 FB E4           C          CMP     BL,0E4H              ; IN AL,DMA ?
C152  74 03              C          JZ      DMA_IN_TRAP          ; JUMP IF YES
C154  E9 C27F R          C          JMP     NMI_EXIT             ; EXIT IF NOT DMA NMI
C157                     C  DMA_IN_TRAP:
C157  E9 C227 R          C          JMP     DMA_IN
                         C
```

8237 DMA to V40 DMA Converter

```
                        C           page
                        C  ;---------------------------------------------------
                        C  ; INPUT                                            .
                        C  ;       (DX) 8237 ADDRESS                          .
                        C  ;       (AL) 8237 OUTPUT COMMAND                   .
                        C  ;---------------------------------------------------
C15A                    C  DMA_OUT:                         ; 8237 OUT TRAP ROUTINE
C15A  83 FA 10          C          CMP     DX,10H           ; CHECK UPPER LIMIT
C15D  72 03             C          JB      DOUT0            ; JUMP IF 8237 REGISTERS
C15F  E9 C1E6 R         C          JMP     DOUTP            ; CHECK IF PAGE REGISTERS
C162  8B DA             C  DOUT0:  MOV     BX,DX            ; SET POINTER
C164  03 DA             C          ADD     BX,DX
C166  2E: FF A7 C207 R  C          JMP     WORD PTR CS:[BX+OFFSET DOUT_TABLE]
                        C  ;
C16B                    C  DOUT2:                           ; 8237 ADDRESS 2
C16B  B0 01             C          MOV     AL,1             ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C16D  EB 12 90          C          JMP     DOUT62           ; GO TO COMMON ROUTINE
                        C  ;
C170                    C  DOUT3:                           ; 8237 ADDRESS 3
C170  B0 01             C          MOV     AL,1             ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C172  EB 14 90          C          JMP     DOUT72           ; GO TO COMMON ROUTINE
                        C  ;
C175                    C  DOUT4:                           ; 8237 ADDRESS 4
C175  B0 02             C          MOV     AL,2             ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C177  EB 08 90          C          JMP     DOUT62           ; GO TO COMMON ROUTINE
                        C  ;
C17A                    C  DOUT5:                           ; 8237 ADDRESS 5
C17A  B0 02             C          MOV     AL,2             ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C17C  EB 0A 90          C          JMP     DOUT72           ; GO TO COMMON ROUTINE
                        C  ;
£17F                    C  DOUT6:                           ; 8237 ADDRESS 6
C17F  B0 03             C          MOV     AL,3             ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C181  B2 14             C  DOUT62: MOV     DL,DMAU+4        ; POINT TO HIGH OR LOW
C183  EB 05 90          C          JMP     DOUT78           ; GO TO COMMON ROUTINE
                        C  ;
C186                    C  DOUT7:                           ; 8237 ADDRESS 7
C186  B0 03             C          MOV     AL,3             ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C188  B2 12             C  DOUT72: MOV     DL,DMAU+2        ; POINT TO HIGH OR LOW
C18A  E6 11             C  DOUT78: OUT     DMAU+1,AL
C18C  33 DB             C          XOR     BX,BX
C18E  02 17             C          ADD     DL,[BX]          ; GET F/F
C190  32 F6             C          XOR     DH,DH
C192  8A C1             C          MOV     AL,CL            ; ADDRESS OR COUNT
C194  EE                C          OUT     DX,AL
C195  80 37 01          C          XOR     BYTE PTR [BX],1  ; CHANGE F/F
C198  E9 C27F R         C          JMP     NMI_EXIT
                        C  ;
C19B  8A C1             C  DOUT8:  MOV     AL,CL            ; GET OUTPUT COMMAND
C19D  E6 18             C          OUT     DMAU+8,AL        ; OUTPUT COMMAND REGISTER
C19F  E9 C27F R         C          JMP     NMI_EXIT
                        C  ;
```

8237 DMA to V40 DMA Converter

```
C1A2  8A C1              C  DOUTA:  MOV   AL,CL            ; COPY MASK COMMAND
C1A4  80 E1 03           C          AND   CL,3
C1A7  B4 01              C          MOV   AH,1             ; POSITION BIT
C1A9  D2 E4              C          SHL   AH,CL            ; SHIFT TO LEFT
C1AB  A8 04              C          TEST  AL,00000100B     ; TEST SET/CLR BIT
C1AD  E4 1F              C          IN    AL,DMAU+0FH      ; GET MASK BITS
C1AF  74 07              C          JZ    DOUTA1           ; JUMP IF CLEAR
C1B1  0A C4              C          OR    AL,AH            ; SET MASK BIT
C1B3  E6 1F              C          OUT   DMAU+0FH,AL
C1B5  E9 C27F R          C          JMP   NMI_EXIT
C1B8  F6 D4              C  DOUTA1: NOT   AH              ; COMPLEMENT AH
C1BA  22 C4              C          AND   AL,AH           ; CLAR MASK BIT
C1BC  E6 1F              C          OUT   DMAU+0FH,AL
C1BE  E9 C27F R          C          JMP   NMI_EXIT
                         C  ;
C1C1  8A C1              C  DOUTB:  MOV   AL,CL           ; COPY MODE REGISTER
C1C3  24 03              C          AND   AL,3            ; MASK CHANNEL SELECT BITS
C1C5  E6 11              C          OUT   DMAU+1,AL       ; SELECT CHANNEL
C1C7  8A C1              C          MOV   AL,CL           ; GET BACK MODE COMMAND
C1C9  E6 1A              C          OUT   DMAU+0AH,AL     ; ISSUE MODE COMMAND
C1CB  E9 C27F R          C          JMP   NMI_EXIT
                         C  ;
C1CE  33 DB              C  DOUTC:  XOR   BX,BX           ; CLEAR F/F AT ADDRESS 0
C1D0  C6 07 00           C          MOV   BYTE PTR [BX],0
C1D3  E9 C27F R          C          JMP   NMI_EXIT
                         C  ;
C1D6  B0 01              C  DOUTD:  MOV   AL,1            ; MASTER RESET
C1D8  E6 10              C          OUT   DMAU+0,AL
C1DA  E9 C27F R          C          JMP   NMI_EXIT
                         C  ;
C1DD  32 C9              C  DOUTE:  XOR   CL,CL           ; CLAR ALL MASKS
C1DF  8A C1              C  DOUTF:  MOV   AL,CL           ; GET MASK BITS
C1E1  E6 1F              C          OUT   DMAU+0FH,AL
C1E3  E9 C27F R          C          JMP   NMI_EXIT
                         C  ;
C1E6  B0 02              C  DOUTP:  MOV   AL,2            ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C1E8  81 FA 0081         C          CMP   DX,81H          ; PAGE REGISTER FOR DMA CHANNEL 2
C1EC  74 10              C          JZ    DOUTP8
C1EE  B0 03              C          MOV   AL,3            ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C1F0  81 FA 0082         C          CMP   DX,82H          ; PAGE REGISTER FOR DMA CHANNEL 3
C1F4  74 08              C          JZ    DOUTP8
C1F6  B0 01              C          MOV   AL,1            ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C1F8  81 FA 0083         C          CMP   DX,83H          ; PAGE REGISTER FOR DMA CHANNEL 1
C1FC  75 06              C          JNZ   DOUTPX
C1FE  E6 11              C  DOUTP8: OUT   DMAU+1,AL
C200  8A C1              C          MOV   AL,CL           ; GET PAGE ADDRESS
C202  E6 16              C          OUT   DMAU+6,AL       ; SET DMA PAGE
C204  EB 79 90           C  DOUTPX: JMP   NMI_EXIT
                         C  ;
```

8237 DMA to V40 DMA Converter

```
C207                    C  DOUT_TABLE     LABEL   WORD
C207  C27F R            C                 OFFSET NMI_EXIT ; REG 0
C209  C27F R            C                 OFFSET NMI_EXIT ; REG 1
C20B  C16B R            C                 OFFSET DOUT2    ; REG 2
C20D  C170 R            C                 OFFSET DOUT3    ; REG 3
C20F  C175 R            C                 OFFSET DOUT4    ; REG 4
C211  C17A R            C                 OFFSET DOUT5    ; REG 5
C213  C17F R            C                 OFFSET DOUT6    ; REG 6
C215  C186 R            C                 OFFSET DOUT7    ; REG 7
C217  C19B R            C                 OFFSET DOUT8    ; REG 8
C219  C27F R            C                 OFFSET NMI_EXIT ; REG 9
C21B  C1A2 R            C                 OFFSET DOUTA    ; REG A
C21D  C1C1 R            C                 OFFSET DOUTB    ; REG B
C21F  C1CE R            C                 OFFSET DOUTC    ; REG C
C221  C1D6 R            C                 OFFSET DOUTD    ; REG D
C223  C1DD R            C                 OFFSET DOUTE    ; REG E
C225  C1DF R            C                 OFFSET DOUTF    ; REG F
                        C
```

8237 DMA to V40 DMA Converter

```
                        C           page
                        C ;-----------------------------------------------------
                        C ; INPUT                                               .
                        C ;       (DX) 8237 ADDRESS                             .
                        C ; OUTPUT                                              .
                        C ;       (AL) 8237 INPUT DATA                          .
                        C ;-----------------------------------------------------
C227                    C DMA_IN:                       ; 8237 IN TRAP ROUTINE
C227  83 FA 09          C         CMP     DX,9          ; CHECK UPPER LIMIT
C22A  73 53             C         JNB     NMI_EXIT
C22C  8B DA             C         MOV     BX,DX         ; SET POINTER
C22E  03 DA             C         ADD     BX,DX
C230  2E: FF A7 C26D R  C         JMP     WORD PTR CS:[BX+OFFSET DIN_TABLE]
                        C ;
C235                    C DIN2:                         ; 8237 ADDRESS 2
C235  B0 01             C         MOV     AL,1          ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C237  EB 12 90          C         JMP     DIN62         ; GO TO COMMON ROUTINE
                        C ;
C23A                    C DIN3:                         ; 8237 ADDRESS 3
C23A  B0 01             C         MOV     AL,1          ; SELECT CHANNEL 1 (8237 CHANNEL 1)
C23C  EB 14 90          C         JMP     DIN72         ; GO TO COMMON ROUTINE
                        C ;
C23F                    C DIN4:                         ; 8237 ADDRESS 4
C23F  B0 02             C         MOV     AL,2          ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C241  EB 08 90          C         JMP     DIN62         ; GO TO COMMON ROUTINE
                        C ;
C244                    C DIN5:                         ; 8237 ADDRESS 5
C244  B0 02             C         MOV     AL,2          ; SELECT CHANNEL 2 (8237 CHANNEL 2)
C246  EB 0A 90          C         JMP     DIN72         ; GO TO COMMON ROUTINE
                        C ;
C249                    C DIN6:                         ; 8237 ADDRESS 6
C249  B0 03             C         MOV     AL,3          ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C24B  B2 14             C DIN62:  MOV     DL,DMAU+4     ; POINT TO HIGH OR LOW
C24D  EB 05 90          C         JMP     DIN78         ; GO TO COMMON ROUTINE
                        C ;
C250                    C DIN7:                         ; 8237 ADDRESS 7
C250  B0 03             C         MOV     AL,3          ; SELECT CHANNEL 3 (8237 CHANNEL 3)
C252  B2 12             C DIN72:  MOV     DL,DMAU+2     ; POINT TO HIGH OR LOW
C254  E6 11             C DIN78:  OUT     DMAU+1,AL     ; SELECT CHANNEL
C256  33 DB             C         XOR     BX,BX
C258  02 17             C         ADD     DL,[BX]       ; GET F/F
C25A  32 F6             C         XOR     DH,DH         ; MAKE UP 16 BIT ADDRESS
C25C  80 37 01          C         XOR     BYTE PTR [BX],1 ; CHANGE F/F
C25F  EC                C         IN      AL,DX         ; GET CONTENTS OF POINTED REGISTER
                        C ;
C260  8B DC             C DINEX:  MOV     BX,SP         ; REPLACE SAVED AL
C262  36: 88 47 08      C         MOV     SS:8[BX],AL
C266  EB 17 90          C         JMP     NMI_EXIT
                        C ;
```

8237 DMA to V40 DMA Converter

```
C269  E4 1B              C  DIN8:   IN      AL,DMAU+0BH     ; 8237 ADDRESS 8
C26B  EB F3              C          JMP     DINEX
                         C  ;
C26D                     C  DIN_TABLE       LABEL   WORD
C26D  C27F R             C          DW      OFFSET NMI_EXIT ; REG 0
C26F  C27F R             C          DW      OFFSET NMI_EXIT ; REG 1
C271  C235 R             C          DW      OFFSET DIN2     ; REG 2
C273  C23A R             C          DW      OFFSET DIN3     ; REG 3
C275  C23F R             C          DW      OFFSET DIN4     ; REG 4
C277  C244 R             C          DW      OFFSET DIN5     ; REG 5
C279  C249 R             C          DW      OFFSET DIN6     ; REG 6
C27B  C250 R             C          DW      OFFSET DIN7     ; REG 7
C27D  C269 R             C          DW      OFFSET DIN8     ; REG 8
                         C
C27F                     C  NMI_EXIT:
C27F  5A                 C          POP     DX                      ; RESTORE ALL REGISTERS
C280  59                 C          POP     CX
C281  5B                 C          POP     BX
C282  1F                 C          POP     DS
C283  58                 C          POP     AX                      ; RESTORE ORIG CONTENTS OF AX
C284  CF                 C          IRET
                         C  NMI_INT ENDP
```

Sequential Fuel Injection Software Using μPD78312

Contents:

Author: K. Funabashi
NEC Tokyo

Persons to contact: R. Cherrington, W. Knippschild
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Product Description
μCOM 78K I / II / III Family
8/16 Bit Microcomputer

Related Products

μPD78310/312	8/16 Bit Microcomputer	CMOS

1. Sequential Fuel Injection

Outline

This sequential fuel injection system is based upon crank angle. Crank angle also controls fuel injection start timing.

When both edges of crank angle (2/pulse) are counted Y times after TDC (fuel injection start timing), X hours (fuel injection time) of output pulse is generated. (See figure 1.)

Hardware

A 16-bit presentable up/down counter (UDCO) and a 16-bit timer (TMO) are used to control sequential fuel injection and spark. (See figure 2.)

(1) RAM

FNCTR	: 1 byte	—	Fuel injection on flag (bit 0 to 5)
FFCTR	: 1 byte	—	Fuel injection off flag (bit 0 to 5)
FYCTR	: 1 byte	—	Fuel injection start timing (the value of Y)
FXTMA[n]	: 3 bytes X 6	—	Fuel injection time (X[n])
FXTMB[n]	: 3 bytes X 6	—	Fuel injection remaining time
FPDAT	: 1 byte	—	Special fuel injection end process data
FPILL	: 1 byte	—	Cylinder correct data
FCHECK	: 1 byte	—	Cylinder correct number
FIDAT	: 1 byte	—	Off cylinder number

Flag

FEXT	: Special fuel injection execution flag
FCHK	: Reference cylinder detection flag
FILL	: Cylinder correction execution flag

(2) 16-bit presentable up/down counter (UDCO) — See figure 2.

UDCO is cleared with the rising edge of TDC. Then UDCO is up-counted at both edges of the crank angle pulse, and when the contents of UDCO coincides with CRO1, interrupt CRFO1 is generated (fuel injection start timing).

At the same time (the rising edge of TDC), the MACRO SERVICE EXIF1 is generated to measure the engine speed.

(3) External interrupt (EXIFO) — See Figure 2.

Interrupt EXIFO occurs with the rising edge of TDC. (Same time as UDCO clear timing and MACRO SERVICE).

(4) 16-bit timer (TMO) — See figure 2, 3.

24 bits of one shot timer is to be realized by using the timer unit and the macro service counter (MSC) (1 bit = 1 μs, 16.8 s Max.) to control fuel injectiom time.

The way this timer is set is shown as figure 3. The value equivalent to the lower 16 bits of the value set in 24 bits minus 1 is set in MDO, and the value equivalent to the value of the upper 8 bits plus 1 is set in the macro service counter (MSC), and then the timer is started in interval mode, and subsequently 0FFFFH (full counts) is set in MDO.

Process

(1) Fuel injection start timing (Y-time counts of crank angle) — See figure 1,2, 4.

In the interrupt EXIFO process which is calculated by the rising edge of TDC, Y is set into CRO1 (figure 4 *1). After then, interrupt CRFO1 occurs when UDCO becomes to equal to CRO1.

(2) Start of sequential fuel injection — See figure 1, 2, 5.

After counting Y times, the corresponding port is set to "H" (figure 5 *1). The timer is started at the same time by software (figure 5 *2). At this time, if the timer is busy counting another cylinder, calculation (fuel injection time) — (time until the timer on counting will count up) is saved in memory (FXTMB) and it goes into a timer waiting status (figure 5 *3).

(3) End of sequential fuel injection — See figure 1, 2, 6.

After fuel injection time, the corresponding port is set to "L" (figure 6 *1). At the same time, if there is a timer waiting status cylinder, the timer is started as the contents of the next FXTMB[n] (figure 6 *2).

If there are more than 2 timer waiting status cylinders, the minimum value of timer starts, and the other's FXTMB[n] minus the contents of TMO is set into FYTMB[n] for new fuel injection remaining time (figure 6 *3).

Note: Timer waiting status: Defined as the contents of FXTMB[n] not equaling "O" (figure 6 *4). As to the details of process, please refer to the flow chart (figure 6).
The example of detailed process for each status is shown as follows.

Example

Fuel injection start process — See figure 5, 7.

(1) In case of other output ports remaining "L".

Corresponding output is set to "H" (figure 5 *1) and fuel injection time (FXTMA) is set in the timer and started (figure 5 *2, figure 7 *1).

(2) In case of other than (1).

Corresponding output is set to "H" (figure 5 *1) and the result of following calculation is stored in FXTMB[n] (figure 5 *3, Figure 7 *2, *3).

$$\text{FXTMB[n]} - \frac{\text{(Fuel injection time FXTMA[n])}}{\text{X[n]}} - \frac{\text{(Current contents of UDCO)}}{\text{TMO}}$$

Fuel injection stop process — See figure 6, 7, 8.

(1) In case of other output ports remaining "L".

Corresponding output is set to "L" and the timer is stopped (figure 6 *1).

(2) In case of other than (1) and not continuously fuel injection.

Corresponding output is set to "L" (figure 6 *1), and the next cylinder fuel injection time (FXTMB[n]) is set in the timer (figure 6 *2, figure 7 *4, *5). Subsequently, the contents of all cylinder remaining fuel injection time (FXTMB[n]) minus the current fuel injection time is re-stored in the FXTMB[n] (figure 6 *3, Figure 7 *1 to *7).

(3) In case of continuous fuel injection.

When the corresponding FXTMB[n] is not "O", this process is regarded as continuous fuel injection. In this case, the corresponding output is not set at "L".

The continuous fuel injection time is set in main flow process for heavy load.

The value set in the timer (TMO) is next to the last re-started output timer (figure 6 *5).

(4) In case of special fuel injection.

The special fuel injection is used to get quick response when quick acceleration is detected.

The figure 9 shows how to start special fuel injection.

Note: * All fuel injection times become invalid when the special fuel injections started.

* When the fuel injection is longer than the special, the fuel injection time which started after the special fuel injection started becomes valid.

* The INTERRUPT DISABLE status must be kept during the special fuel injection start process.

2. Reference Cyclinder Detection

Outline

Pulse width of TDC is measured with crank angle to judge reference cylinders #0 (32°) and #3 (24°).

Hardware — See figure 2

UDCO is up-counted at both edges of crank angle, and is cleared with the rising edge of TDC. Macro service is started with the falling edge of TDC, and the lower 8 bits of UDCO (UDCOL) are transferred into RAM.

Process — See figure 1, 2, 10

Under the EXIF1 interrupt process, the value in RAM (Pulse width) is read to judge the current cylinder. Using the same process, macro service is set for the next detection.

Cylinder Correction Process

Under the interrupt EXIF1 process which is calculated by the falling edge of TDC, pulse width of TDC is measured with crank angle; more than 32° is judged as "#0", and within the range from 24° to 31° is judged as "#3".

In case it is judged as "#0" or "#3", the FCHK flag is set and "1" for "A" or "4" for "B" is set into FCHECK (figure 10 *1).

Under the interrupt EXIF0 process which is calculated by the following edge of TDC, the FCHK flag is tested (figure 4 *2), and if it is "1", FCHECK is compared with FNCTR, and in case where they are not the same, an abnormal status is judged and sent into the cylinder correction mode (FILL flag is set).

FNCTR correction (Next on cylinder)

FNCTR correction is executed as soon as judgement of a abnormal status (figure 4 *3). Therefore the normal count value is obtained since the Y count is set.

FFCTR correction (Next off cylinder)

FFCTR correction is executed in the following 3 ways (depending on the timer condition). The FFCTR correction is done when the timer is not operating.

(1) When timer is not operated — See figure 4 *4.

In case the timer is not operating when the abnormal status is detected, FFCTR is corrected soon. In this case, the FILL flag is not set.

(2) Rest of timer > Normal fuel injection time — See figure 5 *4.

(In cases not 1.) Under interrupt CRFO1 process, the rest of timer is compared with fuel injection time (FXTMA). If the rest of timer is the longer of the two, FFCTR is corrected (figure 5 *4). In this case, all fuel injections are "OFF", excepting normal fuel injection 1 output (set in this process, figure 5 *5).

(3) The other case than above — See figure 6 *5.

(In the cases other than 1 and 2.) Under interrupt TMFO, FFCTR is corrected. In this case, all of fuel injection before setting FILL is reset, and all of them after setting FILL is maintained.

In any case of (1), (2) and (3), when FFCTR is corrected, FILL flag is reset and correction mode is finished.

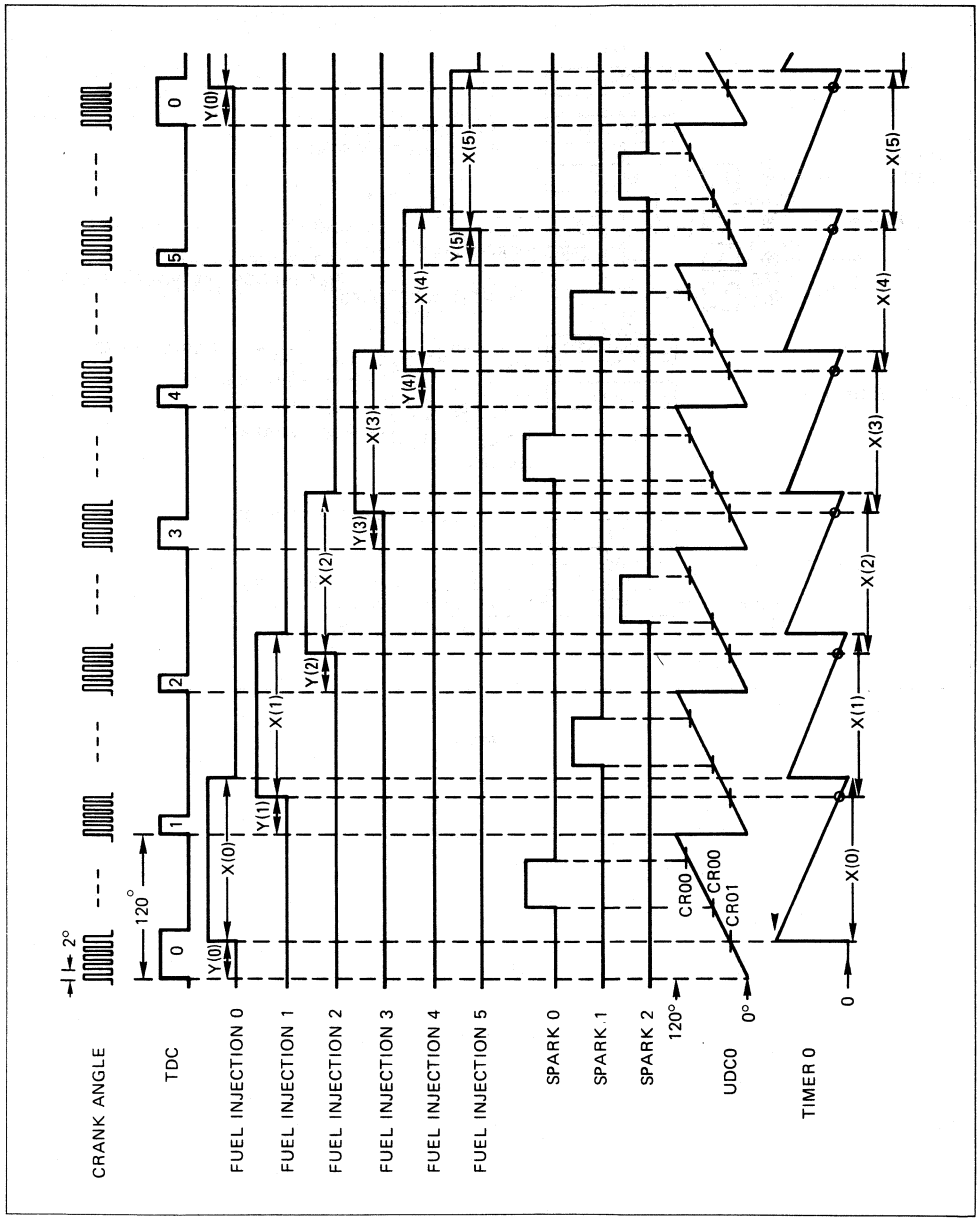

CRANK ANGLE
2°
120°
0
1
2
3
4
5
0
TDC
FUEL INJECTION 0
FUEL INJECTION 1
FUEL INJECTION 2
FUEL INJECTION 3
FUEL INJECTION 4
FUEL INJECTION 5
SPARK 0
SPARK 1
SPARK 2
120°
0°
UDC0
CR00
CR00
CR01
TIMER 0
0
Y(0)
X(0)
Y(1)
X(1)
Y(2)
X(2)
Y(3)
X(3)
Y(4)
X(4)
Y(5)
X(5)

Figure 1

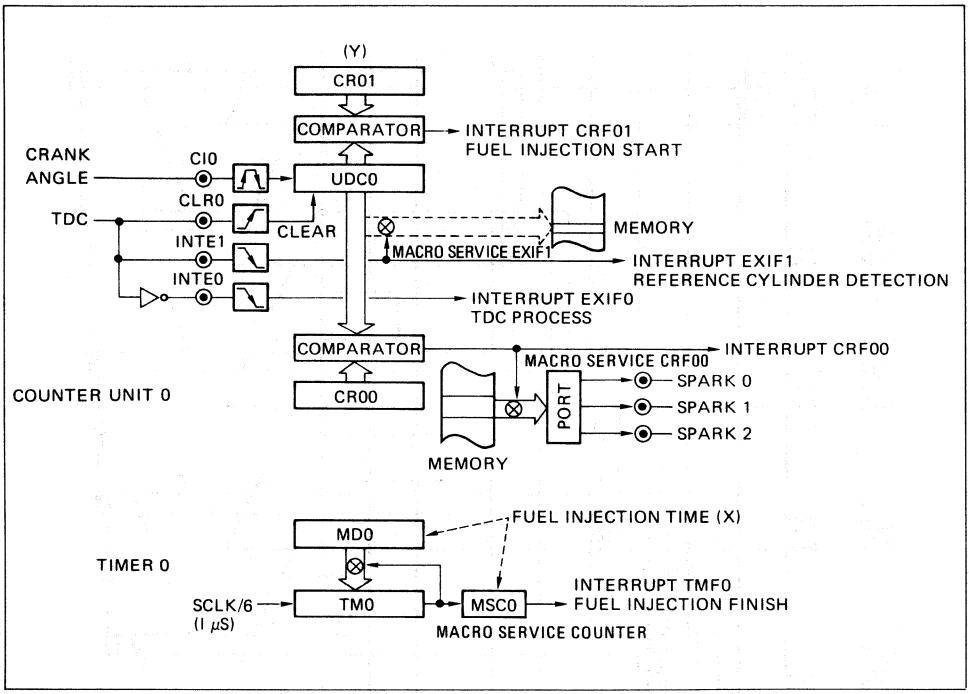
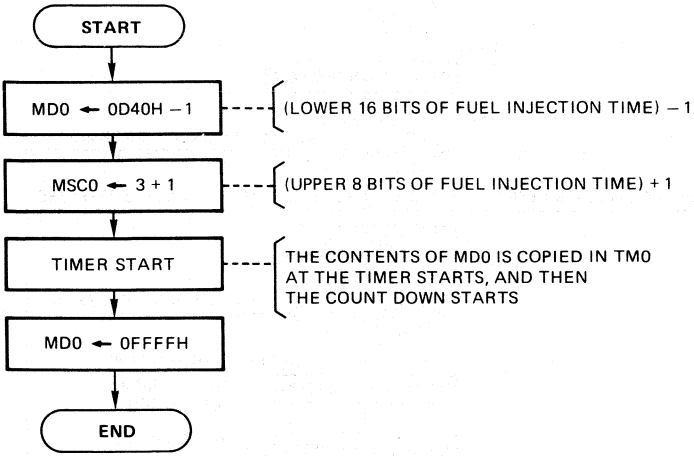

(Y)
CR01
COMPARATOR
INTERRUPT CRF01
FUEL INJECTION START
CRANK
ANGLE
CI0
UDC0
CLR0
TDC
CLEAR
INTE1
INTE0
MACRO SERVICE EXIF1
MEMORY
INTERRUPT EXIF1
REFERENCE CYLINDER DETECTION
INTERRUPT EXIF0
TDC PROCESS
COMPARATOR
INTERRUPT CRF00
MACRO SERVICE CRF00
COUNTER UNIT 0
CR00
PORT
SPARK 0
SPARK 1
SPARK 2
MEMORY
FUEL INJECTION TIME (X)
MD0
TIMER 0
SCLK/6
(1 μS)
TM0
MSC0
INTERRUPT TMF0
FUEL INJECTION FINISH
MACRO SERVICE COUNTER

Figure 2

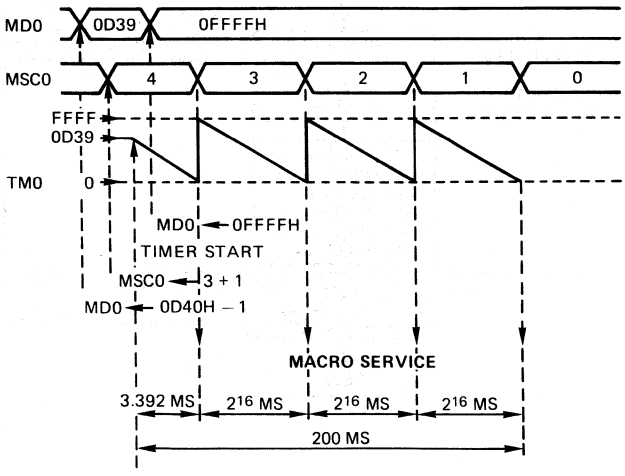
EXAMPLE (200 ms Fuel injection)

Timer starting process



(a) Flowchart

Timing



(b) Timing chart

Figure 3

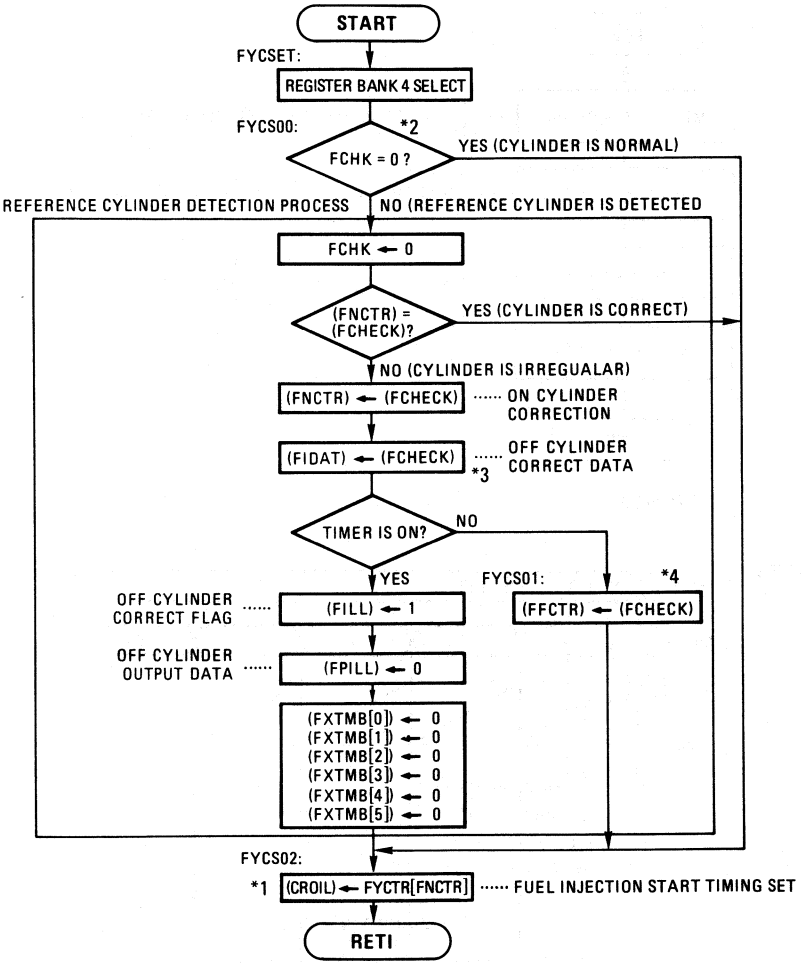

START
FYCSET:
REGISTER BANK 4 SELECT
FYCS00:
*2
FCHK = 0 ?
YES (CYLINDER IS NORMAL)
REFERENCE CYLINDER DETECTION PROCESS
NO (REFERENCE CYLINDER IS DETECTED
FCHK ← 0
(FNCTR) = (FCHECK)?
YES (CYLINDER IS CORRECT)
NO (CYLINDER IS IRREGUALAR)
(FNCTR) ← (FCHECK)
ON CYLINDER CORRECTION
(FIDAT) ← (FCHECK)
OFF CYLINDER CORRECT DATA
*3
TIMER IS ON?
NO
YES
FYCS01:
*4
OFF CYLINDER CORRECT FLAG
(FILL) ← 1
(FFCTR) ← (FCHECK)
OFF CYLINDER OUTPUT DATA
(FPILL) ← 0
(FXTMB[0]) ← 0
(FXTMB[1]) ← 0
(FXTMB[2]) ← 0
(FXTMB[3]) ← 0
(FXTMB[4]) ← 0
(FXTMB[5]) ← 0
FYCS02:
*1
(CROIL) ← FYCTR[FNCTR]
FUEL INJECTION START TIMING SET
RETI

Figure 4

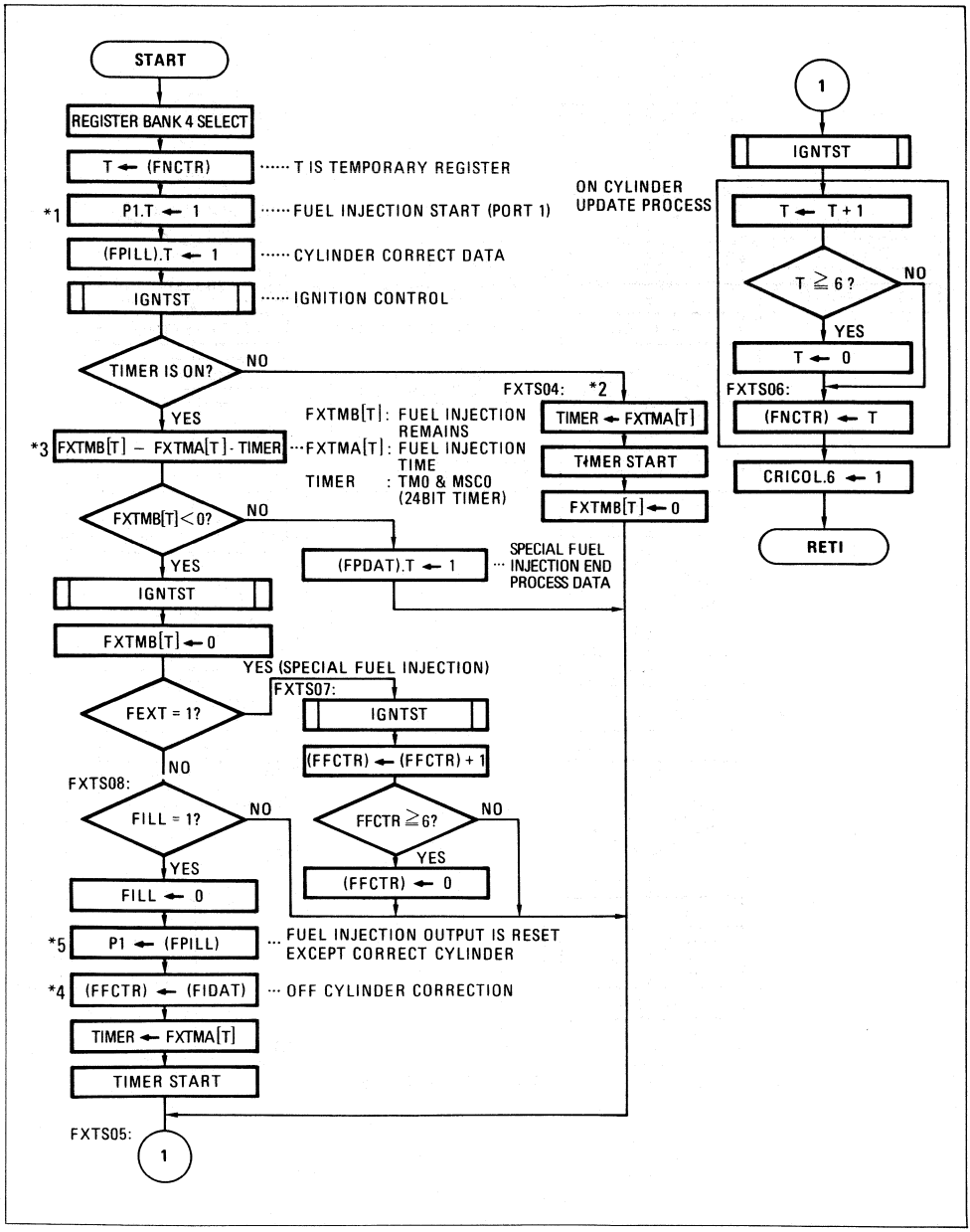

START
REGISTER BANK 4 SELECT
T ← (FNCTR)
..... T IS TEMPORARY REGISTER
*1 P1.T ← 1
..... FUEL INJECTION START (PORT 1)
(FPILL).T ← 1
..... CYLINDER CORRECT DATA
IGNTST
..... IGNITION CONTROL
TIMER IS ON?
NO
YES
*3 FXTMB[T] – FXTMA[T] - TIMER
FXTMB[T]: FUEL INJECTION REMAINS
FXTMA[T]: FUEL INJECTION TIME
TIMER : TM0 & MSC0 (24BIT TIMER)
FXTMB[T]<0?
NO
YES
IGNTST
FXTMB[T] ← 0
YES (SPECIAL FUEL INJECTION)
FEXT = 1?
NO
FXTS08:
FILL = 1?
NO
YES
FILL ← 0
*5 P1 ← (FPILL)
... FUEL INJECTION OUTPUT IS RESET EXCEPT CORRECT CYLINDER
*4 (FFCTR) ← (FIDAT)
... OFF CYLINDER CORRECTION
TIMER ← FXTMA[T]
TIMER START
FXTS05:
1
(FPDAT).T ← 1
... SPECIAL FUEL INJECTION END PROCESS DATA
FXTS04: *2
TIMER ← FXTMA[T]
TIMER START
FXTMB[T] ← 0
FXTS07:
IGNTST
(FFCTR) ← (FFCTR) + 1
FFCTR ≧ 6?
NO
YES
(FFCTR) ← 0
1
IGNTST
ON CYLINDER UPDATE PROCESS
T ← T + 1
T ≧ 6?
NO
YES
T ← 0
FXTS06:
(FNCTR) ← T
CRICOL.6 ← 1
RETI

Figure 5

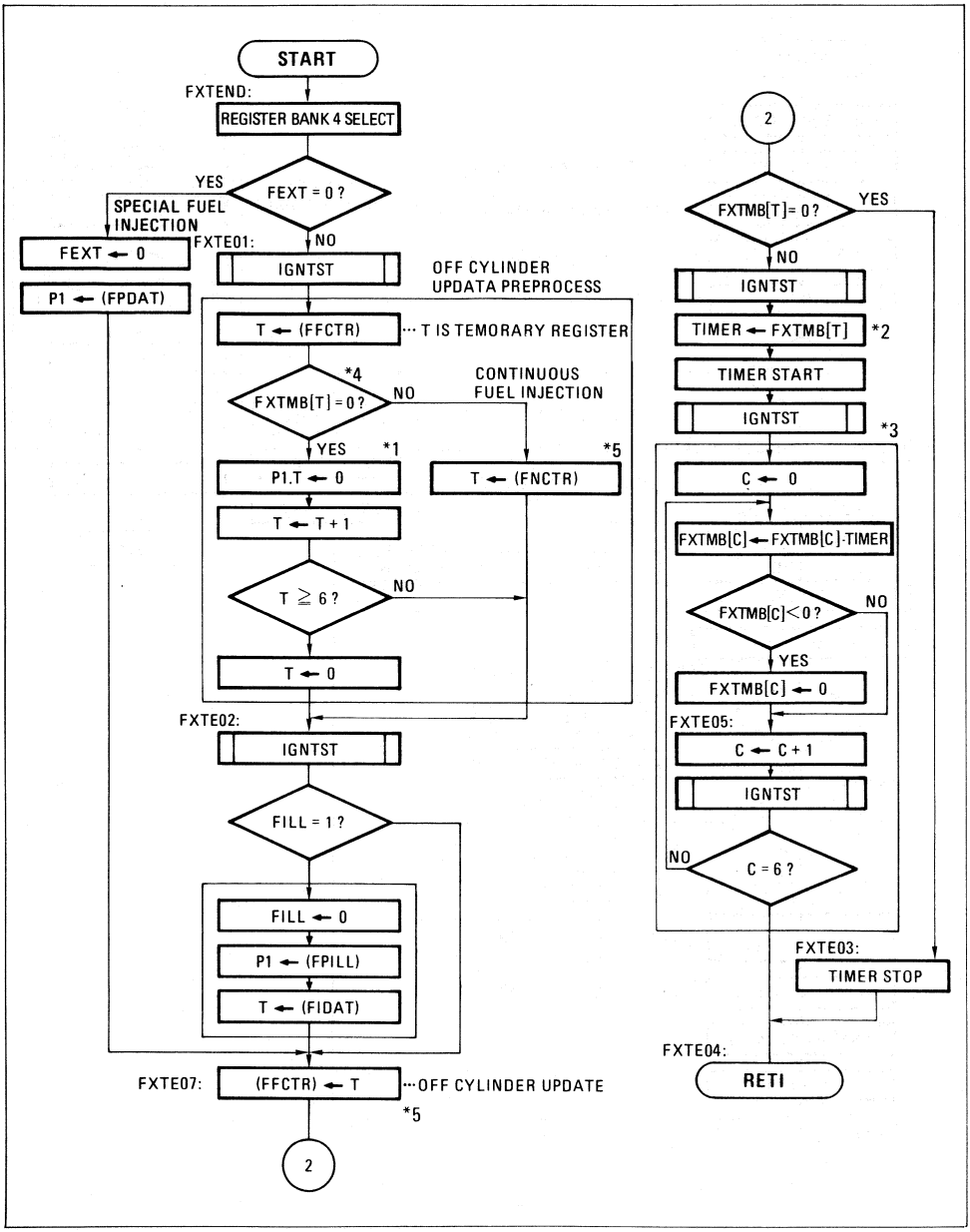

START
FXTEND:
REGISTER BANK 4 SELECT
FEXT = 0 ?
YES
SPECIAL FUEL INJECTION
FEXT ← 0
P1 ← (FPDAT)
NO
FXTE01:
IGNTST
OFF CYLINDER UPDATA PREPROCESS
T ← (FFCTR)
··· T IS TEMORARY REGISTER
*4
FXTMB[T] = 0 ?
NO
CONTINUOUS FUEL INJECTION
YES
*1
*5
P1.T ← 0
T ← (FNCTR)
T ← T + 1
T ≧ 6 ?
NO
T ← 0
FXTE02:
IGNTST
FILL = 1 ?
FILL ← 0
P1 ← (FPILL)
T ← (FIDAT)
FXTE07:
(FFCTR) ← T
··· OFF CYLINDER UPDATE
*5
2
2
FXTMB[T] = 0 ?
YES
NO
IGNTST
TIMER ← FXTMB[T]
*2
TIMER START
IGNTST
*3
C ← 0
FXTMB[C] ← FXTMB[C] - TIMER
FXTMB[C] < 0 ?
NO
YES
FXTMB[C] ← 0
FXTE05:
C ← C + 1
IGNTST
NO
C = 6 ?
FXTE03:
TIMER STOP
FXTE04:
RETI

Figure 6

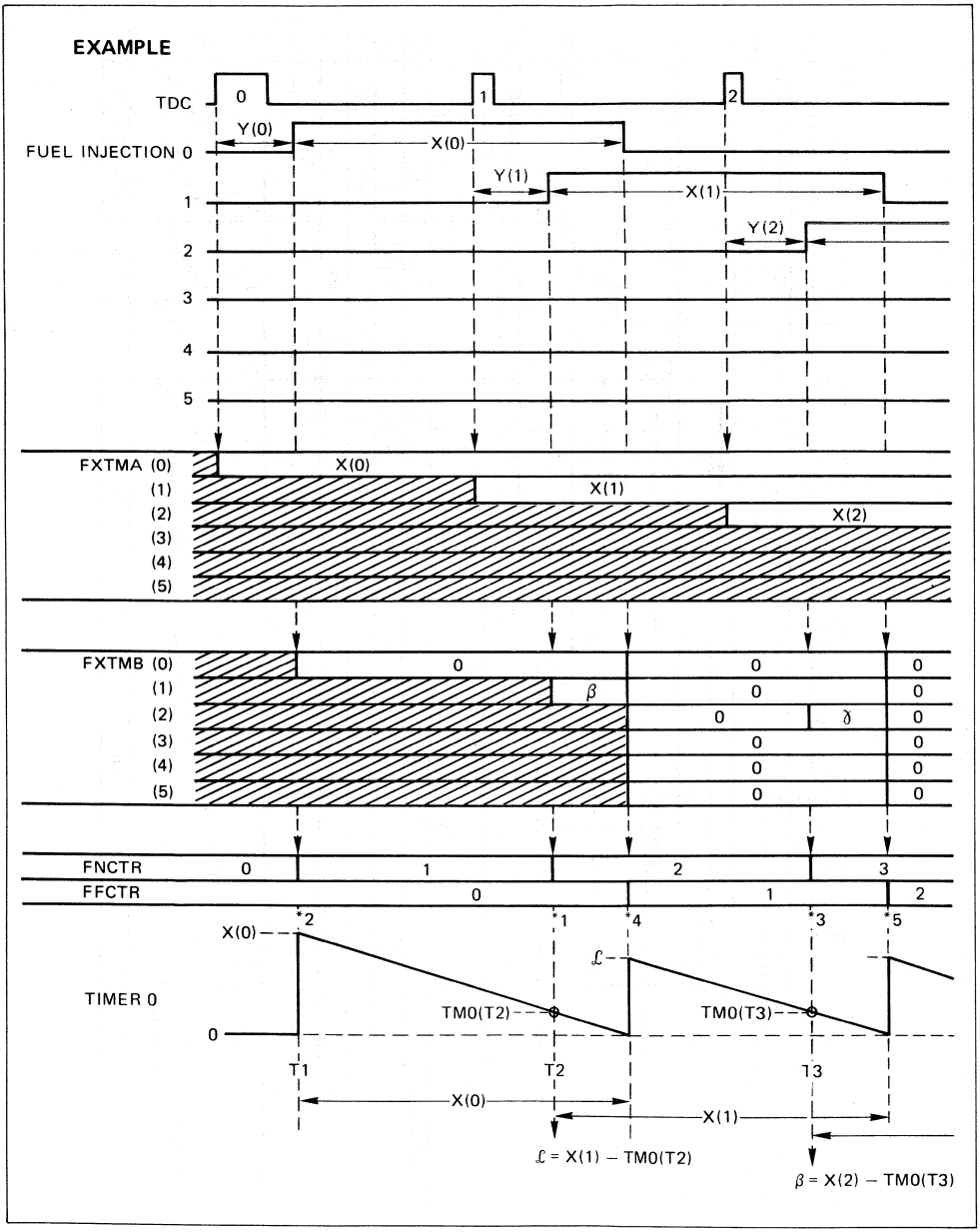

EXAMPLE
TDC
FUEL INJECTION 0
1
2
3
4
5
Y(0)
X(0)
Y(1)
X(1)
Y(2)
FXTMA (0)
(1)
(2)
(3)
(4)
(5)
X(2)
FXTMB (0)
(1)
(2)
(3)
(4)
(5)
β
ϑ
FNCTR
FFCTR
TIMER 0
TM0(T2)
TM0(T3)
T1
T2
T3
ℒ = X(1) − TM0(T2)
β = X(2) − TM0(T3)

Figure 7

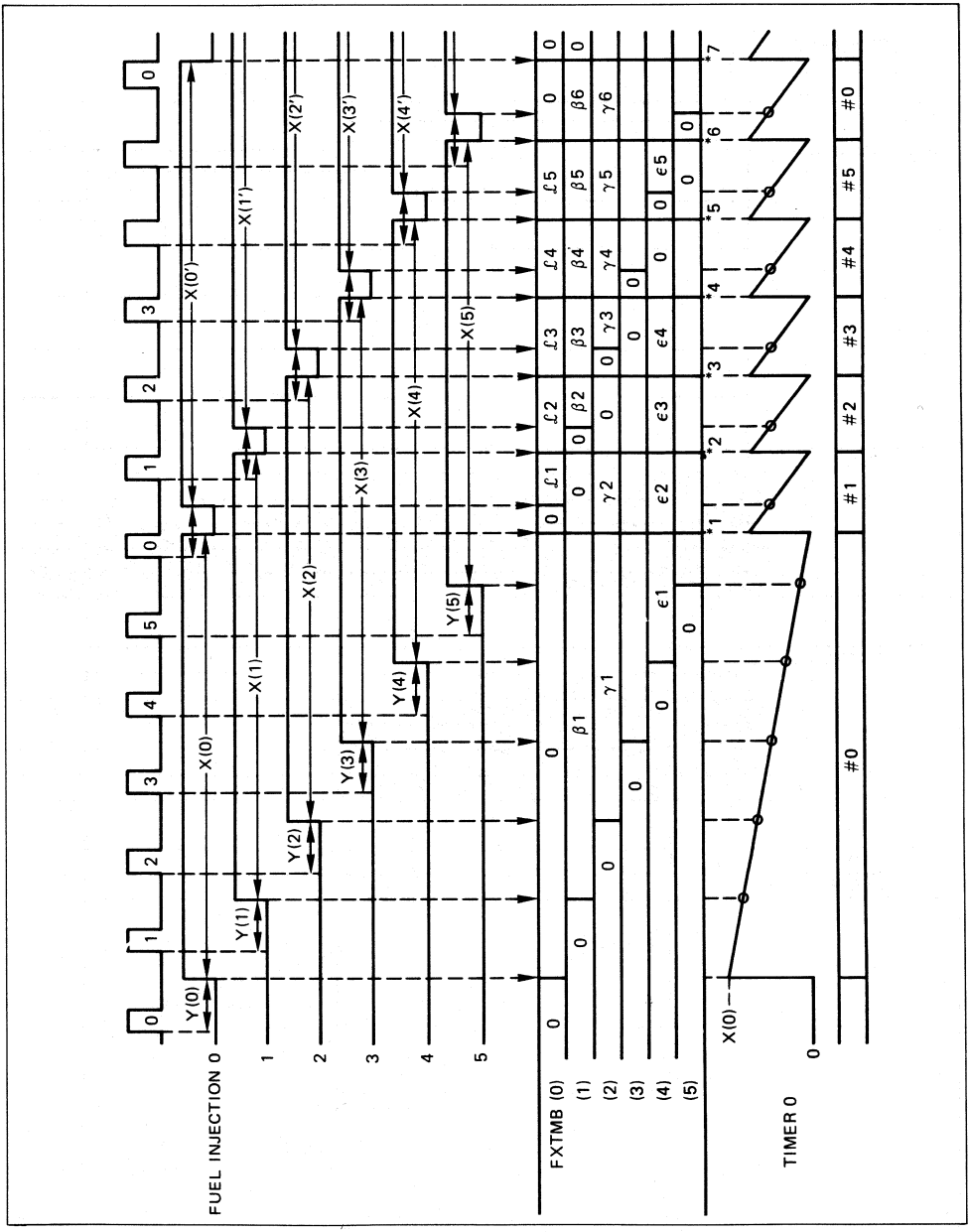

FUEL INJECTION 0
1
2
3
4
5
Y(0)
Y(1)
Y(2)
Y(3)
Y(4)
Y(5)
X(0)
X(1)
X(2)
X(3)
X(4)
X(5)
X(0')
X(1')
X(2')
X(3')
X(4')
FXTMB (0)
(1)
(2)
(3)
(4)
(5)
TIMER 0
#0
#1
#2
#3
#4
#5
#0

Figure 8

Special fuel injection start process

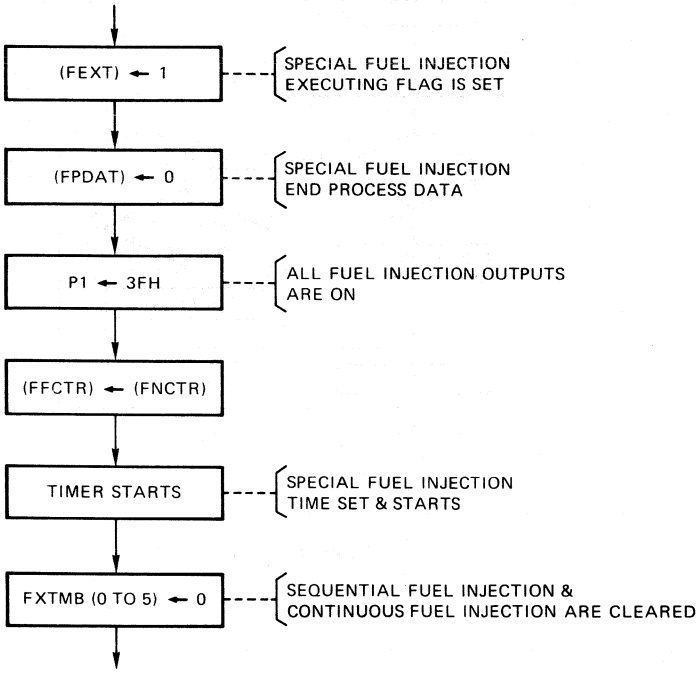


Figure 9

Reference cylinder detection (EXIF1 INTERRUPT)

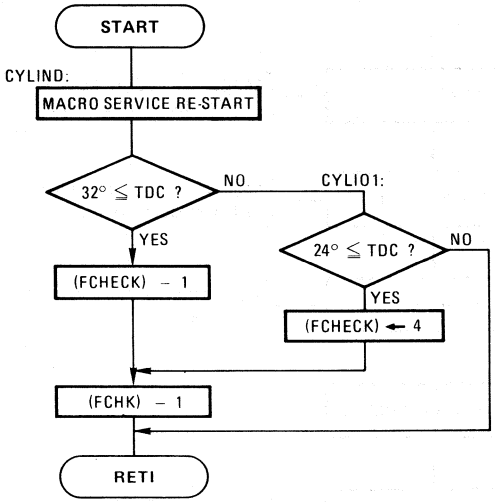


Figure 10

Copyright (C) 1986 NEC Corporation
UPD78310 ASSEMBLER CO.0

SOURCE FILE: INT.DAT
OBJECT FILE: INT.REL
COMMAND : RA310 INT.DAT PC(310) XR

ASSEMBLE LIST

```
STNO  ADRS R OBJECT     M I  SOURCE STATEMENT

   1                          NAME    INT
   2                          PUBLIC  FTBL31
   3                          PUBLIC  FYCSET
   4                          PUBLIC  FXTSET
   5                          PUBLIC  FXTEND
   6                          PUBLIC  D64INT
   7                          PUBLIC  D50INT
   8                          PUBLIC  IGNINT
   9                          PUBLIC  IGNTST
  10                      11                          PUBLIC  ADFINT
  12                         PUBLIC  TMRINT
  13                         PUBLIC  CYLIND
  14                         EXTRN   OBRJOB
                        ;*************************************************************************
                        ;*                                                                       *
                        ;*   INTERRUPT                                                           *
                        ;*                                                                       *
                        ;*      SOURCE  LABEL    I/O  PRIORITY  NOTE                             *
                        ;*                                                                       *
                        ;*      NMI     ------    -     -       unused                          *
                        ;*      EXIF0   FYCSET   I/O    6      fuel injection start            *
                        ;*      EXIF1   CYLIND    I     7      cylinder check                  *
                        ;*      EXIF2   ------    -     8      unused                          *
                        ;*      WDT     ------    -     -      unused                          *
                        ;*      TBF     ------    -     x      unused                          *
                        ;*      TMF0    FXTEND    O     4      fuel injection end              *
                        ;*      TMF1    TMRINT   I/O    5      1 ms  timer interrupt           *
                        ;*      TMF2    ------    -     x      unused                          *
                        ;*      CRF00   IGNINT    O     0      ignite timing                   *
                        ;*      CRF01   FXTSET    O     1      fuel injection output           *
                        ;*      CRF10   D50INT    O     2      duty output 50 ms               *
                        ;*      CRF11   D64INT    O     3      duty output 6.4 ms              *
                        ;*      SEF     ------    -     x      unused                          *
                        ;*      SRF     ------    -     x      unused                          *
                        ;*      STF     ------    -     x      unused                          *
                        ;*      ADF     ADFINT    I     9      internal A/D input              *
                        ;*                                                                       *
                        ;*   MACRO SERVICE                                                       *
                        ;*      NUMBER  SOURCE                  NOTE                             *
                        ;*      0       TM0                     fuel injection X timer routine  *
                        ;*                                              (TM0 overflow counter)  *
                        ;*      2       FXIF1                   CYLINDER CHECK                  *
                        ;*                                              (UDC0 capture)          *
                        ;*      3       CRF10                   duty output 50ms timer routine  *
                        ;*                                              (UDC1 overflow counter) *
                        ;*      4       ADF                     A/D data read                   *
                        ;*                                                                       *
                        ;*                                                                       *
                        ;*************************************************************************
                        ;---------------------------------------
  15        0020        FAPUL   EQU     32                      ; TDC A PULSE
  16        0018        FBPUL   EQU     24                      ; TDC B PULSE
                        ;---------------------------------------
                    1   $       INCLUDE(B:RAM.INC)
  17        E000    1   ADCNV   EQU     0E000H                  ; external A/D converter.
  18        FFB0    1   EXTP0   EQU     0FFB0H                  ; external parallel port 0
  19        FFB1    1   EXTP1   EQU     0FFB1H                  ; external parallel port 1
  20        FFB2    1   EXTP2   EQU     0FFB2H                  ; external parallel port 2
  21        FFB3    1   EXTCM   EQU     0FFB3H                  ; external port command
  22        FE20    1   STACK   EQU     0FE20H                  ; STACK POINTER TOP ADDRESS
  23        FE20    1   FNCTR   EQU     0FE20H                  ; output on counter
  24        FE21    1   FFCTR   EQU     0FE21H                  ; output off counter
  25        FE22    1   FYCTR   EQU     0FE22H                  ; Y COUNTER data area
  26        FE28    1   FXTMA   EQU     0FE28H                  ; X TIMER data A
  27        FE3A    1   FXTMB   EQU     0FE3AH                  ; X TIMER data B
  28        FE4C    1   FPDAT   EQU     0FE4CH                  ; EXT PORT DATA
  29        FE4D    1   FPILL   EQU     0FE4DH                  ; ILLEGAL PORT DATA
  30        FE4E    1   FCHECK  EQU     0FE4EH                  ; CHECK CYLINDER NUMBER
```

UPD78310 ASSEMBLER CO.0

```
                FE4F    1  DMODE   EQU     0FE4FH          ; duty mode.
                FE50    1  DON64   EQU     0FE50H          ; duty on time.   (6.4ms)
                FE52    1  DOF64   EQU     0FE52H          ; duty off time.  (6.4ms)
                FE54    1  DONT1   EQU     0FE54H          ; duty on time.   (50ms:output=P00)
                FE56    1  DONT2   EQU     0FE56H          ; duty on time.   (50ms:output=P01)
                FE58    1  DNXT1   EQU     0FE58H          ; duty on time.   (50ms:work area)
                FE5A    1  DNXT2   EQU     0FE5AH          ; duty off time.  (50ms:work area)
                FE5C    1  IACNT   EQU     0FE5CH          ; timer A
                FE5E    1  IRCNT   EQU     0FE5EH          ; timer R
                FE62    1  REVOL   EQU     0FE62H          ; ENGINE REVOLUTION (r.p.m.)
                FE64    1  RNCPT   EQU     0FE64H          ; NEWEST CAPTURE
                FE67    1  RFOVC   EQU     0FE67H          ; FREE RUNNING COUNTER OVERFOW COUNTER
                FE68    1  ROCPT   EQU     0FE68H          ; BEFORE CAPTURE
                FE6B    1  AWAIT   EQU     0FE6BH          ; INTERNAL A/D. wait data area
                FE6E    1  ADTBL   EQU     0FE6EH          ; A/D data area.
                FE79    1  ACNTR   EQU     0FE79H          ; 100ms EXTERNAL A/D SELECT COUNTER.
                FE7A    1  SCNTR   EQU     0FE7AH          ; SPEED PULSE COUNTER.
                FE7C    1  SCAPT   EQU     0FE7CH          ; SPEED PULSE CAPTURE.
                FE7E    1  STIMR   EQU     0FE7EH          ; 700ms timer
                FE7F    1  TCNTR   EQU     0FE7FH          ; 20ms counter.
                FE80    1  PMDAT   EQU     0FE80H          ; PULSE MOTOR DATA.
                FE81    1  EFLG0   EQU     0FE81H          ; flag 0
                FE82    1  EFLG1   EQU     0FE82H          ; flag 1
                FE83    1  CYDAT   EQU     0FE83H          ; cylinder data
                FE84    1  EWRK0   EQU     0FE84H          ; MACRO SERVISE Dummy RAM
                FE85    1  EFLG2   EQU     0FE85H          ; flag 2
                FE86    1  JFG1    EQU     0FE86H          ;
                FE87    1  JFG2    EQU     0FE87H          ;
                FE88    1  JFG3    EQU     0FE88H          ;
                FE89    1  JFG4    EQU     0FE89H          ;
                FE8A    1  JFGA    EQU     0FE8AH          ;
                FE8B    1  JFG5    EQU     0FE8BH          ;
                FE8C    1  JFG6    EQU     0FE8CH          ;
                FE8D    1  FIDAT   EQU     0FE8DH          ; fuel injection illegal cylinder data.
                FE90    1  T1      EQU     0FE90H          ;
                FE91    1  T3      EQU     0FE91H          ;
                FE92    1  T2      EQU     0FE92H          ;
                FE93    1  T5      EQU     0FE93H          ;
                FE94    1  T4      EQU     0FE94H          ;
                FE95    1  T7      EQU     0FE95H          ;
                FE96    1  T6      EQU     0FE96H          ;
                FE97    1  T8      EQU     0FE97H          ;
                FE98    1  T9      EQU     0FE98H          ;
                FE99    1  T10     EQU     0FE99H          ;
                FEBC    1  B4R12   EQU     0FEBCH          ; RB = 4  R12
                FEBD    1  B4R13   EQU     0FEBDH          ; RB = 4  R13
                FEF0    1  MSP00   EQU     0FEF0H          ; MSP0
                FEF2    1  MSC00   EQU     0FEF2H          ; SFRP0
                FEF3    1  SFRP0   EQU     0FEF3H          ; MSC0
                FEF8    1  MSP02   EQU     0FEF8H          ; MSP2
                FEFA    1  MSC02   EQU     0FEFAH          ; SFRP2
                FEFB    1  SFRP2   EQU     0FEFBH          ; MSC2
                FEFC    1  MSP03   EQU     0FEFCH          ; MSP3
                FEFE    1  MSC03   EQU     0FEFEH          ; SFRP3
                FEFF    1  SFRP3   EQU     0FEFFH          ; MSC3
                FEE0    1  MSP04   EQU     0FEE0H          ; MSP4
                FEE2    1  MSC04   EQU     0FEE2H          ; SFRP4
                FEE3    1  SFRP4   EQU     0FEE3H          ; MSC4
                                   CSEG
                        ;-------------------------------------------------------
                        ;       PULSE (31) PORT TABLE
                        ;            use for fuel injection routine
                        ;-------------------------------------------------------
0000                    FTBL31:
0000    01                      DB      00000001B       ; 0
0001    02                      DB      00000010B       ; 1
0002    04                      DB      00000100B       ; 2
0003    08                      DB      00001000B       ; 3
0004    10                      DB      00010000B       ; 4
0005    20                      DB      00100000B       ; 5
```

```
97                              $       EJECT
                                        CSEG
                                ;*******************************************************
                                ;*      EXIF0  external interrupt 0                    *
                                ;*               PULSE (21) UP EDGE                    *
                                ;*******************************************************
                                ;-------------------------------------------------------
                                ;       fuel injection                  ; Y counter set
                                ;-------------------------------------------------------
 98  0006                       FYCSET:
 99  0006  05AC                           SEL     RB4                     ;
100  0008  6F1C78                         CMP     UDC0L, #120             ; RB = 4 ;
101  000B  8304                           BL      $FYCS00                 ; if( UDC0 >= 120 )
102  000D  0C1C0000                       MOVW    UDC0, #0                ;
103  0011                       FYCS00:                                   ;       UDC = 0 ;
104  0011  2020                           MOV     A, FNCTR                ;
105  0013  08A28525                       BF      EFLG2. 2, $FYCS02       ; if( EFLG2. 2 == 1 ) {
106  0017  A285                           CLR1    EFLG2. 2                ;    EFLG2. 2 = 0 ;
107  0019  9F4E                           CMP     A, FCHECK               ;    if( FNCTR != FCHECK ) {
108  001B  811F                           BE      $FYCS02                 ;       /* error */
109  001D  204E                           MOV     A, FCHECK               ;       FNCTR = FCHECK ;
110  001F  2220                           MOV     FNCTR, A                ;       FIDAT = FHCECK ;
111  0021  228D                           MOV     FIDAT, A                ;
112  0023  08AF8013                       BF      TMC0. 7, $FYCS01        ;       if( timer counting ) {
113  0027  B385                           SET1    EFLG2. 3                ;           EFLG2. 3 = 1 ;
114  0029  3A4D00                         MOV     FPILL, #0               ;           FPILL = 0 ;
115  002C  673AFE                         MOVW    RP7, #FXTMB             ;
116  002F  BB12                           MOV     R3, #18                 ;           FXTMB(0~5) = 0 ;
117  0031  B900                           MOV     R1, #0                  ;
118  0033                       FYCS03:                                   ;
119  0033  51                             MOV     (HL+), A                ;
120  0034  33FD                           DBNZ    B, $FYCS03              ;
121  0036  204E                           MOV     A, FCHECK               ;
122  0038  1402                           BR      $FYCS02                 ;       }
123  003A                       FYCS01:                                   ;       else {
124  003A  2221                           MOV     FFCTR, A                ;           FFCNT = FCHECK ;
125  003C                       FYCS02:                                   ;       }
126  003C  0A1022FE                       MOV     A, FYCTR(A)             ; }
127  0040  220A                           MOV     CR01L, A                ; CR01 = FYCTR( FNCTR )
128  0042  089FC2                         CLR1    CRIC01. 7               ; interrupt flag reset
129  0045  089EC2                         CLR1    CRIC01. 6               ; enable interrupt "CRF01"
                                ;-------------------------------------------------------
                                ;       engine revolution               ; capture data
                                ;-------------------------------------------------------
130  0048                       REVINT:
131  0048  08AF6008                       BF      FRCC. 7, $RINT01        ; if( (FRCC. 7==1) && (CPT0H. 7==1) )
132  004C  771105                         BT      CPT0H. 7, $RINT01       ;
                                                                           ;
133  004F  2667                           INC     RFOVC                   ;     RFOVC = RFOVC + 1
134  0051  089F60                         CLR1    FRCC. 7                 ;     FRCC. 7 = 0
135  0054                       RINT01:
136  0054  1C10                           MOVW    AX, CPT0                ; RNCPT = RFOVC, CPT0
137  0056  1A64                           MOVW    RNCPT, AX               ;
138  0058  386766                         MOV     RNCPT+2, RFOVC          ;
                                ;
                                ;  JOB REGISTRATION ( ENGINE REVOLUTION MAIN ROUTINE )
                                ;
                                ;-------------------------------------------------------
                                ;       A/D START (3)                   ; internal A/D converter
                                ;-------------------------------------------------------
139  005B                       ADSTR3:
140  005B  2020                           MOV     A, FNCTR                ; A = FNCTR
141  005D  AF00                           CMP     A, #0                   ;
142  005F  8104                           BZ      $ADS300                 ; if( FNCTR == 0 OR FNCTR == 3 )
143  0061  AF03                           CMP     A, #3                   ;
144  0063  801E                           BNZ     $ADS302                 ;
145  0065                       ADS300:
146  0065  08BF6811                       BT      ADM. 7, $ADS301         ;    if( CS == 0 )
147  0069  2B6881                         MOV     ADM, #10000001B         ;       ADM = 80H (A/D channel = 0)
148  006C  2BE023                         MOV     ADIC, #23H              ;       ADF flag clear
149  006F  3AE201                         MOV     MSC04, #1               ;       MSC04 = 1
150  0072  0CE071FE                       MOVW    MSP04, #ADTBL+3         ;       MSP04 = ADTBL+3
151  0076  A682                           CLR1    EFLG1. 6                ;       A/D wait flag reset.
152  0078  1409                           BR      $ADS302                 else
153  007A                       ADS301:                                     A/D wait flag set.
154  007A  B682                           SET1    EFLG1. 6                AWAIT = 80H (next A/D channel = 0)
155  007C  3A6B81                         MOV     AWAIT, #10000001B       ;       AWAIT+1 = ADTBL+3 (== next MSP04)
156  007F  0C6C71FE                       MOVW    AWAIT+1, #ADTBL+3
157  0083                       ADS302:
                                ;
                                ;-------------------------------------------------------
                                ;
158  0083  089FC8                         CLR1    EXIC0. 7                ; interrupt flag reset ;
159  0086  2C8E04                         BR      !EITEND                 ;interrupt end
160                                       CSEG
```

UPD78310 ASSEMBLER CO.0

```
                        $       EJECT
                        ;*******************************************************
                        ;*      CRF01  CR01 == UDC0                            *
                        ;*               Y counter count up                    *
                        ;*******************************************************
                        ;-------------------------------------------------------
                        ;       fuel injection                  ; X timer set
                        ;-------------------------------------------------------
  161  0089             FXTSET:                                 ;
  162  0089  05AC              SEL     RB4                      ;
  163  008B  2020              MOV     A, FNCTR                 ; PORT( FNCTR ) = 1 ;
  164  008D  0A100000          MOV     A, FTBL31(A)             ;
  165  0091  2451              MOV     R5, R1                   ;
  166  0093  9E01              OR      A, P1                    ;
  167  0095  2201              MOV     P1, A                    ;
  168  0097  204D              MOV     A, FPILL                 ;
  169  0099  8E15              OR      R1, R5                   ; FPILL.FNCTR = 1 ;
  170  009B  224D              MOV     FPILL, A                 ;
  171  009D  20F2              MOV     A, MSC00                 ; timer read ;
  172  009F  2421              MOV     R2, R1                   ;    /* R2 = MSC0 */
  173  00A1  1188              MOVW    AX, TM0                  ;    /* AX = TM0  */
  174  00A3  03AF09            BF      A.7, $FXTS01             ; if( TM0 under flow && (R2 != 0) )
  175  00A6  08AFCE05          BF      TMIC0.7, $FXTS01         ;
  176  00AA  8C22              AND     R2, R2                   ;
  177  00AC  8101              BZ      $FXTS01                  ;
  178  00AE  CA                DEC     R2                       ;       R2 -- ;
  179  00AF                    FXTS01:                          ;
  180  00AF  8C22              AND     R2, R2                   ; if( R2 == 0 )
  181  00B1  8005              BNZ     $FXTS02                  ;       AX = 0
  182  00B3  600000            MOVW    RP0, #0                  ; else
  183  00B6  1401              BR      $FXTS03                  ;       R2 --
  184  00B8                    FXTS02:                          ;
  185  00B8  CA                DEC     R2                       ;
  186  00B9                    FXTS03:                          ;
  187  00B9  28EF02            CALL    !IGNTST                  ; CALL IGNTST ;
                                                               ;
  188  00BC  24C8              MOVW    RP6, RP0                 ; RP6 = AX  /// R2,R13,R12 <-- TIMER
                                                               ;
  189  00BE  2020              MOV     A, FNCTR                 ;
  190  00C0  2441              MOV     R4, R1                   ;
  191  00C2  2431              MOV     R3, R1                   ; B = FNCTR
  192  00C4  8831              ADD     R3, R1                   ;
  193  00C6  8831              ADD     R3, R1                   ; /*    R2,RP6 : TIMER COUNT  */
  194  00C8  08AF8077          BF      TMC0.7, $FXTS04          ; if( timer counting ) {
                                                               ;
  195  00CC  0A3028FE          MOV     A, FXTMA(B)              ;     FXTMB( FNCTR ) =
  196  00D0  9ABC              SUB     A, B4R12                 ;         FXTMA( FNCTR ) - TM0
  197  00D2  0AB03AFE          MOV     FXTMB(B), A              ;
  198  00D6  0A3029FE          MOV     A, FXTMA+1(B)            ;
  199  00DA  9BBD              SUBC    A, B4R13                 ;
  200  00DC  0AB03BFE          MOV     FXTMB+1(B), A            ;
  201  00E0  0A302AFE          MOV     A, FXTMA+2(B)            ;
  202  00E4  8B12              SUBC    R1, R2                   ;
  203  00E6  0AB03CFE          MOV     FXTMB+2(B), A            ;
  204  00EA  824F              BNC     $FXTS07                  ;    if( FXTMB( FNCTR ) < 0 ) {
  205  00EC  28EF02            CALL    !IGNTST                  ;       CALL IGNTST ;
  206  00EF  B900              MOV     R1, #0                   ;
  207  00F1  0AB03AFE          MOV     FXTMB(B), A              ;       FXTMB( FNCTR ) = 0 ;
  208  00F5  0AB03BFE          MOV     FXTMB+1(B), A            ;
  209  00F9  0AB03CFE          MOV     FXTMB+2(B), A            ;
  210  00FD  08A1850F          BF      EFLG2.1, $FXTS08         ;       if( EFLG2.1 == 1 ) {

  211  0101  28EF02            CALL    !IGNTST                  ;           CALL IGNTST ;
  212  0104  2621              INC     FFCTR                    ;           FFCTR ++ ;
  213  0106  6F2106            CMP     FFCTR, #6                ;
  214  0109  8363              BC      $FXTS05                  ;
  215  010B  3A2100            MOV     FFCTR, #0                ;
  216  010E  145E              BR      $FXTS05                  ;       else {
  217  0110                    FXTS08:                          ;
  218  0110  08A3855A          BF      EFLG2.3, $FXTS05         ;           if( EFLG2.3 == 1 ) {
  219  0114  A385              CLR1    EFLG2.3                  ;               EFLG2.3 = 0 ;
  220  0116  388D21            MOV     FFCTR, FIDAT             ;               FFCTR = FIDAT ;
  221  0119  D5                MOV     A, R5                    ;
  222  011A  2201              MOV     P1, A                    ;               PORT = 1 ;
  223  011C  0A302AFE          MOV     A, FXTMA+2(B)            ;
  224  0120  C1                INC     R1                       ;               timer = FXTMA( FNCTR ) ;
  225  0121  22F2              MOV     MSC00, A                 ;
  226  0123  0A3028FE          MOV     A, FXTMA(B)              ;
```

```
UPD78310 ASSEMBLER CO.0

227  0127  2401              MOV     R0,R1                   ;
228  0129  0A3029FE          MOV     A,FXTMA+1(B)            ;
229  012D  138A              MOVW    MD0,AX                  ;
230  012F  2BCE21            MOV     TMIC0,#21H              ;                timer start ;
231  0132  2B8084            MOV     TMC0,#84H               ;
232  0135  0B8AFFFF          MOVW    MD0,#0FFFFH             ;                }
233  0139  1433              BR      $FXTS05                 ;           }
234  013B                  FXTS07:                           ;     }
235  013B  204C              MOV     A,FPDAT                 ;     else {
236  013D  8E15              OR      R1,R5                   ;        FPDAT.FNCTR = 1 ;
237  013F  224C              MOV     FPDAT,A                 ;     }
238  0141  142B              BR      $FXTS05                 ; }
239  0143                  FXTS04:                           ; else {
240  0143  0A302AFE          MOV     A,FXTMA+2(B)            ;        MSC0,MD0 = FXTMA( FNCTR ) ;
241  0147  C1                INC     R1                      ;
242  0148  22F2              MOV     MSC00,A                 ;
                                                           ;
243  014A  0A3028FE          MOV     A,FXTMA(B)              ;
244  014E  2401              MOV     R0,R1                   ;
245  0150  0A3029FE          MOV     A,FXTMA+1(B)            ;
246  0154  138A              MOVW    MD0,AX                  ;
247  0156  2BCE21            MOV     TMIC0,#21H              ;
248  0159  2B8084            MOV     TMC0,#84H               ;        timer start ;
249  015C  0B8AFFFF          MOVW    MD0,#0FFFFH             ;        MD0 = 0FFFFH    ( full count ) ;
                                                           ;
250  0160  B900              MOV     R1,#0                   ;        FXTMB( FNCTR ) = 0 ;
251  0162  0AB03AFE          MOV     FXTMB(B),A              ;
252  0166  0AB03BFE          MOV     FXTMB+1(B),A            ;
253  016A  0AB03CFE          MOV     FXTMB+2(B),A            ; }
254  016E                  FXTS05:                           ;

255  016E  28EF02            CALL    !IGNTST                 ; CALL IGNTST ;
256  0171  D4                MOV     A,R4                    ; FNCTR ++ ;
257  0172  C1                INC     R1                      ;
258  0173  AF06              CMP     A,#6                    ;
259  0175  8302              BC      $FXTS06                 ;
260  0177  B900              MOV     R1,#0                   ;
261  0179                  FXTS06:                           ;
262  0179  2220              MOV     FNCTR,A                 ;
                        ;
                        ;
                        ;
263  017B  089FC2            CLR1    CRIC01.7                ; interrupt flag reset ;
264  017E  088EC2            SET1    CRIC01.6                ; disable interrupt "CRF01"
265  0181  2C8E04            BR      !EITEND                 ; return ;
                        $       EJECT
266                             CSEG
                        ;*********************************************************
                        ;*      TM0 timer count up interrupt                     *
                        ;*              X timer count up                         *
                        ;*********************************************************
                        ;---------------------------------------------------------
                        ;       fuel injection                  ; injection output end
                        ;---------------------------------------------------------
267  0184                  FXTEND:                           ;
268  0184  05AC              SEL     RB4                     ;
269  0186  08A1850A          BF      EFLG2.1,$FXTE01         ;
270  018A  A185              CLR1    EFLG2.1                 ;
271  018C  204C              MOV     A,FPDAT                 ;
272  018E  2201              MOV     P1,A                    ;
273  0190  2021              MOV     A,FFCTR                 ;
274  0192  143E              BR      $FXTE07                 ;
275  0194                  FXTE01:                           ;

276  0194  28EF02            CALL    !IGNTST                 ;
277  0197  2021              MOV     A,FFCTR                 ; if( FXTMB( FFCTR ) == 0 )
278  0199  2431              MOV     R3,R1                   ;
279  019B  8831              ADD     R3,R1                   ;
280  019D  8831              ADD     R3,R1                   ;
281  019F  0A303AFE          MOV     A,FXTMB(B)              ;
282  01A3  0A3E3BFE          OR      A,FXTMB+1(B)            ;
283  01A7  0A3E3CFE          OR      A,FXTMB+2(B)            ;
284  01AB  2020              MOV     A,FNCTR                 ;
285  01AD  8015              BNZ     $FXTE02                 ;
286  01AF  2021              MOV     A,FFCTR                 ;        PORT( FFCTR ) = 0
287  01B1  0A100000          MOV     A,FTBL31(A)             ;
288  01B5  ADFF              XOR     A,#0FFH                 ;
289  01B7  9C01              AND     A,P1                    ;
290  01B9  2201              MOV     P1,A                    ;
```

UPD78310 ASSEMBLER C0.0

```
291  01BB  2021          MOV     A,FFCTR          ;        A = FFCTR ;
292  01BD  C1            INC     R1               ;        A = A + 1 ;
293  01BE  AF06          CMP     A,#6             ;
294  01C0  8302          BC      $FXTE02          ;        if( A >= 6 )
295  01C2  B900          MOV     R1,#0            ;            A = 0
296  01C4              FXTE02:                    ;        FFCTR = A

297  01C4  28EF02        CALL    !IGNTST          ;
298  01C7  08A38507      BF      EFLG2.3,$FXTE07  ;       if( EFLG2.3 == 1 ) {
299  01CB  A385          CLR1    EFLG2.3          ;           EFLG2.3 = 0 ;
300  01CD  384D01        MOV     P1,FPILL         ;           P1 = FPILL ;
301  01D0  208D          MOV     A,FIDAT          ;           FFCTR = FIDAT ;
302  01D2              FXTE07:                    ;       }
303  01D2  2221          MOV     FFCTR,A          ; else
304  01D4  2431          MOV     R3,R1            ;        FFCTR = FNCTR
305  01D6  8831          ADD     R3,R1            ;
306  01D8  8831          ADD     R3,R1            ;
                                                 ;
307  01DA  0A303CFE      MOV     A,FXTMB+2(B)     ; if( FXTMB( FFCTR ) != 0 )
308  01DE  0A3E3BFE      OR      A,FXTMB+1(B)     ;
309  01E2  0A3E3AFE      OR      A,FXTMB(B)       ;
310  01E6  814C          BZ      $FXTE03          ;
                                                 ;
311  01E8  28EF02        CALL    !IGNTST          ;
312  01EB  0A303CFE      MOV     A,FXTMB+2(B)     ; MSC0,MD0 = FXTMB( FFCTR )
313  01EF  2421          MOV     R2,R1            ;
314  01F1  C1            INC     R1               ;
315  01F2  22F2          MOV     MSC00,A          ;
316  01F4  0A303AFE      MOV     A,FXTMB(B)       ;
317  01F8  2401          MOV     R0,R1            ;
318  01FA  0A303BFE      MOV     A,FXTMB+1(B)     ;
319  01FE  138A          MOVW    MD0,AX           ;
320  0200  2448          MOVW    RP2,RP0          ;                    /// R2,R5,R4 == TIMER
321  0202  2BCE21        MOV     TMIC0,#21H       ; timer start
322  0205  2B8084        MOV     TMC0,#84H        ;
323  0208  0B8AFFFF      MOVW    MD0,#0FFFFH      ; MD0 = 0FFFFH    ( full count )
                                                 ;
324  020C  28EF02        CALL    !IGNTST          ;
325  020F  673AFE        MOVW    RP7,#FXTMB       ; HL = FXTMB
326  0212  BB06          MOV     R3,#6            ; B = 6 ( COUNTER )
327  0214              FXTE05:                    ; while( B != 0 )
328  0214  D4            MOV     A,R4             ;
329  0215  169A          SUB     (HL+),A          ; FXTMB(B) = FXTMB(B) - timer
330  0217  D5            MOV     A,R5             ;                           (R2,R5,R4)
331  0218  169B          SUBC    (HL+),A          ;
332  021A  D2            MOV     A,R2             ;
333  021B  169B          SUBC    (HL+),A          ;
334  021D  820E          BNC     $FXTE06          ;    if( FXTMB(B) < 0 )

335  021F  B900          MOV     R1,#0            ;        FXTMB(B) = 0
336  0221  0AA0FDFF      MOV     -3(HL),A         ;
337  0225  0AA0FEFF      MOV     -2(HL),A         ;
338  0229  0AA0FFFF      MOV     -1(HL),A         ;
339  022D              FXTE06:                    ;
340  022D  28EF02        CALL    !IGNTST          ;
341  0230  33E2          DBNZ    B,$FXTE05        ;    B = B - 1
                                                 ;
342  0232  1403          BR      $FXTE04          ;
343  0234              FXTE03:                    ; else
344  0234  2B8004        MOV     TMC0,#04H        ;        timer stop
345  0237              FXTE04:                    ;
                       ;
                       ;
                       ;
346  0237  089FCE        CLR1    TMIC0.7          ; interrupt flag reset ;
347  023A  2C8E04        BR      !EITEND          ; interrupt end
```

UPD78310 ASSEMBLER CO. 0

```
348                             $       EJECT
                                        CSEG
                                ;*******************************************************
                                ;*      CRF11   CR11 == UDC1                           *
                                ;*******************************************************
                                ;-------------------------------------------------------
                                ;       DUTY 6.4ms                      ; timer set
                                ;-------------------------------------------------------
349  023D                       D64INT:
350  023D  05AC                        SEL     RB4                     ; SELECT REGISTER BANK 4
351  023F  720013                      BT      P0.2, $D6IT01

352  0242  1C1E                        MOVW    AX, UDC1
353  0244  2428                        MOVW    RP1, RP0
354  0246  1D50                        ADDW    AX, DON64
355  0248  1A0E                        MOVW    CR11, AX                ; COMPARE REGISTER SET
356  024A  240A                        MOVW    RP0, RP1                ;   CR11 = UDC1 + DON64 ( port on time )
357  024C  2D0032                      ADDW    AX, #3200H              ; PORT OFF TIME SET
358  024F  1A52                        MOVW    DOF64, AX               ;   DOF64 = UDC1 + 3200H ( 6.4ms )
359  0251  B200                        SET1    P0.2                    ; HIGH LEVEL OUTPUT
360  0253  1406                        BR      $D6IT11
361  0255                       D6IT01:
362  0255  1C52                        MOVW    AX, DOF64               ; COMPARE REGISTER SET
363  0257  1A0E                        MOVW    CR11, AX                ;   CR11 = DOF64  ( port off time )
364  0259  A200                        CLR1    P0.2                    ; LOW LEVEL OUTPUT
365  025B                       D6IT11:
366  025B  089FC6                      CLR1    CRIC11.7                ; interrupt flag reset ;
367  025E  57                          RETI                            ; RETURN TO MAIN ROUTINE
                                $       EJECT
368                                     CSEG
                                ;*******************************************************
                                ;*      CRF10  CR10 == UDC1 and MACRO SERVICE END      *
                                ;*             timer count up                          *
                                ;*******************************************************
                                ;-------------------------------------------------------
                                ;       DUTY 50ms                       ; timer set
                                ;-------------------------------------------------------
369  025F                       D50INT:
370  025F  05AC                        SEL     RB4                     ;
371  0261  714F56                      BT      DMODE.1, $D5IT20
372  0264  704F47                      BT      DMODE.0, $D5IT10        ;
373  0267                       D5IT06:
                                                                       ; DUTY MODE == 0
374  0267  B000                        SET1    P0.0                    ; P00, P01 <-- HIGH LEVEL
375  0269  B100                        SET1    P0.1                    ;

376  026B  1C54                        MOVW    AX, DONT1               ;
377  026D  1F56                        CMPW    AX, DONT2               ; DONT1 : DONT2
378  026F  8119                        BE      $D5IT01                 ;       =           GOTO D5IT01
379  0271  8326                        BL      $D5IT02                 ;       <           GOTO D5IT02
                                                                       ;       >
                                ;-------------------------------------------------------
380  0273  1C56                        MOVW    AX, DONT2               ; DONT1 > DONT2
                                                                       ;  TIMER SET "DONT2"
381  0275  28D902                      CALL    !DTRSET                 ;    duty timer set
382  0278  1C54                        MOVW    AX, DONT1               ; DNXT1 = DONT1 - DONT2
383  027A  1E56                        SUBW    AX, DONT2               ;
384  027C  1A58                        MOVW    DNXT1, AX               ;
385  027E  6050C3                      MOVW    RP0, #0C350H            ; DNXT2 = 0C350H - DONT1
386  0281  1E54                        SUBW    AX, DONT1               ;      ( 0C350H = 50ms )
387  0283  1A5A                        MOVW    DNXT2, AX               ;
388  0285  3A4F01                      MOV     DMODE, #1               ; DUTY MODE = 1
389  0288  144B                        BR      $D5IT40                 ;
                                ;-------------------------------------------------------
390  028A                       D5IT01:                                 ; DONT1 = DONT2
                                                                       ;  TIMER SET "DONT1"
391  028A  28D902                      CALL    !DTRSET                 ;      duty timer set
392  028D  6050C3                      MOVW    RP0, #0C350H            ;
393  0290  1E54                        SUBW    AX, DONT1               ; DNXT2 = 0C350H - DONT1
394  0292  1A5A                        MOVW    DNXT2, AX               ;
395  0294  3A4F03                      MOV     DMODE, #3               ; DUTY MODE = 3
396  0297  143C                        BR      $D5IT40                 ;
                                ;-------------------------------------------------------
397  0299                       D5IT02:                                 ; DONT1 < DONT2
                                                                       ;  TIMER SET "DONT1"
398  0299  28D902                      CALL    !DTRSET                 ;      duty timer set
399  029C  1C56                        MOVW    AX, DONT2               ;
400  029E  1E54                        SUBW    AX, DONT1               ; DNXT1 = DONT2 - DONT1
401  02A0  1A58                        MOVW    DNXT1, AX               ;
402  02A2  6050C3                      MOVW    RP0, #0C350H            ;
```

UPD78310 ASSEMBLER CO.0

```
403  02A5  1E56          SUBW    AX, DONT2               ; DNXT2 = 0C350H - DONT2
404  02A7  1A5A          MOVW    DNXT2, AX               ;
405  02A9  3A4F02        MOV     DMODE, #2               ; DUTY MODE = 2
406  02AC  1427          BR      $D5IT40                 ;

                    ;-------------------------------------------------------
407  02AE               D5IT10:                           ; DUTY MODE == 1
408  02AE  A100              CLR1    P0.1                 ; P01 = LOW LEVEL
409  02B0  1C58              MOVW    AX, DNXT1            ; duty timer set "DNXT1"
410  02B2  28D902            CALL    !DTRSET              ;
411  02B5  3A4F03            MOV     DMODE, #3            ; DUTY MODE = 3
412  02B8  141B              BR      $D5IT40              ;
                    ;-------------------------------------------------------
413  02BA               D5IT20:                           ; DUTY MODE == 2 or 3
414  02BA  704F0C            BT      DMODE.0, $D5IT30     ;
                    ;-------------------------------------------------------
                                                          ; DUTY MODE == 2
415  02BD  A000              CLR1    P0.0                 ; P00 = LOW LEVEL
416  02BF  1C58              MOVW    AX, DNXT1            ; duty timer set "DNXT1"
417  02C1  28D902            CALL    !DTRSET              ;
418  02C4  3A4F03            MOV     DMODE, #3            ; DUTY MODE = 3
419  02C7  140C              BR      $D5IT40              ;
                    ;-------------------------------------------------------
420  02C9               D5IT30:
421  02C9  A000              CLR1    P0.0                 ; P00, P01 = LOW LEVEL
422  02CB  A100              CLR1    P0.1                 ;
423  02CD  1C5A              MOVW    AX, DNXT2            ; duty timer set "DNXT2"
424  02CF  28D902            CALL    !DTRSET              ;
425  02D2  3A4F00            MOV     DMODE, #0            ; DUTY MODE = 0
                    ;-------------------------------------------------------
426  02D5               D5IT40:
427  02D5  089FC4            CLR1    CRIC10.7             ; interrupt flag reset ;
428  02D8  57                RETI                         ; RETURN TO MAIN ROUTINE
                    ;-------------------------------------------------------
                    ;       duty timer set subroutine
                    ;-------------------------------------------------------
                    ;       MACRO SERVICE
                    ;               MSC03       : OVER FLOW COUNTER
                    ;               MSP03, SFRP3 : UN USED
                    ;-------------------------------------------------------
429  02D9               DTRSET:                           ;
430  02D9  8808              ADDW    RP0, RP0             ;
431  02DB  3AFE02            MOV     MSC03, #2            ; if( RP0 * 2 > OFFFFH )
432  02DE  8303              BC      $DTRS01              ;       MSC03 = 2
433  02E0  3AFE01            MOV     MSC03, #1            ; else
434  02E3               DTRS01:                           ;       MSC03 = 1
435  02E3  2BC420            MOV     CRIC10, #20H
436  02E6  1D1E              ADDW    AX, UDC1             ;
437  02E8  1A0C              MOVW    CR10, AX             ; CR10 = RP0 * 2
438  02EA  56                RET                          ;
                    $       EJECT
439                         CSEG
                    ;*********************************************************
                    ;*      IGNITE TIMING CONTROL                            *
                    ;*      CR00 INTERRUPT                                   *
                    ;*********************************************************
440  02EB               IGNINT:
441  02EB  28F302            CALL    !IGNITS              ;
442  02EE  57                RETI                         ;
                    ;---------------------------------------
                    ;---------------------------------------
443  02EF               IGNTST:
444  02EF  08AFC022          BF      CRIC00.7, $IGIT04
                    ;---------------------------------------
445  02F3               IGNITS:
446  02F3  3501              PUSH    AX                   ;
447  02F5  49                PUSH    PSW                  ;
448  02F6  730006            BT      P0.3, $IGIT01        ;

449  02F9  B300              SET1    P0.3                 ; IGNITE PORT = LOW LEVEL
450  02FB  1C5C              MOVW    AX, IACNT            ; AX = IACNT
451  02FD  140E              BR      $IGIT03
452  02FF               IGIT01:
453  02FF  A300              CLR1    P0.3                 ; IGNITE PORT = HIGH LEVEL
454  0301  1C5C              MOVW    AX, IACNT            ;
455  0303  1D5E              ADDW    AX, IRCNT            ;
456  0305  2F7800            CMPW    AX, #120             ;
457  0308  8303              BL      $IGIT03              ;
458  030A  2E7800            SUBW    AX, #120             ;
459  030D               IGIT03:
460  030D  1A08              MOVW    CR00, AX             ; INEXT = IRCNT
```

UPD78310 ASSEMBLER CO.0

```
461  030F                    IGIT02:
                             ;       MOV     MSC01, #1               ;
                             ;       MOV     CRIC00, #20H            ; MACRO SERVICE RESTART
462  030F  089FC0                    CLR1    CRIC00.7                ; interrupt flag reset ;
463  0312  48                        POP     PSW                     ;
464  0313  3401                      POP     AX                      ;
465  0315                    IGIT04:
466  0315  56                        RET                             ;

                             $       EJECT
467                          ;       CSEG
                             ;*********************************************************
                             ;*      INTERNAL A/D CONVERTER INTERRUPT                 *
                             ;*      ADF INTERRUPT                                    *
                             ;*********************************************************
                             ;
                             ;       MACRO SERVICE
                             ;               MSC04   : 1 ( ONE TIME )
                             ;               MSP04   : ADTBL + ( 3 , 5 , 6 or 10 )
                             ;               SFRP4   : ADCR
                             ;
468  0316                    ADFINT:
469  0316  08D68204                  BTCLR   EFLG1.6, $AINT01        ; if( A/D WAIT ?)

470  031A  089F68                    CLR1    ADM.7                   ;  A/D CONVERTER STOP
471  031D  57                        RETI                            ;
472  031E                    AINT01:
473  031E  05AC                      SEL     RB4                     ; SELECT REGISTER BANK 4
474  0320  3AE201                    MOV     MSC04, #1               ; MSC04 = 1
475  0323  3C6CE0                    MOVW    MSP04, AWAIT+1          ; MSP04 = ADTBL + ( 3 , 5 , 6 , 10 )
476  0326  206B                      MOV     A, AWAIT                ; ADM = AWAIT ( A/D CONTROL CODE )
477  0328  1268                      MOV     ADM, A                  ;
478  032A  2BE023                    MOV     ADIC, #23H              ; ADF ( A/D end of convert ) flag clear
                                                                     ; macro service restart
479  032D  089FE0                    CLR1    ADIC.7                  ; interrupt flag reset ;
480  0330  57                        RETI                            ; return to main routine

                             $       EJECT
481                          ;       CSEG
                             ;*********************************************************
                             ;*      TMF1   TIMER INTERRUPT ( 1ms )                   *
                             ;*              interval timer                           *
                             ;*********************************************************
482  0331                    TMRINT:
483  0331  05AC                      SEL     RB4                     ;
                             ;-----------------------------------------------------
                             ;       speed count                     ; speed counter increment
                             ;-----------------------------------------------------
484  0333                    SCOUNT:                                 ; SPEED PULSE COUNT
485  0333  080782                    MOV1    CY, EFLG1.7             ; if( up_edge or down_edge )
486  0336  086103                    XOR1    CY, P3.1                ;       increment( speed_pulse_counter )
487  0339  8206                      BNC     $SCNT01                 ;                       (SCNTR)
488  033B  07E87A                    INCW    SCNTR                   ;
489  033E  087782                    NOT1    EFLG1.7                 ;
490  0341                    SCNT01:
                             ;-----------------------------------------------------
                             ;       timer routine                   ; timer branch
                             ;-----------------------------------------------------
491  0341  207F                      MOV     A, TCNTR                ;
492  0343  8811                      ADD     R1, R1                  ; PC = ( TMRTBL + TCNTR*2 )
493  0345  2401                      MOV     R0, R1                  ;
494  0347  B900                      MOV     R1, #0                  ;
495  0349  675003                    MOVW    RP7, #TMRTBL            ;
496  034C  88E8                      ADDW    RP7, RP0                ;
497  034E  056F                      BR      (RP7)                   ;

                             ;-----------------------------------------------------
498  0350                    TMRTBL:
499  0350  A203                      DW      ADSTR1                  ; 0:A/D START (1)
500  0352  D803                      DW      ADSTR6                  ; 1:A/D START (6)
501  0354  A303                      DW      ADINP1                  ; 2:A/D INPUT (1) , A/D START (5)
502  0356  F803                      DW      ADST10                  ; 3:A/D START (10)
503  0358  A203                      DW      ADSTR1                  ; 4:A/D START (1)
504  035A  1804                      DW      PLMOUT                  ; 5:PULSE MOTOR CONTROL
505  035C  AB03                      DW      ADINP1                  ; 6:A/D INPUT (1) , A/D START (5)
506  035E  3B04                      DW      ADSTRT                  ; 7:A/D START (2, 4, 7, 8, 9)
507  0360  7803                      DW      ADINPT                  ; 8:A/D INPUT (2, 4, 7, 8, 9), A/D START (1)
508  0362  2204                      DW      ROVCHK                  ; 9:FRC OVERFLOW CHECK
509  0364  AB03                      DW      ADINP1                  ;10:A/D INPUT (1) , A/D START (5)
510  0366  D803                      DW      ADSTR6                  ;11:A/D START (6)
511  0368  A203                      DW      ADSTR1                  ;12:A/D START (1)
512  036A  F803                      DW      ADST10                  ;13:A/D START (10)
513  036C  AB03                      DW      ADINP1                  ;14:A/D INPUT (1) , A/D START (5)
```

```
514  036E  1804              DW      PLMOUT                  ;15:PULSE MOTOR CONTROL
515  0370  A203              DW      ADSTR1                  ;16:A/D START (1)
516  0372  4E04              DW      SCAPTR                  ;17:SPEED
517  0374  AB03              DW      ADINP1                  ;18:A/D INPUT (1) , A/D START (5)
518  0376  2204              DW      ROVCHK                  ;19:FRC OVERFLOW CHECK

                        $       EJECT
                        ;-------------------------------------------------------
                        ;       A/D INPUT (2, 4, 7, 8, 9)   external
                        ;-------------------------------------------------------
519  0378               ADINPT:                             ;
520  0378  2079                MOV     A, ACNTR            ;

521  037A  AF00                CMP     A, #0               ;
522  037C  8005                BNZ     $AINP01             ;
523  037E  6770FE              MOVW    RP7, #ADTBL+2       ; A/D INPUT (2)
524  0381  141A                BR      $AINP05             ;
525  0383               AINP01:
526  0383  2431                MOV     R3, R1
527  0385  3305                DBNZ    B, $AINP02          ;
528  0387  6772FE              MOVW    RP7, #ADTBL+4       ; A/D INPUT (4)
529  038A  1411                BR      $AINP05             ;
530  038C               AINP02:
531  038C  3305                DBNZ    B, $AINP03          ;
532  038E  6775FE              MOVW    RP7, #ADTBL+7       ; A/D INPUT (7)
533  0391  140A                BR      $AINP05             ;
534  0393               AINP03:
535  0393  3305                DBNZ    B, $AINP04          ;
536  0395  6776FE              MOVW    RP7, #ADTBL+8       ; A/D INPUT (8)
537  0398  1403                BR      $AINP05             ;
538  039A               AINP04:
539  039A  6777FE              MOVW    RP7, #ADTBL+9       ; A/D INPUT (9)
540  039D               AINP05:
541  039D  09F001E0            MOV     A, !ADCNV+1         ;
542  03A1  55                  MOV     (HL), A             ;
                        ;-------------------------------------------------------
                        ;       A/D START (1)           external
                        ;-------------------------------------------------------
543  03A2               ADSTR1:
544  03A2  B900                MOV     R1, #0              ;
545  03A4  09F100E0            MOV     !ADCNV, A                   ; COMMAND SET
546  03A8  2C5F04              BR      !TINT10             ;

                        $       EJECT
                        ;-------------------------------------------------------
                        ;       A/D INPUT (1)           external
                        ;-------------------------------------------------------
547  03AB               ADINP1:
548  03AB  09F000E0            MOV     A, !ADCNV                   ;
549  03AF  2401                MOV     R0, R1              ;
550  03B1  09F001E0            MOV     A, !ADCNV+1         ;
551  03B5  1A6E                MOVW    ADTBL, AX           ;
                        ;-------------------------------------------------------
                        ;       A/D START (5)           internal
                        ;-------------------------------------------------------
552  03B7               ADSTR5:
553  03B7  08BF6811            BT      ADM.7, $ADS501      ; if( CS == 0 )
554  03BB  2B6883              MOV     ADM, #10000011B     ;     ADM = 83H (A/D channel = 1)
555  03BE  2BE023              MOV     ADIC, #23H          ;     ADF flag clear
                                                          ;     macro service start
556  03C1  3AE201              MOV     MSC04, #1           ;     MSC04 = 1
557  03C4  0CE073FE            MOVW    MSP04, #ADTBL+5     ;     MSP04 = ADTBL+5
558  03C8  A682                CLR1    EFLG1.6             ;     A/D wait flag reset.
559  03CA  1409                BR      $ADS502             ;
560  03CC               ADS501:                     ; else
561  03CC  B682                SET1    EFLG1.6     ;     A/D wait flag set.
562  03CE  3A6B83              MOV     AWAIT, #10000011B ;   AWAIT = 83H (next A/D channel = 1)
563  03D1  0C6C73FE            MOVW    AWAIT+1, #ADTBL+5       ;     AWAIT+1 = ADTBL+5 (== next MSP04)
564  03D5               ADS502:
565  03D5  2C5F04              BR      !TINT10     ;
```

UPD78310 ASSEMBLER CO. 0

```
                                $       EJECT
                                ;-------------------------------------------------------
                                ;       A/D START (6)           internal
                                ;-------------------------------------------------------
566  03D8                       ADSTR6:
567  03D8  08BF6811                     BT      ADM. 7, $ADS601         ; if( CS == 0 )
568  03DC  2B6885                       MOV     ADM, #10000101B         ;    ADM = 85H (A/D channel = 2)
569  03DF  2BE023                       MOV     ADIC, #23H              ;    ADF flag clear
                                                                       ;    macro service start
570  03E2  3AE201                       MOV     MSC04, #1               ;    MSC04 = 1
571  03E5  0CE074FE                     MOVW    MSP04, #ADTBL+6         ;    MSP04 = ADTBL+6
572  03E9  A682                         CLR1    EFLG1. 6                ;    A/D wait flag reset.
573  03EB  1409                         BR      $ADS602                 ;
574  03ED                       ADS601:                                 ; else
575  03ED  B682                         SET1    EFLG1. 6                ;    A/D wait flag set.
576  03EF  3A6B85                       MOV     AWAIT, #10000101B       ;    AWAIT = 85H (next A/D channel = 2)
577  03F2  0C6C74FE                     MOVW    AWAIT+1, #ADTBL+6       ;       AWAIT+1 = ADTBL+6 (== next MSP04)
578  03F6                       ADS602:
579  03F6  1467                         BR      $TINT10                 ;

                                $       EJECT
                                ;-------------------------------------------------------
                                ;       A/D START (10)          internal
                                ;-------------------------------------------------------
580  03F8                       ADST10:
581  03F8  08BF6811                     BT      ADM. 7, $AD1001         ; if( CS == 0 )
582  03FC  2B6887                       MOV     ADM, #10000111B         ;    ADM = 87H (A/D channel = 3)
583  03FF  2BE023                       MOV     ADIC, #23H              ;    ADF flag clear
                                                                       ;    macro service start
584  0402  3AE201                       MOV     MSC04, #1               ;    MSC04 = 1
585  0405  0CE078FE                     MOVW    MSP04, #ADTBL+10        ;    MSP04 = ADTBL+10
586  0409  A682                         CLR1    EFLG1. 6                ;    A/D wait flag reset.
587  040B  1409                         BR      $AD1002                 ;
588  040D                       AD1001:                                 ; else
589  040D  B682                         SET1    EFLG1. 6                ;    A/D wait flag set.
590  040F  3A6B87                       MOV     AWAIT, #10000111B       ;    AWAIT = 87H (next A/D channel = 3)
591  0412  0C6C78FE                     MOVW    AWAIT+1, #ADTBL+10      ;    AWAIT+1 = ADTBL+10 (== next MSP04)
592  0416                       AD1002:
593  0416  1447                         BR      $TINT10                 ;

                                $       EJECT
                                ;-------------------------------------------------------
                                ;       PULSE MOTOR CONTROL
                                ;-------------------------------------------------------
594  0418                       PLMOUT:
595  0418  2080                         MOV     A, PMDAT                ;
596  041A  3149                         ROL     R1, 1                   ;
597  041C  2280                         MOV     PMDAT, A                ;
598  041E  123B                         MOV     P0H, A                  ;
599  0420  143D                         BR      $TINT10                 ;

                                $       EJECT
                                ;-------------------------------------------------------
                                ;       FREE RUNNING COUNTER OVER FLOW CHECK
                                ;-------------------------------------------------------
600  0422                       ROVCHK:
601  0422  08AF6013                     BF      FRCC. 7, $RCHK01        ; if( over flow )
602  0426  089F60                       CLR1    FRCC. 7                 ;    over flow flag reset
603  0429  2667                         INC     RFOVC                   ;    over flow counter increment
604  042B  2067                         MOV     A, RFOVC                ;
605  042D  9A6A                         SUB     A, ROCPT+2              ;
606  042F  AF10                         CMP     A, #10H                 ;
607  0431  8306                         BC      $RCHK01                 ; engine stop check
608  0433  0C620000                     MOVW    REVOL, #0               ; set( engine stop data )
609  0437  A085                         CLR1    EFLG2. 0                ;
610  0439                       RCHK01:
611  0439  1424                         BR      $TINT10
```

UPD78310 ASSEMBLER C0.0

```
                                $       EJECT
                                ;-------------------------------------------------------
                                ;       A/D START (2, 4, 7, 8, 9)    external
                                ;-------------------------------------------------------
 612  043B                      ADSTRT:
 613  043B  2679                       INC     ACNTR                   ; ACNTR increment
 614  043D  6F7905                     CMP     ACNTR, #5               ; if( ACNTR > 4 )
 615  0440  8303                       BC      $ASTR01                 ;       ACNTR = 0 ;
 616  0442  3A7900                     MOV     ACNTR, #0               ;
 617  0445                      ASTR01:
 618  0445  2079                       MOV     A, ACNTR                ;
 619  0447  C1                         INC     R1                      ;
 620  0448  09F100E0                   MOV     !ADCNV, A               ; A/D channel set ( ACNTR+1 )
 621  044C  1411                       BR      $TINT10                 ;
                                $       EJECT
                                ;-------------------------------------------------------
                                ;       SPEED DATA CAPTURE
                                ;-------------------------------------------------------
 622  044E                      SCAPTR:
 623  044E  267E                       INC     STIMR                   ; STIMR increment
 624  0450  6F7E23                     CMP     STIMR, #35              ; if( STIMR > 34 )   -- 700 ms
 625  0453  830A                       BC      $SCPT01                 ;

 626  0455  3C7A7C                     MOVW    SCAPT, SCNTR            ;       SCAPT = SCNTR
 627  0458  0C7A0000                   MOVW    SCNTR, #0               ;       SCNTR = 0
 628  045C  3A7E00                     MOV     STIMR, #0               ;       STIMR = 0
 629  045F                      SCPT01:
                                ;       BR      !TINT10                 ;
                                $       EJECT
                                ;-------------------------------------------------------
                                ;       TIMER INTERRUPT END
                                ;-------------------------------------------------------
 630  045F                      TINT10:
 631  045F  267F                       INC     TCNTR                   ; TCNTR increment
 632  0461  6F7F14                     CMP     TCNTR, #20              ; if( TCNTR > 19 )
 633  0464  8303                       BC      $TINT11                 ;

 634  0466  3A7F00                     MOV     TCNTR, #0               ;       TCNTR = 0
 635  0469                      TINT11:
 636  0469  089FD0                     CLR1    TMIC1.7                 ; interrupt flag reset ;
 637  046C  57                         RETI                            ;
                                $       EJECT
 638                                    CSEG
                                ;*********************************************************
                                ;*      EXIF1  external interrupt 1                      *
                                ;*             cylinder check                            *
                                ;*********************************************************
 639  046D                      CYLIND:
 640  046D  05AC                       SEL     RB4                     ;

 641  046F  3AFA01                     MOV     MSC02, #1               ;
 642  0472  2BCA20                     MOV     EXIC1, #20H             ;
 643  0475  2083                       MOV     A, CYDAT                ;

 644  0477  AF20                       CMP     A, #FAPUL               ; if( TDC > A )
 645  0479  8307                       BL      $CYLI01                 ;       EFLG2.2 = 1
 646  047B  3A4E01                     MOV     FCHECK, #1              ;
 647  047E  B285                       SET1    EFLG2.2                 ;
 648  0480  1409                       BR      $CYLI02                 ;
 649  0482                      CYLI01:
 650  0482  AF18                       CMP     A, #FBPUL               ;
 651  0484  8305                       BL      $CYLI02                 ;
 652  0486  3A4E04                     MOV     FCHECK, #4              ;
 653  0489  B285                       SET1    EFLG2.2                 ;
 654  048B                      CYLI02:
                                ;
                                ;
                                ;
                                ;
                                ;
 655  048B  089FCA                     CLR1    EXIC1.7                 ; interrupt flag reset ;
                                ;       BR      $EITEND                 ;
```

UPD78310 ASSEMBLER CO.0

```
                                $       EJECT
656                                     CSEG
                                ;*******************************************************
                                ;*      INTERRUPT END ROUTINE                          *
                                ;*******************************************************
657   048E                      EITEND:
658   048E    758209                    BT      EFLG1.5,$EITE01

659   0491    660020                    MOVW    RP3,#2000H            ;REGISTAR BANK = 2
                                                                      ;USER FLAG     = 0
                                                                      ;INTERRUPT     = DI
660   0494    640000                    MOVW    RP2,#OBRJOB
661   0497    290000                    RETCS   !0
662   049A                      EITE01:
663   049A    57                        RETI
664                                     END
```

```
TARGET CHIP: UPD78310

ASSEMBLY COMPLETE        0 Error(s) found

Maximum Stack-Size  : 001BH


  Segment Informations:
  ---------------------

    ADRS    LEN    NAME

    0000    049BH  CSEG



Cross-Reference Listing:

NAME        ADDR  R ATTR SEG    REFERENCES
-----------------------------------------

ACNTR       FE79H    EQU          46#   520   613   614   616   618
AD1001      040DH R LOC  CSEG     581   588#
AD1002      0416H R LOC  CSEG     587   592#
ADCNV       E000H    EQU          17#   541   545   548   550   620
ADFINT      0316H R PUB  CSEG      11   468#
ADINP1      03ABH R LOC  CSEG     501   505   509   513   517   547#
ADINPT      0378H R LOC  CSEG     507   519#
ADS300      0065H R LOC  CSEG     142   145#
ADS301      007AH R LOC  CSEG     146   153#
ADS302      0083H R LOC  CSEG     144   152   157#
ADS501      03CCH R LOC  CSEG     553   560#
ADS502      03D5H R LOC  CSEG     559   564#
ADS601      03EDH R LOC  CSEG     567   574#
ADS602      03F6H R LOC  CSEG     573   578#
ADST10      03F8H R LOC  CSEG     502   512   580#
ADSTR1      03A2H R LOC  CSEG     499   503   511   515   543#
ADSTR3      005BH R LOC  CSEG     139#
ADSTR5      03B7H R LOC  CSEG     552#
ADSTR6      03D8H R LOC  CSEG     500   510   566#
ADSTRT      043BH R LOC  CSEG     506   612#
ADTBL       FE6EH    EQU          45#   150   156   523   528   532   536   539   551   557   563
                                  571   577   585   591
AINP01      0383H R LOC  CSEG     522   525#
AINP02      038CH R LOC  CSEG     527   530#
AINP03      0393H R LOC  CSEG     531   534#
AINP04      039AH R LOC  CSEG     535   538#
AINP05      039DH R LOC  CSEG     524   529   533   537   540#
AINT01      031EH R LOC  CSEG     469   472#
ASTR01      0445H R LOC  CSEG     615   617#
AWAIT       FE6BH    EQU          44#   155   156   475   476   562   563   576   577   590   591
B4R12       FEBCH    EQU          75#   196
B4R13       FEBDH    EQU          76#   199
CYDAT       FE83H    EQU          54#   643
CYLI01      0482H R LOC  CSEG     645   649#
CYLI02      048BH R LOC  CSEG     648   651   654#
CYLIND      046DH R PUB  CSEG      13   639#
D50INT      025FH R PUB  CSEG       7   369#
D5IT01      028AH R LOC  CSEG     378   390#
D5IT02      0299H R LOC  CSEG     379   397#
D5IT06      0267H R LOC  CSEG     373#
D5IT10      02AEH R LOC  CSEG     372   407#
D5IT20      02BAH R LOC  CSEG     371   413#
D5IT30      02C9H R LOC  CSEG     414   420#
D5IT40      02D5H R LOC  CSEG     389   396   406   412   419   426#
D64INT      023DH R PUB  CSEG       6   349#
D6IT01      0255H R LOC  CSEG     351   361#
D6IT11      025BH R LOC  CSEG     360   365#
DMODE       FE4FH    EQU          31#   371   372   388   395   405   411   414   418   425
DNXT1       FE58H    EQU          36#   384   401   409   416
DNXT2       FE5AH    EQU          37#   387   394   404   423
DOF64       FE52H    EQU          33#   358   362
DON64       FE50H    EQU          32#   354
DONT1       FE54H    EQU          34#   376   382   386   393   400
```

Cross-Reference Listing:

NAME	ADDR	R	ATTR	SEG	REFERENCES
DONT2	FE56H		EQU		35# 377 380 383 399 403
DTRS01	02E3H	R	LOC	CSEG	432 434#
DTRSET	02D9H	R	LOC	CSEG	381 391 398 410 417 424 429#
EFLG0	FE81H		EQU		52#
EFLG1	FE82H		EQU		53# 151 154 469 485 489 558 561 572 575 586 589 658
EFLG2	FE85H		EQU		56# 105 106 113 210 218 219 269 270 298 299 609 647 653
EITE01	049AH	R	LOC	CSEG	658 662#
EITEND	048EH	R	LOC	CSEG	159 265 347 657#
EWRK0	FE84H		EQU		55#
EXTCM	FFB3H		EQU		21#
EXTP0	FFB0H		EQU		18#
EXTP1	FFB1H		EQU		19#
EXTP2	FFB2H		EQU		20#
FAPUL	0020H		EQU		15# 644
FBPUL	0018H		EQU		16# 650
FCHECK	FE4EH		EQU		30# 107 109 121 646 652
FFCTR	FE21H		EQU		24# 124 212 213 215 220 273 277 286 291 303
FIDAT	FE8DH		EQU		64# 111 220 301
FNCTR	FE20H		EQU		23# 104 110 140 163 189 262 284
FPDAT	FE4CH		EQU		28# 235 237 271
FPILL	FE4DH		EQU		29# 114 168 170 300
FTBL31	0000H	R	PUB	CSEG	2 90# 164 287
FXTE01	0194H	R	LOC	CSEG	269 275#
FXTE02	01C4H	R	LOC	CSEG	285 294 296#
FXTE03	0234H	R	LOC	CSEG	310 343#
FXTE04	0237H	R	LOC	CSEG	342 345#
FXTE05	0214H	R	LOC	CSEG	327# 341
FXTE06	022DH	R	LOC	CSEG	334 339#
FXTE07	01D2H	R	LOC	CSEG	274 298 302#
FXTEND	0184H	R	PUB	CSEG	5 267#
FXTMA	FE28H		EQU		26# 195 198 201 223 226 228 240 243 245
FXTMB	FE3AH		EQU		27# 115 197 200 203 207 208 209 251 252 253 281 282 283 307 308 309 312 316 318 325
FXTS01	00AFH	R	LOC	CSEG	174 175 177 179#
FXTS02	00B8H	R	LOC	CSEG	181 184#
FXTS03	00B9H	R	LOC	CSEG	183 186#
FXTS04	0143H	R	LOC	CSEG	194 239#
FXTS05	016EH	R	LOC	CSEG	214 216 218 233 238 254#
FXTS06	0179H	R	LOC	CSEG	259 261#
FXTS07	013BH	R	LOC	CSEG	204 234#
FXTS08	0110H	R	LOC	CSEG	210 217#
FXTSET	0089H	R	PUB	CSEG	4 161#
FYCS00	0011H	R	LOC	CSEG	101 103#
FYCS01	003AH	R	LOC	CSEG	112 123#
FYCS02	003CH	R	LOC	CSEG	105 108 122 125#
FYCS03	0033H	R	LOC	CSEG	118# 120
FYCSET	0006H	R	PUB	CSEG	3 98#
FYCTR	FE22H		EQU		25# 126
IACNT	FE5CH		EQU		38# 450 454
IGIT01	02FFH	R	LOC	CSEG	448 452#
IGIT02	030FH	R	LOC	CSEG	461#
IGIT03	030DH	R	LOC	CSEG	451 457 459#

Cross-Reference Listing:

```
NAME        ADDR   R ATTR SEG    REFERENCES
-----------------------------------------

IGIT04      0315H  R LOC  CSEG     444    465#
IGNINT      02EBH  R PUB  CSEG       8    440#
IGNITS      02F3H  R PUB  CSEG      10    441    445#
IGNTST      02EFH  R PUB  CSEG       9    187    205    211    255    276    297    311    324    340    443#
INT                MOD               1
IRCNT       FE5EH     EQU            39#    455
JFG1        FE86H     EQU            57#
JFG2        FE87H     EQU            58#
JFG3        FE88H     EQU            59#
JFG4        FE89H     EQU            60#
JFG5        FE8BH     EQU            62#
JFG6        FE8CH     EQU            63#
JFGA        FE8AH     EQU            61#
MSC00       FEF2H     EQU            78#    171    225    242    315
MSC02       FEFAH     EQU            81#    641
MSC03       FEFEH     EQU            84#    431    433
MSC04       FEE2H     EQU            87#    149    474    556    570    584
MSP00       FEF0H     EQU            77#
MSP02       FEF8H     EQU            80#
MSP03       FEFCH     EQU            83#
MSP04       FEE0H     EQU            86#    150    475    557    571    585
OBRJOB                EXT            14#    660
PLMOUT      0418H  R LOC  CSEG     504    514    594#
PMDAT       FE80H     EQU            51#    595    597
RCHK01      0439H  R LOC  CSEG     601    607    610#
REVINT      0048H  R LOC  CSEG     130#
REVOL       FE62H     EQU            40#    608
RFOVC       FE67H     EQU            42#    133    138    603    604
RINT01      0054H  R LOC  CSEG     131    132    135#
RNCPT       FE64H     EQU            41#    137    138
ROCPT       FE68H     EQU            43#    605
ROVCHK      0422H  R LOC  CSEG     508    518    600#
SCAPT       FE7CH     EQU            48#    626
SCAPTR      044EH  R LOC  CSEG     516    622#
SCNT01      0341H  R LOC  CSEG     487    490#
SCNTR       FE7AH     EQU            47#    488    626    627
SCOUNT      0333H  R LOC  CSEG     484#
SCPT01      045FH  R LOC  CSEG     625    629#
SFRP0       FEF3H     EQU            79#
SFRP2       FEFBH     EQU            82#
SFRP3       FEFFH     EQU            85#
SFRP4       FEE3H     EQU            88#
STACK       FE20H     EQU            22#
STIMR       FE7EH     EQU            49#    623    624    628
T1          FE90H     EQU            65#
T10         FE99H     EQU            74#
T2          FE92H     EQU            67#
T3          FE91H     EQU            66#
T4          FE94H     EQU            69#
T5          FE93H     EQU            68#
T6          FE96H     EQU            71#
T7          FE95H     EQU            70#
T8          FE97H     EQU            72#
T9          FE98H     EQU            73#
TCNTR       FE7FH     EQU            50#    491    631    632    634
TINT10      045FH  R LOC  CSEG     546    565    579    593    599    611    621    630#
TINT11      0469H  R LOC  CSEG     633    635#
TMRINT      0331H  R PUB  CSEG      12    482#
TMRTBL      0350H  R LOC  CSEG     495    498#
```

Interfacing NEC's CMOS Microprocessors V40 and V50 to the Numeric Data Processor (NDP) 8087

Contents:

Author: Shyam Gupta
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD70208/216 User's Manual
Data Book Microprocessors and Peripherals

Related Products

μPD70208	16 bit Microprocessor	CMOS
μPD70216	16 bit Microprocessor	CMOS

1. Introduction

The 16 bit CMOS microprocessors μPD70208/70216 (V40/V50) from NEC incorporate a powerful set of peripherals with a CPU to crease the highest possible computing power.

In order to support the CMOS microprocessor family V20, V30, V40 and V50, NEC has announced a CMOS floating point processor 8087, the numeric data processor (NDP).

This note describes an interface between 8087 and μPD70208/70216 (hereinafter called V40/V50) with a minimum of additional hardware overhead that allows a small piggy board to be built which would fit in the 8087 socket as a temporary solution.

This note is written for the reader who is familiar with the function of a floating point processor.

2. Interface Consideration

For interfacing 8087 to V40/V50 the following point must be considered:

— The bus request protocol of the V40/V50's and the 8087 are different. Whereas the 8087 uses a bidirectional RQ/GT pin, the V40/V50 microprocessors use two separate 'HLDRW' and 'HLDAK' signals for bus exchange protocol.

There are various ways to get around these shortcomings such as using a PAL with additional external logic. Another possibility is to use an 82188 bus controller with the 8087 to interface it with the V40/V50's. The 82188 not only generates the proper bus exchange protocol, but also supplies the required bus control signals.

However even in this case some additional hardware is unavoidable, mainly due to the following reasons:

a) As in the case of the V40/V50's, the 8087 has no feature such as inserting programmable number and wait states in a bus cycle. Therefore the 8087 must be operated with an external ready input.

The V40/V50 microprocessors support insertion of a programmable (0—3) number of wait states in a bus cycle. The bus cycle can optionally be extended by using an external ready signal. After reset, the V40 and V50 microprocessors insert three wait states (default value) in every bus cycle.

The 82188, which was designed to interface the 8087 with 80186/80188 (supporting also a programmable number of (0—3) wait states, with default as 3 wait states after reset), inserts three wait states in the first 256 bus cycles and supplies a synchronized ready output "SRO" to the 8087.

Since the 8087 must be controlled by external ready signal, the V40/V50's must set the (internal) programmable wait states zero within the first 256 bus cycles and then use an external ready signal like the 8087.

b) **Another point to be considered is the difference between the ready signal sampling time between the V40/V50 microprocessors and the 8087.**

Whereas the V40/V50 microprocessors sample the ready signal at the rising edge of T2, T3, TW . . . etc., the 8087 samples the same signal at the end of T2, T3, TW . . . etc.

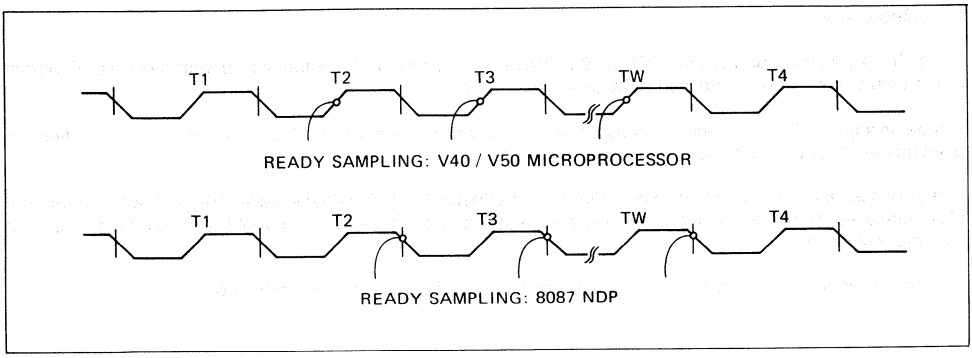
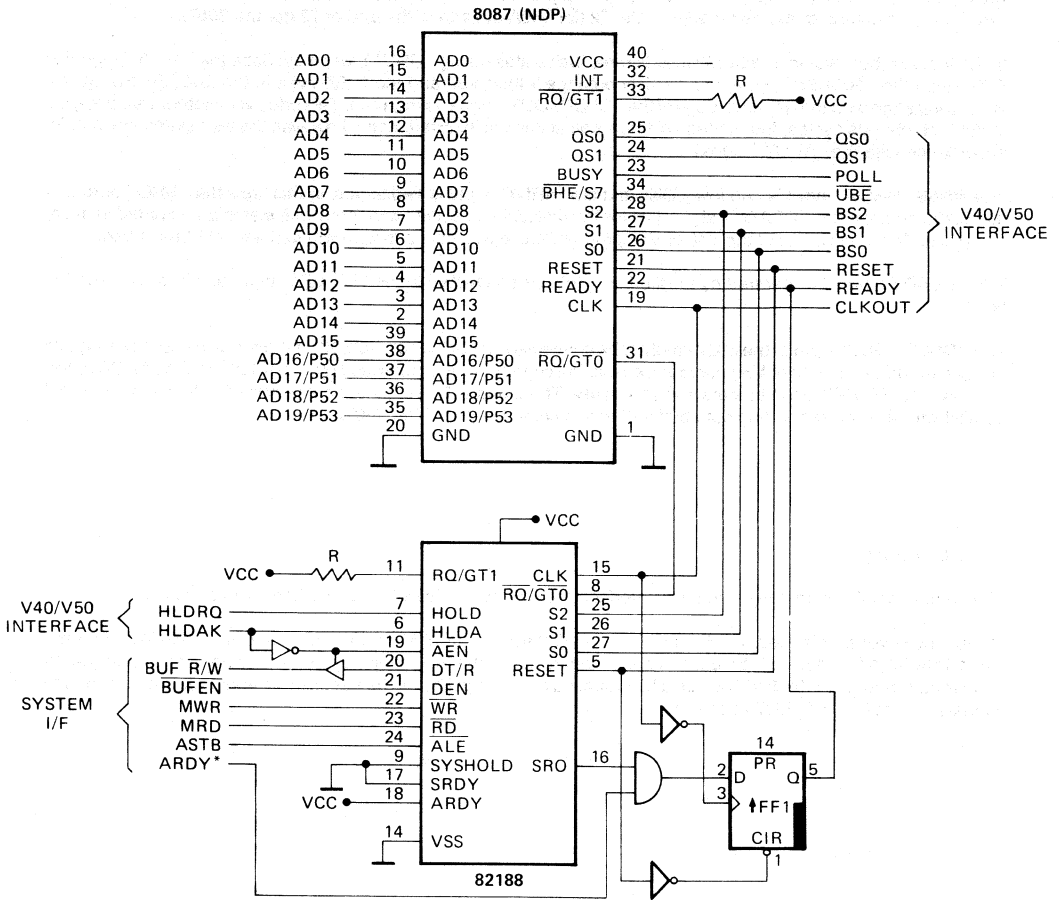


Figure 1. Ready Signal Sampling

Taking the points a) and b) into account, a minimum circuit for interfacing the 8087 with the V40 and V50 microprocessors may be realized as shown in figure 2.



Note: * Asynchronous READY from System

Figure 2. V40/V50 VS 8087 Interface

The handshake protocol is generated by the 82188.

The system uses asynchronous "ARDY" which is synchronized by the negative edge of the "CLOCKOUT" signal via flip-flop 1 (FF1). The FF1 generates a "READY" signal for the V40 or V50 and the 8087 (NDP) which satisfies the "READY" setup time to the rising edge of the T2 (for V40/V50) and to the end of T2 (for the 8087).

In the first 256 bus cycles, the 82188 inserts three wait states (using "SRO") which overlaps the "ARDY" signal to FF1. Since the "READY" signal must satisfy both the V40 or V50 and the 8087, even in the first 256 bus cycles, synchronization with the negative edge of the "CLOCKOUT" signal is required. This adds yet another wait state, i.e. in the first 256 bus cycles, four instead of three wait states are inserted. Later, however, the wait states are in strict accordance with the "ARDY" signal.

The 82188 "SRDY" input is tied low (GND) and the "ARDY" input is tied to Vcc. In this case the "SRO" output of the 82188 controls the "READY" signal just for the first 256 bus cycles and three wait states are inserted in every bus cycle. The user must set the V40 or V50 programmable wait states to zero within these 256 bus cycles.

The "READY" signal generated by the flip-flop 1 (FF1) is connected to the "READY" input of V40 of V50 and the 8087.

The "QS0", "QS1" input of the 8087 need not be connected to the "QS0I", "QS0I" of the 82188 since these signals are generated by the V40 or V50 microprocessors. All control signals generated by the 82188 required for this interface (except "DT/R-") are in tri-state when not in use. This signal ("DT/R-") therefore needs a tri-state buffer to avoid conflict with the corresponding data buffer direction control signal of the V40 or V50.

3. Summary

This proposal allows an easy solution to interface the 8087 with NEC's V40 and V50 microprocessors.

It must however be clarified that for a 5 MHz V40/V50 system, an 8 MHz 8087, and an 8 MHz V40/V50 system a 10 MHz 8087 (NDP) device is be used. Among other reasons, the restriction takes into consideration the long setup time requirements for the 8087 device. The proposed circuit supplies all the necessary handshake and bus control signals for V40/V50 vs. 8087 interface.

V-Series DMA Controller μPD71071

Contents:

Author: E. Goldberg
NEC Electronics Inc. NATICK/USA

Persons to contact: S. Gupta, B. Peters
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Data Book Microprocessor and Periperals
V20, V30 User's Manual
V40, V50 User's Manual

Related Products

μPD71071 DMA Controller	CMOS
V20 16 bit Microprocessor	CMOS
V30 16 bit Microprocessor	CMOS
V40 16 bit Microprocessor	CMOS
V50 16 bit Microprocessor	CMOS

1. Introduction

The μPD71071 is a general purpose CMOS DMA controller (DMAC). It has a 24-bit address bus, a 16-bit data bus, and operates with an input clock frequency of up to 8 MHz. It contains a standby mode (for low power applications) which is entered by shutting off the clock to the DMAC and exited by turning on the clock.

The μPD71071 may be used in a variety of microprocessor based systems and is ideally suited for use in an NEC μPD70108/70116 (V20/V30) design or an 8088/8086 based design. It may also be used in an INTEL 80286 based design.

In the examples in this application note, the DMA controller performs transfers between memory and I/O and the CPU uses block string move instructions to perform transfers from memory to memory.

This application note is a design guide which answers certain conceptual design application issues. The note focuses on using the μPD71071 in a V30 application and an INTEL 80286 application.

2. μPD71071 in a V30 System

Figure 1 shows the μPD71071 in a V30 based large scale microprocessor system. A bus controller (μPD71088) is required when using the V30 in the large system mode.

Bus Interface Components

The μPD71071 interface to the V30 address bus requires two 8-bit latches, one 8-bit transceiver, and two gates to control the enabling and direction of the transceiver. The data bus interface requires two 8-bit transceivers and one gate to control the direction of the transceivers.

Since the DMAC is not used for memory-to-memory transfers, its data transceivers will only be enabled when the V30 is programming the DMAC. Therefore, the μPD71071 chip select input can also be used to enable the data transceivers. Unless a CPU read is performed, the direction of the data transceivers should always be controlled as if the CPU were writing to the DMAC since the only time that the transceivers will source the data bus is when the CPU reads the DMAC. Using the IORD signal will prevent a wired-OR on condition between the DMAC and CPU data bus transceivers.

The direction of the address bus transceiver depends upon whether the DMAC is a bus master or a bus slave. When the DMAC is a bus master, the AEN signal enables the address transceivers and latches. The transceiver sources address bits A0—A7 and the address latches source address bits A8—A19. When the DMAC is a bus slave and is being programmed by the V30, the address latches are not enabled and the DMAC transceiver direction control must be set so it places A0—A7 from the system bus on the local DMAC A0—A7 address bus. Bits A0—A3 are used by the DMAC to select one of its registers for programming.

Bus Interface Timing

When the DMAC is a bus master and is not in the compressed mode, the timing for a μPD71071 bus cycle and for a V30 bus cycle are conceptually the same. Each bus cycle requires four clock periods (S1, S2, S3, and S4). The compressed mode requires three clock periods (see the μPD71071 DMA Controller Data Sheet).

During S1, the DMA generates the memory address. During S2 and S3, MRD or MWR access the memory and IORD or IOWR access an I/O controller. The I/O controller uses its DMAAKn (n = 0, 1, 2, or 3) to identify that it is the source or destination of the memory transfer. Figures 2 and 3 show transfers from memory to I/O and I/O to memory.

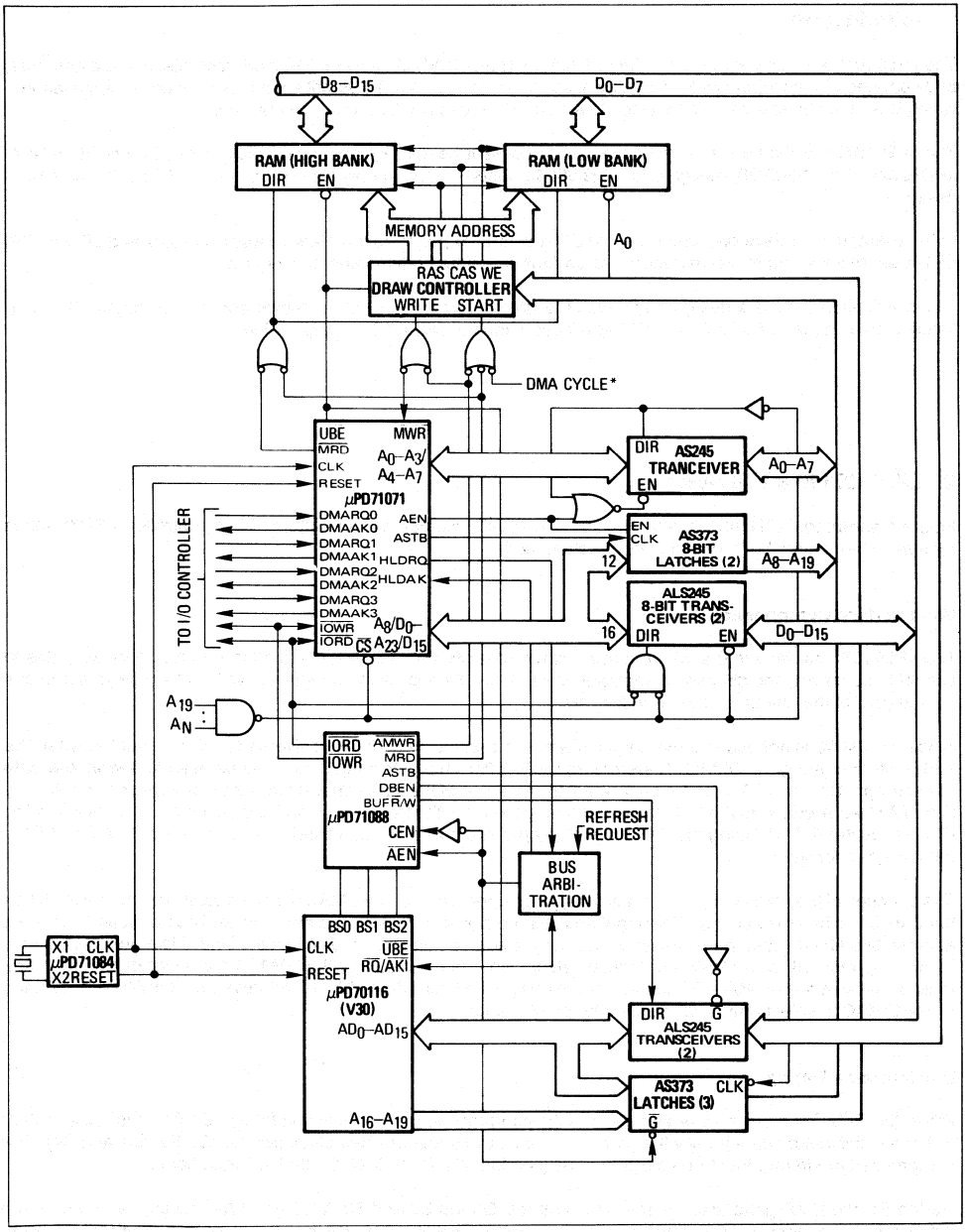


Figure 1. μPD71071 DMA Controller in V30 Microprocessor System

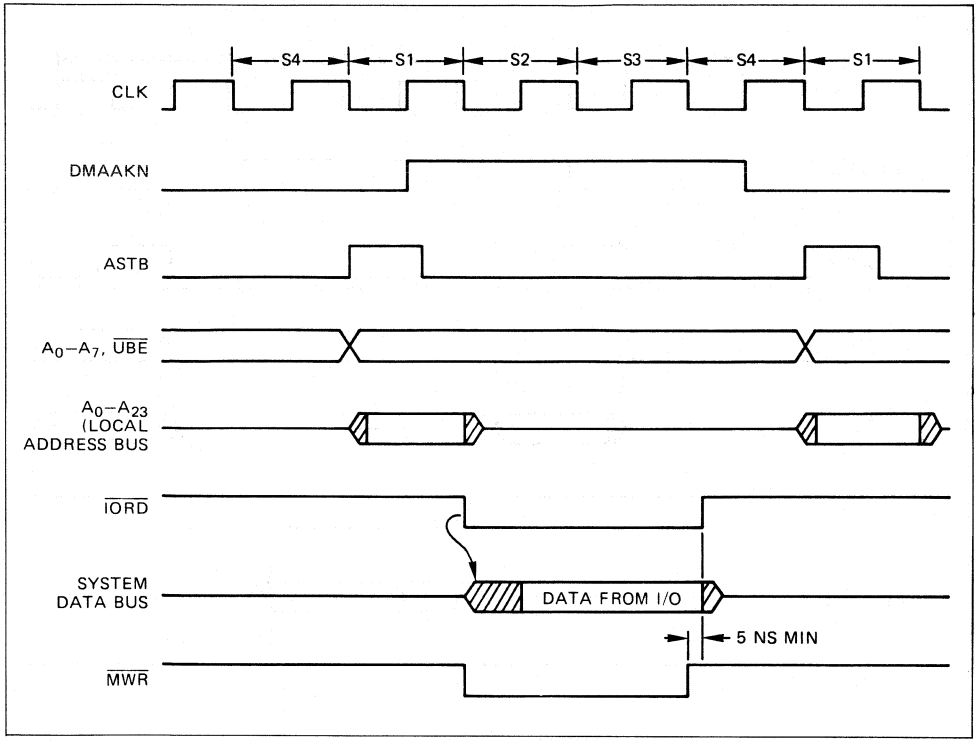


Figure 2. Memory to I/O Transfer

During an I/O to memory transfer, the I/O controller must put valid data on the data bus during S2, S3, and S4. During a memory to I/O transfer, memory puts data on the data bus during S2 and S3. The I/O controller will strobe in the data on the low to high transition of IOWR. When using a DRAM main memory system, MWR should be programmed for the normal write mode (valid during S3) so that it will not become active at the DRAM write input before the data is stable. When using MWR in the extended mode, it will become active during S2 and the I/O controller may not have stable data on the data bus.

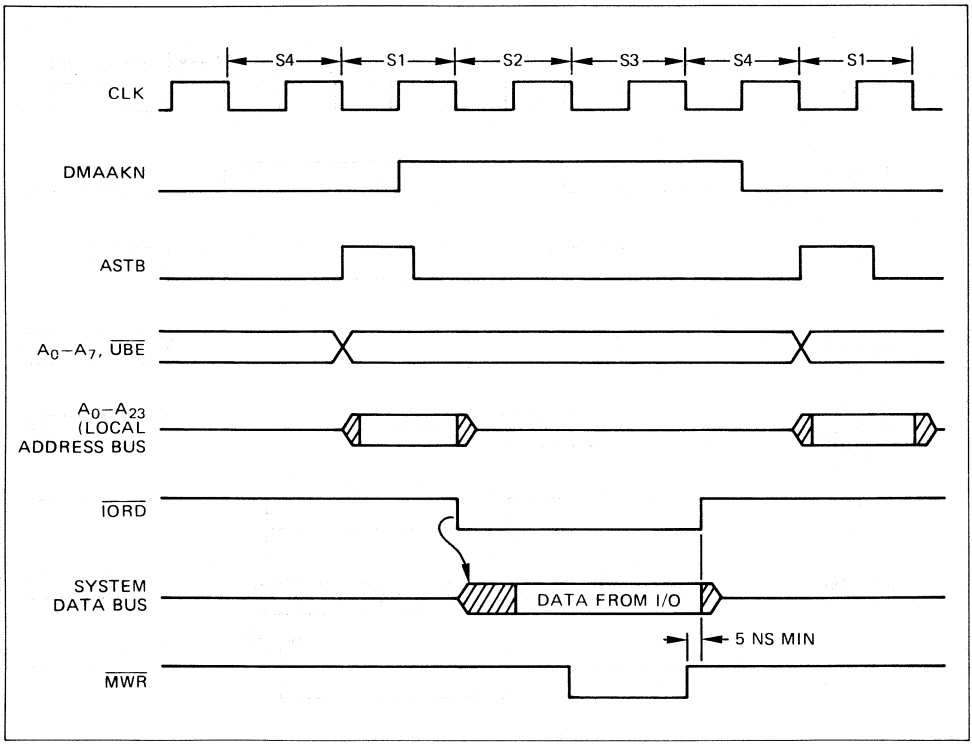


Figure 3. I/O to Memory Transfer

The V30 MRD and DMAC MRD signals enable RAM data onto the data bus during a memory read operation. Note that a data hold of five ns (tDWHRH) is built into the DMAC timing of transfers between memory and I/O. This insures that the read pulse (MRD or IORD) will always remain active for at least five ns after the write strobe (IOWR or MWR) becomes inactive.

Note that in figure 1, MRD of the V30 and MRD of the DMAC are not connected. Also, AMWR of the V30 and MWR of the DMAC are not connected. This is to take advantage of the 8 MHz speed of the DMAC when used with a DRAM. The DRAM controller uses MRD or AMWR of the V30 to start a memory cycle. The DMA cycle signal is used to start DMA DRAM bus cycles. Figure 4 shows a logic circuit where the DMA cycle signal is started on the high to low transition of the DMAC ASTB signal. The low to high transition of the DMAC MRD or MWR signals terminates the DMA cycle signal.

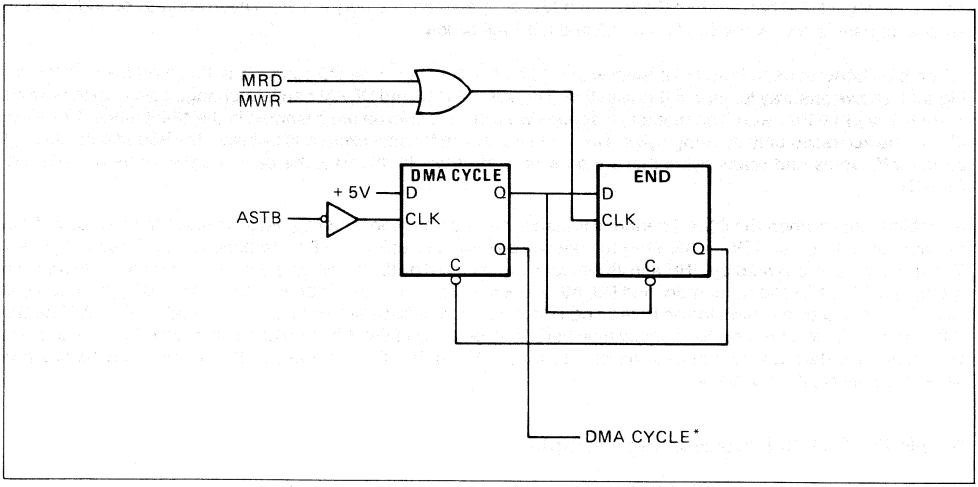


Figure 4. DMA Cycle Logic

Obtaining and Using the System Bus

The DMAC obtains use of the system bus by using its HLDRQ/HLDAK signals. When the DMAC receives a DMA request (DMARQn) on one or more of its four request inputs and it currently does not have control of the sytem bus, it raises the HLDRQ signal to a high. The CPU gives control of the system bus to the DMAC by raising the HLDAK input to a high. When the V30 is used in the large scale mode as in figure 1, one pin (RQ/AK) serves both as a bus request and acknowledge. Since the DMAC is a two-pin interface (HLDRQ/HLDAK), some bus arbritration interface logic is required in order to make the two types of interfaces compatible. When the V30 is in the small scale mode, two pins of the V30 (HOLD and HLDA) are used for bus access. HLDRQ is connected to HOLD, HLDA is connected to HLDAK, and no interface logic is required.

One portion of a system which may cause bus efficiency problems is dynamic (RAM (DRAM) refresh. How much of a problem this is depends upon what kind of DRAM controller is being used. When a refresh request occurs and the DMAC is the bus master, the DMAC can be forces to give up the bus by making the HLDAK signal invalid. DRAM refresh can then be performed. After refresh is completed, HLDAK could again be made active, thereby making the DMAC the bus master again and allowing DMA transfers to continue.

The problem with the sequence just described is the time overhead resulting from continually making HLDAK false and then true again. This is particularly true when the DMAC is transferring very large blocks (up to 64K words) of data. One way to avoid constand DMAC interruption by refresh is to refresh all 256 DRAM rows back to back in one continuous block of time. The time to actually refresh the DRAMs would not change, but the time overhead of removing HLDAK and then reasserting it after refresh would be reduced to once every four ms as opposed to 256 times every four ms. In this manner, 7.7K words of data could be transferred uninterrupted.

Data Bus Size

The DMAC may be programmed to operate in an 8- or 16-bit mode. In 16-bit mode, some registers are written or read as words (D0—D15) and others as bytes (D0—D7 or D8-D15). In 8-bit mode, all registers are accessed as bytes using data bits D0—D7. The UBE signal is an input when the DMAC is being programmed and an output when the DMAC is a bus master. UBE designates whether the upper byte of the data bus (bits D8—D15) is valid. In the 8-bit mode, UBE is three-state off. In the 16-bit mode, the upper byte of the data bus is valid when UBE is low. The V30 UBE signal and the DMAC UBE signal should be connected together as in figure 1. The UBE signal should be used to select the upper RAM bank. When the 16-bit mode is programmed, memory words on even address boundaries will always be transferred by the DMAC and A0 and UBE will be low.

When the DMAC is used in a V30 system with a 16-bit memory and 8-bit I/O controllers, the interface is different. Figure 5 shows one way to handle this situation. The V30, DMAC, and DRAM can interchange data in byte or word format. The μPD71071 would be memory mapped I/O so that is could be programmed in the 16-bit mode. However, I/O can be accessed only by using bytes. This is not a problem for data exchange between the V30 and I/O as long as the V30 writes and reads bytes that are on even addresses. In this way, the data will always be on data bits D0—D7.

A problem arises when the DMAC performs transfers between memory and I/O. Here bytes are transferred to/from memory on D0—D7 and D8—D15. The problem can be solved by using one 8-bit transceiver as shown in figure 5. The transceiver is enabled by UBE and its direction is controlled by IORD. When a byte is transferred from an even memory address between memory and I/O, A0 = 0 and UBE = 1. The I/O data is exchanged with data bus bits Do—D7. When a byte is transferred from an odd memory address between and I/O, A0 = 1 and UBE = 0. Here the 8-bit transceiver is enabled, the memory high bank is selected, and the 8-bit I/O data is connected to data bus bits D8—D15. This data will also appear on data bits D0—D7, but it will be ignored by the memory low bank since A0 = 1 deselects the low bank.

3. μPD71071/V30 Programming Example

The following shows how the μPD71071 would typically be programmed in a system. Figure 6 shows a system with a μPD71071 DMAC in a V30-based system. This specific illustration of the system shown one disk and one communications controller (two of a possible four I/O controllers with DMA cpability). When the count register in the DMAC is zero, the END/TC output will set a flip-flop which will cause a V30 interrupt via the μPD71059 interrupt controller.

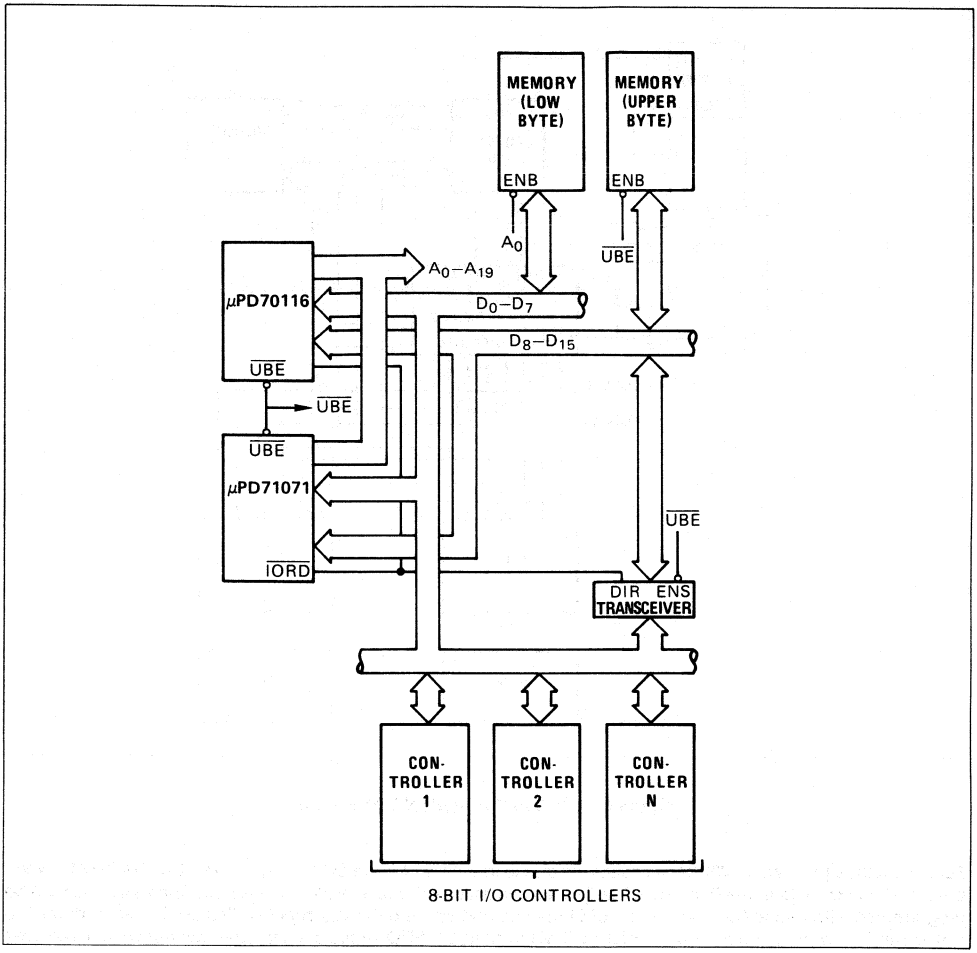


Figure 5. μPD71071 with 8-bit I/O Controllers

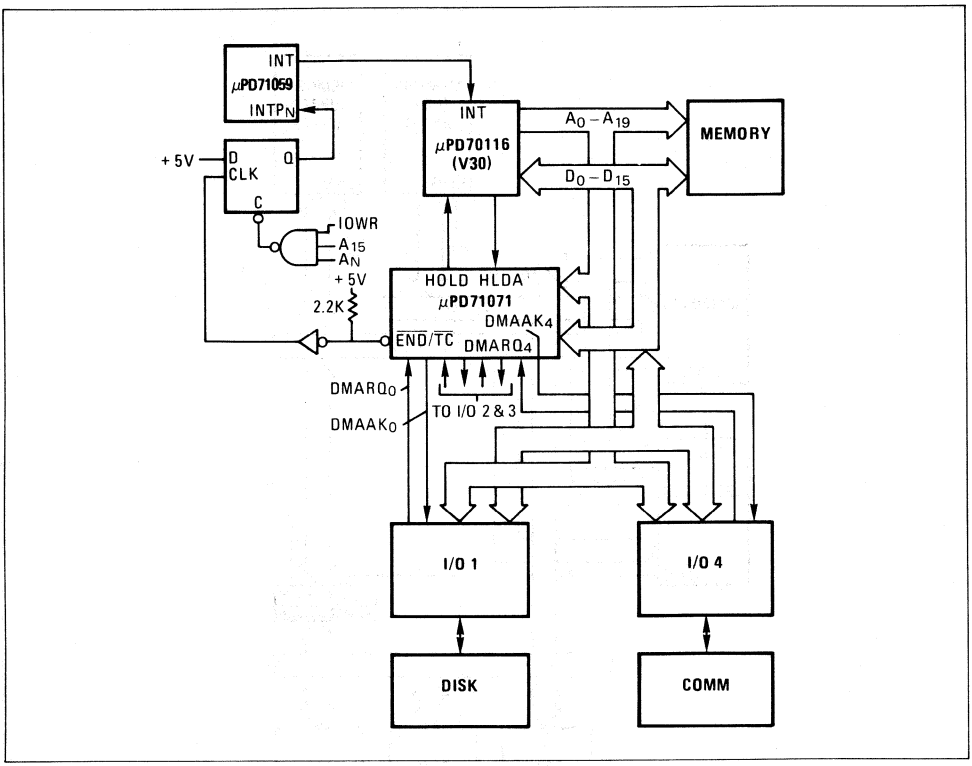


Figure 6. μPD71071 in Typical V30 System

For simplicity, a program in V30 code causes the disk to transfer a block of data using channel 0 of the DMAC. When the transfer is complete, the END/TC will output a pulse which will cause a DMAC interrupt. Channel 0 will then be programmed with a new count value and DMA starting address and the interrupt f lip-flop will be reset. When the disk is ready to transfer another block of data, it will raise its DMARQ signal and start the transfer when it receives a DMAAK.

Figure 7 shows a simplified flow chart of this program. The program listings for the boxes marked INIT 71071 and IO TO MEM are shown in the listing sections "Symbolic I/O Port Addresses", "INIT_71071:", and IO_TO_MEM:". These boxes represent the bulk of the DMAC programming.

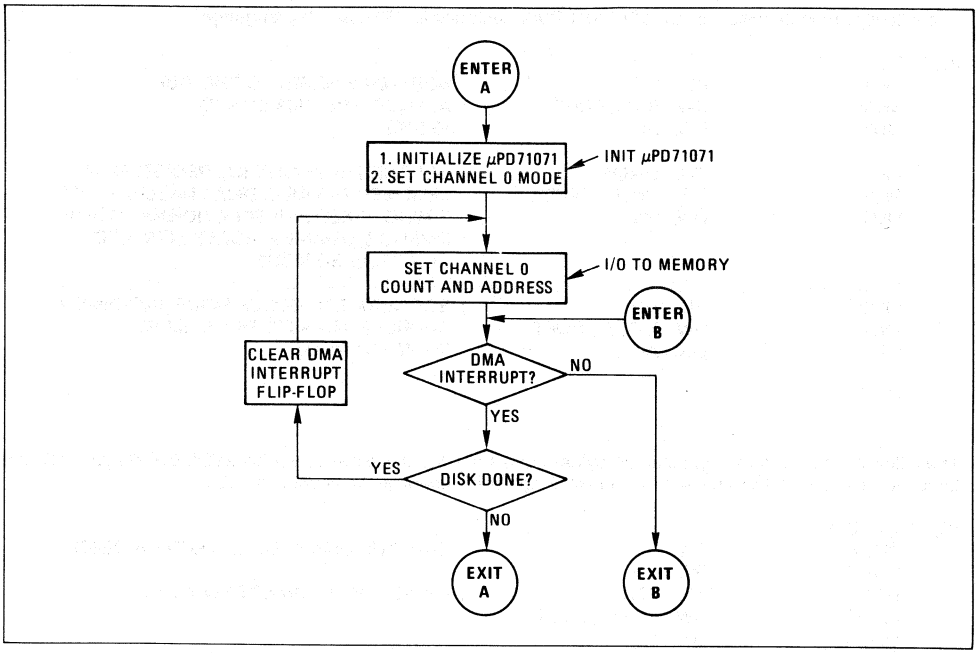


Figure 7.DMA Flow Chart

The following is a symbolic listing definition of the μPD71071 I/O port addresses.

Symbolic I/O Port Addresses

```
INIT_PORT       EQU  00H    ; INITIALIZE
CH_PORT         EQU  01H    ; CHANNEL REGISTER READ/WRITE
COUNT_PORT      EQU  02H    ; COUNT REGISTER READ/WRITE
ADDR_PORTL      EQU  04H    ; LOWER 16 BITS OF MEMORY START ADDR
ADDR_PORTH      EQU  06H    ; UPPER 8 BITS OF MEMORY START ADDR
CNTL_PORT       EQU  08H    ; DEVICE CONTROL REGISTER READ/WRITE
MODE_PORT       EQU  0AH    ; MODE CONTROL REGISTER
STATUS_PORT     EQU  0BH    ; STATUS REGISTER
REQUEST_PORT    EQU  0EH    ; REQUEST REGISTER
MASK_PORT       EQU  0FH    ; MASK REGISTER
```

The following module initializes the μPD71071 DMA Controller in V30 assembly language.

```
INIT_71071:
        MOV     AL,  03H            ; SOFTWARE RESET TO THE 71071
        MOV     DW,  INIT_PORT      ; AND SET THE DATA BUS TO
        OUT     DW,  AL             ; 16-BITS

        MOV     AW,  0040H          ; SET THE 71071 CONTROL REGISTER TO
        MOV     DW,  CNTL_PORT      ; MEM-MEM DISABLE, DMA ENABLE, NORM
        OUT     DW,  AW             ; TIMING, FIXED PRIORITY, NORMAL WRITE,
                                    ; DMARQ & DMAAK = ACTIVE LOW, AND
                                    ; BUS RELEASE MODE

        MOV     AL,  89H            ; SET CHAN 0 TO BLOCK MODE, INCREMENT
        MOV     DW,  MODE_PORT      ; ADDRESS, NO AUTO-INIT, MEM TO
        OUT     DW,  AL             ; I/O, WORD MODE

        RET
```

The following is a module to transfer CW bytes of data from disk I/O to memory address DL:BW. Register DL contains the segment and BW contains the offset. Register DH contains the channel number.

```
IO_TO_MEM:
        PUSH    DW                  ; SAVE THE DMA ADDRESS AND CHANNEL
        PUSH    BW
        MOV     AL,  0FH            ; MASK OFF ALL DMAC REQUESTS
        MOV     DW,  MASK_PORT
        OUT     DW,  AL
        MOV     AL,  DH             ; SET THE 71071 TO POINT
        MOV     DW,  CH_PORT        ; TO THE CHANNEL TO BE PROGRAMMED
        OUT     DW,  AL             ; WITH THE NEW COUNT AND ADDRESS
        MOV     AW,  CW             ; SEND THE NEW COUNT OF THE
        MOV     DW,  COUNT_PORT     ; CJAMMEÖ SEÖECTED.
        OUT     DW,  AW
        POP     AW
        MOV     DW,  ADDR_PORTL     ; SEND LOW 16-BIT OF NEW DMA
        OUT     DW,  AW             ; MEMORY ADDRESS BITS.
        POP     AW                  ; SEND UPPER 8-BITS OF NEW
        MOV     DW,  ADDR_PORTH     ; DMA MEMORY ADDRESS
        OUT     DW,  AL
        MOV     AL,  00H            ; UNMASK ALL DMA REQUESTS
        MOV     DW,  MASK_PORT
        OUT     DW,  AL

        RET
```

4. μPD71071 in an INTEL 80286 System

Figure 8 shows the μPD71071 in an INTEL 80286 based microprocessor system. The 80286 always requires a clock generator/driver (82284) and a system bus controller (82288).

Bus Interface

Functionally, the 80286 bus signals and 8086 bus signals are very similar. The major difference is that the address

and data buses are not multiplexed in the 80286. The remainder of the bus control signals are very similar. These signals are MRDC, MWTC, IORC, IOWC, ALE, DEN, DT/R, BHE, HOLD, and HLDA.

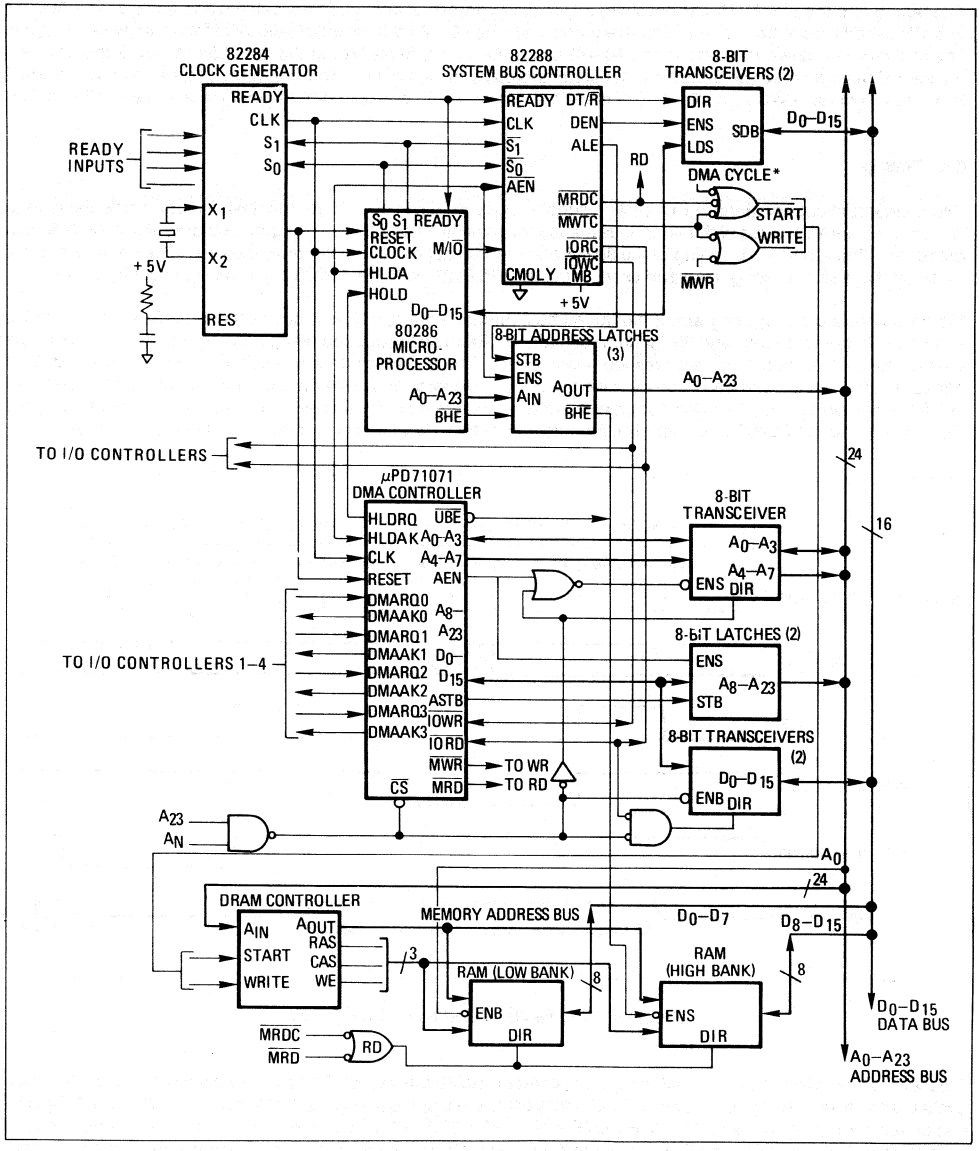


Figure 8. μPD71071 DMA Controller in an Intel 80286 System

The 80286 requires 24 bits of address latching and a latch or flip-flop bit for the BHE signal. It also requires 16 bits of transceivers for the data bus. The address and data bus interface for the μPD71071 is essentially the same as it was in the V30 application previously described.

The 80286 and the μPD71071 cannot share any of their address or data bus transceivers and latches. The A0—A7 bits of the 80286 must be latched. This is required since the 80286 address is not held valid for a complete bus cycle. The μPD71071 requires a transceiver for A0—A7 since A0—A3 are bidirectional. Address bits A8—A23 cannot share the same latch because the 80286 has separate address and data buses. Sharing these address bits would result in a wire OR of A8—A23 with D0—D15 of the 80286, which would corrupt both the address and data of the 80286.

Bus Timing

The conceptual observations in this section suggest possible basic design approaches. No one particular design is presented here because of the enormous number of potential variations and unique problems which each design presents. Although the bus timing of the 80286 and the V30 appear to be different, they are in fact quite similar. If these similarities are recognized and used, using the μPD71071 with the 80286 could be significantly easier.

Figure 9 shows a timing diagram of the 80286 bus. Fundamentally, a 80286 bus cycle is divided into two states known as Ts (send status) and Tc (perform command). Each state requires two periods of the input clock to the 80286. Hence, one bus cycle requires four clock cycles. The first clock period is called φ1 and the second φ2. If necessary, wait states can be added by using the READY signal to add additional Tc states. A command delay (RD or WR strobe delay) may be added to provide additional setup time for addresses or data. This is done using the CMDLY input to the 82288 bus controller. For this application, no wait states or command delays are shown.

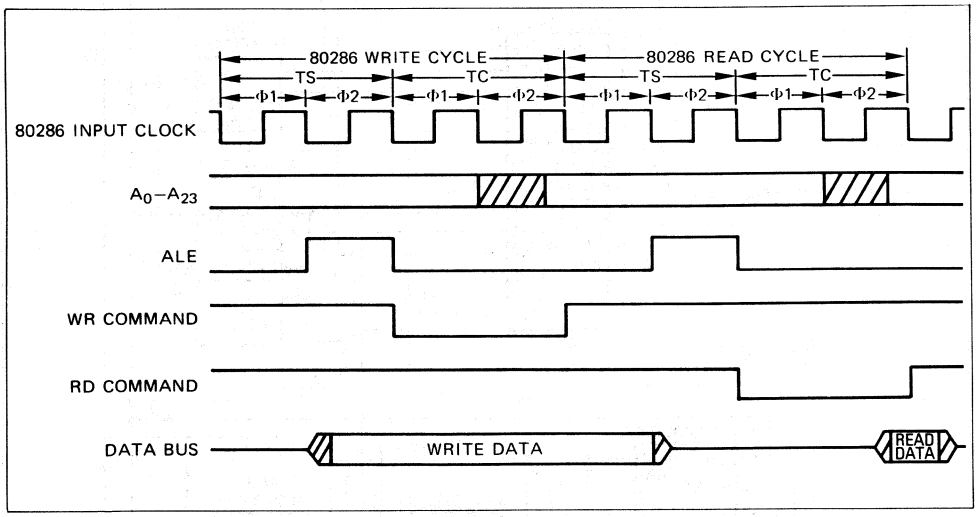


Figure 9. 80286 Read and Write Bus Cycles

Figure 10 shows μPD71071 bus cycles superimposed on 80286 bus cycles. The starting point of the μPD71071 bus cycle is shown skewed by one clock cycle with respect to the starting point of a 80286 bus cycle. By visualizing the two buses in this way, the address, ALE (ASTB in the μPD71071), RD, WR, and data buses are very similar. In fact, the timing of ALE, RD, WR, and RD data buses re conceptually the same. Their timing differs only by the specification values.

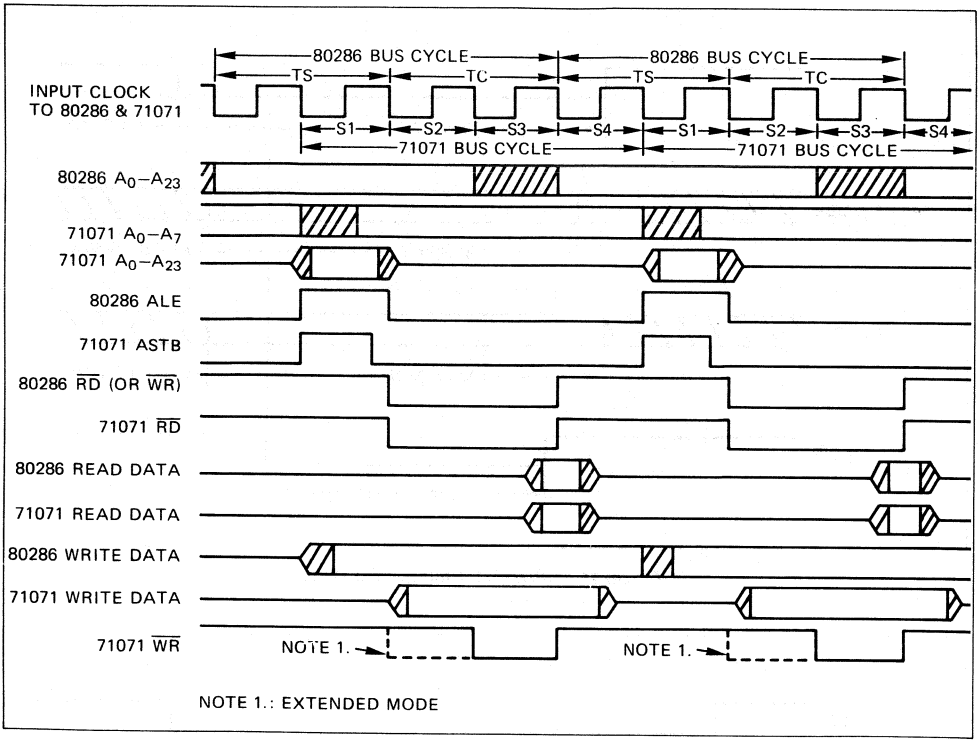


Figure 10. μPD71071 and 80286 Bus Cycles

The address and WR data buses are very similar. The key portion of the system which must interface to both of these bus masters (80286 of μPD71071) are the RAMs and I/O controllers. The control logic timing for these areas could be simplified if, for example, the starting point of a bus cycle were keyed off the ALE signals. In this situation, the address bus, data bus, and control signals are either conceptually identical or sufficiently close so as to provide commonality in bus interfacing.

When an 80286 (with an input clock of 8 MHz) is a bus master and is programming the μPD71071, the timing is such that no wait states or command delays are necessary. The μPD71071 programming timing requirements are shown in figures 11 and 12. The logic configuration shown in figure 8 can be used.

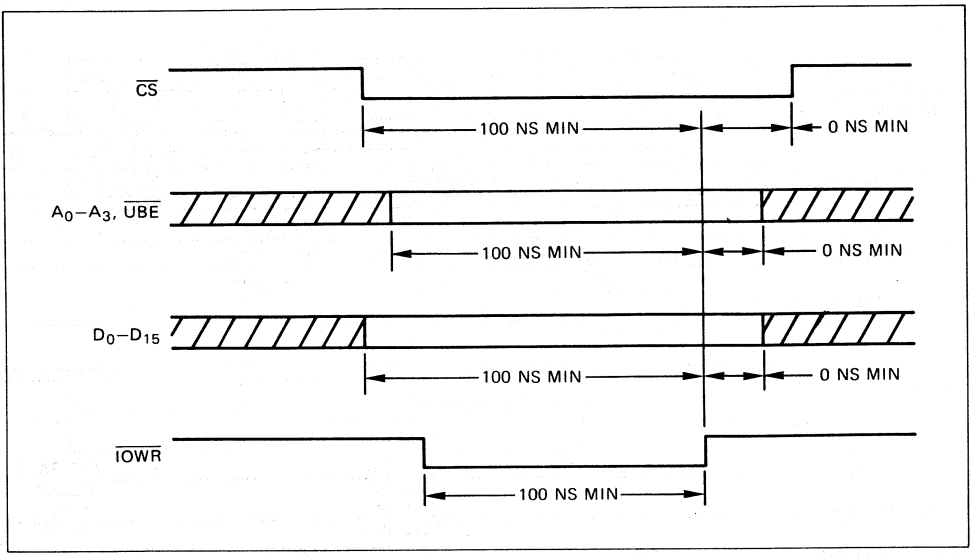


Figure 11. μPD71071 Timing, CPU Write

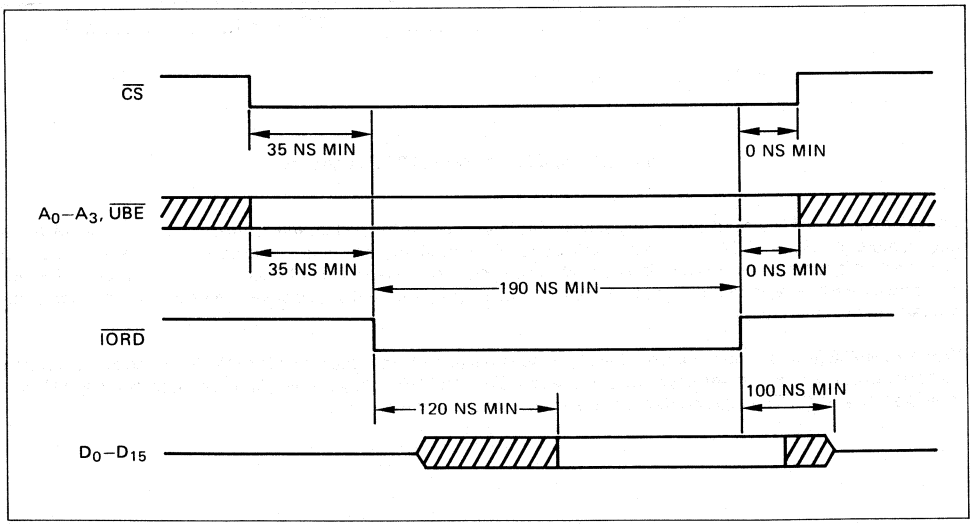


Figure 12. μPD71071 Timing, CPU Read

Single Chip Keybord μPD75104

Contents:

Author: B. Gough
NEC Electronics (Europe) GmbH

Related Documentation

μCOM 75X Family Product Description

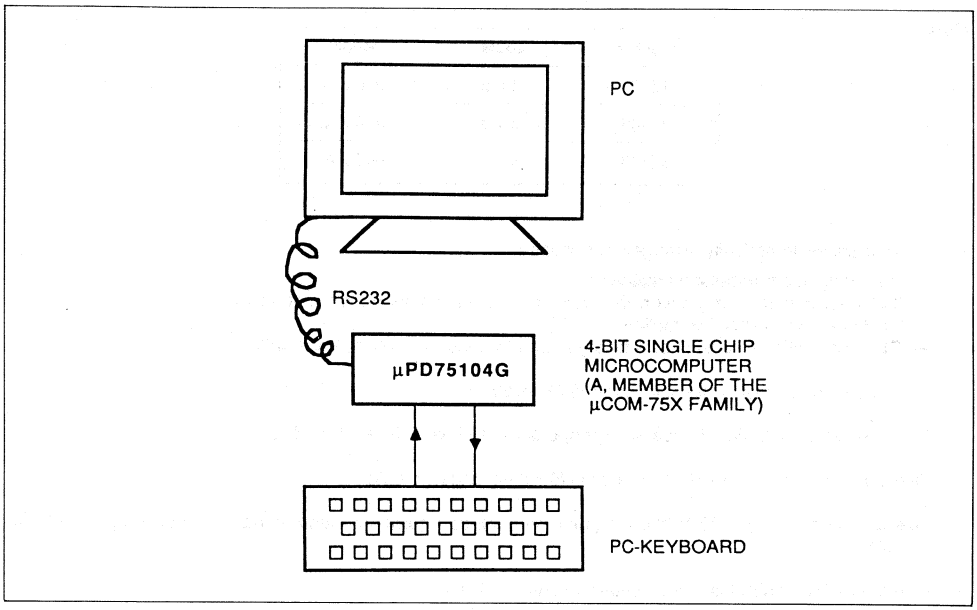
Related Products

μPD75004 / 006 / 008
μPD75106 / 108
μPD75206 / 208 / 216A
μPD75308

1. Introduction

This Application is based upon NEC's μCOM-75X Family using the μPD75104G.

1.1 A Single Chip Keyboard



Why use the μPD75104G?

Because:

- 4K ROM
- 58 I/O Lines
- 32 Pins drive leds
- 120 ms total sink
- Comparator input
- 12 I/O lines with pull up resistors, mask option. Ideal for key boards.
- Fast instruction cycle 0.95 μs
- 3 timers they could be used as follows:
  - — 1 for baud rate up to 9.6 K baud
  - — 1 for debounce
  - — 1 for repeat timing
- 1-, 4-, and 8-bit instructions
- UVPROM and OTPROMS
- 64 pin flat pack

1.2 The μPF75104G has the following features:

The μPD75104/106/108 are high end 4-bit microcomputers. High end in terms of size, number of peripherals on chip, and speed.

In addition to high-speed operation, they are able to manipulate data in 1-, 4-, and 8-bit units. I/O data control is notably enhanced by a variety of bit manipulation instructions.

Note:

Device	ROM	RAM
75104	4 x 8	320 x 4
75106	6 x 8	320 x 4
75108	8 x 8	512 x 4

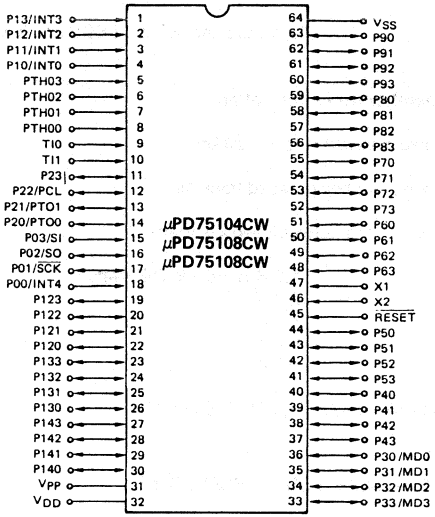
- 43 separate systematically arranged instructions
  - — Numerous bit manipulation instructions
  - — 8-bit data transfer, comparison, operation, and increment/decrement instructions
  - — 1-byte relative branch instructions
  - — GETI instruction that realizes arbitrary 2- or 3-byte instructions in 1-byte units
- High-speed instruction cycle: 0.95 μs / 4.19 MHz, 5V
- Programmable cycle time for low voltage operation: 1.91 μs, 15.3 μs / 4.19 MHz
- Size of program memory (ROM): (4 x 8 bit) (6 x 8 bit) and (8 x 8 bit)
- Size of data memory (RAM): 320 x 4 (memory bank configuration: memory-mapped input/output) 512 x 4 (75108)
- Memory for bit manipulation (bit-sequential buffer: 16 bits)
- General purpose register: 4 bits x 8 x 4 banks
- Accumulators:

  Bit accumulator (CY)
  4-bit accumulator (A)
  8-bit accumulator (XA)
- 58 I/O Lines
  - — CMOS In/Out pins (32 lines: drive LED directly)
  - — Middle-high voltage N-ch open drain In/Out pins
    (12 lines: drive LED directly; on-chip pullup resistor in bit units by mask option)
  - — Variable Threshold Voltage Analog Input Pins (4 lines)
- Vector interrupt function capable of multiple interrupts
  (3 external vector interrupts, 2 external test inputs, 4 internal vector interrupts)
- Two 8-bit timer/event counters
- 8-bit serial interface
  - — Either LSB first or MSB first can be selected for data transfer
  - — Two transfer modes (transmit/receive and receive-only mode)

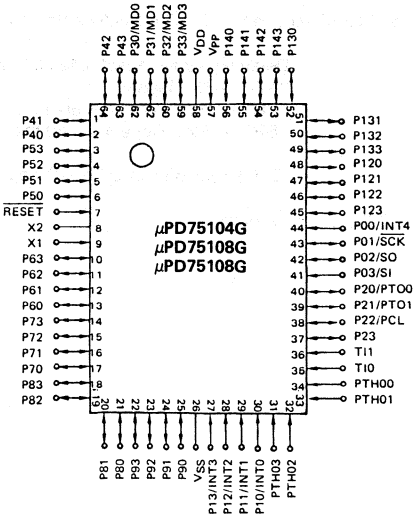
- Power-on reset (mask option)
- Internal clock oscillator for crystal/ceramic resonator
- Standby operation (STOP/HALT)
- CMOS technology
- Low power consumption

PIN CONFIGURATIONS

64-PIN PLASTIC SHRINK DIP



64-PIN PLSTIC FLAT PACKAGE



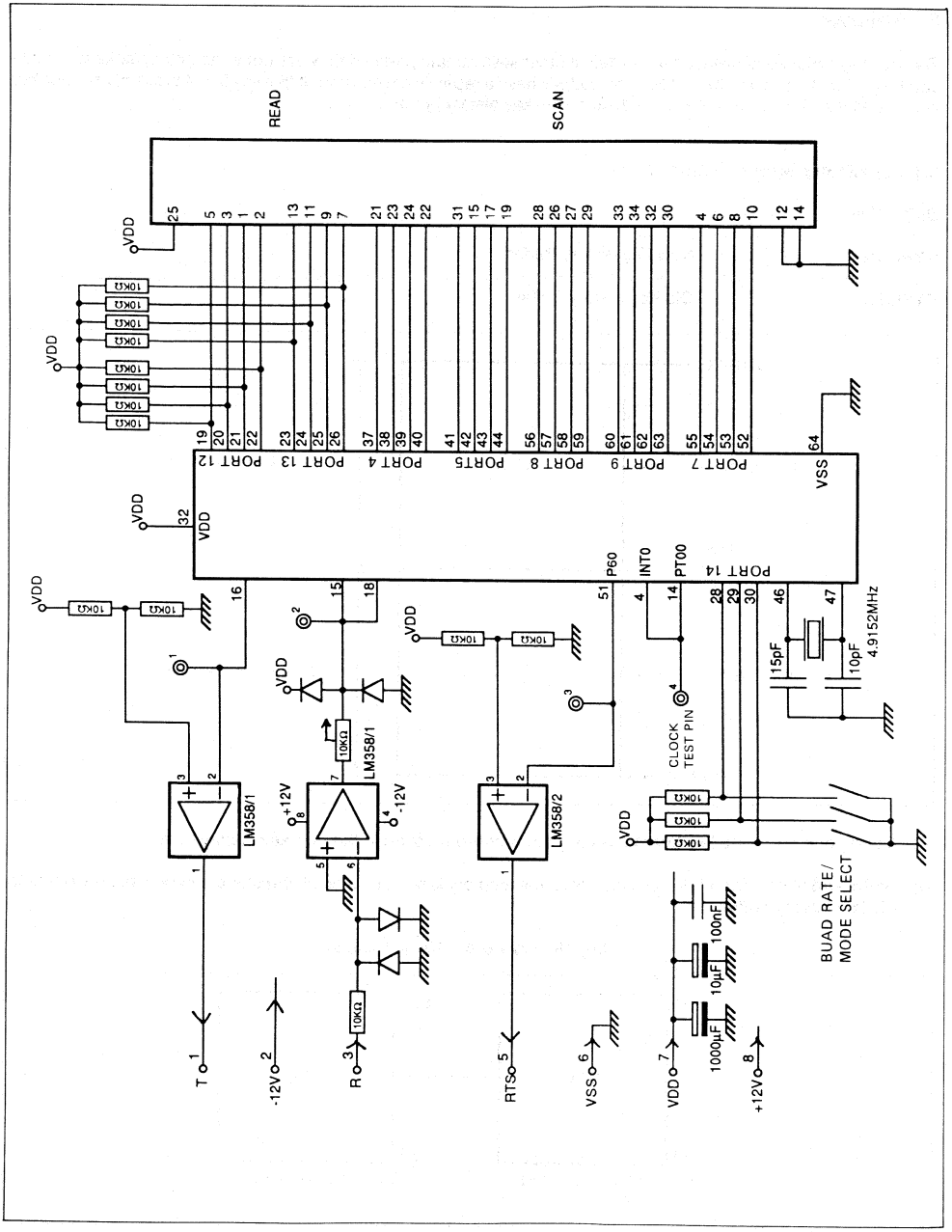
1.3 Major Components List

1)	μPD75104G x 1	NEC 4-Bit Microcomputer (Demo board uses the μPD75P108 UVPROM version)
2)	LM348N x 1	OP-AMP (only for RS232, up to 4.8 K)
3)	5.6 KΩ x 6	Resistors
4)	4146 x 4	Diodes
5)	128 key	Key board, including 4 LEDS and key diodes

1.4 General Specification Information

- Key board size matric 8 x 16 = 128 keys
- 32 key roll over, last key pressed transmit
- Debounce time 1 ms between two key reads
- Bad rates: 1.2 K Baud (via DIP switch)
  2.4 K
  4.8 K
  9.6 K
  19.2 K
- Mode ASCII only, UNSHIFT, SHIFT, CTRL **
- Data Formate
  1 Start-bit, 8 data-bits, (LSB . . . MSB) 1 Stip-bit
- Asynchronous 1/2 duplex
- 1 Handshake: RTS (Ready to Send), 1.6 ms before start bit
- Repeat: Delay time until repeat time 580 ms (± 10 %)
  "NR" — No Repeat
  "FR" — Fast Repeat = 50 ms ± 10 %
  "SR" — Slow Repeat = 180 ms ± 10 %
- Rom used: 1920 Kbyte, note μPD75104G has 4K ROM.

Note: ** SHIFT, CTRL are control keys.


READ
SCAN
VDD
VSS
PORT 12
PORT 13
PORT 4
PORT5
PORT 8
PORT 9
PORT 7
PORT 14
P60
INT0
PT00
10KΩ
LM358/1
LM358/2
+12V
-12V
15pF
10pF
4.9152MHz
CLOCK
TEST PIN
BUAD RATE/
MODE SELECT
100nF
10μF
1000μF
T
R
RTS

2. Rollover

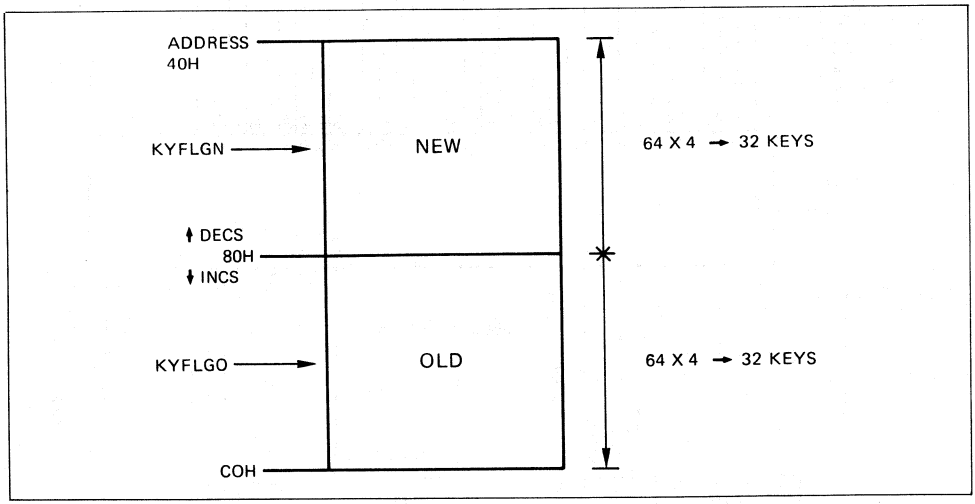
The last key pressed is always transmitted, if other keys remain pressed they are not transmitted. 32 keys at once can be pressed and only the last key pressed will have a repeat transmission. If this key is not pressed no other key is transmitted. The storing of keys is done on the key history buffer.

2.1 Key History Buffer Memory Bank 0

Buffer Pointers:

KYFLGN: New Key History Buffer

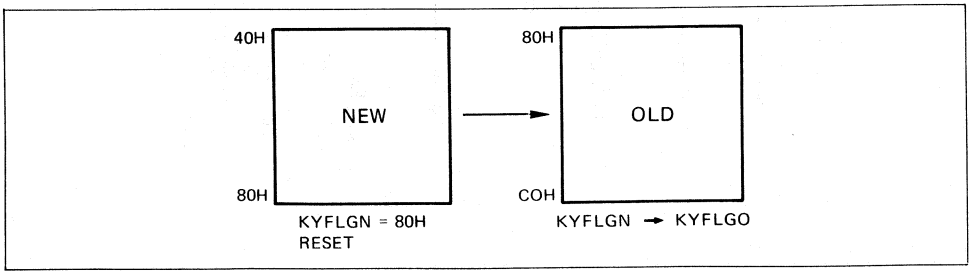
KYFLGO: Old Key History Buffer



A) During one complete key board scan any key pressed is saved in the new key history buffer.

B) At the end of one complete key board scan the memory is reset by a block transfer of new key history buffer to old key history buffer.

Key History Buffer Block Transfer



C) On the next key scan the old key history buffer is used as a comparison for repeat, no repeat and transmission of only last key pressed.

2.2 Last Key Pressed Register

This 8-bit register "LASKYP" is used for storing the last key pressed, but not control keys.

3. Using μPD75104G Timers

The μPD75104 has 3 timers, 1 basic internal timer and 2 timer counters.

3.1 Basic Interval Timer

Is used as a repeat timer with a 4.9152 MHz crystal.

Max. delay 213 ms
Min. delay 1.66 ms

3.2 Timer Counter 0

Is used as a baud rate generator from 150 to 19,600 baud, this supplies clock pulse to serial interface (only when using 4.9152 MHz crystal).

3.3 Timer Counter 1

Is used as the debounce timer 1 ms between double reading of a key pressed.

4. Interrupt and REG Bank Structure

4.1 Interrupts

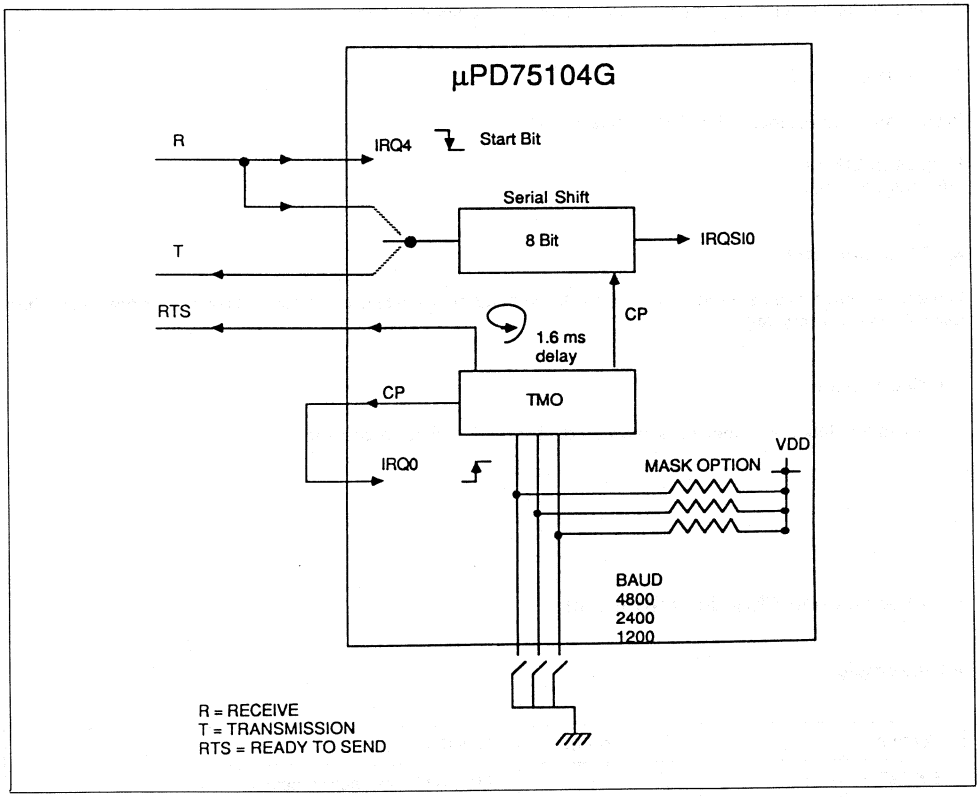
Vectors	Priority	Function
EX INT → IRQ4	1	Start Bit '0' for receive data
EX INT0 → IRQ0	2	Used for timing of data transmission/receive via T/C TMO.
IN INTSI0 → IRQSI0	3	Used for end of serial trans/rec.

All other interrupts are tested.

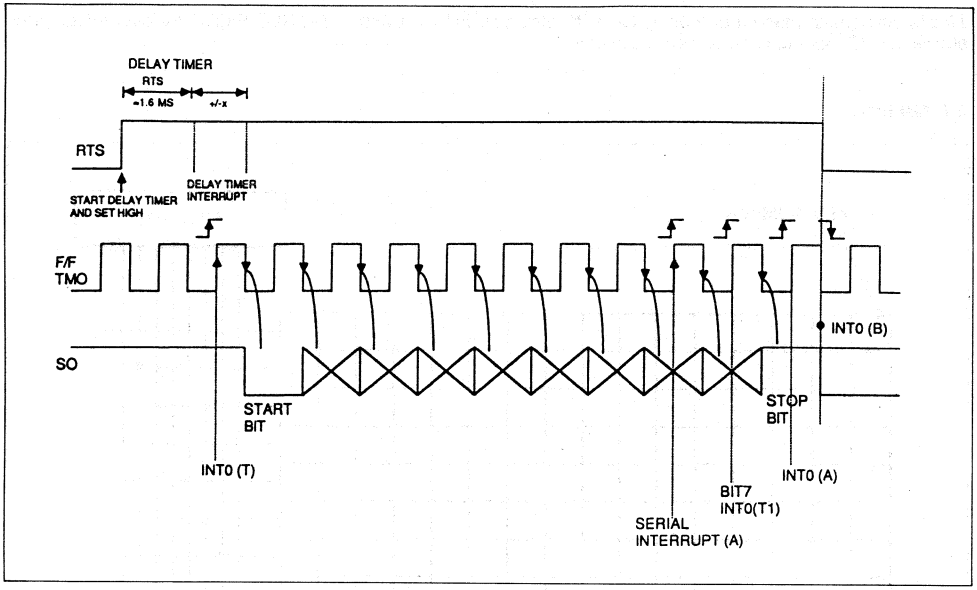
4.2 Reg. Banks

REG. Bank No.	Function
RB0	All Interrupts
RB1/RB2	Main Scan Program
RB3	All Subroutines

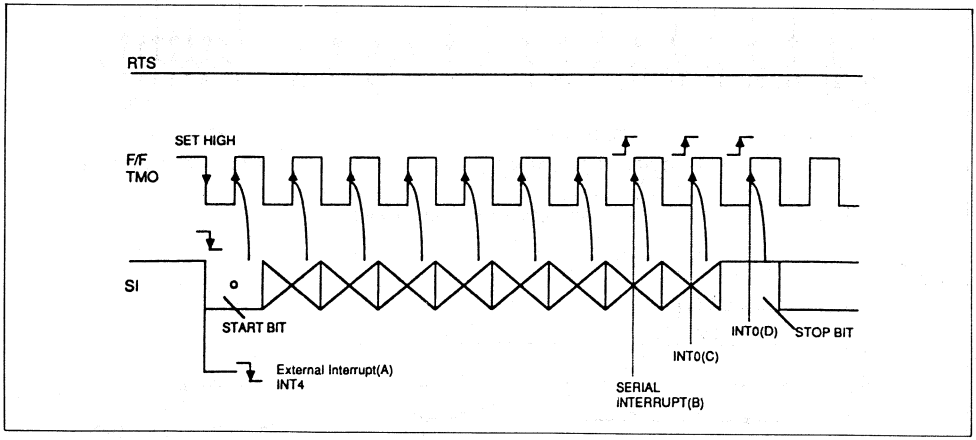
5. Hardware Blocks On-Chip for Serial Interface



5.1 TRANS (T)



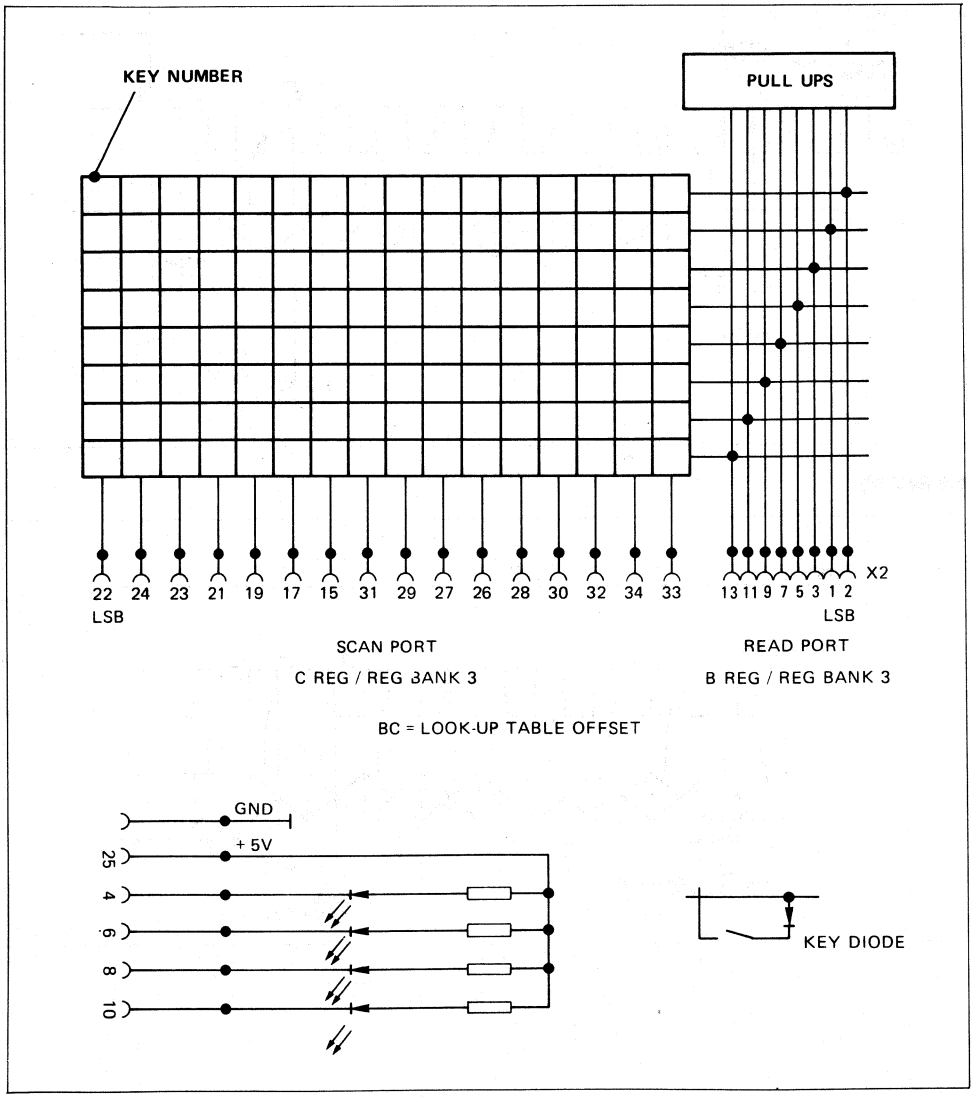
5.2 REC (R)



6. Key Scan Off Set Calculation

This is the pointer used for all look up-table off sets and is in REG BANK 3, BC REG PAIR. The following diagram shows the BC look-up table off set calculation.

6.1 128 Keys



7. Look-up Table Flags

TABLE 1	−>	UNSHIFT	TABLE 4 REPEAT FLAGS
TABLE 2	−>	SHIFT	TABLE 4 REPEAT FLAGS
TABLE 3	−>	CTRL	TABLE 5 REPEAT FLAGS

7.1 Table 4/5 Codes

01	−>	NR	
02	−>	FR	Every 60 ms transmission
03	−>	SR	Every 180 ms transmission

Notes: a) All table off sets are given in BC REG in RB3.

b) **EE HEX** means no transmission.

8. ASCII Tables

KEY NO.	BC OFF SET	TABLE 1	TABLE 2	TABLE 3	01 TABLE 4	TABLE 5
	00	B0	B0	B0	02	01
←	01	B1	21	EEX	01	01
	02	89	89	89	01	01
	03	A2	A2	A2	02	01
C	04	63	43	03	02	01
9	05	36	26	EEX	02	01
M	06	6D	4D	0D	01	02
	07	A8	A8	A8	01	01
	08	8B	8B	8B	02	01
↵	09	0D	0D	0D	01	02
	0A	AB	AB	AB	01	01
	0B	B6	B6	B6	02	01
	0C	95	95	95	01	02
	0D	C3	C3	C3	01	01
	0E	CB	CB	CB	01	01
	0F	D3	D3	D3		01
					01	
BREAK	10	FF	FF	FF	01	01
	11	A0	A0	A0	02	01
	12	80	81	EEX	02	01
D	13	64	44	04	02	01
F	14	66	46	06	02	01
Z	15	7A	5A	1A	02	01
J	16	6A	4A	0A	02	02
P	17	30	3D	EEX	02	01
.	18	2D	5F	1F	02	01
↑	19	08	08	08	01	02
	1A	88	88	88	03	01
	1B	98	98	98	03	03
	1C	92	92	92	01	03
	1D	C4	C4	C4	01	01
	1E	CC	CC	CC	01	01
	1F	D4	D4	D4		01

Note: 1) BC is off set REg for all look-up
2) EE = do not trans

KEY NO.	BC	TABLE 1	TABLE 2	TABLE 3	TABLE 4	TABLE 5
	20	1B	1B	1B	01	01
	21	3C	3E	EEX	02	01
0	22	71	51	11	02	01
m	23	65	45	05	02	01
m	24	72	52	12	02	01
H	25	68	48	08	02	02
C	26	75	55	15	02	01
.	27	2E	3A	EEX	02	01
0	28	7C	5C	1C	02	01
	29	AA	AA	AA	01	01
	2A	7F	7F	7F	02	02
	2B	B5	B5	B5	01	01
	2C	BD	BD	BD	01	01
	2D	C5	C5	C5	01	01
	2E	CD	CD	CD	01	01
	2F	D5	D5	D5	01	01
CODE	30	EEX	EEX	EEX	01	01
	31	B1	B1	B1	01	01
	32	8A	8A	8A	01	01
X	33	78	58	18	02	01
	34	74	54	14	02	01
	35	20	20	EEX	02	01
−	36	69	49	09	02	02
-	37	2C	3B	EEX	02	01
Y	38	78	5B	1B	02	01
	39	8C	8C	8C	01	01
STOP	3A	B8	B8	B8	01	01
←	3B	97	97	97	02	02
	3C	C0	C0	C0	01	01
	3D	C6	C6	C6	01	01
	3E	CE	CE	CE	01	01
	3F	D6	D6	D6	01	01

LED

KEY NO.	BC	TABLE 1	TABLE 2	TABLE 3	TABLE 4	TABLE 5
	40	B2	B2	B2	01	01
	41	09	09	09	02	02
A	42	61	41	01	02	01
3	43	33	40	00	02	01
4	44	34	24	EEX	02	01
N	45	6E	4E	0E	02	01
7	46	37	2F	EEX	02	01
	47	A7	A7	A7	01	01
	48	70	50	10	02	01
	49	5E	60	EEX	02	01
	4A	96	96	96	03	03
START	4B	B9	B9	B9	01	01
	4C	BA	BA	BA	01	01
	4D	C7	C7	C7	01	01
	4E	CF	CF	CF	01	01
	4F	D7	D7	DF	01	01
	50	B3	B3	B3	01	01
	51	EEX	EEX	EEX	01	01
	52	A1	A1	A1	01	01
	53	A3	A3	A3	01	01
5	54	35	25		02	01
	55	A5	A5	A5	01	01
	56	A6	A6	A6	01	01
L	57	6C	4C	0C	02	01
8	58	7E	3F	1E	02	01
+	59	2B	2A	EEX	02	01
↑	5A	93	93	93	02	02
	5B	BC	BC	BC	01	01
	5C	B7	B7	B7	01	01
	5D	C8	C8	C8	01	01
	5E	D0	D0	D0	01	01
	5F	D8	D8	D8	01	01

LED

LED(L/C)

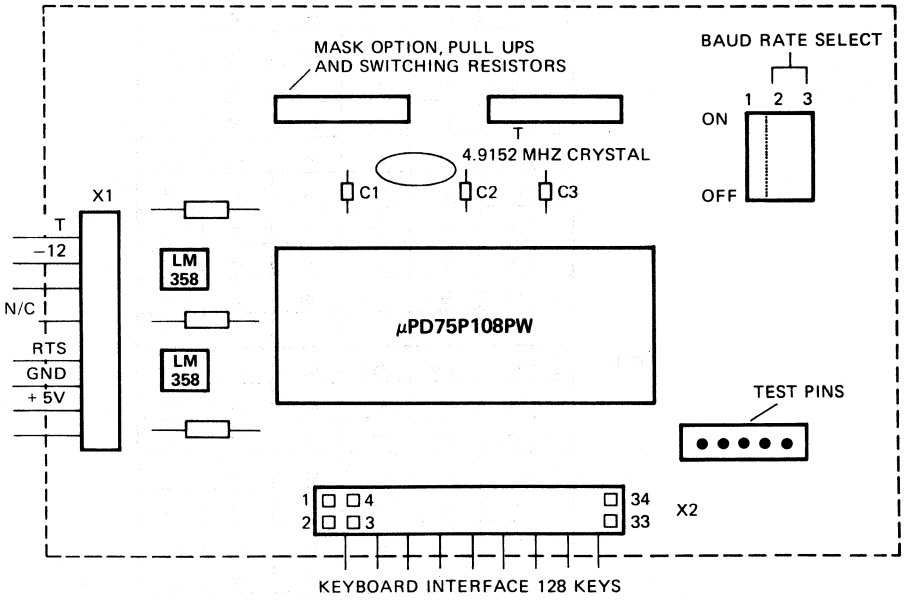
	KEY NO.	BC	TABLE 1	TABLE 2	TABLE 3	TABLE 4	TABLE 5
LED		60	B4	B4	B4	01	01
SHIFT		61	EEX	EEX	EEX	01	01
	2	62	32	22	EEX	02	01
	W	63	77	57	17	02	01
		64	A4	A4	A4	01	01
	D	65	67	47	07	02	01
	8	66	38	28		02	01
	O	67	6F	47	0F	02	01
		68	A9	A9	A9	01	01
	#	69	23	27		02	01
		6A	90	90	90	03	03
	→	6B	91	91	91	02	02
		6C	C1	C1	C1	01	01
		6D	C9	C9	C9	01	01
		6E	D1	D1	D1	01	01
		6F	EEX	EEX	EEX	01	01
CTRL	CTRL	70	EEX	EEX	EEX	01	01
		71	8D	8D	8D	01	01
	Y	72	79	59	19	02	01
	S	73	73	53	13	02	01
	V	74	76	56	16	02	01
	B	75	62	42	02	02	01
	X	76	6B	4B	0B	02	02
	9	77	39	29	EEX	02	01
	C	78	7D	5D	1D	02	01
		79	EEX	EEX	EEX	01	01
SHIFT		7A	BB	BB	BB	01	01
	↗	7B	94	94	94	01	01
		7C	C2	C2	C2	01	01
		7D	CA	CA	CA	01	01
		7E	D2	D2	D2	01	01
		7F	EEX	EEX	EEX	01	01

9. Receive-Codes

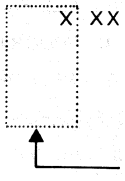
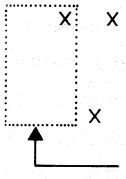
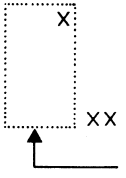
HEX	FUNCTION
18	RESET

10. Demo Board

NEC-EE demo board is as shown below:



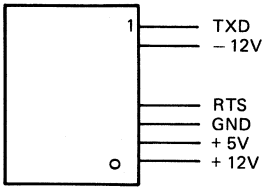
10.1 Demo Board Baud Rate Selection

BAUD RATE	KEY SWITCH
1.2K	OFF ON 
2.4K	OFF ON 
4.8K	OFF ON 

10.2 Connectors on Demo Board

a) From Demo Board to Keyboard (X2).

b) RS232 from Demo Board to PC (X1).

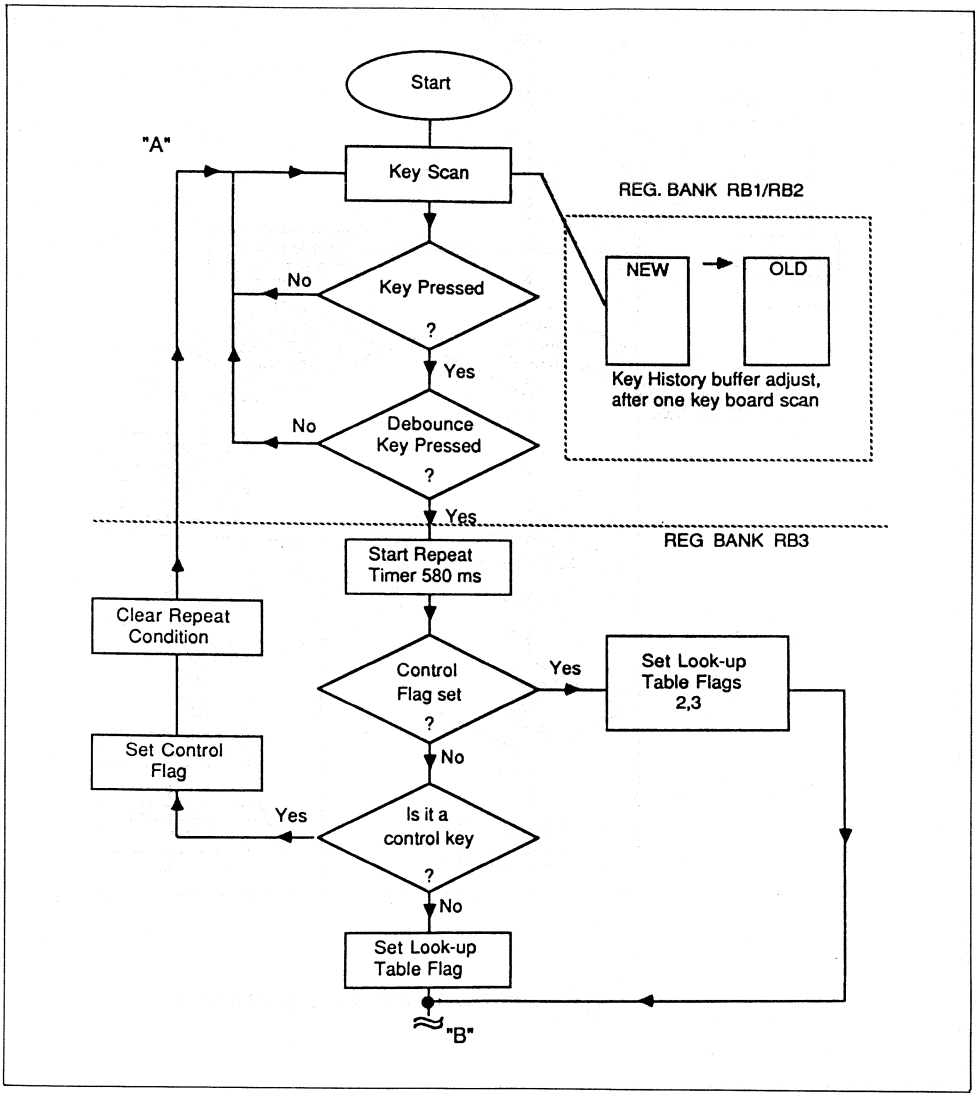


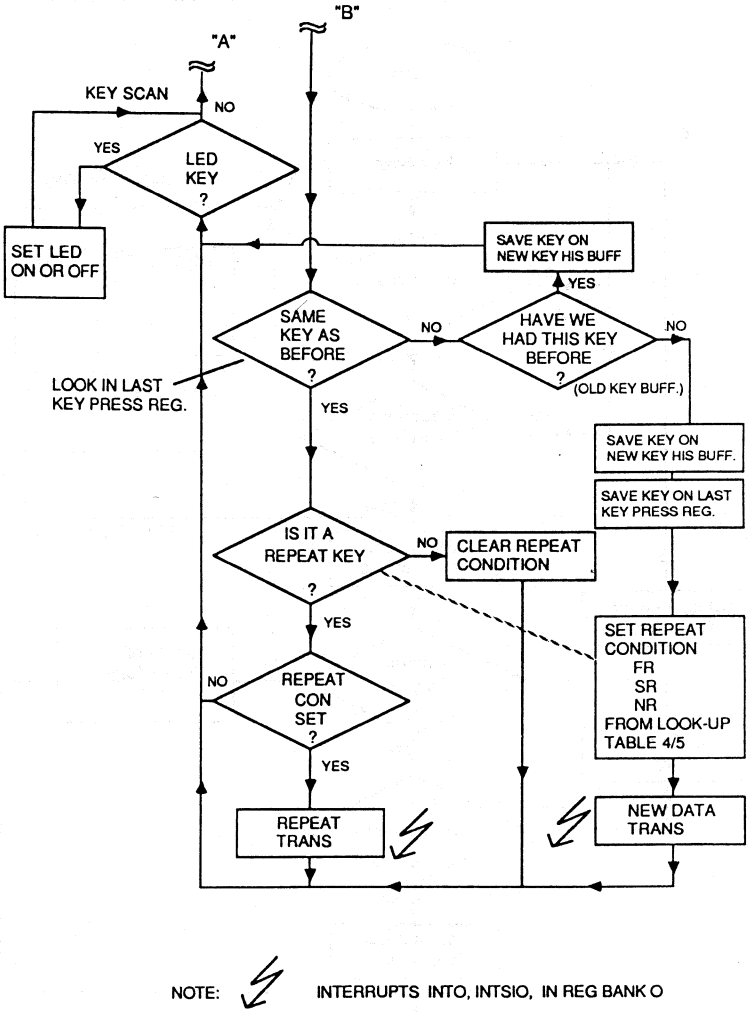
11. Software

The demo program is approximately 2K bytes long. It consists of 18 moduals, as follows:

No.	Modual Name	Function
1	NECIN	INITIALIZE PROCESSOR
2	NECMA	MAIN PROGRAM KEY SCAN
3	NECKEYP	KEY PRESS PROGRAM SUB FROM NECM
4	NECREPT	CHECKS FOR A REPEAT KEY SUB FROM NECKEYP
5	NECKEYCH	CHECKS OLD KEY HISTORY BUFFER SUB FROM NECKEYP
6	NECLED	LED CONTROL PROGRAM SUB FROM NECKEYP
7	NECBASIC	REPEAT TIMING PROGRAM SUB FROM NECKEYP
9	NECINTO	TRANSMIT AND RECEIVE TIMING PROGRAM INTERRUPT PROGRAM
10	NECBAUD	BAUD RATE PROGRAM
11	NECSERL	SERIAL INPUT/OUTPUT PROGRAM INTERRUPT PROGRAM
12	NECREC	SERIAL INPUT START PROGRAM INTERRUPT PROGRAM
13	NECACDAT	ACTION ON RECEIVED DATA PROGRAM SUB FOR NECINTO
14—18	NECTABLE 1—5	LOOK-UP TABLES, ASCII ONLY

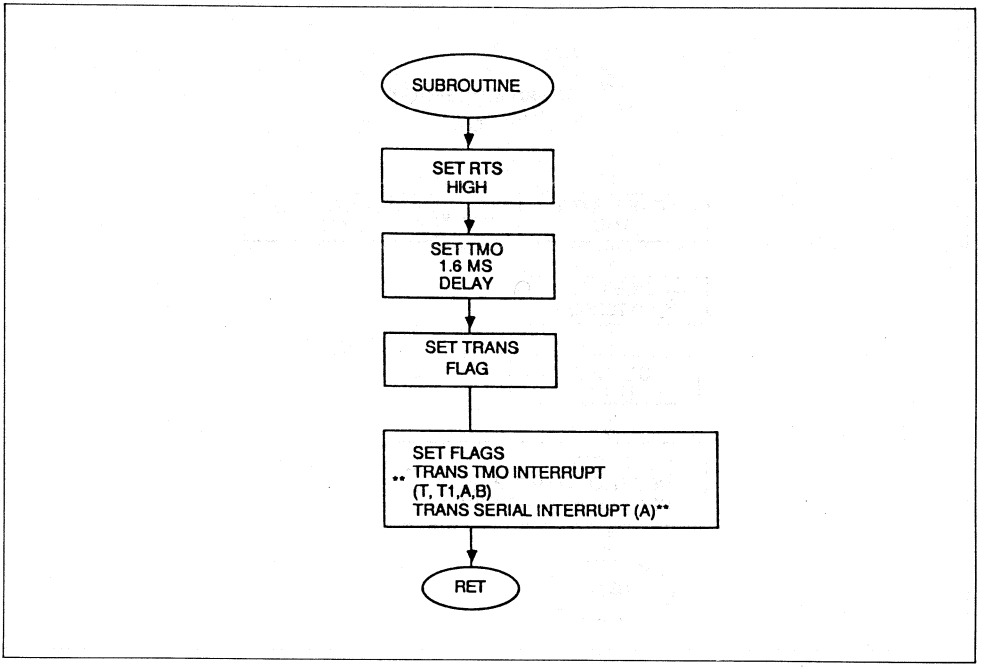
11.2 Key Board Flow Diagram





NOTE: INTERRUPTS INTO, INTSIO, IN REG BANK O

11.2 Transmission (T)



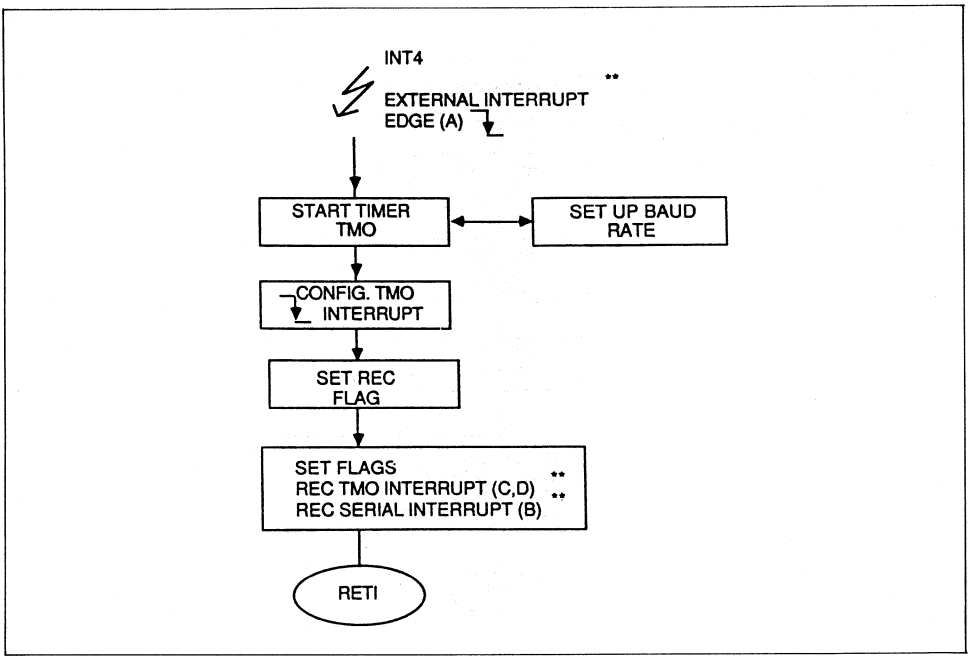
Notes: The 16-bit sequential buffer is used as a record of which flag is set in transmission/receive.

* BSB0/BSB1	INT/0
* BSB2	INTSI0

→
→

**See Section 5, Sub Section 5.1

11.3 Receive (R)



Note: **See Section 4, Sub Section 5.2

Using the μPD70208/70216 Interrupt Controller

Contents:

Author: R. Naro
NEC Electronics NATICK/USA

Persons to contact: S. Gupta, B. Peters
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD70208/216 User's Manual
Data Book Microprocessors and Peripherals

Related Products

μPD70208 16 bit Microprocessor	CMOS
μPD70216 16 bit Microprocessor	CMOS
μPD71059 Interrupt controller	CMOS

1. Introduction

Because of the importance of interrupts in real-time applications, a powerful yet flixible interrupt controller is required by most real-time applications. Applications using the μPD70208/216 microprocessors have access to a powerful programmable interrupt controller called the ICU (interrupt control unit). The ICU is a general purpose interrupt controller with liks to the μPD71059 and μPD8259A interrupt controllers. The ICU contains a number of advanced features which enhance its use in a wide variety of applications. These features include:

- eight interrupt inputs (1 internal, 7 external)
- easy cascade with slave μPD71059 and μPD8259A controllers
- software compatible with the μPD71059 and μPD8259A
- edge or level trigger operation
- fixed or rotating interrupt priorities
- software polling mode

The capabilities and features of the μPD70208/216 ICU quickly translate into tangible benefits for users of these devices. For the vast majority of μPD70208/216 applications, the eight hardware interrupts are sufficient. However, even in the event tht eight channels cannot meet the requirements of a particular application, interrupts can be expanded to a total of 57 independently vectored interrupt sources by the use of slave μPD71059 interrupt controllers operating in the cascade mode.

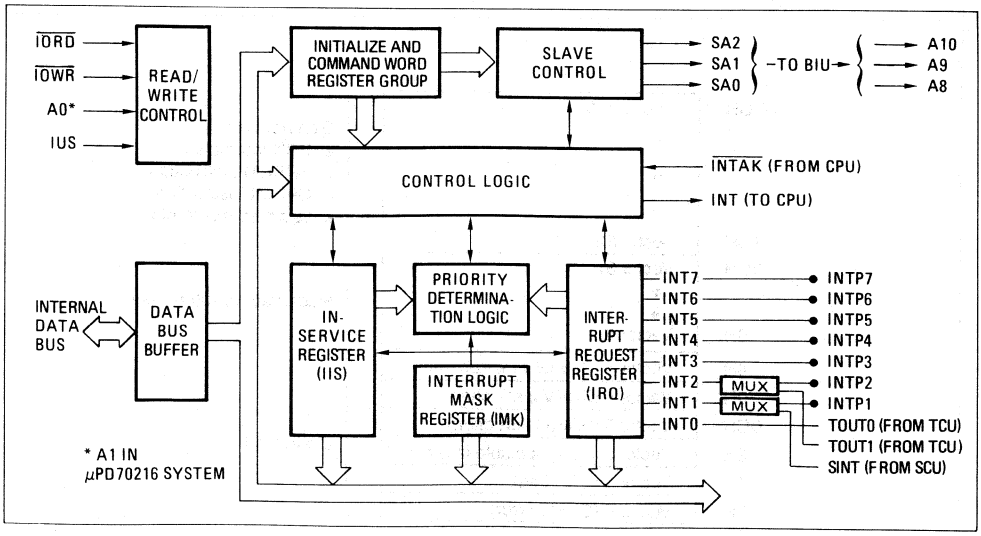


Figure 1. ICU Block Diagram

Also important for both the migration of older designs to low power, high integration microprocessors like the μPD70208/216 as well as new designs is that software compatibility with the μPD71059 and μPD8259A interrupt controllers is maintained. The design of the ICU, like that of the other LSI peripherals integrated on the μPD70208/216, is modeled after an existing discrete LSI peripheral. The significance of compatibility is evident when a design is ported from an existing μPD8086/88 system using the μPD8259A. Rather than having to completely re-write the interrupt controller initialization and driver software, all modifications are limited to enabling and positioning the ICU in the I/O map at the same I/O addresses. Thus the porting of the original software generally involves no modifications of existing software but instead only requires a small module to initialize the μPD70208/216 internal peripheral registers and transfer contrl to the original initialization software.

The remaining features are what make the ICU a complete interrupt controller. The ability to select the triggering mode, control interrupt priorities and other operating modes allows the ICU to operate in a wide variety of environments. Since a number of viable options are always available, the ICU can always be fit to a particular application without concern for introducing incompatibilities.

2. Interrupt Processing

Before delving into the detailed operation of the ICU, a brief overview of the operation of interrupts on the μPD70208/216 is in order. The μPD70208/216 support a variety of interrupt sources which are divided into hardware and software interrupts. Hardware interrupts do not have any temporal relationship to the execution of a program and are considered asynchronous events. On the other hand, the software interrupts always occur as a direct result of program execution and re considered to be synchronous events. Other than having different interrupt sources, hardware and software interrupts are serviced identically by the software interrupt handlers.

Each interrupt source is assigned a unique vector which is used by the CPU to locate the associated interrupt service routine. When an interrupt occurs and is acknowledged, the CPU must either accept (maskable interrupts) or compute internally (all others) the interrupt vector used to index the interrupt vector table. The interrupt vector table is located at the base of the memory address space and contains 256 doubleword interrupt vectors. The format of the interrupt vector table is shown in figure 2.

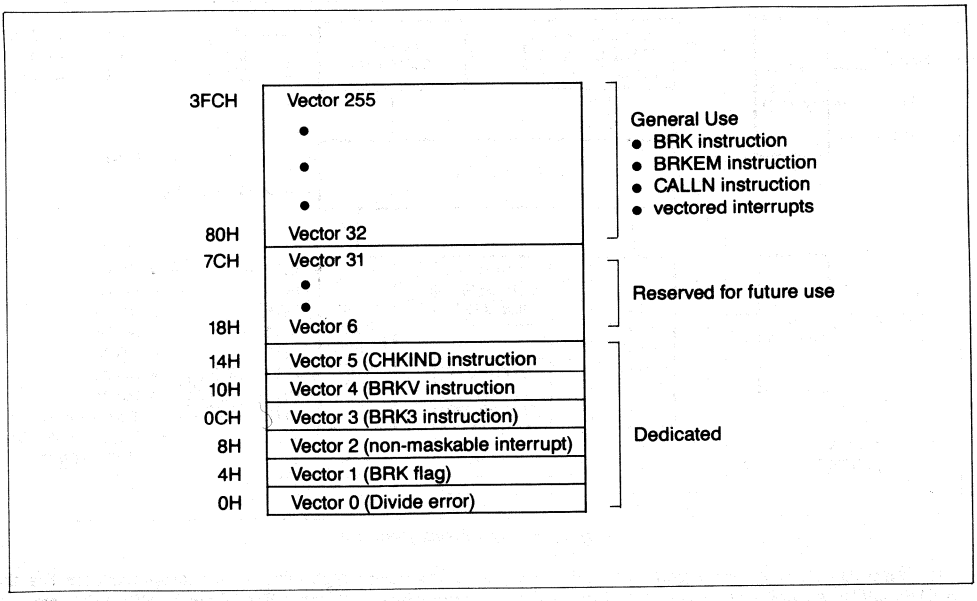


Figure 2. Interrupt Vector Table

Each of the entries in the interrupt vector table contains a far pointer to the interrupt handler responsible for servicing the interrupt. The low order word contains the program counter of the interrupt handler while the upper word contains the program segment.

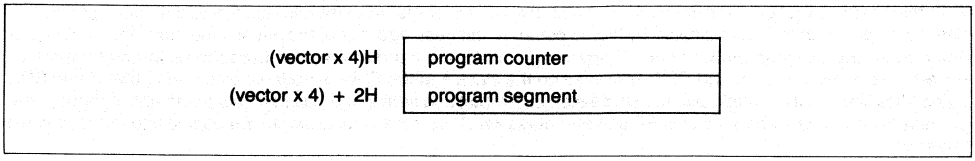


Figure 3. Interrupt Vector Format

There are two separate classes of hardware interrupts. An NMI (non-maskable interrupt) is a dedicated rising edge-triggered input which cannot be disabled or masked by software. Only one NMI interrupt exists in a system although different events can be made to generate an NMI through the use of external logic. Once recognized, the CPU responds to the NMI by transferring control to the interrupt handler pointed to by vector 2 in the interrupt vector table.

The other class of hardware interrupts are the maskable interrupts. Maskable interrupts are different from the non-maskable interrupt since they must be first processed by the ICU before being presented to the CPU. However, unlike the NMI input, the CPU can choose to ignore maskable interrupts by clearing the IE flag in the PSW. Maskable interrupts are also special because they can supply any one of the 256 vectors to be used to service the interrupt. Exactly which vector number will be returned is dependent on the initialization of the ICU and the highest priority unmasked interrupt input requesting service.

If maskable interrupts are enabled and an unmasked interrupt input requests service, the CPU responds by running a pair of interrupt acknowledge bus cycles. The first of these bus cycles tells the ICU and any slave μPD71059 interrupt controlers that the CPU is preparing to receive the vector number and that the highest priority device requesting interrupt service should be determined. In response to the first interrupt acknowledge bus cycle the ICU generates the HOLD signal. HOLD is used internally within the ICU to latch the state of the interrupt request inputs in the interrupt request register for the subsequent priority determination phase.

The second interrupt acknowledge bus cycle is then used to read the actual interrupt vector. The BUSLOCK output is asserted for the duration the two interrupt acknowledge bus cycles to prevent interference in a multi-master system. The bus timing for a maskable interrupt is shown in the following figure.

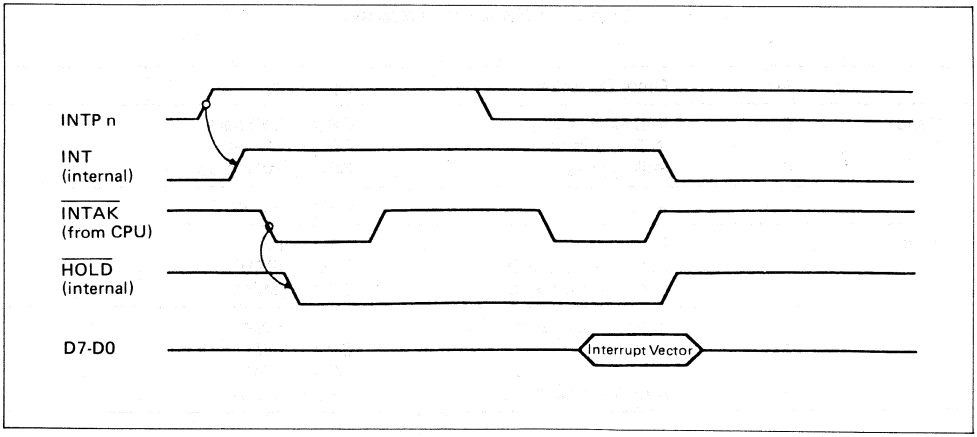


Figure 4. Interrupt Timing

With the freshly supplied interrupt vector in hand, the CPU is prepared to transfer control to the interrupt handler. The interrupt vector is first multiplied by four to compute the base address of the associated interrupt vector table entry. Next, the program counter and program segment for the handler are read from the table and temporarily stored. The current PC, PS and PSW are saved on the stack and the PSW is updated by clearing the IE and BRK fields (disabling further maskable and single-step interrupts) and setting the MD field (selecting native mode). The process is completed when the new PC and PS values which were previously stored are copied into the respective registers.

Following the execution of the interrupt handler, control is returned to the point of interruption by the IRET instruction. The IRET instruction simply pops the PC, PS and PSW off the stack and returns the CPU to the state at the point of interruption.

3. ICU Register Set

The ICU contains a number of internal registers which control and define its operation as well as additional external registers which affect its operation. To use the ICU, it must first be enabled by setting the IS field in the OPSEL (On-Chip Peripheral Selection) register. In a similar manner, the ICU must be fixed in the I/O address space by the OPHA (On-Chip Peripheral High Address) and IULA (ICU Low Address) registers. Finally, since there are a total of ten internal and external interrupt sources, the OPCN (On-Chip Peripheral Connection) register must be used to select which eight of the ten possible interrupt sources will be connected to the ICU. These registers are all external to the ICU.

Input/Output instructions to the I/O addresses selected by the OPHA and IULA registers are used to read from and write to the ICU registers. The internal addresses of the ICU registers are determined by address line A0 for the μPD70208 and address line A1 for the μPD70216. Because there are more registers then addresses, both state sequences and programmed control of the read/write registers are used to support access to all registers. The following table contains the ICU register and the corresponding address.

Table 1. ICU Register Addresses

	A0	Other Condition	Operation
Read	0	IMD selects IRQ	CPU ← IRQ data
	0	IMD selects IIS	CPU ← IIS data
	0	Polling phase	CPU ← Polling data
	1	—	CPU ← IMKW
Write	0	D4 = 1	CPU → IIW1
	0	(D4 = 0) · (D3 = 0)	CPU → IPFW
	0	(D4 = 0) · D3 = 1)	CPU → IMDW
	1	Initialize Sequence	CPU → IIW2
	1		CPU → IIW3
	1		CPU → IIW4
	1	After Initialization	CPU → IMKW

Note 1: In polling phase, polling data has priority over the contents of the IRQ or IIS register when read.

Note 2: A1 in μPD70216 systems.

Interrupt Request Register

The IRQ (interrupt request) register reflects the current state of each of the interrupt inputs. Whenever an interrupt input is asserted, the corresponding IRQ bit is set and if not masked by the IMK register, the interrupt is passed through to the ICU priority determination logic. Although the IMK register masks the propagation of an interrupt request, reads of the IRQ register are unaffected by the contents of the IMK register and always reflect the state of the interrupt inputs. Applications using fewer than the seven external interrupts can take advantage of this situation and mask the unused inputs. The masked inputs can then be used as a general purpose input pins without any possibility of generating an interrupt.

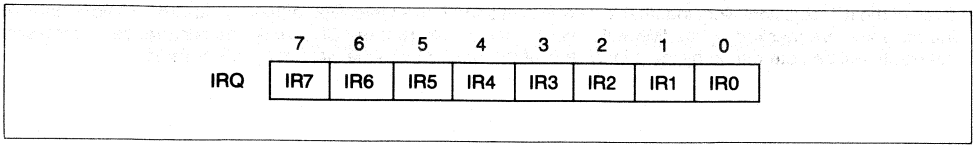


Figure 5. IRQ Register

Interrupt Mask Register

The IMK (interrupt mask) register contains a mask word with each bit corresponding to a physical interrupt input. When a bit within the IMK register is set, the corresponding interrupt input is masked and cannot propagate any further within the ICU. Thus the IMK register allows individual interrupts to disabled locally on a case by case basis by software without affecting other interrupt inputs. When the ICU is reset by writing the first initialization word, the IMK register is cleared and all interrupts are enabled.

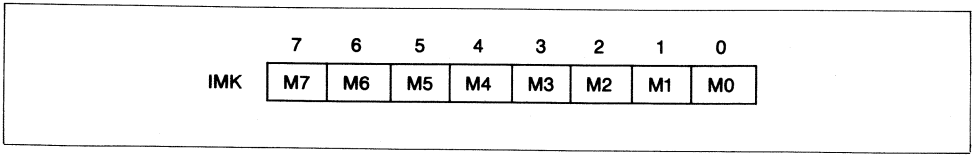


Figure 6. IMK Register

Active interrupt inputs with their IRQ bit set but masked are only temporarily ignored. An interrupt will occur at a later time if software unmasks the interrupt input and the IRQ bit is still set. If the interrupt disappears before the next interrupt acknowledge, no interrupt will be generated.

Interrupt In-Service Register

The IIS (interrupt in-service) register contains information on the interrupt levels currently being serviced. When an interrupt is acknowledged, the corresponding bit within the IIS register is set and used to prevent further interrupts from the same or lower priority levels. Also, the state of the IIS register is used to determine if a new CPU interrupt request should be made if another higher priority interrupt input is asserted. The nesting of interrupts depends on the operating mode of the ICU and in the exceptional nesting mode, the nesting of both higher and lower priority interrupts is supported. This is explained in more detail in the section describing the nesting modes.

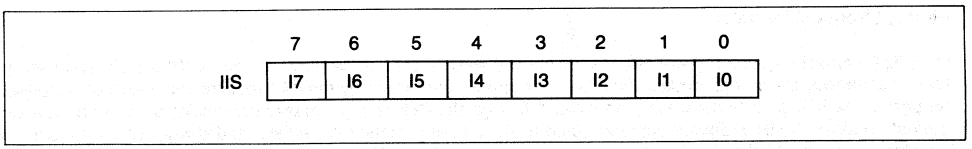


Figure 7. IIS Register

Each of the ICU registers work together to determine if and when an interrupt request to the CPU will be generated. Inputs which are masked by the IMK will never be able to interrupt the CPU while the occurrence of unmasked interrupts will be determined by the priority level of the interrupt and the state of the IIS register.

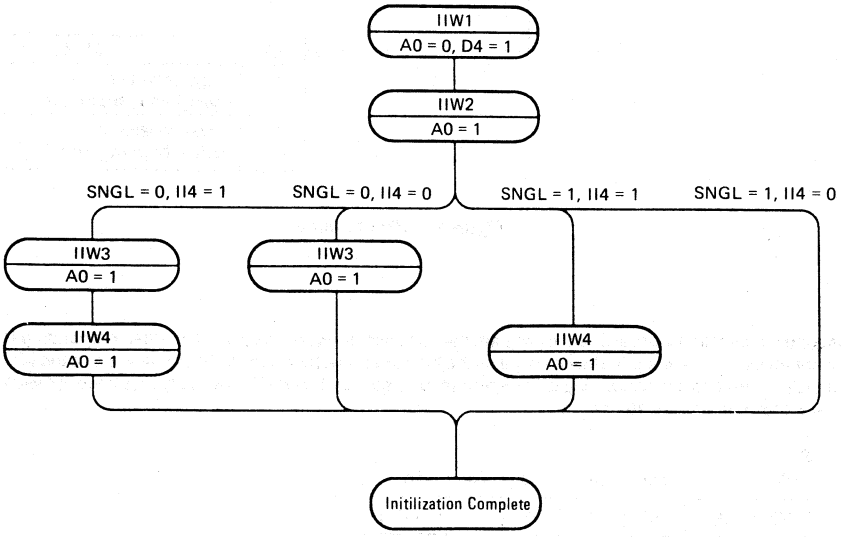
4. Initialization Words

Before enabling the CPU to respond to maskable interrupts, two to four initialization words must be written to the ICU to select the operating characteristics. All of the initialization words must be written in order to correctly initialize the ICU and the final two initialization words are optional and used only if selected by the IIW1 command. Because a state machine is used to accept the initialization words, ICU initialization software must always start from the first initialization word (IIW1).

IIW1 also has special significance since it automatically generates an internal reset signal for the ICU whenever it is addressed. This reset signal places the ICU in the following state:

- normal nesting is selected
- edge-triggered logic is reset (a low-to-high transition is required)
- IMK, IIS, IIW4 ← 0
- INT0 is highest priority, INT7 is the lowest priority
- exceptional nesting mode is released
- IRQ selected for reading

An example of a typical ICU initialization sequence can be found in appendix C. Once the initialization sequence is complete, command to read and write the operational control registers can be issued.



Note: In μPD70216 systems, A1 is used in place of A0.

Figure 8. Initialization Sequences

IIW1

IIW1 is used to select the triggering mode, specify whether or not slaves are present in the system and if IIW4 will be part of the initialization sequence. IIW1 is the only initialization word that can be written at any time and is specified by setting the address input to 0 (A0 for the μPD70208 and A1 for the μPD70216) and bit 4 in the initialization word to 1. Applications have the option to use the ICU in either the edge-triggered or level-triggered modes as determined by the LEV field. If the extended mode is selected, IIW3 must be written to specify the connection of the slave μPD71059 interrupt controllers used in the system. Finally, the II4 field selects whether the ICU should expect the IIW4 command to be issued.

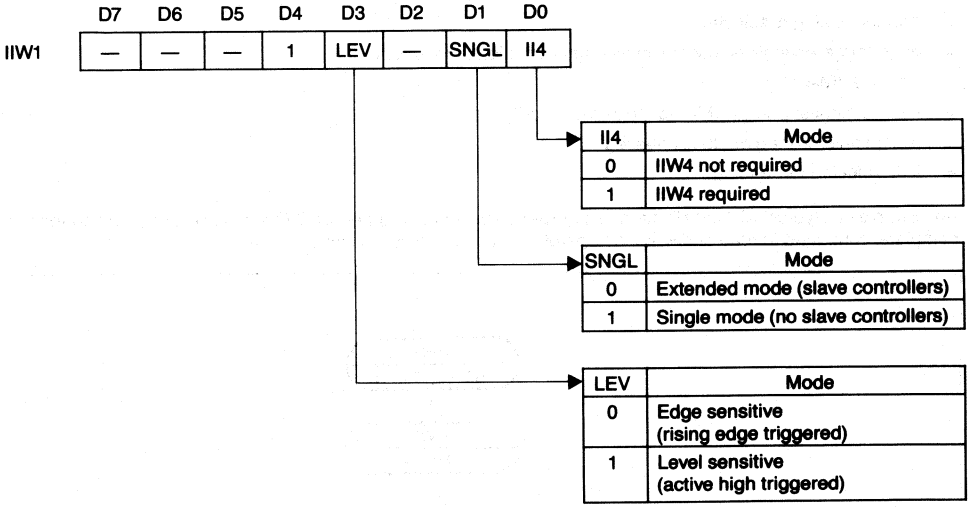


Figure 9. IIW1 Format

IIW2

The second initialization word is used to program the base vector returned by the ICU when responding to the CPU request for an interrupt vector. Software specifies the high order five bits of the interrupt vector and the ICU appends the low order three bits correcponding to the interrupt to be serviced. The IIW2 word and the interrupt vector assignments are shown in figures 10 and 11.

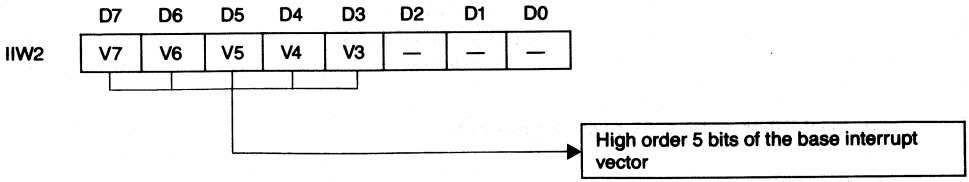


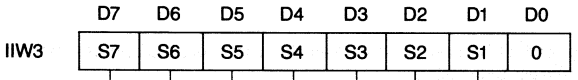
Figure 10. IIW2 Format

Interrupt Levels	Output Vector							
	D7	D6	D5	D4	D3	D2	D1	D0
INT0	V7	V6	V5	V4	V3	0	0	0
INT1	V7	V6	V5	V4	V3	0	0	1
INT2	V7	V6	V5	V4	V3	0	1	0
INT3	V7	V6	V5	V4	V3	0	1	1
INT4	V7	V6	V5	V4	V3	1	0	0
INT5	V7	V6	V5	V4	V3	1	0	1
INT6	V7	V6	V5	V4	V3	1	1	0
INT7	V7	V6	V4	V4	V3	1	1	1

Figure 11. Interrupt Vector Assignments

IIW3

IIW3 is only required in cascade mode applications which use slave μPD71059 interrupt controllers. Each slave controller is attached to one of the INTP1—INTP7 interrupt inputs with the corresponding bit in IIW3 set to indicate a slave channel connection. Note that unlike the μPD71059 and μPD8259A interrupt controllers, there is no provision for the ICU to operate in the slave mode.



Sn	Mode
0	INTPn is not a slave input
1	INTPn is a slave input

Figure 12. IIW3 Format

IIW4

IIW4 is programmed only if the II4 field in IIW1 was set. IIW4 selects the nesting mode and the finish interrupt mode. The operation of these modes is described in a subsequent section of this MicroNote.

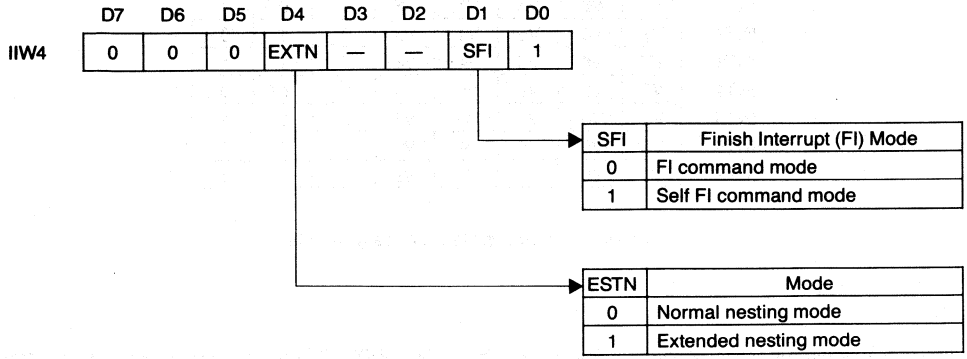


Figure 13. IIW4 Format

5. Command Words

Command words are used modify the operation of the ICU. Commands are available to mask and unmask interrupt inputs, change priority levels and terminate interrupt processing.

IMKW

The IMKW (interrupt mask word) is used to selectively enable or disable interrupts by controlling the state of the IMK register. Each bit in the IMK register enables or disables the corresponding interrupt request from propagating further within the ICU. The typical scenario for masking or unmasking an interrupt involves reading the IMK register, setting or clearing the appropriate bits and writing the result back to the IMK register.

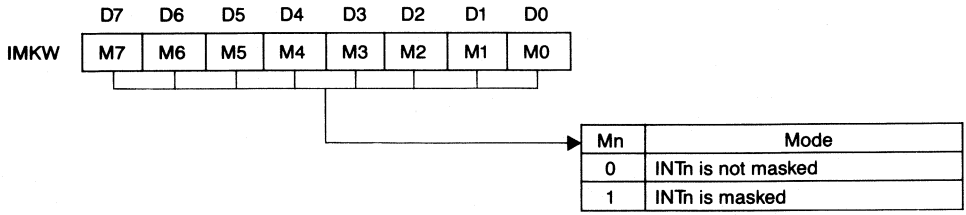


Figure 14. IMWK Format

For masking or unmasking of single interrupts, the SET1 and CLR1 instructions are used. To enable or disable multiple interrupts, the logical AND or OR instructions with a prepared mask can be used. Both methods of masking and unmasking interrupts at the ICU are shown below in the example.

Example 1. Masking/Unmasking Interrupts

```
. . .
MOV     DX,  IMKW           ; Point to the IMK register
IN      AL,  DX             ; Read the current contents
CLR1    AL,  1              ; Clear (unmask) M1
OUT     DX,  AL             ; Write the new mask back to the IMK register
. . .
. . .
. . .
MOV     DX,  IMWK           ; Point to the IMK register
IN      AL,  DX             ; Read the current contents
        OR   AL, 00000111b  ; Mask interrupts M0, M1 and M2
OUT     DX,  AL             ; Write the new mask back to the IMK register
. . .
```

The IMK register is also used to dynamically after the interrupt priorities in the exceptional nesting mode. Normally all lower priority interrupts are masked during the processing of an interrupt. However, in the exceptional nesting mode, both lower and higher priority interrupts can be serviced. The use of the exceptional nesting mode is covered in detail in a later section.

IPFW

The IPFW (interrupt finish and priority and finish word) consists of four fields which the CPU manipulates to generate the desired command. The RP field is set whenever software desires to force a rotation of the interrupt priorities. The SIL field in turn controls whether the interrupt level is supplied in the IL2—IL0 field (specific) or if the highest priority interrupt as determined by the IIS register (non-specific). The FI field is set if a finish interrupt command is selected, otherwise it indicates that some other ICU command is being issued.

	D7	D6	D5	D4	D3	D2	D1	D0
IPFW	RP	SIL	FI	0	0	IL2	IL1	IL0

IL2	IL1	IL0	
0	0	0	Interrupt level 0
0	0	1	Interrupt level 1
0	1	0	Interrupt level 2
0	1	1	Interrupt level 3
1	0	0	Interrupt level 4
1	0	1	Interrupt level 5
1	1	0	Interrupt level 6
1	1	1	Interrupt level 7

RP	SIL	FI				
0	0	1	FI Command	No level specification	No rotation	Formal FI command
1	0	1			Rotation	Normal rotation FI command
0	1	1		Level specification	No rotation	Specific FI command
1	1	1			Rotation	Specific rotation FI command
0	1	0	Non-FI Command		No rotation	No operation
1	1	0			Rotation	Specific rotation command
0	0	0		No level specification	No rotation	Self FI mode rotation reset
1	0	0			Rotation	Self FI mode rotation set

Figure 15. IPFW Format

Each of the IPFW commands are explained in detail below.

Normal FI Command

The normal FI command is used to inform the ICU that interrupt processing has completed and that the highest priority bit in the IIS register is to be reset. Note that this command requires that the ICU be able to guarantee that the highest priority IIS bit corresponds to the interrupt handler terminating service. This is not the case if the handler has changed priority levels or is executing in the exceptional nesting mode. These cases require the use of the specific FI command.

	7	6	5	4	3	2	1	0
PFCW =	0	0	1	0	0	—	—	—

Figure 16. Normal FI Command

Specific FI Command

The specific FI command operates identically to the normal FI command but with the following exception. Rather than resetting the highest priority bit in the IIS register, the specific FI command encodes the interrupt level in the IL2—IL0 field of the command and the specified IIS bit can be unambiguously reset. Because no ambiguities exist concerning the interrupt priority level or nesting mode, the specific FI command can be used in all of the ICU operatig modes.

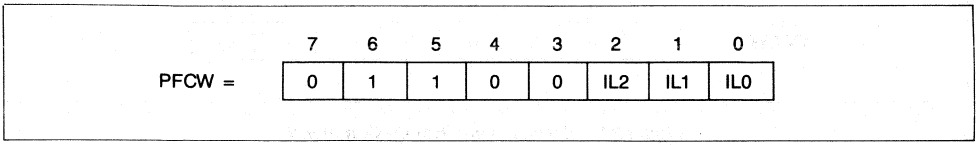


Figure 17. Specific FI Command

Self FI Mode

The self FI mode (enabled by SFI field in IIW4) takes a slightly different tack. Rather than requiring the interrupt handler to clear the appropriate IIS bit at the completion of interrupt processing, the self FI mode will automatically perform a normal FI command at the end of the second interrupt acknowledge bus cycle. Although the use of the self FI mode can lead to shorter interrupt handlers (since there is no need to issue a FI command), other problems can arise when using the self FI mode.

First, since the IIS bit for the interrupt is service is immediately cleared, the IIS register no longer indicates which, if any, interrupt levels are being services. The priority ranking of the interrupts is also disturbed because if maskable interrupts were to be re-enabled by the interrupt handler, both higher and lower priority interrupts could be recognized and serviced. In the level-triggered mode, it is also possible for an interrupt to interrupt itself if the CPU does not leave interrupts disaled until the end of interrupt processing. Because of the potential for problems, it is best to leave maskable interrupts disabled for the duration of service when the self FI mode is used.

Self FI Rotation

The self FI mode by itself does not alter the interrupt priorities during of after servicing. The ICU allows the self FI mode to automatically change the priority of the serviced interrupt to the lowest priority when the IIS bit is reset through the use of the self FI rotate set and clear commands.

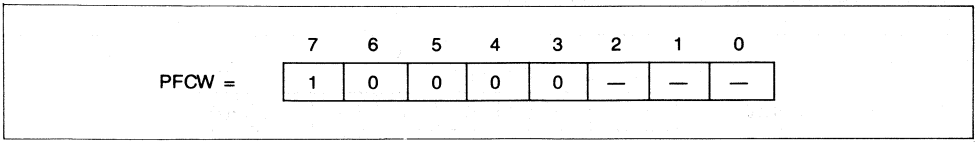


Figure 18. Self FI Rotation Set Command

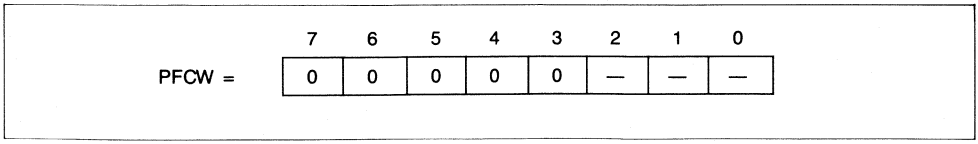


Figure 19. Self FI Rotation Clear Command

Normal FI and Rotate

The normal FI and rotate command is used to terminate interrupt processing and change interrupt priorities. Like the normal FI command, the highest priority IIS bit is reset but a change in the interrupt priority levels occurs with current interrupt level being assigned to the lowest priority level.

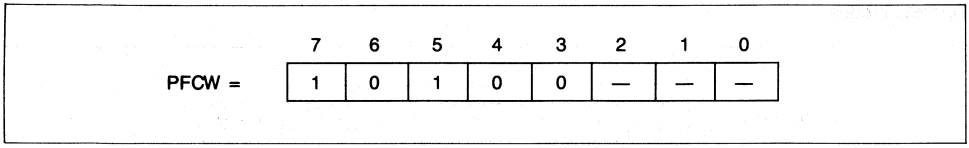


Figure 20. Normal FI and Rotate Command

Specific FI and Rotate Command

Like the specific FI command, the IIS bit specified in the IL2—IL0 field is reset but is also assigned the lowest interrupt priority level. This command can be used to automatically rotate the interrupt priorities at the end of service to allow all interrupts fais access to the CPU.

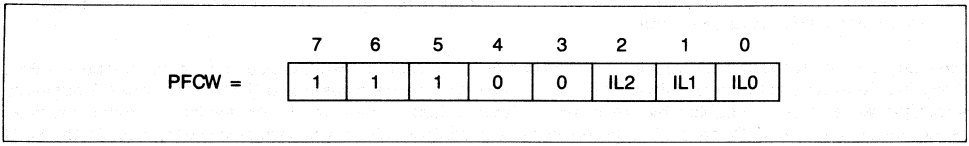


Figure 21. Specific Rotation FI Command

Rotate Command

The rotate command is used to force a change in interrupt priorities without issuing a finish interrupt command. The rotate command specifies the interrupt level to be set to the lowest priority in the IL2—IL0 field with all other interrupt levels assigned new priorities. Because this command causes a change in priorities, a specific FI command should used in place of the normal FI command.

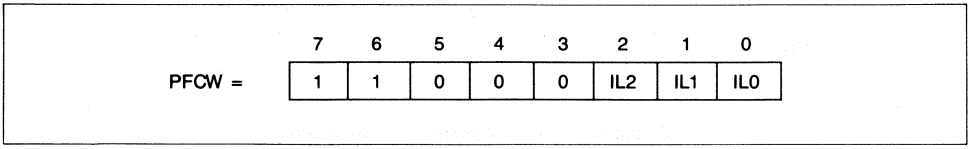
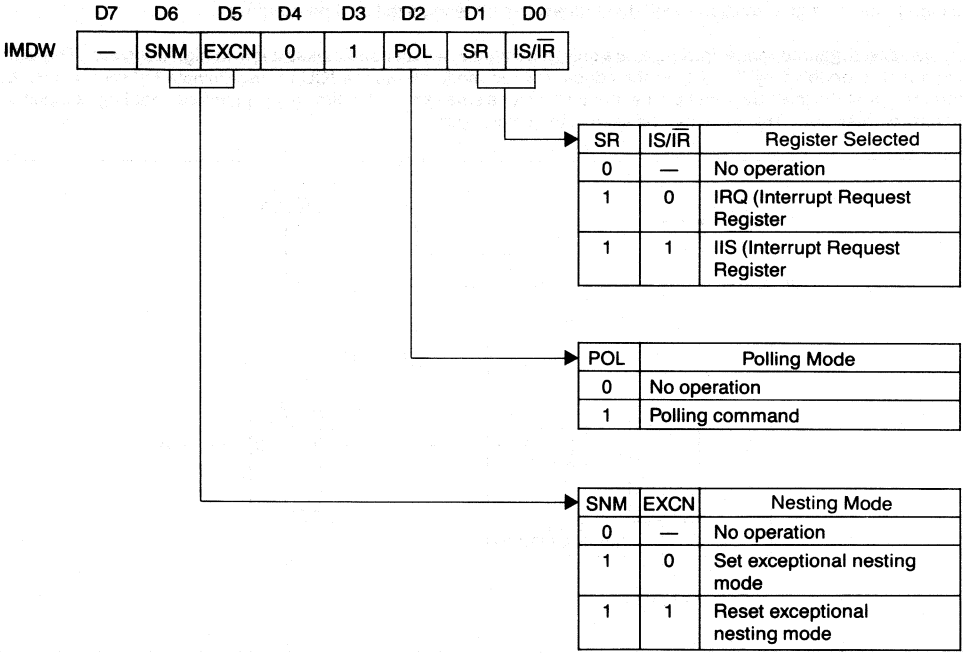


Figure 22. Rotate Command

IMDW

The IMDW (interrupt mode word) controls the operating mode of the ICU. With the IMDW, software has the ability to select the exceptional nesting mode, polling mode and the reading of the IRQ and IIS registers. The exceptional nesting mode is enabled and disabled by the SNM bit within the IMDW. If SNM is set, then the EXCN field is used to set or release the exceptional nesting mode. The SR and IS/IR fields are used to enable reading of the the IIS and and IRQ registers. If the SR bit is set, the contents of the register determined by the IS/IR field will be returned during a read. The ICU polling mode is selected by the POL field. When set, the POL field will override a read of the IRQ/IIS registers and instead return special polling data which indicates the highest priority active interrupt.



SR	IS/IR	Register Selected
0	—	No operation
1	0	IRQ (Interrupt Request Register
1	1	IIS (Interrupt Request Register

POL	Polling Mode
0	No operation
1	Polling command

SNM	EXCN	Nesting Mode
0	—	No operation
1	0	Set exceptional nesting mode
1	1	Reset exceptional nesting mode

Figure 23. IMDW Format

6. Triggering Modes

The ICU processes interrupts in either the edge-triggered or level-triggered mode as selected by the IIW1 LEV field. The selection of the triggering mode is usually dependent on the type of peripherals designed into the system as well as other factors such as a desire to minimize the static power dissipation of the μPD70208/216.

The level-triggered interrupt mode will repeatedly generate a CPU interrupt request while the following two conditions are satisfied:

- the INTn input is held high
  and
- a FI command is issued in the interrupt handler.

Thus it is a necessary requirement that an interrupt handler in the level-triggered mode clear the interrupt condition before issuing the FI command and exiting the interrupt handler (by the RETI instruction). A special note of caution about the self-FI mode with level-triggered interrupts is also required. Because an automatic FI command is issued by the ICU at the completion of the interrupt acknowledge bus cycles, it is possible for additional interrupts to occur if the CPU were to re-enable maskable interrupts before clearing the source of the interrupt.

Level-triggered interrupts enjoy an advantage over edge-triggered interrupts because they permit the automatic generation and nesting of interrupts so long as the interrupt input pin is asserted. This allows the level-triggered mode to service multiple requests from the same source or multiple sources attached to a single interrupt input. These interrupt schemes are not available in the edge-triggered mode since an interrupt input is required to make a high-to-low-to-high transition before the request can be recognized as a new interrupt.

In the edge-triggered mode, interrupts are only generated when an input makes a low-to-high transition. The extra logic that is enabled by the ICU in the edge-triggered mode allows the ICU to discriminate between events by requiring that the interrupt input pin be brought low for a small amount of time (t_{IPIPL}) before enabling recognition of further interrupts. This logic can be seen in the below figure.

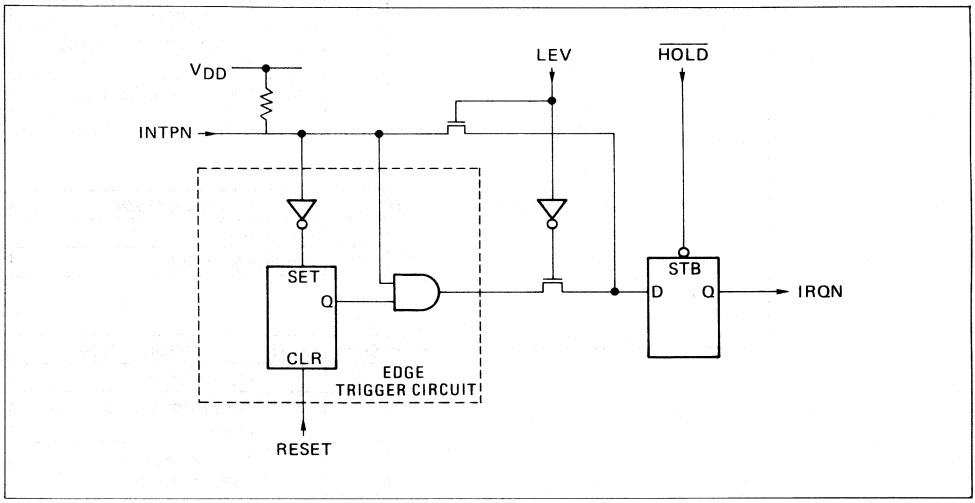


Figure 24. INTPn Input Logic

Edge-triggered interrupts are useful when the interrupt source is a short pulse rather than a continuous level. This pulse must be active-low because following the generation of the interrupt request on the rising edge of the input signal, the input must remain at a high level until the CPU interrupt acknowledge occurs. Edge-triggered interrupts are also slightly easier to use since often the interrupt source uses the edge and not level in requesting interrupt service.

Perhaps the best reason to use edge-triggered interrupts is due to the design requirements of low power systems. Because each of the INTP1—INTP7 inputs has a weak pull-up to V_{CC}, a small current (<300μA per input) is sourced by the ICU whenever an interrupt input is held at a low level. To avoid continuous sourcing of this pull-up current, the edge-triggered mode can be used to leave the interrupt inputs at the high level and having the input briefly go low then high again to generate an interrupt. Since it is the rising edge and not the level which causes an interrupt, the static power requirement of the μPD70208/216 can be reduced over that of level-triggered designs.

7. Interrupt Priorities and Nesting

The ICU has the capability to operate in a number of nesting modes using either a fixed or rotating priority schemes. These modes are set each time the ICU is initialized and can be changed dynamically to suit changing environments. This section describes how the various priority and nesting modes are used in a variety of applications.

Normal Nesting Mode

The most common interrupt nesting mode is the normal nesting mode. The normal nesting mode is selected automatically during an initialization sequence if IIW4 is not used or by the EXTN field in IIW4 being cleared. The normal nesting mode arranges the interrupt inputs in a strict priority sequence with INT0 being assigned the highest priority and INT7 being assigned the lowest priority.

While operating in the normal nesting mode, interrupts are serviced in order of priority. During an interrupt acknowledge cycle, the IIS bit corresponding to the highest priority unmasked requester (determined by the IRQ and IMK registers) is set and the interrupt vector placed on the data bus. At the same time, interrupts at the same or lower priority levels are automatically masked out until an FI command is issued. Higher priority interrupt requests are sent on to the CPU but servicing of these requests re dependent on the interrupt handler to re-enable interrupts by executing an enable interrupt instruction.

For example, if INTP3 and INTP6 are both requesting service, INTP3 will be serviced first and only after the completion of the INTP3 handler will INTP6 be acknowledged and serviced. If timer/counter 0 (INTP0) were to generate an interrupt request during the service of INTP3, the request would be sent to the CPU but would only be acknowledged if the INTP3 interrupt handler had re-enabled maskable interrupts. If that were the case, the INTP3 handler would be interrupted and timer/counter 0 serviced.

An application does not have to keep the fixed INT0—INT7 priority structure during operation. At any time a program can use the IPFW to issue a rotate command. A rotate command causes the interrupt level specified in the IL2—IL0 field to become the lowest priority interrupt level. Following the rotate command, the normal nesting mode is still in force but the order of priorities for the IIS register has changed. In the following example, a rotate priority command has been issued specifying interrupt level 3 has the lowest priority interrupt and that interrupt level 4 has now assumed the role as the highest priority interrupt.

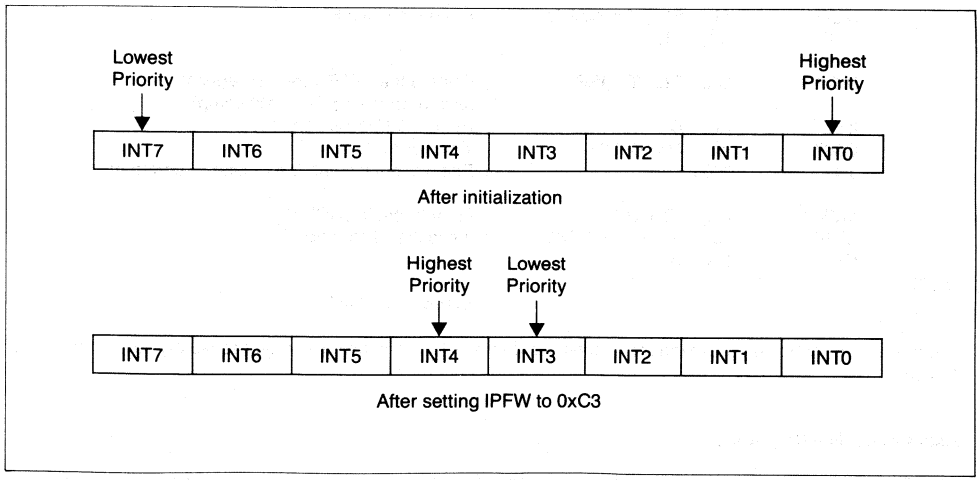


Figure 25. Rotate Priority Command

The rotate priority command can be used in an interrupt handler to change the priority but some restrictions apply. Because the normal FI command resets the highest priority IIS bit, the possibility exists for the wrong IIS bit to cleared if more than one interrupt handler was active. In this case, the correct method is to issue a specific FI command which will clear the specified IIS bit regardless of any change in priority levels. This restriction does not apply in the self-FI mode (which is selected in IIW4) since the correct IIS bit is reset automatically at the end of the interrupt acknowledge sequence and before a rotate priority command can be issued.

Extended Nesting Mode

Extended nesting is used by the ICU to maintain a strict interrupt priority ranking across the slave interrupts. When the ICU is programmed to operate in the normal mode (EXTN field in IIW4 is cleared), slave μPD71059 interrupt controllers are viewed as a single priority level by the ICU. This situation presents a potential problem because higher priority slave interrupts cannot interrupt lower priority interrupts from the same slave interrupt controller. If it is desirable to maintain the strict priority ranking through the slave controllers, the extended nesting mode must be used.

The extended nesting mode is enabled by setting the EXTN field in IIW4 of the master (ICU) interrupt controller. In a cascade mode design, FI commands must be sent to both the master and slave interrupt controllers to reset the interrupts in both sets of IIS registers. Now if the FI command was sent to both the master and slave, the possibility of a lower priority interrupt sneaking in exists. To guarantee proper operation of the extended nesting mode, the slave controller interrupt handlers must check to see if other interrupts from the same controller are being serviced. Each handler can easily determine if other interrupt handlers are active by reading the slave μPD71059 ISR (In-Service Register) and testing if any bits are still set. If the μPD71059 ISR contents are non-zero, then the handler should only issue the slave controller FI command. When the last handler completes execution, it will find the ISR cleared and it can issue the FI command to the ICU.

Example 2. Slave FI Command

```
        ...
        CLI                             ; Disable further maskable interrupts

        MOV     DX,  SLAVE IPF          ; Poit to the slave IPF register
        MOV     AL,  FI_COMMAND         ; Issue the slave FI
        OUT     DX,  AL

        MOV     DX,  SLAVE_ISR          ; Point to the μPD71059 ISR register
                                        ; (assumed to be previously setup)
        IN      AL,  DX                 ; Read the ICU IIS contents
        CMP     AL,  00000000B          ; Test if zero
        JNZ     EXIT                    ; Exit without issuing the master FI command

        MOV     DX,  ICU IPF            ; Point to the ICU IPF register
        MOV     AL,  FI_COMMAND         ; Issue the FI command
        OUT     DX,  AL
EXIT:
        ...                             ; Clean up and exit
```

Additional information on the extended nesting mode can be found in the section on cascading slave interrupt controllers.

Exceptional Nesting Mode

The ICU will not recognize any lower priority interrupts while operating in the normal or extended nesting modes. In the event that it is necessary to recognize and service lower priority interrupts during the servicing of higher priority

interrupts, the exceptional nesting mode must be used. The exceptional nesting mode is set and released on demand by the SNM nd EXCN fields in the IMDW command. By setting the mask for the current interrupt in the IMK register and selecting the exceptional nesting mode, all unmasked interrupts at priority levels above and below the current interrupt priority level will be enabled.

Issuing an normal FI command in the exceptional nesting mode requires a small amount of caution. Because the IIS bits of masked interrupts are unmodified by a normal FI command, a specific FI command for the current interrupt level must be issued unless the exceptional nesting mode is released, in which case the normal FI command may be used. An example of the exceptional nesting mode is shown in the example below.

Example 3. Exceptional Nesting Mode

```
...
MOV     DX,  IMKW       ; Point to the interrupt mask register
IN      AL,  DX         ; Read the current contents
SET1    AL,  3          ; Mask the current interrupt level
OUT     DX,  AL

MOV     DX,  IMDW       ; Point to the interrupt mode register
MOV     AL,  01100000B  ; Set exceptional nesting mode command
OUT     DX,  AL
...
...                     ; Core of the interrupt handler
...
MOV     DX,  IPFW       ; Point to the interrupt priority and finish register
MOV     AL,  01100011B  ; Specific FI for level 3
OUT     DX,  AL

MOV     DX,  IMDW       ; Point to the interrupt mode register
MOV     AL,  01000000B  ; Release exceptional nesting mode command
OUT     DX,  AL

MOV     DX,  IMKW       ; Point to the interrupt mask register
IN      AL,  DY         ; Read the current contents
CLR1    AL,  3          ; Unmask the current interrupt level
OUT     DX,  AL
...
```

8. Incomplete Interrupts

Interrupt requests from peripheral devices must be held at a high level until the completion of the first interrupt acknowledge bus cycle. The failure of an interrupt input to meet this hold requirement will cause an incomplete interrupt to occur. An incomplete interrupt is a useful event for detecting and servicing any spurious interrupts requests by a special interrupt handler.

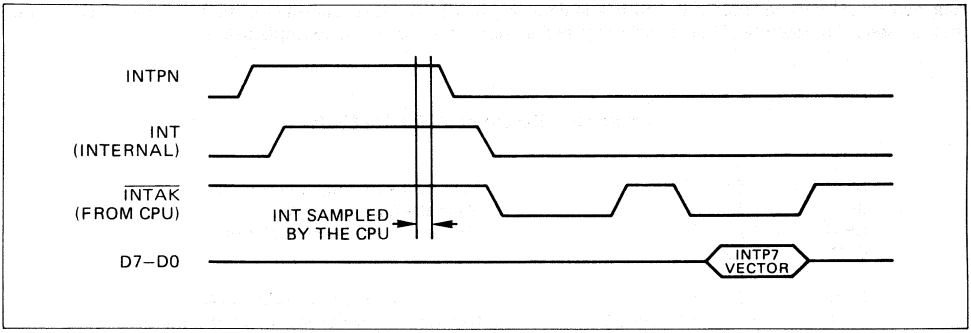


Figure 26. Incomplete Interrupts

In the event of an incomplete interrupt, the ICU responds as if an INTP7 interrupt occurred but with some minor differences. If INTP7 is not used in the sytem, the corresponding interrupt vector can be used as the vector to the incomplete interrupt handler. If the INTP7 is used as an interrupt source, then the associated handler must check the IIS register to determine the source of the interrupt. A regular INTP7 interrupts request always sets bit 7 within the IIS register. This fact can be used by the interrupt handler to distinguish between an INTP7 interrupt (IIS:7=1) or a spurious interrupt (IIS:7=0) as shown in the following example.

Example 4. Incomplete Interrupt Handler

```
...
MOV     DX,  IIS            ; Point to the in-service register
IN      AL,  DX             ; Read the current contents
TEST1   AL,  7              ; Test bit 7 of the IIS register
JZ      INCOMPLETE          ; Process the incomplete interrupt

CMP     IN_INTP7, TRUE      ; Test if in the middle of an INTP7 service
JE      INCOMPLETE          ; If true, this must be an incomplete interrupt
...
```

Since bit 7 in the IIS register is set by incomplete interrupt, the interrupt handler should not issue an FI command but should instead exit the handler with a IRET instruction. The only other possibility is if an incomplete interrupt should occur during the processing of an INTP7 interrupt. In this case, if the handler were to examine bit 7 in the IIS register it would be set and the handler would be fooled into thinking an INTP7 interrupt occurred. To avoid this potential problem, the interrupt handler should keep an additional piece of state information which allows the handler to determine if an INTP7 interrupt service is in progress.

9. Polling Operation

The ICU can also be operated in a software polling mode. To use the polling mode, the CPU should disable maskable interrupts and issue a polling command using the IMDW. The polling mode is a special mode that the ICU enters for the next ICU read command. The next read bus cycle will be treated as if it was an interrupt acknowledge bus cycle and the IIS register will be updated according to the state of the interrupt inputs and the IMK register contents.

In addition to the setting of the IIS register, the ICU will respond to a read bus cycle with the ICU address input low with the polling data. Polling data is an encoding of the current interrupt status and includes two fields as shown in figure 27. The INT field indicates whether any interrupt input is actively requesting service (INT=1) or if no interrupts are active (INT=0). If set, the low order three bits (PL2—PL0) of the polling data contain the interrupt level of the highest priority interrupt requesting service.

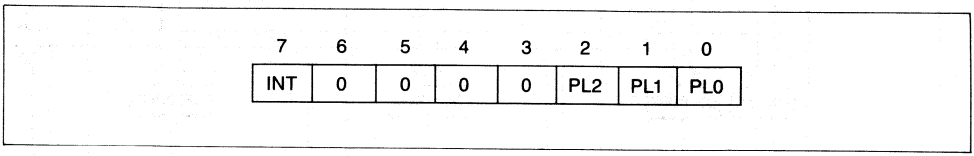


Figure 27. Polling Data

The poll command finds use in a number of applications. Because the there is no limit on the number of interrupts that can be serviced in the polling mode, it provides a method of expanding beyond the 57 direct vectored interrupts supported by the μPD70208/216. Remote μPD71059's can be used in the polling mode when a cascade mode connection is not possible due to a lack of an appropriate bus interface. Whatever the application, both vectored and the polling modes can be intermexed within an application as desired.

10. Interrupt Cascading

The section on the extended nesting mode touched on the ability of the ICU to support additional vectored interrupts in the cascade mode. Expanding the seven external interrupts of the μPD70208/216 is easily accomplished using the μPD71059 interrupt controller and a small amount of hardware. The block diagram of a typical cascade mode design is shown in figure 28. This section requires a detiled understanding of the operation of the μPD71059 interrupt controller. However, because of the similarity between the ICU and the μPD71059, all references to μPD71059 registers and commands use the same mnemonics as the ICU and the μPD71059 will be specifically mentioned.

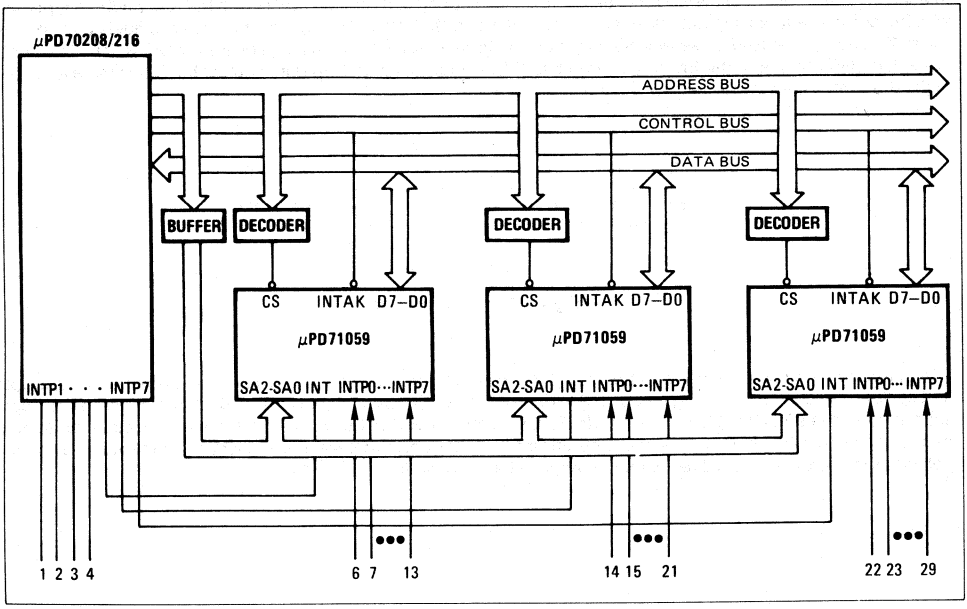


Figure 28. Cascade Mode Block Diagram

When operating in the cascade mode, the sequence of events when a peripheral device requests interrupt service is shown in figure 29. The sequence starts when a peripheral device sets an unmasked bit in a slave μPD71059 interrupt request register. This causes the μPD71059 INT output to be asserted which sets a bit in the ICU IRQ register. If there are no other higher priority requests, the CPU interrupt input (which is an internal μPD70208/216 signal) will be asserted by the ICU. When the CPU responds by running the first interrupt acknowledge bus cycle, the ICU as the master interrupt controller will output a slave address corresponding to the interrupt input being acknowledged on the address lines for all slave μPD71059 controllers to compare. Each of the slave μPD71059 interrupt controllers compare this address with the slave address written during their IIW3 initialization word. The slave controller with the matching address will then place the interrupt vector on the data bus during the second interrupt acknowledge bus cycle. Both the slave controller and the ICU will set the corresponding bits within the IIS registers for the interrupt levels being serviced.

The remainder of the interrupt processing takes place under the control of the interrupt handler. At the end of the interrupt handler, an FI command is issued to the slave μPD71059 to reset the bit in the IIS register for the interrupt completing service. Next, the handler will issue an FI command for the master (ICU) which resets the slave in-service bit in the ICU IIS register. Finally the IRET instruction is executed and control returns to the point of interruption. This process is described in detail in the section on the extended nesting mode or in the source code listings in the appendix.

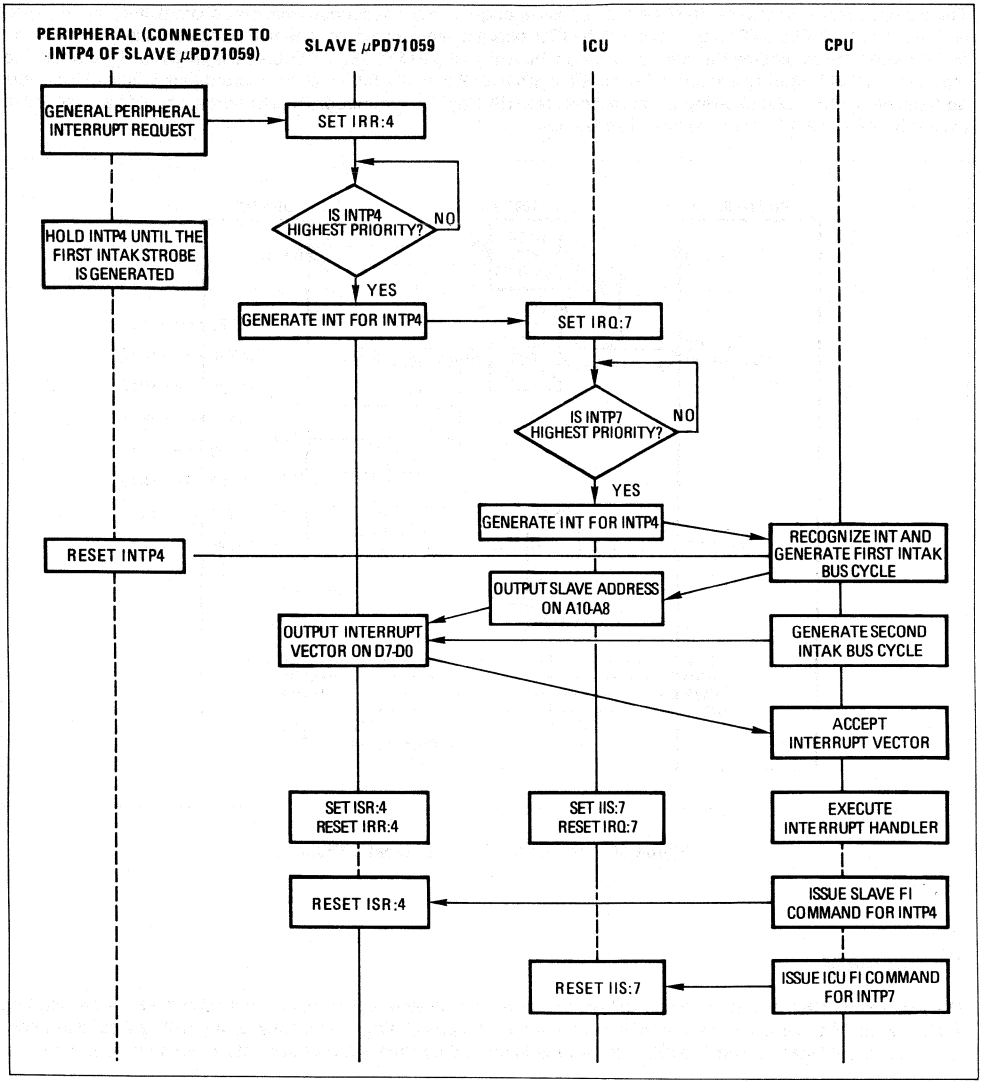


Figure 29. Slave Interrupt Processing Flow

For the hardware design of a cascade mode system, a couple of additional requirements are introduced. First, multiple slave interrupt controllers must be able to uniquely determine which controller is to respond during the first interrupt acknowledge bus cycle. Secondly, interrupt requests from slave controllers must be routed through the master so they can passed on to the CPU.

The cascade mode hardware interface seems easy enough. The second requirement is accomplished by simply attaching the μPD71059 INT output to one of the ICU external interrupt inputs. The ICU cooperates to help meet the first requirement by placing the slave address on the A10—A8 (μPD70208) or AD10—AD8 (μPD70216) outputs. The small amount of buffering is needed because the μPD71059 and μPD8259A don't guarantee the state of the slave address pins SA2—SA0 to be inputs, even if the slave (SV) input is grounded. Thus buffering should be provided to isolate these signals from the system address bus.

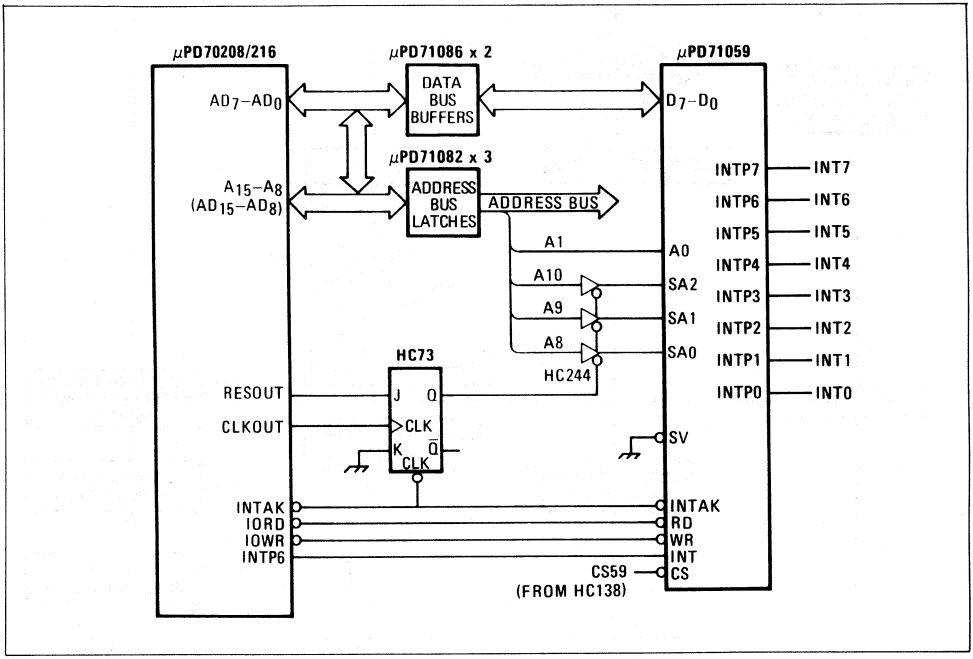


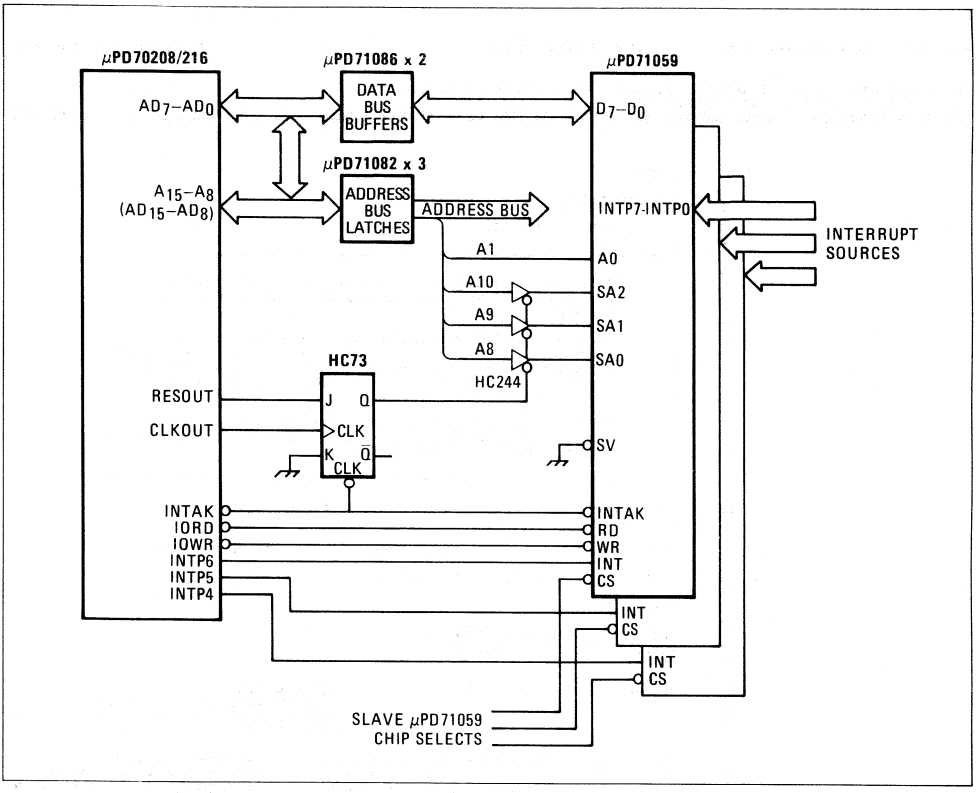
Figure 30. Cascade Mode Schematic Diagram

11. Summary

The μPD70208/216 ICU has been shown to provide the flexibility and expansibility required in the most demanding applications. The combination of hardware support for the processing of interrupts along with the software configuration capabilities, makes the ICU a powerful solution to the processing of interrupts in real-time environments.

Appendix A: ICU Cascade Mode Schematic

The addition of one or more slave μPD71059 interrupt controllers to a μPD70208/216 system is a simple matter. As shown below, one need only buffer the slave address from the interrupt controllers and attach each μPD71059 INT output to one of the μPD70208/216 INTP inputs. Since the μPD70208/216 generates the interrupt acknowledge output and the I/O read/write strobes, a sytem with up to 57 vectored interrupt sources is easily constructed.



Operation of the buffering logic is as follows. The buffer enable flip-flop is set during the μPD70208/216 reset state. This forces the Q output to the high state which disables the slave address buffers. Once the slave μPD71059 has been initialized and the SA2—SA0 pins placed in the input state, the slave address buffers can be enabled. Enabling of these buffers can be accomplished by software and a programmable latch or as shown in this example, the first interrupt acknowledge bus cycle can be used to reset the flip-flop and enabling the buffers.

Appendix B: ICU Ports Definitions

This assembly language program allows the addresses of the ICU registers to be defined separately from the source files that reference them. The use of this method delays the binding of I/O port addresses until the linking stage, adding additional flexibility in the selection and relocation of the internal peripherals.

```
8086/87/88/186 MACRO ASSEMBLER    ICU I/O PORT DEFINITIONS                    07:19:39  12/22/86  PAGE 1

IBM PC-DOS 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE ICUPORTS
OBJECT MODULE PLACED IN OBJ\ICUPORTS.OBJ
ASSEMBLER INVOKED BY:  ASM86 ICUPORTS.ASM EP(ICUPORTS.ERR) OJ(OBJ\ICUPORTS.OBJ) PR(LIST\ICUPORTS.LST) PW(132) M1 NOGE


LINE     SOURCE

   1     $TITLE(ICU I/O Port Definitions)
   2
   3             NAME    ICUPORTS
   4
   5     ;-------------------------------------------------------------------------
   6     ;                                                                         :
   7     ;       This file contains the I/O port definitions for the ICU           :
   8     ;       and should be linked with any module requiring to access          :
   9     ;       the ICU.                                                          :
  10     ;                                                                         :
  11     ;       Note that the I/O port addressing changes depending on the        :
  12     ;       processor in the system.  V40 systems have internal peripheral    :
  13     ;       I/O ports that are contiguous while V50 systems alternate         :
  14     ;       addresses, forcing the I/O ports to always to on the upper        :
  15     ;       or lower byte of the data bus.                                    :
  16     ;                                                                         :
  17     ;       The symbol ICU_BASE is used to locate the base address of the     :
  18     ;       ICU and must be used by the IPR initialization software to        :
  19     ;       correctly setup the IULA register.                                :
  20     ;                                                                         :
  21     ;-------------------------------------------------------------------------
  22
  23     $*DEFINE (V40) (1)        $' Address scaling for V40 systems
         $*DEFINE (V50) (2)        $' Address scaling for V50 systems

  24     $SET(SCALE, $V50)         $' Change this to V40 or V50 depending on the target

  25     IO_PORTS        SEGMENT WORD    PUBLIC
  26
  27     PUBLIC  ICU_BASE
  28
  29     PUBLIC  IRQ, IIS, IMKW, IPFW, IMDW
  30     PUBLIC  IIW1, IIW2, IIW3, IIW4
  31
  32
  33     ICU_BASE        EQU     20H             ; ICU Base Address
  34                                             ; Make ICU_BASE even if the ICU is to be
  35                                             ; located on D7-D0 or odd if the ICU is to
  36                                             ; be on D15-D8
  37
  38     IRQ     EQU     ICU_BASE + $EVAL(0 * $SCALE)    ; Interrupt Request Register
  39     IIS     EQU     ICU_BASE + $EVAL(0 * $SCALE)    ; Interrupt In-Service Register
  40     IPFW    EQU     ICU_BASE + $EVAL(0 * $SCALE)    ; Interrupt Priority and Finish Word
  41     IMDW    EQU     ICU_BASE + $EVAL(0 * $SCALE)    ; Interrupt Mode Word
  42     IMKW    EQU     ICU_BASE + $EVAL(1 * $SCALE)    ; Interrupt Mask Register
  43
  44     IIW1    EQU     ICU_BASE + $EVAL(0 * $SCALE)    ; Interrupt Initialize Word 1
  45     IIW2    EQU     ICU_BASE + $EVAL(1 * $SCALE)    ; Interrupt Initialize Word 2
  46     IIW3    EQU     ICU_BASE + $EVAL(1 * $SCALE)    ; Interrupt Initialize Word 3
  47     IIW4    EQU     ICU_BASE + $EVAL(1 * $SCALE)    ; Interrupt Initialize Word 4
  48
  49     IO_PORTS        ENDS
  50
  51             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Appendix C: ICU and Slave μPD71059 Initialization

This first set of files is routines to initialize the ICU and the slave μPD71059 interrupt controller.

```
8086/87/88/186 MACRO ASSEMBLER    V40/V50 ICU INITIALIZATION MODULE                    09:10:48  11/26/86  PAGE 1

IBM PC-DOS 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE ICUINIT
OBJECT MODULE PLACED IN OBJ\ICUINIT.OBJ
ASSEMBLER INVOKED BY:  ASM86 ICUINIT.ASM EP(ICUINIT.ERR) OJ(OBJ\ICUINIT.OBJ) PR(LIST\ICUINIT.LST) PW(132) M1


                LINE    SOURCE

                   1 +1 $TITLE(V40/V50 ICU Initialization Module)
                   2
                   3            NAME    ICUINIT
                   4
                   5    ;*****************************************************************************
                   6    ;                                                                            :
                   7    ; Module Name                                                               :
                   8    ;                                                                           :
                   9    ;       ICU Initialization                                                  :
                  10    ;                                                                           :
                  11    ;                                                                           :
                  12    ; Module Description                                                        :
                  13    ;                                                                           :
                  14    ;       This module is responsible for the initialization of the            :
                  15    ;       V40/V50 Interrupt Control Unit.  In this example, the ICU           :
                  16    ;       is setup to interface to a slave uPD71059 for a total of            :
                  17    ;       15 interrupts (7 internal, 8 external).  Because it is              :
                  18    ;       desired to preserve the interrupt priority structure, the           :
                  19    ;       extended nesting mode (in IIW4) is selected.                        :
                  20    ;                                                                           :
                  21    ; Inputs                                                                    :
                  22    ;                                                                           :
                  23    ;       None                                                                :
                  24    ;                                                                           :
                  25    ; Outputs                                                                   :
                  26    ;                                                                           :
                  27    ;       None                                                                :
                  28    ;                                                                           :
                  29    ; Registers Modified                                                        :
                  30    ;                                                                           :
                  31    ;       None                                                                :
                  32    ;                                                                           :
                  33    ;*****************************************************************************
                  34 +1 $EJECT

                        V40/V50 ICU INITIALIZATION MODULE                    09:10:48  11/26/86  PAGE 2

                LINE    SOURCE

                  35 +1 $INCLUDE(icu.defs)
=1                36 +1 $NOLIST
                  83
                  84    IIW1_INIT       EQU     EXTENDED_MODE + EDGE_TRIGGER + SELECT_IIW4
                  85    IIW2_INIT       EQU     80H
                  86    IIW3_INIT       EQU     INT7
                  87    IIW4_INIT       EQU     FI_MODE + EXTEND_NESTING
                  88    IMKW_MASK       EQU     NOT (INT0 + INT7)
                  89
                  90    IO_PORTS        SEGMENT WORD    PUBLIC
                  91
                  92    EXTRN   IIW1 : ABS, IIW2 : ABS, IIW3 : ABS, IIW4 : ABS, IMKW : ABS
                  93
                  94    IO_PORTS        ENDS
                  95
                  96
                  97    CODE    SEGMENT WORD    PUBLIC
                  98
                  99    PUBLIC  ICU_INIT
                 100    EXTRN   INIT_71059 : NEAR
                 101
                 102            ASSUME  CS:CODE
                 103
                 104    ICU_INIT        PROC    NEAR
                 105
```

```
                106             PUSH    AX                      ; Save the working registers
                107             PUSH    DX
                108
            E   109             MOV     DX, IIW1                ; Point to IIW1
                110             MOV     AL, IIW1_INIT           ; Get the initialization word
                111             OUT     DX, AL                  ; IIW1 is programmed to select edge triggering
                112                                             ; and extended mode
                113
            E   114             MOV     DX, IIW2                ; Point to IIW2
                115             MOV     AL, IIW2_INIT           ; Get the initialization word
                116             OUT     DX, AL                  ; IIW2 contains the vector base for the
                117                                             ; the internal interrupts (80H to 86H)
                118
            E   119             MOV     DX, IIW3                ; Point to IIW3
                120             MOV     AL, IIW3_INIT           ; Get the initialization word
                121             OUT     DX, AL                  ; IIW3 selects the slave controllers which in
                122                                             ; example is attached to INTP7
                123
            E   124             MOV     DX, IIW4                ; Point to IIW4
                125             MOV     AL, IIW4_INIT           ; Get the initialization word
                126             OUT     DX, AL                  ; This example uses extended nesting and each
                127                                             ; handler issues an FI command
                128
            E   129             MOV     DX, IMKW                ; Point to mask register
                130             MOV     AL, IMKW_MASK           ; Enable two interrupts and mask off the others
                131             OUT     DX, AL                  ; Initialize the IMKW register
                132
            E   133             CALL    INIT_71059              ; Initialize the slave uPD71059 controller
                134
                135             POP     DX                      ; Restore the registers
                136             POP     AX
                137
                138             RET                             ; Return
                139
                140     ICU_INIT        ENDP
                141
                142     CODE    ENDS
                143
                144             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Note that in the above code, the addresses of IIW3 and IIW4 are redundant because IIW2, IIW3 and IIW4 all share the same port address. They are shown in the example strictly for clarity and may be omitted if desired.

```
8086/87/88/186 MACRO ASSEMBLER    CASCADE MODE UPD71059 INITIALIZATION                    12:32:39  11/24/86  PAGE 1

IBM PC-DOS 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE INIT59
OBJECT MODULE PLACED IN OBJ\INIT59.OBJ
ASSEMBLER INVOKED BY:  ASM86 INIT59.ASM EP(INIT59.ERR) OJ(OBJ\INIT59.OBJ) PR(LIST\INIT59.LST) PW(132) M1


                LINE    SOURCE

                   1 +1 $TITLE(Cascade Mode uPD71059 Initialization)
                   2
                   3           NAME    INIT59
                   4
                   5    ;*****************************************************************************
                   6    ;                                                                            :
                   7    ; Module Name                                                               :
                   8    ;                                                                           :
                   9    ;       V40/V50 Cascade Mode uPD71059 Initialization                        :
                  10    ;                                                                           :
                  11    ; Module Description                                                        :
                  12    ;                                                                           :
                  13    ;       This module is responsible for the initialization of the            :
                  14    ;       slave (cascade) mode uPD71059 used in the example.                  :
                  15    ;                                                                           :
                  16    ; Inputs                                                                    :
                  17    ;                                                                           :
                  18    ;       None                                                                :
                  19    ;                                                                           :
                  20    ; Outputs                                                                   :
                  21    ;                                                                           :
                  22    ;       None                                                                :
                  23    ;                                                                           :
                  24    ; Registers Modified                                                        :
                  25    ;                                                                           :
                  26    ;       None                                                                :
                  27    ;                                                                           :
                  28    ;*****************************************************************************
                  29 +1 $EJECT

                CASCADE MODE UPD71059 INITIALIZATION                    12:32:39  11/24/86  PAGE 2

                LINE    SOURCE

                  30 +1 $INCLUDE(uPD71059.def)
=1                31 +1 $NOLIST
                  73
                  74    IW1_INIT        EQU     EXTENDED_MODE + EDGE_TRIGGER + SELECT_IW4
                  75    IW2_INIT        EQU     88H
                  76    IW3_INIT        EQU     07H
                  77    IW4_INIT        EQU     FI_MODE + NORMAL_NESTING
                  78    IMW_MASK        EQU     NOT INTP0
                  79
                  80    IO_PORTS        SEGMENT WORD    PUBLIC
                  81
                  82    EXTRN   IW1 : ABS, IW2 : ABS, IW3 : ABS, IW4 : ABS
                  83    EXTRN   IMW : ABS
                  84
                  85    IO_PORTS        ENDS
                  86
                  87
                  88    CODE    SEGMENT WORD    PUBLIC
                  89
                  90    PUBLIC  INIT_71059
                  91
                  92            ASSUME  CS:CODE
                  93
                  94    INIT_71059      PROC    NEAR
                  95
                  96            PUSH    AX                      ; output to IW3
                  97            PUSH    DX
                  98                                            ; Access via boundary address to access 8259
E                 99            MOV     DX, IW1                 ; IW1 selects the page to the slave controller
                 100            MOV     AL, IW1_INIT            ; Get the initialization word
                 101            OUT     DX, AL                  ; output to IW1
                 102
                 103                                            ; Determines that this is a slave
E                104            MOV     DX, IW2                 ; Extended mode is selected and the V5 bit
                 105            MOV     AL, IW2_INIT            ; Get the initialization word
                 106            OUT     DX, AL                  ; output to IW2
                 107
                 108
E                109            MOV     DX, IW3                 ; Save the working register
```

```
                110             MOV     AL, IW3_INIT            ; Get the initialization word
                111             OUT     DX, AL                  ; Set the slave address to show that this
                112                                             ; uPD71059 is used in the slave mode
                113
              E 114             MOV     DX, IW4                 ; Point to IW4
                115             MOV     AL, IW4_INIT            ; Get the initialization word
                116             OUT     DX, AL                  ; Like the master, normal nesting is selected
                117
              E 118             MOV     DX, IMW                 ; Point to the mask register
                119             MOV     AL, IMW_MASK            ; Select the interrupts to be enabled
                120             OUT     DX, AL                  ; Unmask the interrupts
                121
                122             POP     DX                      ; Restore the registers
                123             POP     AX
                124
                125             RET                             ; Return
                126
                127     INIT_71059      ENDP
                128
                129     CODE    ENDS
                130
                131     END
```

ASSEMBLY COMPLETE, NO ERRORS FOUND

Appendix D: ICU and Slave Interrupt Handlers

This appendix contains shells of interrupt handlers for both master and slave interrupts.

```
8086/87/88/186 MACRO ASSEMBLER    V40/V50 ICU MASTER INTERRUPT HANDLER                    14:48:28  11/24/86  PAGE 1

IBM PC-DOS 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE MASTER
OBJECT MODULE PLACED IN OBJ\MASTER.OBJ
ASSEMBLER INVOKED BY:  ASM86 MASTER.ASM EP(MASTER.ERR) OJ(OBJ\MASTER.OBJ) PR(LIST\MASTER.LST) PW(132) M1


                    LINE     SOURCE

                       1 +1  $TITLE(V40/V50 ICU Master Interrupt Handler)
                       2
                       3             NAME    MASTER
                       4
                       5     ;*************************************************************************
                       6     ;                                                                        :
                       7     ; Module Name                                                            :
                       8     ;                                                                        :
                       9     ;       Master Mode Interrupt Handler                                    :
                      10     ;                                                                        :
                      11     ;                                                                        :
                      12     ; Module Description                                                     :
                      13     ;                                                                        :
                      14     ;       This module serves as the master mode interrupt handler for      :
                      15     ;       the TCU timer/counter 0 interrupt.  Since this is a master       :
                      16     ;       interrupt handler, no special handling other re-enabling         :
                      17     ;       maskable interrupts to allow higher priority interrupts to       :
                      18     ;       be recognized and serviced.                                      :
                      19     ;                                                                        :
                      20     ; Inputs                                                                 :
                      21     ;                                                                        :
                      22     ;       None                                                             :
                      23     ;                                                                        :
                      24     ; Outputs                                                                :
                      25     ;                                                                        :
                      26     ;       None                                                             :
                      27     ;                                                                        :
                      28     ; Registers Modified                                                     :
                      29     ;                                                                        :
                      30     ;       None                                                             :
                      31     ;                                                                        :
                      32     ;*************************************************************************
                      33 +1  $EJECT
8086/87/88/186 MACRO ASSEMBLER    V40/V50 ICU MASTER INTERRUPT HANDLER                    14:48:28  11/24/86  PAGE 2

                    LINE     SOURCE

                      34 +1  $INCLUDE(icu.defs)
=1                    35 +1  $NOLIST
                      82
                      83     IO_PORTS        SEGMENT WORD    PUBLIC
                      84
                      85     EXTRN   IPFW : ABS
                      86
                      87     IO_PORTS        ENDS
                      88
                      89
                      90     CODE    SEGMENT WORD    PUBLIC
                      91
                      92     PUBLIC  ICU_HANDLER
                      93
                      94             ASSUME  CS:CODE
                      95
                      96     ICU_HANDLER     PROC    FAR
                      97
                      98             PUSHA                           ; Save the working registers
                      99
                     100     ;
                     101     ;       Save any additional state information (such as segment registers) here...
                     102     ;
                     103
                     104             STI                             ; Re-enable interrupts if it is desired
                     105                                             ; to service higher priority interrupts
```

```
                        106
                        107     ;
                        108     ;       The body of the interrupt handler will normally reside here ...
                        109     ;
                        110
                        111             CLI                                     ; Disable interrupts and finish up
                        112
                    E   113             MOV     DX, IPFW                        ; Point to the IPFW register
                        114             MOV     AL, FI_COMMAND                  ; Get the FI command
                        115             OUT     DX, AL                          ; Issue the FI command to the ICU
                        116
                        117             POPA                                    ; Restore the previous state
                        118
                        119             IRET                                    ; Return
                        120
                        121     ICU_HANDLER     ENDP
                        122
                        123     CODE    ENDS
                        124
                        125             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
8086/87/88/186 MACRO ASSEMBLER    V40/V50 ICU SLAVE INTERRUPT HANDLER                14:46:47  11/24/86  PAGE 1

IBM PC-DOS 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE SLAVE
OBJECT MODULE PLACED IN OBJ\SLAVE.OBJ
ASSEMBLER INVOKED BY:  ASM86 SLAVE.ASM EP(SLAVE.ERR) OJ(OBJ\SLAVE.OBJ) PR(LIST\SLAVE.LST) PW(132) M1


                        LINE     SOURCE

                           1 +1  $TITLE(V40/V50 ICU Slave Interrupt Handler)
                           2
                           3             NAME    SLAVE
                           4
                           5     ;*************************************************************************
                           6     ;                                                                        :
                           7     ; Module Name                                                            :
                           8     ;                                                                        :
                           9     ;       Slave Mode Interrupt Handler                                     :
                          10     ;                                                                        :
                          11     ;                                                                        :
                          12     ; Module Description                                                     :
                          13     ;                                                                        :
                          14     ;       This module serves as the slave mode interrupt handler for       :
                          15     ;       the uPD71054 timer/counter 2 interrupt.                          :
                          16     ;                                                                        :
                          17     ; Inputs                                                                 :
                          18     ;                                                                        :
                          19     ;       None                                                             :
                          20     ;                                                                        :
                          21     ; Outputs                                                                :
                          22     ;                                                                        :
                          23     ;       None                                                             :
                          24     ;                                                                        :
                          25     ; Registers Modified                                                     :
                          26     ;                                                                        :
                          27     ;       None                                                             :
                          28     ;                                                                        :
                          29     ;*************************************************************************
                          30 +1  $EJECT
                                 V40/V50 ICU SLAVE INTERRUPT HANDLER                14:46:47  11/24/86  PAGE 2

                        LINE     SOURCE

                          31 +1  $INCLUDE(upd71059.defs)
                    =1    32 +1  $NOLIST
                          78
                          79     IO_PORTS        SEGMENT WORD    PUBLIC
                          80
                          81     EXTRN   IPFW : ABS, PFCW : ABS, MCW : ABS
                          82
                          83     IO_PORTS        ENDS
                          84
                          85
                          86     CODE    SEGMENT WORD    PUBLIC
                          87
                          88     PUBLIC  SLAVE_HANDLER
                          89
                          90             ASSUME  CS:CODE
                          91
                          92     SLAVE_HANDLER   PROC    FAR
                          93
                          94             PUSHA                           ; Save the working registers
                          95
                          96     ;
                          97     ;       Save any additional state (such as the segment registers) here...
                          98     ;
                          99
                         100             STI                             ; Allow responding to other interrupts
                         101
                         102     ;
                         103     ;       The body of the interrupt handler will normally reside here ...
                         104     ;
                         105
                         106             CLI                             ; Disable further interrupts
                         107
                    E    108             MOV     DX, PFCW                ; Point to the uPD71059 PFCW
                         109             MOV     AL, FI_COMMAND          ; Get the FI command
                         110             OUT     DX, AL                  ; Issue the FI command for the slave (uPD71059
                         111
                    E    112             MOV     DX, MCW                 ; Point to the Mode Control Word
                         113             MOV     AL, ISR_SELECT          ; Specify a read ISR command
```

```
                    114                 OUT     DX, AL                  ; Do it
                    115
                    116                 IN      AL, DX                  ; Read the ISR register
                    117                 CMP     AL, 00000000B           ; Test for other active interrupts
                    118                 JNZ     EXIT                    ; Exit others are active without issuing a
                    119                                                 ; master FI
                    120
                    121
                E   122                 MOV     DX, IPFW                ; Point to the ICU IPFW register
                    123                 MOV     AL, FI_COMMAND          ; Get the FI command
                    124                 OUT     DX, AL                  ; Issue the FI command for the master (ICU)
                    125
                    126         EXIT:
                    127                 POPA                            ; Restore the state prior to the interrupt
                    128
                    129                 IRET                            ; Return
                    130
                    131         SLAVE_HANDLER   ENDP
                    132
                    133         CODE    ENDS
                    134
                    135                 END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Appendix E: Assembly Language Include Files

These include files are provided to allow symbolic definition and generation of initialization and control words for the μPD70208/216 ICU and the μPD71059. Modules desiring to use these include files need only place an include statement in the assembly language source file.

```
$NOLIST

;-------------------------------------------------------------------------
;                                                                        :
;       ICU Register and Bit Definitions                                 :
;                                                                        :
;       This include file contains the definitions for accessing and     :
;       manipulating the ICU registers.  Commands may be formed by       :
;       ORing or adding the selected modes together.                     :
;                                                                        :
;       Note that either one of the following equates must be specified  :
;       for the following registers to be properly initialized:          :
;                                                                        :
;               IIW1 - EDGE_TRIGGER/LEVEL-TRIGGER                        :
;               IIW4 - NORMAL_NESTING/EXTEND_NESTING                     :
;                                                                        :
;-------------------------------------------------------------------------

EDGE_TRIGGER    EQU     10H             ; Select Edge Triggered Interrupts
LEVEL_TRIGGER   EQU     18H             ; Select Level Triggered Interrupts

EXTENDED_MODE   EQU     00H             ; Select Extended Mode
SINGLE_MODE     EQU     02H             ; Select Single Mde

SELECT_IIW4     EQU     01H             ; Select IIW4

NORMAL_NESTING  EQU     01H             ; Select Normal Nesting Mode
EXTEND_NESTING  EQU     11H             ; Select Extended Nesting Mode

FI_MODE         EQU     00000000B       ; FI Command Mode
SELF_FI_MODE    EQU     00000010B       ; Self FI Mode

FI_COMMAND      EQU     20H             ; Normal Finish Interrupt Command

IRQ_SELECT      EQU     0AH             ; Select IRQ register for reading
IIS_SELECT      EQU     0BH             ; Select IIS register for reading
POLLING_COMMAND EQU     0CH             ; Polling command

INT0            EQU     01H             ; Interrupt 0
INT1            EQU     02H             ; Interrupt 1
INT2            EQU     04H             ; Interrupt 2
INT3            EQU     08H             ; Interrupt 3
INT4            EQU     10H             ; Interrupt 4
INT5            EQU     20H             ; Interrupt 5
INT6            EQU     40H             ; Interrupt 6
INT7            EQU     80H             ; Interrupt 7

$LIST
```

This file contains the definitions for the bit fields within the μPD71059 interrupt controller. While the registers are compatible, the mnemonics are changed to conform to the standards of the data sheet for this device.

```
$NOLIST

;-------------------------------------------------------------------------
;                                                                        :
;       uPD71059 Register and Bit Definitions                            :
;                                                                        :
;       This include file contains the definitions for accessing and     :
;       manipulating the uPD71059 registers.  Commands may be formed by  :
;       ORing or adding the selected modes together.                     :
;                                                                        :
;       Note that either the edge or level sensitive mode must be        :
;       specified use this set of definitions.  Also note that           :
;       no provision is made for CALL (8080/8085) mode applications.     :
;                                                                        :
;-------------------------------------------------------------------------

EDGE_TRIGGER    EQU     10H             ; Select Edge Triggered Interrupts
LEVEL_TRIGGER   EQU     18H             ; Select Level Triggered Interrupts

EXTENDED_MODE   EQU     00H             ; Select Extended Mode
SINGLE_MODE     EQU     02H             ; Select Single Mde

SELECT_IW4      EQU     01H             ; Select IIW4

NORMAL_NESTING  EQU     01H             ; Select Normal Nesting Mode
EXTEND_NESTING  EQU     11H             ; Select Extended Nesting Mode

FI_MODE         EQU     00H             ; Finish Interrupt Command Mode
SELF_FI_MODE    EQU     02H             ; Self Finish Interrupt Mode

FI_COMMAND      EQU     20H             ; Normal Finish Interrupt Command

ISR_SELECT      EQU     0AH             ; Select ISR register for reading
IRR_SELECT      EQU     0BH             ; Select IRR register for reading
POLLING_COMMAND EQU     0CH             ; Polling command

INTP0           EQU     01H             ; Interrupt 0
INTP1           EQU     02H             ; Interrupt 1
INTP2           EQU     04H             ; Interrupt 2
INTP3           EQU     08H             ; Interrupt 3
INTP4           EQU     10H             ; Interrupt 4
INTP5           EQU     20H             ; Interrupt 5
INTP6           EQU     40H             ; Interrupt 6
INTP7           EQU     80H             ; Interrupt 7

$LIST
```

MASM Code Macros for V-Series Processors V20, V30, V40, V50, V25

Contents:

Author: R. Naro
NEC Electronics NATICK/USA

Persons to contact: S. Gupta, B. Peters
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

V20, V30, V40, V50 User's Manual
V25, V35 User's Manual
Date book microprocessors and peripherals

Related Products

V20	16 bit microprocessor	CMOS
V30	16 bit microprocessor	CMOS
V40	16 bit microprocessor	CMOS
V50	16 bit microprocessor	CMOS
V25	16 bit microcomputer	CMOS
V35	16 bit microcomputer	CMOS

1. Introduction

The ability to use standard PC-based software development tools like the Microsoft® Assembler (MASM) is one of the most significant opportunities available to software designer's using one of the V-Series microprocessor family (V20—V50 and V25). Yet, in the past, the opportunity to use IBM PC-based software development tools has been limited by a number of constraints which either made their use impractical or often even impossible. This first MicroNote in a software development series takes the first step in the right direction for the leveraging of PC-based tools in the embedded system marketplace.

The traditional problems in the past have been:

- inhability to use the complete V-Series instruction set
- inhability to use high level languages like C, Pascal or Forth
- inhability to locate the resulting code
- inhability to test and debug the application software on an in-circuit emulator
- Inhability to place the finish code in an EPROM

This new series of software development MicroNotes will present in-house solutions to some of the problems as well as explore existing third party solutions. The goal of the software development MicroNote series is to demonstrate how through the proper application of existing tools, a dramatic reduction in application software development time can be achieved.

Since there are many software issues that need to be addressed, this MicroNote focuses on the lowest common denominator in an embedded application, the assembler. In any embedded application, an assembler is usually required to either work in the stand-alone mode if the application is modest or together with a high level language such as C in a more ambitious undertaking.

In the world of MS-DOS, the first assembler that comes to mind is the ubiquitous Microsoft assembler. The Microsoft assembler (MASM), is a powerful tool for the development of assembly language modules for the 8086/186/286 microprocessors as well as the 8087/287 numeric coprocessors. While this is great, support for the powerful new V-Series instructions is lacking. Since it is our goal to have complete support for the V-Series using MASM, something must be done to modify the behavior of MASM to accept and correctly assemble the new instructions. This Micro-Note presents such a solution using a set of MASM code macros.

Benefits

Why use the Microsoft assembler when NEC and other vendors offer similar products. The primary benefits of using MASM are:

- full macro assembler with conditional assembly directives
- generate standard MS-DOS object files
- includes linker, librarian and symbolic debugger and other utilities
- NEC in-circuit emulator compatible (outputs symbols for debugging)
- low cost

A final note before getting technical. While the focus of this MicroNote is on the Microsoft assembler, these code macros can be adapted to virtually any macro assembler supporting the 80186 instruction set..

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

IBM is a registered trademark of International Business Machines Corporation.

V20™, V30™, V40™, V50™ and V25™ are registered trademarks of NEC Electronics Inc.

MASM

MASM, the acronym for Microsoft Assembler, is the most popular assembler currently used in the PC environment. However, for all it's popularity with the masses, MASM has not been able to penetrate the embedded system marketplace due to a perceived lack of support for ROMable code. Note that this is not the fault of MASM for it certainly does support ROMable code but rather it is a lack of tools and utilities that support the physical address location and linking of object file produced by MASM. While this MicroNote shows how to use MASM with the V-Series microprocessors, it won't be till a future MicroNote when the solutions for location and in-circuit emulator will be presented.

To enable MASM to recognize and assemble any of the V-Series instructions requires modifications to the assembler. Since this is usually impractical and likely impossible, the next best step is to use the conditional assembly and macro capabilities of MASM to convert the NEC instruction mnemonics to their equivalent machine code. In the appendices of this MicroNote are four MASM include file listings which contain the set of V-Series instruction code macros for the V20—V50 family and the V25 microprocessor. Used in conjunction with the MASM .186 switch, these code macros enable MASM to recognize the entire V20—V50 or V25 instruction sets.

What's a code macro?

A code macro is nothing more than an assembler macro used to define a new instruction to a macro assembler. Unlike a regular macro which is typically used to generate a sequence of instructions, the code macro is used to generate an entirely new instruction. What makes code macros difficult to for an assembler like MASM is that macros were not meant to add complex instructions with multiple addressing modes to an assembler, so the code generation and error checking capabilities are limited to what takes place during the first pass of the assembler.

What makes life a bit easier is that the new V-Series instructions that utilize the complex addressing mode all have an equivalent instruction (or alias) in the 80186 instruction subset. This allows the V-Series macros to play a trick on the assembler by generating any prefix portion of the instruction, generating the alias instruction, then telling the assembler to back up and insert the true opcode of the new instruction. While all of this sounds complicated, it is really quite simple as shown in the following macro fragment.

```
        db      0fh
opcode: rol     arg1 arg2 arg3, arg4
next:   org     opcode
        db      arg5 + .adj + .immed
        org     next
```

The first line in the above example instructs the assembler to generate the escape opcode for the extended instruction set. The second line defines a local label o p c o d e which marks the opcode byte of the alias instruction when it is assembled and then assembles the alias instruction using the arguments supplied in the macro call. Next, the assembler is instructed to remember the current position and back up to the opcode field of the alias instruction so the fourth line can insert the correct V-Series opcode. Finally, the assembler is instructed to return to the point where the next instruction should be assembled and the process is complete.

Other instructions are much easier. For example, the μPD70320/322 STOP instruction can be codes simply as

```
stop    macro   arg1
        ifb     <arg1>                  ; if arg1 is blank
                db  0fh, 9eh            ; assemble the instruction
        else
               .errnb  <arg1>           ; force an error
                db  90h, 90h            ; and assemble some NOPs
        endif
        endm
```

What makes the code macros in the appendix appear overly complex is the large amount of error checking performed on the arguments to catch any errors that might try to sneak through and be undetected. If such as error did occur, it might be difficult to catch once it has been placed unflagged in the object file. In the above example, note that the macro call is checked that no arguments are specified in the macro call.

V-Series Code Macros

The new code macros support the bit manipulation, BCD arithmetic and the new μPD70320/322 (V25) instructions and are supplied in four separate include files. Separation of the code macros into logically related groups minimizes the possibility of a conflict with an independently defined label and increases the flexibility:

BITS.MAC	bit and bit field manipulation
BCD.MAC	BCD arithmetic
V25.MAC	μPD70320/322 unique instructions
MISC.MAC	miscellaneous instructions and prefixes

Where to get them

The complete source code listings for the code macros can be found in the appendices. To receive an electronically distributed copy of the MASM code macros and other V-Seriers related software utilities, contact your local NEC sales office or log on to the V-Series Microprocessor BBS. The BBS can be reached at (617)655-5615 using a username of MOCRO-BBS. From that point, follow the instructions for using the bulletin board.

2. V-Series Bit Manipulation Instruction Code Macros

```
.xlist
.186

comment /*

© Copyright 1987 NEC Electronics, Inc — All Rights Reserved
These code macros can be freely copied and used without restriction.

Written by:     Rick Naro
                V-Series Microprocessor Group
                NEC Electronics Inc.
                Natick Technology Center
                Natick, MA 01760

These code macros were developed and tested using MASM 4.0 from Microsoft. Note the following restrictions on
the use of the bit manipulation code macros:

— no instruction prefices

   Because of the way MASM processes the code macros, segment overrides and the lock prefix cannot be
   handled in the traditional manner. Be especially careful with using the BP register as a base pointer since this
   defaults to the stack segment and the assembler can insert a segment override prefix to access an operand
   within the current DS segment.

— ins code macro

   The ins macro prohibits the use of the .186 INS (input string) instruction. If you need to use the .186 INS instruc-
   tion, rename the insert bit field instruction macro to a name of your choosing.

— forward references

   Avoid the use of forward references since they are defined defined until pass 2 of the assembler and will be
   defined during any macro processing.

— memory references

   Because of the way the macros handle memory references, no white space can be between the addressing
   mode operands. All other formats work correctly (since they always have two arguments). For example:

   test 1    byte ptr [bx+di], cl        ; OK
   test 1    byte ptr [bx + si], cl      ; Error

— .186 pseudo op

   The bit manipulation code macros enable the assembly of 186 instructions as a side effect.

*/
```

```
        if1

        ifndef  .ax
            .val = 0
            irpc    z, acdb
        .&z&x   equ 1
            endm
        endif

        ifndef  .si
            .val = 6
            irpc    z, sd
        .&z&i   equ 1
            endm
        endif

        ifndef  .sp
            .val = 4
            irpc    z, sb
        .&z&p   equ 1
            endm
        endif

        ifndef  .al
            .val = 0
            irpc    z, acdb
        .&z&l   equ 0
        .&z&h   equ 0
            endm
        endif

_bitf   macro   arg1, arg2, arg3
        local   opcode, next, .immed

        ifdef   .&arg2
            .immed = 0
        else
            .immed = 8
        endif

        db  0fh
        opcode: add arg1, arg2
        next:   org opcode
            db  arg3 + .immed
            org next
        endm


 _bit    macro   arg1, arg2, arg3, arg4, arg5
     local   opcode, next,.ok, .adj,.immed

     .ok = 0
     ife type arg1
         ifdef .&arg1
             .adj = .&arg1
             .ok = 1
         endif
     else
         .adj = type arg1 - 1
         .ok = 1
     endif
```

```
ife .ok
    ifidn   <arg1>, <byte>
        .adj = 0
    endif
    ifidn   <arg1>, <BYTE>
        .adj = 0
    endif
    ifidn   <arg1>, <word>
        .adj = 1
    endif
    ifidn   <arg1>, <WORD>
        .adj = 1
    endif
endif

.immed = 8
ifnb    <arg4>
    ifidn   <arg4>, <cl>
        .immed = 0
    endif

    ifidn   <arg4>, <CL>
        .immed = 0
    endif
else
    ifidn   <arg2>, <cl>
        .immed = 0
    endif

    ifidn   <arg2>, <CL>
        .immed = 0
    endif
endif

ifnb    <arg4>
    ifdif   <arg4>, <1>
            db  0fh
        opcode: rol arg1 arg2 arg3, arg4
        next:   org opcode
            db  arg5 + .adj + .immed
            org next
    else
            db  0fh
        opcode: rol arg1 arg2 arg3 ,2
        next:   org opcode
            db  arg5 + .adj + .immed
            org $+1
            db  1
            org next
    endif
else
    ifdif   <arg2>, <1>
            db  0fh
        opcode: rol arg1, arg2
        next:   org opcode
            db  arg5 + .adj + .immed
            org next
```

```
        else
                db  0fh
            opcode: rol arg1, 2
            next:   org opcode
                db  arg5 + .adj + .immed
                org $+1
                db  1
                org next
        endif
    endif
    endm

ext macro   arg1, arg2, arg3
    ifnb    <arg1>
        ifnb    <arg2>
            ifb <arg3>
                _bitf   arg1, arg2, 33h
            else
                .errnb  <arg3>
                db  3 dup (90h)
            endif
        else
            .errb   <arg2>
            db  3 dup (90h)
        endif
    else
        .errb   <arg1>
        db  3 dup (90h)
    endif
    endm

   ins macro   arg1, arg2, arg3
       ifnb    <arg1>
           ifnb    <arg2>
               ifb <arg3>
                   _bitf   arg1, arg2, 31h
               else
                   .errnb  <arg3>
                   db  3 dup (90h)
               endif
           else
               .errb   <arg2>
               db  3 dup (90h)
           endif
       else
           .errb   <arg1>
           db  3 dup (90h)
       endif
       endm
```

```
test1   macro   arg1, arg2, arg3, arg4, arg5
    ifnb    <arg1>
        ifnb    <arg2>
            ifnb    <arg3>
                ifnb    <arg4>
                    ifb <arg5>
                        _bit    arg1, arg2, arg3, arg4, 10h
                    else
                        .errnb  <arg5>
                        db  3 dup (90h)
                    endif
                else
                    .errb   <arg4>
                    db  3 dup (90h)
                endif
            else
                _bit    arg1, arg2, arg3, arg4, 10h
            endif
        else
            .errb   <arg2>
            db  3 dup (90h)
        endif
    else
        .errb   <arg1>
        db  3 dup (90h)
    endif
    endm


clr1    macro   arg1, arg2, arg3, arg4, arg5
    ifnb    <arg1>
        ifnb    <arg2>
            ifnb    <arg3>
                ifnb    <arg4>
                    ifb <arg5>
                        _bit    arg1, arg2, arg3, arg4, 12h
                    else
                        .errnb  <arg5>
                        db  3 dup (90h)
                    endif
                else
                    .errb   <arg4>
                    db  3 dup (90h)
                endif
            else
                _bit    arg1, arg2, arg3, arg4, 12h
            endif
        else
            .errb   <arg2>
            db  3 dup (90h)
        endif
    else
        .errb   <arg1>
        db  3 dup (90h)
    endif
    endm
```

```
set1    macro   arg1, arg2, arg3, arg4, arg5
    ifnb    <arg1>
        ifnb    <arg2>
            ifnb    <arg3>
                ifnb    <arg4>
                    ifb <arg5>
                        _bit    arg1, arg2, arg3, arg4, 14h
                    else
                        .errnb  <arg5>
                        db  3 dup (90h)
                    endif
                else
                    .errb   <arg4>
                    db  3 dup (90h)
                endif
            else
                _bit    arg1, arg2, arg3, arg4, 14h
            endif
        else
            .errb   <arg2>
            db  3 dup (90h)
        endif
    else
        .errb   <arg1>
        db  3 dup (90h)
    endif
    endm


not1    macro   arg1, arg2, arg3, arg4, arg5
    ifnb    <arg1>
        ifnb    <arg2>
            ifnb    <arg3>
                ifnb    <arg4>
                    ifb <arg5>
                        _bit    arg1, arg2, arg3, arg4, 16h
                    else
                        .errnb  <arg5>
                        db  3 dup (90h)
                    endif
                else
                    .errb   <arg4>
                    db  3 dup (90h)
                endif
            else
                _bit    arg1, arg2, arg3, arg4, 16h
            endif
        else
            .errb   <arg2>
            db  3 dup (90h)
        endif
    else
        .errb   <arg1>
        db  3 dup (90h)
    endif
    endm

    endif
    .list
```

2.1 V-Series BCD Arithmetic Instruction Code Macros

```
.xlist

comment /*

© Copyright 1987 NEC Electronics, Inc — All Rights Reserved
These code macros can be freely copied and used without restriction.

Written by:     Rick Naro
                V-Series Microprocessor Group
                NEC Electronics Inc.
                Natick Technology Center
                Natick, MA 01760

These code macros were developed and tested using MASM 4.0 from Microsoft. Note the following restrictions on
the use of the BCD Code macros:

— no instruction prefixes

  Because of the way MASM processes the code macros, segment overrides and the lock prefix cannot be
  handled in the traditional manner. Be especially careful with using the BP register as a base pointer since this
  defaults to the stack segment and the assembler can insert a segment override prefix to access an operand
  within the current DS segment.

— forward references

  Avoid the use of forward references since they are defined defined until pass 2 of the assembler and will be
  defined during any macro processing.

— memory references

  Because of the way the macros handle memory references, no white space can be between the addressing
  mode operands. All other formats work correctly (since they always have two arguments). For example:

  rol4    byte ptr [bx+di]        ; OK
  rol4    byte ptr [bx + si]      ; Error

*/
```

```
    if1

add4s   macro   arg1
    ifb <arg1>
        db  0fh, 20h
    else
        .errnb  <arg1>
        db  90h, 90h
    endif
    endm

sub4s   macro   arg1
    ifb <arg1>
        db  0fh, 22h
    else
        .errb   <arg1>
        db  90h, 90h
    endif
    endm

cmp4s   macro   arg1
    ifb <arg1>
        db  0fh, 26h
    else
        .errb   <arg1>
        db  90h, 90h
    endif
    endm

_rot    macro   arg1, arg2, arg3
    local   opcode, next

    ifnb    <arg1>
        ifnb    <arg2>
            ifb <arg3>
                db  0fh
                opcode: add al, arg1
                next:   org opcode
                    db  arg2
                    org next
            else
                .errnb  <arg3>
                db  3 dup (90h)
            endif
        else
            .errb   <arg2>
            db  3 dup (90h)
        endif
    else
        .errb   <arg1>
        db  3 dup (90h)
    endif
    endm

  rol4    macro   arg1
      _rot    arg1, 28h
      endm

  ror4    macro   arg1
      _rot    arg1, 2ah
      endm

      endif
      .list
```

2.2 μPD70320/322 Unique Instruction Code Macros

```
.xlist

comment /*

© Copyright 1987 NEC Electronics, Inc — All Rights Reserved
These codemacros can be freely copied and used without restriction.

Written by:     Rick Naro
                V-Series Microprocessor Group
                NEC Electronics Inc.
                Natick Technology Center
                Natick, MA 01760

These code macros were developed and tested using MASM 4.0 from Microsoft. Note the following restrictions on
the use of these V25 unique code macros:

— no instruction prefixes

   Because of the way MASM processes the code macros, segment overrides and the lock prefix cannot be
   handled in the traditional manner. Be especially careful with using the BP register as a base pointer since this
   defaults to the stack segment and the assembler can insert a segment override prefix to access an operand
   within the current DS segment.

— forward references

   Avoid the use of forward references since they are defined defined until pass 2 of the assembler and will be
   defined during any macro processing.

*/
```

```
if1

ifndef  .P0
    .val = 0
    irpc    z, 012
.P&z    equ .val + 0
.PM&z   equ .val + 1
.PMC&z  equ .val + 2
    .val = .val + 8
    endm
endif

ifndef  .PT
.PT equ 38h
.PMT    equ 3bh
endif

ifndef  .INTM
.INTM   equ 40h
endif

ifndef  .EMS0
    .val = 44h
    irpc    z, 012
.EMS&z  equ .val + 0
.EXIC&z equ .val + 8
    .val = .val + 1
    endm
endif

ifndef  .RXB0
    .val = 60h
    irpc    z, 01
.RXB&z  equ .val + 0
.TXB&z  equ .val + 2
.SRMS&z equ .val + 5
.STMS&z equ .val + 6
.SCM&z  equ .val + 8
.SCC&z  equ .val + 9
.BRG&z  equ .val + 0ah
.SCE&z  equ .val + 0bh
.SEIC&z equ .val + 0ch
.SRIC&z equ .val + 0dh
.STIC&z equ .val + 0eh
    .val = .val + 10h
    endm
endif

ifndef  .TM0
    .val = 80h
    irpc    z, 01
.TM&z   equ .val + 0
.MD&z   equ .val + 2
    .val = .val + 8
    endm
endif
```

```
        ifndef  .TMC0
            .val = 90h
            irpc    z, 01
        .TMC&z  equ .val + 0
            .val = .val + 1
            endm
        endif

        ifndef  .TMMS0
            .val = 94h
            irpc    z, 012
        .TMMS&z equ .val + 0
        .TMIC&z equ .val + 8
            .val = .val + 1
            endm
        endif

        ifndef  .DMAC0
            .val = 0a0h
            irpc    z, 01
        .DMAC&z equ .val + 0
        .DMAM&z equ .val + 1
            .val = .val + 2
            endm
        endif

        ifndef  .DIC0
        .DIC0   equ 0ach
        .DIC1   equ 0adh
        endif

        ifndef  .STBC
        .STBC   equ 0e0h
        .RFM    equ 0e1h
        .WTC    equ 0e8h
        .FLAG   equ 0eah
        .PRC    equ 0ebh
        .TBIC   equ 0ech
        .IDB    equ 0ffh
        endif


stop    macro   arg1
        ifb <arg1>
            db  0fh, 9eh
        else
            .errnb  <arg1>
            db  90h, 90h
        endif
        endm
```

```
fint    macro   arg1
    ifb <arg1>
        db  0fh, 92h
    else
        .errnb  <arg1>
        db  90h, 90h
    endif
    endm

retrbi  macro   arg1
    ifb <arg1>
        db  0fh, 91h
    else
        .errnb  <arg1>
        db  90h, 90h
    endif
    endm

btclr   macro   arg1, arg2, arg3, arg4
    local   opcode, next

    ifnb    <arg1>
        ifnb    <arg2>
            ifnb    <arg3>
                ifb <arg4>
                    db  0fh, 9ch, .&arg1
                    opcode: jz  arg3
                    next:   org opcode
                        db  arg2
                        org next
                else
                    .errnb  <arg4>
                    db  5 dup (90h)
                endif
            else
                .errb   <arg3>
                db  5 dup (90h)
            endif
        else
            .errb   <arg2>
            db  5 dup (90h)
        endif
    else
        .errb   <arg1>
        db  5 dup (90h)
    endif
    endm

    endif
    .list
```

2.3 Miscellaneous Instruction Code Macros

```
.xlist

comment /*

© Copyright 1987 NEC Electronics, Inc — All Rights Reserved
These code macros can be freely copied and used without restriction.

Written by:     Rick Naro
                V-Series Microprocessor Group
                NEC Electronics Inc.
                Natick Technology Center
                Natick, MA 01760

These code macros were developed and tested using MASM 4.0 from Microsoft. Note the following restrictions on
the use of the repeat code macros:

— segment overrides on string instructions

  Segment overrides on string instructions using the REPC or REPNC prefixes must be coded in the following
  format to be correctly interpreted:

  repc    cmps    byte ptr es: [di], byte ptr cs: [si]

  where the cs: override can be any valid segment register.

This should not be too much trouble since the REPC and REPNC are only used on the compare string instruction
(all others either do not affect the cary flag or in the case of SCAS, do not accept a segment override prefix).

*/
```

```
    if1

repnc   macro   arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8
    .errb   <arg1>
    ifnb    <arg1>
        ifnb    <arg2>
            ifnb    <arg3>
                ifnb    <arg4>
                    ifnb    <arg5>
                        ifnb    <arg6>
                            ifnb    <arg7>
                                ifb <arg8>
                                    db  64h
                                    arg1 arg2 arg3 arg4, arg5 arg6 arg7
                                else
                                    .errnb  <arg8>
                                    db  90h, 90h
                                endif
                            else
                                .errb   <arg7>
                                db  90h, 90h
                            endif
                        else
                            .errb   <arg6>
                            db  90h, 90h
                        endif
                    else
                        .errb   <arg5>
                        db  90h, 90h
                    endif
                else
                    .errb   <arg4>
                    db  90h, 90h
                endif
            else
                .errb   <arg3>
                db  90h, 90h
            endif
        else
            db  64h
            arg1
        endif
    else
        .errb   <arg1>
        db  90h, 90h
    endif
    endm

repc    macro   arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8
    .errb   <arg1>
    ifnb    <arg1>
        ifnb    <arg2>
            ifnb    <arg3>
                ifnb    <arg4>
                    ifnb    <arg5>
                        ifnb    <arg6>
                            ifnb    <arg7>
                                ifb <arg8>
                                    db  65h
                                    arg1 arg2 arg3 arg4, arg5 arg6 arg7
```

```
                                    else
                                        .errnb  <arg8>
                                        db  90h, 90h
                                    endif
                                else
                                    .errb   <arg7>
                                    db  90h, 90h
                                endif
                            else
                                .errb   <arg6>
                                db  90h, 90h
                            endif
                        else
                            .errb   <arg5>
                            db  90h, 90h
                        endif
                    else
                        .errb   <arg4>
                        db  90h, 90h
                    endif
                else
                    .errb   <arg3>
                    db  90h, 90h
                endif
            else
                db  65h
                arg1
            endif
        else
            .errb   <arg1>
            db  90h, 90h
        endif
        endm

brkem   macro   arg1, arg2
        ifnb    <arg1>
            ifb <arg2>
                db  0fh, 0ffh, arg1
            else
                .errnb  <arg2>

             db  3 dup (90h)
         endif
     else
         .errb   <arg1>
         db  3 dup (90h)
     endif
     endm

     endif
     .list
```

3. Demonstration Program Listing

Microsoft® Macro Assembler Veresion 4.00 4/3/87 15:50:47

MASM Bit Manipulation CodeMacros Test Routine

```
                        page 44, 132
                        title   MASM Bit Manipulation CodeMacros Test Routine
                        name    testmac
                        .186

                C       include bits.mac
                C       .list

                C       include v25.mac
                C       .list

                C       include bcd.mac
                C       .list

                C       include misc.mac
                C       .list

                        .sall

                comment /*

                        Note that the data segment declarations are placed in front of
                        the executable code to avoid any forward references in the program.

                        */

0000            data    segment para    public  'DATA'

                byte_data       db      ?
0000  ??        word_data       dw      ?
0001  ????

0003            data    ends
```

```
Microsoft (R) Macro Assembler  Version 4.00              4/3/87 15:50:47
MASM Bit Manipulation CodeMacros Test Routine            Page     2-1


                        page+
0000                    code    segment para    public  'CODE'

                        public  main

                                assume  cs:code, ds:data, es:data

0000                    main    proc    far

                        ;
                        ;       Bit field instructions
                        ;

                                ext     ch, cl                  ; Extract bit field
                                ext     dh, 3

                                ins     bh, dh                  ; Insert bit field
                                ins     bl, 15

                        ;
                        ;       Bit manipulation instructions
                        ;

                                test1   byte ptr [si], 0        ; Test bit
                                test1   word ptr [bx+di], cl
                                test1   word_data, cl

                                clr1    bx, 1                   ; Clear bit
                                clr1    al, 4

                                set1    byte_data, 3            ; Set bit
                                set1    word_data, 15

                                not1    ch, cl                  ; Complement bit
                                not1    bp, cl
                                not1    byte_data, cl

                        ;
                        ;       BCD instructions
                        ;
```

```
                        add4s                           ; Add BCD strings
                        sub4s                           ; Subtract BCD strings
                        cmp4s                           ; Compare BCD strings

                        rol4    ch                      ; Rotate BCD digit left
                        ror4    byte_data               ; Rotate BCD digit right

                ;
                ;       V25 unique instructions
                ;

                        stop                            ; Stop oscillator
                        retrbi                          ; Return from bank switched interrupt
                        fint                            ; Finish interrupt

                        btclr   P0, 0, after            ; Bit test and clear
0052                before:
                        btclr   TBIC, 3, middle         ; Bit test and clear
0057                middle:
                        btclr   DMAM1, 7, before        ; Bit test and clear
005C                after:

                ;
                ;       Misc. V-Series instructions
                ;

                        repnc   cmpsb
                        repc    movs    byte ptr es:[di], byte ptr cs:[si]

                        brkem   80h
```

```
Microsoft (R) Macro Assembler  Version 4.00                4/3/87 15:50:47
MASM Bit Manipulation CodeMacros Test Routine              Page     3-1


                                        page+
                                ;
                                ;       Test the error detection abilities
                                ;

                                        repc                            ; No instruction to repeat
                                1       .errb   <>
testmac.ASM(103) : error 94: Forced error - string blank
                                1               .errb   <>
testmac.ASM(103) : error 94: Forced error - string blank
                                        brkem                           ; No vector number
                                1               .errb   <>
testmac.ASM(104) : error 94: Forced error - string blank

                                        btclr                           ; 0 args
                                1               .errb   <>
testmac.ASM(106) : error 94: Forced error - string blank
                                        btclr   p0                      ; 1 arg
                                1                       .errb   <>
testmac.ASM(107) : error 94: Forced error - string blank
                                        btclr   p0, 2                   ; 2 args
                                1                               .errb   <>
testmac.ASM(108) : error 94: Forced error - string blank
                                        btclr   p0, 2, before, 4        ; 4 args
                                1                               .errnb  <4>
testmac.ASM(109) : error 95: Forced error - string not blank

                                        fint    3                       ; 1 arg
                                1               .errnb  <3>
testmac.ASM(111) : error 95: Forced error - string not blank
                                        stop    2, 3                    ; 2 args
                                1               .errnb  <2>
testmac.ASM(112) : error 95: Forced error - string not blank
                                        retrbi  1                       ; 1 arg
                                1               .errnb  <1>
testmac.ASM(113) : error 95: Forced error - string not blank

                                        test1   al
                                1                       .errb   <>
testmac.ASM(115) : error 94: Forced error - string blank
```

```
Microsoft (R) Macro Assembler  Version 4.00                 4/3/87 15:50:47

MASM Bit Manipulation CodeMacros Test Routine               Page     3-2


                                        set1    byte ptr [bx]
                                1                                       .errb   <>
testmac.ASM(116) : error 94: Forced error - string blank

                                        ins     cl
                                1                       .errb   <>
testmac.ASM(118) : error 94: Forced error - string blank
                                        ext
                                1               .errb   <>
testmac.ASM(119) : error 94: Forced error - string blank
                                        ins     cl, dl, 3
                                1                               .errnb  <3>
testmac.ASM(120) : error 95: Forced error - string not blank

                                        add4s   test, test1
                                1               .errnb  <test>          ; No argument expected
testmac.ASM(122) : error 95: Forced error - string not blank
                                        rol4    cl, dl

                              main      endp

0097                          code      ends

                                        end     main

   905 Source  Lines
  2540 Total   Lines
   206 Symbols

 43782 Bytes symbol space free

     0 Warning Errors
    16 Severe  Errors
```

Demo Board μPD75308

Contents:

Author: B. Gough
NEC Electronics (Europe) GmbH

Related Documentation

μCOM 75X Family Product Description

Related products

μPD75308 4-Bit Microcomputer

1. Introduction

The idea of this application note is to introduce the development engineer to the 75X Family and μPD75308 which is a member of that family.

In the field of low power LCD applications the μPD75308 is a "State of the Art" single chip microcomputer from NEC.

The μPD75308 is the ideal choice in many application areas such as feature phones, pagers, remote controllers etc.

The μPD75308 is an 8K mask programmable microcomputer, but in the demo board circuit a UVPROM version is used (μPD75P308). This device is one to one with the mask version except in voltage range and as shown in this demo board, it is ideal for prototyping.

2. Demo Board Overview

From figure 3 we can see the application is split into 4 basic hardware blocks which are keyboard interface, LCD interface, LED interface and power up/down control circuit interface.

2.1 Key Board Interface

The keyboard interface consists of 16 keys with a 4 x 4 key matrix. Ports 7 and 6 are ideal for a key board interface because the read Port 7 has a falling edge key interrupt circuit. Port 7 also has programmable pull up resistors, hence a cost reduction in the overall design. Port 6 is used as the scan output port with 4 diodes being used for protection of outputs from short circuit.

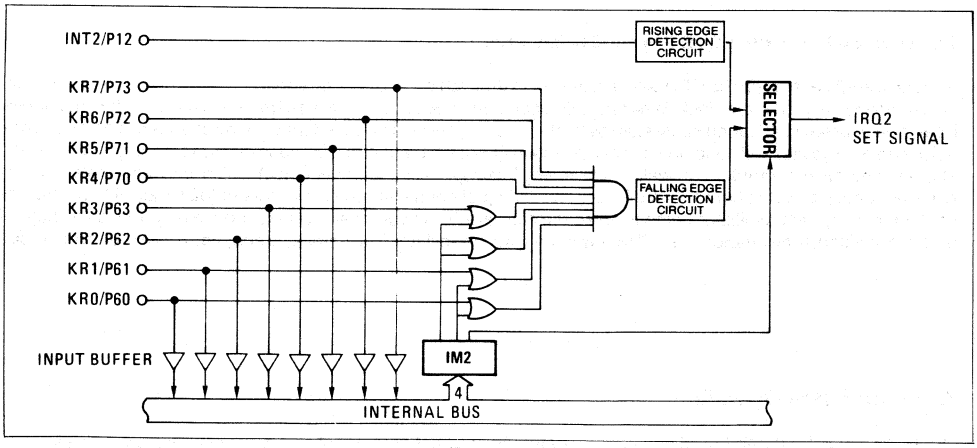


Figure 1.

Figure 1 shows the key interrupt circuit on ports 6/7, but in this application only port 7 is used.

2.2 LCD Interface

The μPD75308's LCD controller driver has 32 segments and 4 commons, so it can drive (32 x 1 = 32), (32 x 2 = 64), (32 x 3 = 96) and (32 x 4 = 128) picture elements. In this application a static method is used, hence 32 picture elements giving 4 characters.

2.3 LED Interface

Ports 4/5 are ideal for driving LEDs because you may select per bit open drain or a mask option pull up resistor on each port bit. On the (UVPROM and OTPROM) versions there is open drain with a max. sink current of 15 mA per pin, 60 mA per port max. and 120 mA max. for the chip. Ports 4/5 also have a maximum voltage of 10V.

2.4 Power Up/Down Detection Circuit Interface

Section 8, figure 6/7 shows the power up/down interrupt. The circuit is interfaced into interrupt input INT0. On a "warm" power up, a rising edge interrupts the processor and switches the subsystem oscillator (32 KHz) to the main system (4 MHz). On power down, the circuit provides a falling edge interrupt which switches the processor from main system oscillator (4 MHz) to subsystem oscillator (32 KHz).

Note: On a cold power up, the processor is reset and runs on the main system oscillator (4 MHz).

Power sources are:

Mains		Main
	=	
Power		System Osc
SUP	=	SUB
	=	
CAP (1F)		System

2.5 What will the 75308's SUB System Oscillator Run?

32 KHz is sufficient to run the CPU with a very slow instruction cycle of 122 μs and the watch timer. The watch timer also supplies the LCD base clock. When the microcomputer does not require hardware blocks other than the above, it can power down to 40 μA on subsystem oscillator. The display is maintained and can be changed. The microcomputer can also go into halt condition and can maintain the watch timer and LCD base clock, so display is maintained at 5 μA. The microcomputer can wake-up from any interrupt (testable or vectorize) which will generate a standby release signal (i.e. watch timer). The CPU is now running under the subsystem oscillator (32 KHz), with the change back to main system oscillator being handled the software. The user must first take the main system oscillator out of stop condition, then change the CPU to the main system oscillator. The timer also supplies a 2 KHz buzzer output.

3. Demo Board Operation

Figure 4 shows the demo board component layout. As shown it consists of an HEX key board, LCD display and buzzer (2 KHz).

On cold power up the display is clear. If the key(s) is pressed, then the HEX numbers are shifted right to left across the screen. The buzzer will also output a 2 KHz tone and the LEDs will change.

The disable reset switch should now be put in the on position (so now all resets are disabled).

It is helpful to put a scope on TP1, this enables you see the main system oscillator switch off in a stop condition when the subsytem is running the CPU.

The main system changes to the subsystem (32 KHz) when you switch off your mains supply. When the μPD75308 is in subsystem you can still display data and enter new data into the display. In subsystem the LEDs and buzzer are switched off to save power. Switching the main power back on puts you back into main system. The LEDs and buzzer are also enabled. You can see this switching of sub to main system at TP1 on your scope.

4. Circuit Diagram

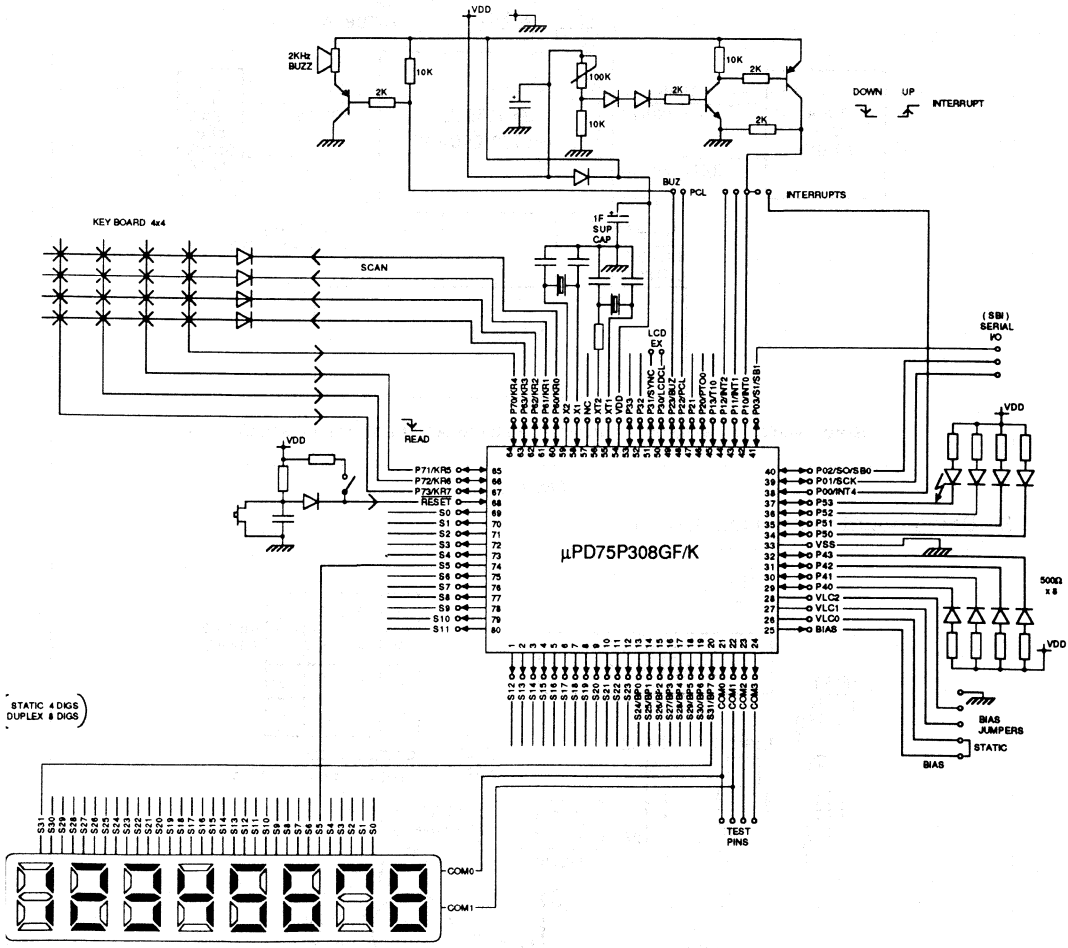
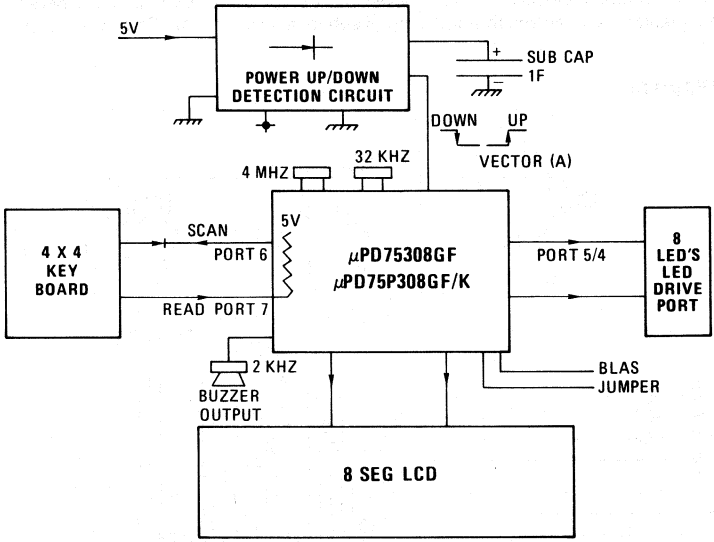


Figure 2.

4a. Block Diagram of Application



Note: Vector (A) is the power up/down, from the power up/down detection circuit.

Figure 3.

5. Demo Board Component Layout

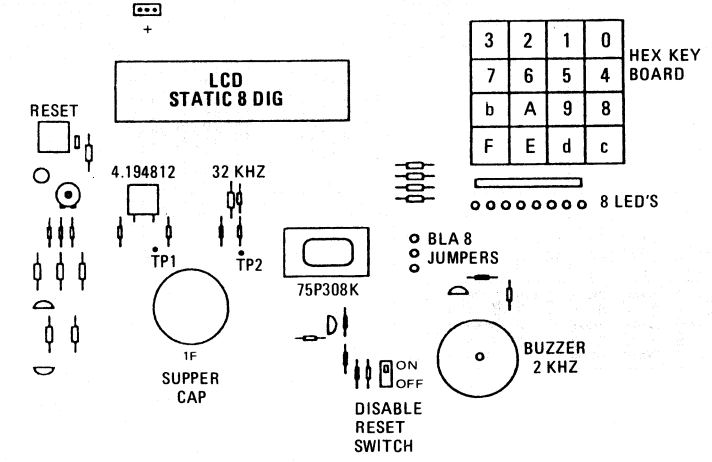


Figure 4.

6. Main Design Points summary

This application shows the main feature of the μPD75308 which are in the main design points summary as follows:

1) Key interrupt ports
2) Programmable pull up ports
3) Open drain ports used for LEDs (15 mA max. 60 mA per port max., 120 mA per chip max.)
4) Mask option bias resistors
5) High frequency noise filter, interrupt input (used for power up/down int.)
6) Low power consumption, 32 KHz sub system oscillator, run CPU and LCD off supper cap
7) LCD controller/driver on chip
8) LCD expansion port
9) 2 KHz (buzzer) output P23
10) 3 timers on chip (one watch)

7. μPD75308 Demo Software

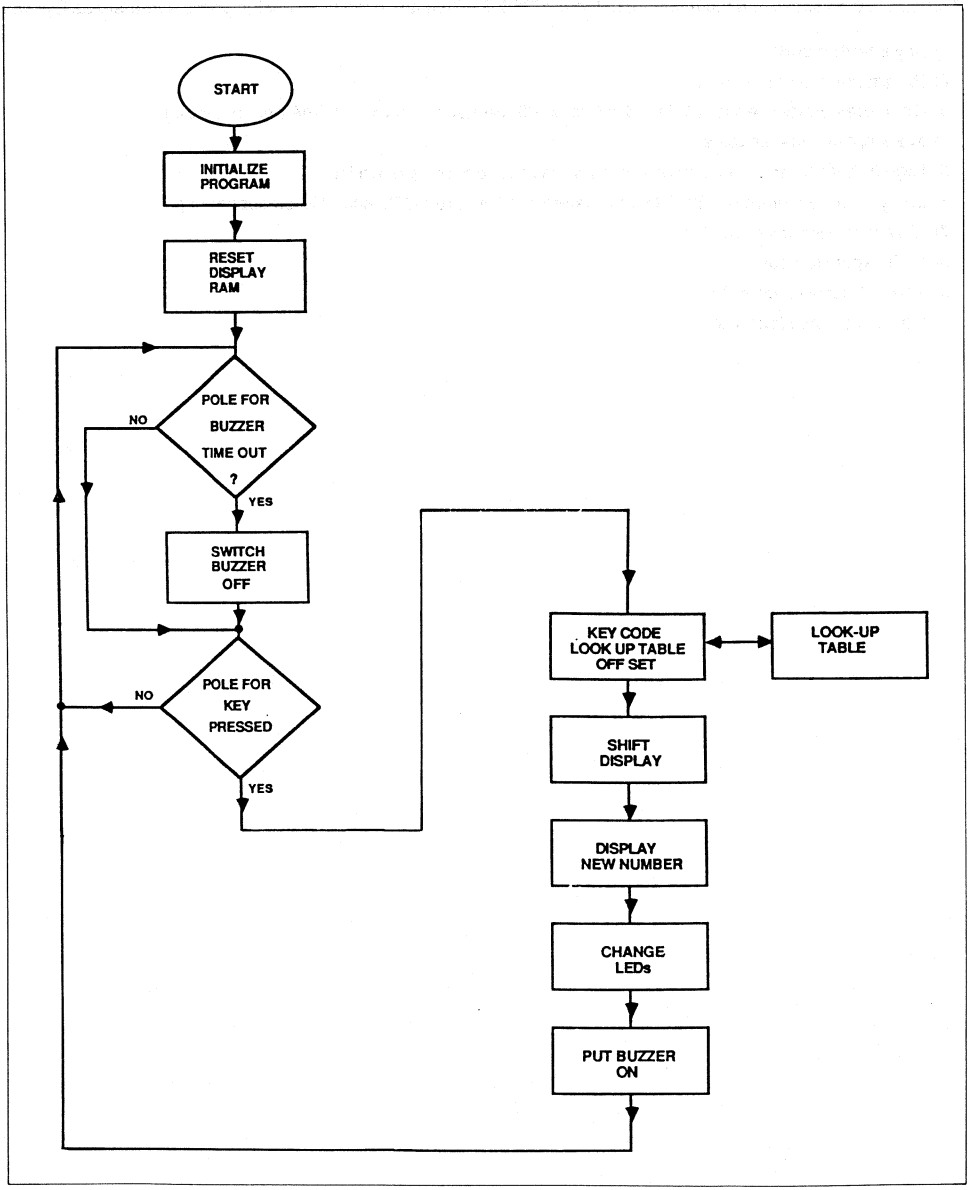


Figure 5.

8. Demo Board Power Up/Down Circuit

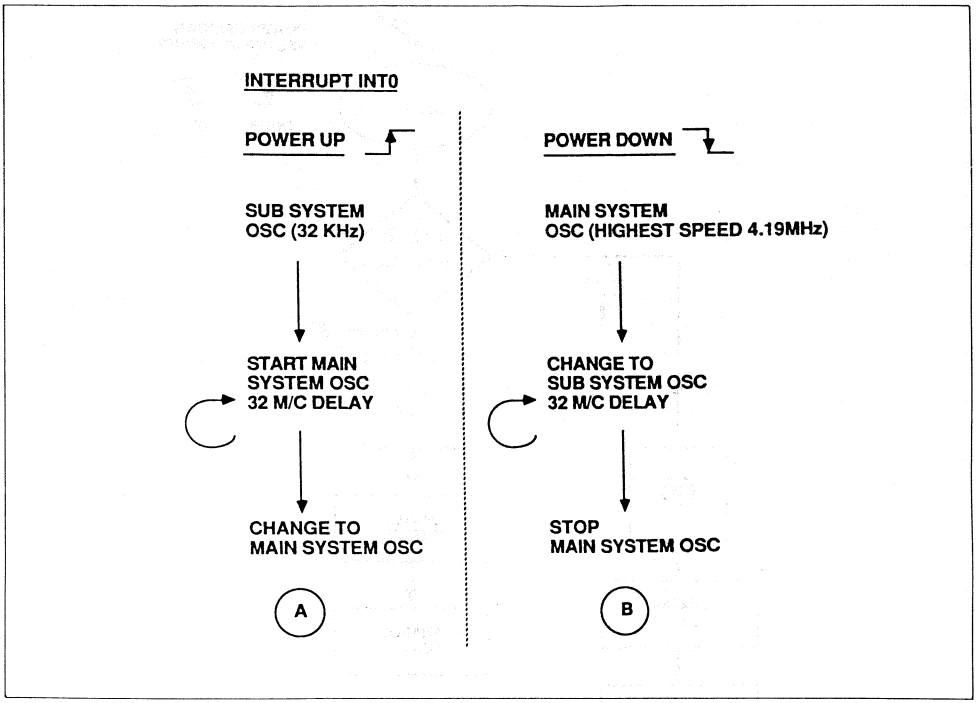


Figure 6.

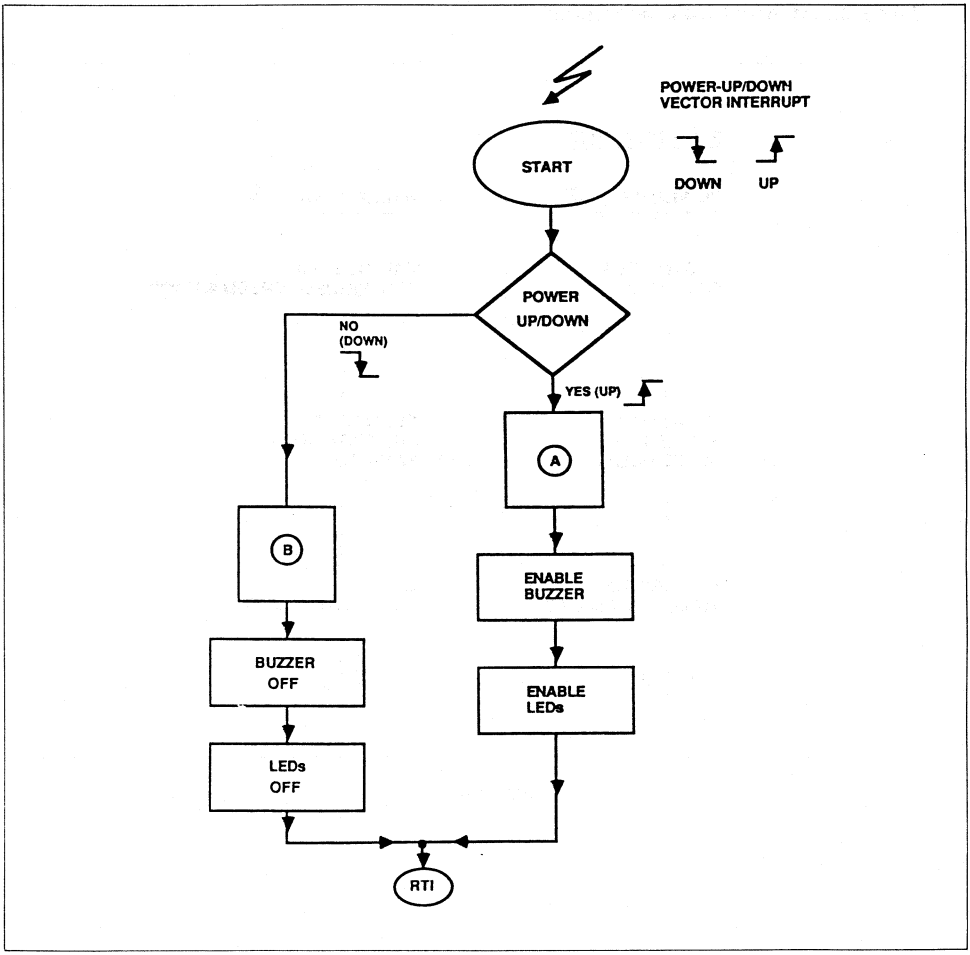

POWER-UP/DOWN
VECTOR INTERRUPT
DOWN
UP
START
POWER
UP/DOWN
NO
(DOWN)
YES (UP)
A
B
ENABLE
BUZZER
BUZZER
OFF
ENABLE
LEDs
LEDs
OFF
RTI

Figure 7.

9. Software Listing

SOURCE STATEMENT

```
        NAME    LCDIN
;LCDBOARD SET UP PROGRAM WRITTEN FOR 75308 BY B.GOUGH 4/87
;*************************************************************************
;NAME            :LCDIN
;DESC            :LCDBOARD SET UP PROGRAM
;DEVICE          :75308
;MEMORY BANK     :15
;REGISTER BANK   :-
;USED HARDWARE   :PORTS 6/7/2 BSB0,BSB2,BSB3
;INPUT           :-
;OUTPUT          :-
;DESTROY         :XA
;CALLS           :BUZZOF,RETDIS,KEYIN,CONVER,SHIFT,DISPLY,LED,BUZZZ
;*************************************************************************
        PUBLIC  SELM0,SELM1,SELM15
        EXTRN   CODE(BUZZOF,RETDIS,KEYIN,CONVER,SHIFT,DISPLY,LED,BUZZZ)
        EXTRN   CODE(LCDPD)
;************VECTOR TABLE*****************************************************
        STKLN   40H
        VENT0   MBE=1,RBE=0,LCDIN       ;START VECTOR
        VENT2   MBE=1,RBE=0,LCDPD       ;POWER DOWN/UP VECTOR
;************GETI TABLE*******************************************************
        GET1    CSEG    IENT
        SELM0:  SEL     MB0
        SELM1:  SEL     MB1
        SELM15: SEL     MB15
;*************************************************************************
C1SEG   CSEG    INBLOCK
LCDIN:  GETI    SELM15
        MOV     XA,#STACK       ;SET STACK
        MOV     SP,XA
        MOV     A,#0011B        ;SET HIGH SPEED SYSTEM CLOCK
        MOV     PCC,A
        MOV     A,#1001B        ;SET EXTERNAL CLOCK
        MOV     CLOM,A
;************SET LCD BASE FREQUENCE FROM WATCH*******************************
        MOV     XA,#07H         ;FROM SUBSYSTEM OSC 32KHZ
        MOV     WM,XA
;************SET PORTS UP****************************************************
        MOV     XA,#11110000B   ;SET PORT6 OUTPUTS
        MOV     PMGA,XA
        MOV     XA,#00110100B   ;SET PORTS 5/4/2 OUTPUTS AND PORT7 INPUT
        MOV     PMGB,XA
        MOV     A,#0H           ;SET PORT6 LOW
        OUT     PORT6,A
        MOV     A,#0FH          ;SET PORT2 HIGH
        OUT     PORT2,A
```

**

SOURCE STATEMENT

```
        MOV     XA,#0FFH
        OUT     PORT4,XA        ;SET PORT4/5 HIGH
        DI                      ;DISABLE INTERRUPTS
        MOV     A,#1H           ;SET PORT7 AS KEY BOARD INPUT
        MOV     IM2,A
        MOV     XA,#80H         ;SET PULL UPS ON PORT7
        MOV     POGA,XA
        MOV     A,#9H           ;SET POWER DOWN EDGE HIGH/LOW
        MOV     IM0,A           ;SAMPLE FREQENCE FXX/64
        SET1    IE0             ;ENABLE POWER DOWN INTERRUPT
        CLR1    IRQ0            ;POWER DOWN/UP FLAG CLEAR
        SET1    BSB0.3          ;BUZZ IN MAIN SYSTEM ONLY
        SET1    BSB0.1          ;SET LEDS ON FLAG
        SET1    BSB0.0          ;POWER DOWN FLAG SET
        MOV     A,#0H           ;CLEAR 60MS BUZZ DELAY COUNTER
        MOV     BSB3,A
        MOV     BSB2,A
        EI                      ;ENABLE ALL INTERRUPS
;*************MAIN PROGRAM*****************************************************
        CALL    !RETDIS         ;RESET DISPLAY RAM
POLE1:  SKF     IRQW            ;CHECK BUZZ TIME OUT 4 MSEC
        CALL    !BUZZOF         ;PUT BUZZER OFF
        SKTCLR  IRQ2            ;WAITING FOR KEY INPUT
        BR      POLE1
        CALL    !KEYIN          ;KEY BOARD SCAN PROGRAM
        BR      POLE1           ;BAD KEY NO DISPLAY(DEBOUNCE)
        CALL    !CONVER         ;KEY CODE LOOK UP TABLE OFF SET
        CALL    !SHIFT          ;SHIFT DISPLAY
        CALL    !DISPLY         ;KEY PRESSED ON DISPLAY
        CALL    !LED            ;CHANGE LED PROGRAM
        CALL    !BUZZZ          ;PUT BUZZER ON
        BR      POLE1
        END
```

**

(COMMAND FILE :)

SOURCE STATEMENT

```
        NAME    LCDRET
;DISPLAY RAM RESET PROGRAM WRITTEN FOR 75308 BY B.GOUGH
;6/87
;*********************************************************
;NAME            :RETDIS
;DESC            :RESET DISPLAY RAM 1E0,1FF
;DEVICE          :75308
;MEMORY BANK     :1
;REGISTER BANK   :-
;USED HARDWARE   :DISPLAY RAM
;INPUT           :-
;OUTPUT          :-
;DESTROY         :A,HL
;CALLS           :-
;*********************************************************
        PUBLIC  RETDIS
        EXTRN   CODE(SELM1)

C9SEG   CSEG    SENT
RETDIS: GETI    SELM1
        MOV     HL,#0E0H        ;LOAD HL WITH HL ADDRESS POINTER
        MOV     A,#0H
NEXFL:  MOV     @HL,A           ;FILL DISPLAY RAM WITH 00H
        INCS    L               ;NEXT NIBBLE
        BR      NEXFL
        MOV     HL,#0F0H        ;NEXT PART OF DISPLAY RAM
NEXFL1: MOV     @HL,A           ;FILL DISPLAY RAM WITH 00H
        INCS    L               ;NEXT NIBBLE
        BR      NEXFL1
        RET
        END
```

**

(COMMAND FILE :)

SOURCE STATEMENT

```
         NAME    LCDBUZOF
;SWITCH OFF BUZZER AFTER 300 MSEC WRITTEN FOR 75308 BY
;B.GOUGH 6/87
;*********************************************************
;NAME             :BUZZOF
;DESC             :SWITCH OFF BUZZER
;DEVICE           :75308
;MEMORY BANK      :15
;REGISTER BANK    :-
;USED HARDWARE    :BUZZER 2KHZ, BSB0,BSB3
;INPUT            :-
;OUTPUT           :-
;DESTROY          :XA
;CALLS            :-
;*********************************************************
         PUBLIC  BUZZOF
         EXTRN   CODE(SELM15)
C10SEG   CSEG    SENT
BUZZOF:  GETI    SELM15
         CLR1    IRQW              ;CLEAR TIMER OUT FLAG
         MOV     A,BSB2            ;60MS BUZZ DELAY REG
         INCS    A
         BR      NEXDL0
         BR      NEXDL1
NEXDL0:  MOV     BSB2,A            ;STORE DELAY COUNT LSN
         RET
NEXDL1:  MOV     BSB2,A            ;ZERO BSB2 REG STORE LSN
         MOV     A,BSB3
         INCS    A
         SKE     A,#03H
         BR      NEXDL2
         BR      NEXDL3
NEXDL2:  MOV     BSB3,A            ;STORE DELAY COUNT MSN
         RET
NEXDL3:  MOV     XA,#07H           ;DISABLE BUZZER AFTER 300MS
         MOV     WM,XA
         SET1    PORT2.3           ;MAKE SURE PORT BUZZER IS HIGH
         RET
         END
```

**

(COMMAND FILE :)

SOURCE STATEMENT

```
        NAME    LCDKEY
;PROGRAM FOR KEYBOARD SCAN  RESULT IN BC REG WRITTEN FOR 75308
;BY B.GOUGH 4/87
;*************************************************************************
;NAME            :KEYIN
;DESC            :KEYBOARD PROGRAM
;DEVICE          :75308
;MEMORY BANK     :15
;REGISTER BANK   :-
;USED HARDWARE   :PORT6,PORT7,BT,BSB0
;INPUT           :-
;OUTPUT          :-
;DESTROY         :L,A,BC,PSW
;CALLS           :-
;*************************************************************************

        PUBLIC  KEYIN
        EXTRN   CODE(SELM15)
C2SEG   CSEG    SENT
KEYIN:  GETI    SELM15
        SKT     BSB0.1
        BR      NODEY
        MOV     A,#0FH          ;SET 2MS DEBOUNCE DELAY
        MOV     BTM,A
POLE2:  SKTCLR  IRQBT
        BR      POLE2
NODEY:  IN      A,PORT7         ;SAVE READ PORT7
        SKE     A,#0FH
        BR      TRKEY           ;TRUE KEY
        CLR1    IRQ2            ;FALSE KEY
        RET
TRKEY:  XCH     A,C             ;SAVE READ VALUE IN C REG
        MOV     A,#0FH          ;SET PORT6 HIGH
        OUT     PORT6,A
        MOV     L,#08H          ;SET PORT6 OFF SET
NEKEY:  CLR1    PORT6.0L        ;SET SCAN BIT LOW
        IN      A,PORT7
        SKE     A,#0FH          ;CHECK READ PORT6 FOR KEY PRESS
        BR      KEYP            ;KEY IS PRESSED
        SET1    PORT6.0L        ;SET HIGH LAST SCAN BIT
        INCS    L               ;NEXT SCAN BIT
        SKE     L,#0CH          ;CHECK FOR BAD KEY
        BR      NEKEY           ;SCAN NEXT KEY
        MOV     A,#04H          ;SET PORT6 LOW
        OUT     PORT6,A
        CLR1    IRQ2            ;CLEAR KEY BOARD FLAG
        RET                     ;BAD KEY
```

**

SOURCE STATEMENT

```
KEYP:   IN      A,PORT6         ;SAVE SCAN VALUE IN B REG
        XCH     A,B
        MOV     A,#0H           ;SET PORT6 LOW
        OUT     PORT6,A
        CLR1    IRQ2            ;CLEAR KEY BOARD FLAG
        RETS                    ;BC REG BANK 3 HAS RESULT
        END
```

```
                                        **

        (COMMAND FILE :                  )

SOURCE STATEMENT


        NAME    LCDCON
;KEY CODE TO LOOK UP TABLE OFF SET  PROGRAM WRITTEN FOR 75308
;BY B.GOUGH 4/87
;*************************************************************************
;NAME            :CONVER
;DESC            :KEY CODE LOOK UP TABLE
;DEVICE          :75308
;MEMORY BANK     :15
;REGISTER BANK   :-
;USED HARDWARE   :-
;INPUT           :-
;OUTPUT          :-
;DESTROY         :A,BC PSW
;CALLS           :-
;*************************************************************************
        PUBLIC  CONVER
        EXTRN   CODE(SELM15)

CSEG    CSEG    SENT
CONVER: GETI    SELM15
;*************INVERT KEY BOARD DATA*****************************************
        XCH     A,B             ;INVERT B
        NOT     A
        XCH     A,B
        XCH     A,C             ;INVERT C
        NOT     A
        XCH     A,C
;*************CAL NEW VALUE OF BC*******************************************
        SKE     B,#1H
        BR      NETV
        SKE     C,#1H
        BR      NEXC
        MOV     B,#0H           ;SET B VALUE
        MOV     C,#0H           ;SET C VALUE
        RET
NEXC:   SKE     C,#2H
        BR      NEXC1
        MOV     B,#0H           ;SET B VALUE
        MOV     C,#8H           ;SET C VALUE
        RET
NEXC1:  SKE     C,#4H
        BR      NEXC2
        MOV     B,#1H           ;SET B VALUE
        MOV     C,#0H           ;SET C VALUE
        RET
NEXC2:  MOV     B,#1H           ;SET B VALUE
        MOV     C,#8H           ;SET C VALUE
```

**

```
SOURCE STATEMENT

        RET
NETV:   SKE     B,#2H
        BR      NETV1
        SKE     C,#1H
        BR      NEXC3
        MOV     B,#2H           ;SET B VALUE
        MOV     C,#0H           ;SET C VALUE
        RET
NEXC3:  SKE     C,#2H
        BR      NEXC4
        MOV     B,#2H           ;SET B VALUE
        MOV     C,#8H           ;SET C VALUE
        RET
NEXC4:  SKE     C,#4H
        BR      NEXC5
        MOV     B,#3H           ;SET B VALUE
        MOV     C,#0H           ;SET C VALUE
        RET
NEXC5:  MOV     B,#3H           ;SET B VALUE
        MOV     C,#8H           ;SET C VALUE
        RET
NETV1:  SKE     B,#4H
        BR      NETV2
        SKE     C,#1H
        BR      NEXC6
        MOV     B,#4H           ;SET B VALUE
        MOV     C,#0H           ;SET C VALUE
        RET
NEXC6:  SKE     C,#2H
        BR      NEXC7
        MOV     B,#4H           ;SET B VALUE
        MOV     C,#8H           ;SET C VALUE
        RET
NEXC7:  SKE     C,#4H
        BR      NEXC8
        MOV     B,#5H           ;SET B VALUE
        MOV     C,#0H           ;SET C VALUE
        RET
NEXC8:  MOV     B,#5H           ;SET B VALUE
        MOV     C,#8H           ;SET C VALUE
        RET
NETV2:  SKE     C,#1H
        BR      NEXC9
        MOV     B,#6H           ;SET B VALUE
        MOV     C,#0H           ;SET C VALUE
        RET
NEXC9:  SKE     C,#2H
        BR      NEXC10
        MOV     B,#6H           ;SET B VALUE
        MOV     C,#8H           ;SET C VALUE
        RET
```

**

SOURCE STATEMENT

```
NEXC10: SKE     C,#4H
        BR      NEXC11
        MOV     B,#7H           ;SET B VALUE
        MOV     C,#0H           ;SET C VALUE
        RET
NEXC11: MOV     B,#7H           ;SET B VALUE
        MOV     C,#8H           ;SET C VALUE
        RET
        END
```

```
                                        **

        (COMMAND FILE :                  )


SOURCE STATEMENT


         NAME    LCDSHIFT
;THIS PROGRAM SHIFTS THE LCD DISPLAY 1 TO THE LEFT
;WRITTEN BY B.GOUGH 6/87
;*********************************************************
;NAME             :SHIFT
;DESC             :SHIFT DISPLAY
;DEVICE           :75308
;MEMORY BANK      :1
;REGISTER BANK    :-
;USED HARDWARE    :DISPLAY RAM,BSB1
;INPUT            :-
;OUTPUT           :-
;DESTROY          :XA,HL,D,E,BSB1.0
;CALLS            :-
;*********************************************************
         PUBLIC  SHIFT
         EXTRN   CODE(SELM15,SELM1)

C8SEG    CSEG    SENT
SHIFT:   GETI    SELM1
;***********MOVE BLOCK 3 TO 4**************************
         MOV     D,#0H           ;CLEAR NIBBLE BLOCK POINTER
         MOV     HL,#0F8H        ;BOTTOM BLOCK POINTER
         PUSH    HL
         MOV     HL,#0F0H        ;ABOVE BLOCK POINTER
BLOC1:   SKE     D,#8H
         BR      BLOC2
         BR      BLOC3
BLOC2:   MOV     A,@HL           ;MOVE ABOVE BLOCK NIBBLE
         XCH     A,E             ;E REG HAS SHIFT DATA
         INCS    L               ;POINT NEXT NIBBLE IN ABOVE BLOCK
         NOP                     ;STOP SKIP CONDITION
         POP     XA              ;GET POINTER FOR BOTTOM BLOCK
         PUSH    HL              ;SAVE NEW ABOVE BLOCK POINTER
         XCH     XA,HL
         XCH     A,E             ;RECOVER DATA
         MOV     @HL,A           ;MOVE DATA FROM ABOVE BLOCK TO BOTTOM
         INCS    L               ;POINT NEXT NIBBLE IN BOTTOM BLOCK
         NOP                     ;STOP SKIP CONDITION
         XCH     XA,HL           ;STORE NEW NIBBLE BOTTOM BLOCK POINTER
         POP     HL              ;RECOVER ABOVE BLOCK POINTER
         PUSH    XA              ;SAVE NEW BOTTOM BLOCK POINTER
         INCS    D               ;NEXT BLOCK NIBBLE
         BR      BLOC1           ;NEXT NIBBLE BLOCK SHIFT
         BR      BLOC4           ;BRANCH ON OVERFLOW OF D REG
;***********MOVE BLOCK 2 TO 3**************************
BLOC3:   GETI    SELM15
         SKT     BSB1.0          ;CHECK FOR EXIT FLAG
```

**

SOURCE STATEMENT

```
          BR      BLOC5
          CLR1    BSB1.0          ;CLEAR EXIT FLAG
          POP     XA              ;BALANCE STACK
          RET                     ;SHIFT FINISHED
BLOC5:    GETI    SELM1
          POP     XA              ;BALANCE STACK
          MOV     HL,#0F0H        ;BOTTOM BLOCK POINTER
          PUSH    HL
          MOV     HL,#0E8H        ;ABOVE BLOCK POINTER
          BR      BLOC2           ;NEXT NIBBLE BLOCK SHIFT
;**********MOVE BLOCK 3 TO 4*****************************
BLOC4:    GETI    SELM15
          SET1    BSB1.0          ;SET EXIT FLAG
          GETI    SELM1
          MOV     D,#0H           ;CLEAR NIBBLE BLOCK POINTER
          POP     XA              ;BALANCE STACK
          MOV     HL,#0E8H        ;BOTTOM BLOCK POINTER
          PUSH    HL
          MOV     HL,#0E0H        ;ABOVE BLOCK POINTER
          BR      BLOC2           ;NEXT NIBBLE BLOCK SHIFT
;*************************************************************
          END
```

**

(COMMAND FILE :)

SOURCE STATEMENT

```
        NAME    LCDDIS
;LCD DISPLAY PROGRAM WRITTEN FOR 75308 BY B.GOUGH 4/87
;*****************************************************************
;NAME            :DISPLY
;DESC            :LCD DISPLAY PROGRAM FIRST DIGIT ONLY BC HAS DIG CODE
;DEVICE          :75308
;MEMORY BANK     :1
;REGISTER BANK   :-
;USED HARDWARE   :LCD DRIVER/CONTROLLER
;INPUT           :-
;OUTPUT          :-
;DESTOY          :XA,BC,HL,D
;CALLS           :TABLE1
;*****************************************************************
        PUBLIC  DISPLY
        EXTRN   CODE(SELM15,SELM1)
        EXTRN   CODE(TABLE1)

C4SEG   CSEG    SENT
DISPLY: GETI    SELM1
        MOV     D,#08H          ;LOOK UP TABLE 8 BIT NIBBLE COUNT
        MOV     HL,#0E0H        ;TOP OF DISPLAY TABLE
NETL2:  MOV     XA,BC           ;LOAD XA WITH LOOK UP TABLE OFF SET
        PUSH    XA
NETL:   CALL    !TABLE1
        MOV     @HL,A           ;LOAD DISPLAY RAM
        INCS    L               ;NEXT DISPLAY NIBBLE
        SKE     L,#8H           ;SKIP IF RAM IS FULL
        BR      NETL1
        GETI    SELM15
        MOV     XA,#3CH         ;ENABLE LCD,STATIC,FRAME 512HZ
        MOV     LCDM,XA
        MOV     A,#1H           ;POWER TO BIAS RESISTOR NETWORK
        MOV     LCDC,A
        POP     XA              ;BALANCE STACK
        RET                     ;DISPLAY RAM FULL
NETL1:  POP     XA              ;LOAD LOOK UP TABLE OFF SET
        INCS    A               ;NEXT LOCTION IN LOOK UP TABLE
        NOP
        PUSH    XA              ;SAVE NEW LOOK UP TABLE LOCATION
        INCS    D               ;INC 8 NIBBLE COUNT
        BR      NETL
        POP     XA              ;BALANCE STACK
        END
```

```
                                        **

        (COMMAND FILE :                 )

SOURCE STATEMENT

                    NAME    LCDBUZZ
;LCD BUZZER PROGRAM WRITTEN FOR 75308 BY B.GOUGH 4/87
;*****************************************************************
;NAME               :BUZZZ
;DESC               :PUTS 2KHZ BUZZER ON/OFF
;DEVICE             :75308
;MEMORY BANK        :15
;REGISTER BANK      :-
;USED HARDWARE      :BUZZER 2KHZ,BSB0
;INPUT              :-
;OUTPUT             :-
;DESTROY            :XA
;CALLS              :-
;*****************************************************************
        PUBLIC  BUZZZ
        EXTRN   CODE(SELM15)

C7SEG   CSEG    SENT
BUZZZ:  GETI    SELM15
        SKT     BSB0.3              ;CHECK MAIN SYSTEM ,SUB SYSTEM FLAG
        RET                         ;RETURN ,IN SUB NOW
        MOV     XA,#87H             ;ENABLE BUZZER
        MOV     WM,XA
        CLR1    PORT2.3             ;SET BUZZER PORT2.3 LOW
        MOV     A,#0H               ;RESET DELAY REGS
        MOV     BSB2,A
        MOV     BSB3,A
        RET
        END
```

```
                                        **

        (COMMAND FILE :                  )


SOURCE STATEMENT


         NAME    LCDLED
;LED CONTROL PROGRAM WRITTEN FOR 75308 BY B.GOUGH 4/87
;*********************************************************************
;NAME            :LED
;DESC            :LED CONTROL PROG
;DEVICE          :75308
;MEMORY BANK     :15
;REGISTER BANK   :-
;USED HARDWARE   :PORTS 4/5
;INPUT           :-
;OUTPUT          :-
;DESTROY         :XA PSW
;CALLS           :-
;*********************************************************************
         PUBLIC  LED
         EXTRN   CODE(SELM15)

C5SEG    CSEG    SENT
LED:     GETI    SELM15
         SKT     BSB0.1          ;CHECK IF LED S ON OR OFF
         RET
         SKT     PORT5.0
         BR      LEDOF1
         MOV     XA,#0AAH        ;PUT PATTERN 10101010 ON LED S
         OUT     PORT4,XA
         RET
LEDOF1:  MOV     XA,#55H         ;PUT PATTERN 01010101 ON LED S
         OUT     PORT4,XA
         RET
         END
```

```
                                        **

        (COMMAND FILE :                  )

SOURCE STATEMENT


         NAME    LCDPD
;POWER UP/DOWN INTERRUPT PROGRAM WRITTEN FOR 75308 BY B.GOUGH
;4/87
;*********************************************************************
;NAME             :LCDPD
;DESC             :POWER UP/DOWN PROGRAM
;DEVICE           :75308
;MEMORY BANK      :15
;REGISTER BANK    :-
;USED HARDWARE    :INTERRUPT INPUT, SUBSYSTEM OSC 32KHZ,BSB0
;INPUT            :-
;OUTPUT           :-
;DESTROY          :A,CY
;CALLS            :-
;*********************************************************************
         PUBLIC  LCDPD
         EXTRN   CODE(SELM15)
C6SEG    CSEG    SENT
LCDPD:   GETI    SELM15
         DI                          ;DISABLE INTERRUPTS
         SKT     BSB0.0              ;CHECK IF P/D FLAG IS SET
         BR      POWUP               ;TO POWER UP
         SET1    SCC.0               ;SWITCH TO SUB SYSTEM OSC 32KHZ
         MOV     A,#0H               ;SET UP 32 M/C DELAY
NEXD1:   INCS    A
         BR      NEXD1
         MOV     A,#0H
NEXD2:   INCS    A
         BR      NEXD2
         MOV     A,#0H
NEXD3:   INCS    A
         BR      NEXD3
         SET1    SCC.3               ;STOP MAIN SYSTEM OSC 4.192 MHZ
         MOV     XA,#0FFH
         OUT     PORT4,XA            ;SWITCH PORT LEDS OFF
         MOV     A,#0H               ;SET LOW TO HIGH EDGE POWER UP
         MOV     IM0,A
         CLR1    IRQ0                ;CLEAR POWER UP/DOWN FLAG
         CLR1    BSB0.0              ;SET POWER UP FLAG
         CLR1    BSB0.1              ;SET LED OFF FLAG
         CLR1    BSB0.3              ;DISABLE BUZZER
         EI
         RETI
POWUP:   CLR1    SCC.3               ;START MAIN SYSTEM OSC 4.192 MHZ
         MOV     A,#0H               ;START 32 M/C DELAY
NEXA1:   INCS    A
         BR      NEXA1
```

**

SOURCE STATEMENT

```
        MOV     A,#0H
NEXA2:  INCS    A
        BR      NEXA2
        MOV     A,#0H
NEXA3:  INCS    A
        BR      NEXA3
        CLR1    SCC.0           ;CHANGE FROM SUB TO MAIN SYSTEM OSC
        MOV     A,#9H           ;SET HIGH TO LOW EDGE POWER DOWN
        MOV     IM0,A
        MOV     XA,#04H         ;SWITCH MAIN SYSTEM CLOCK FOR LCD BASE FREQU
        MOV     WM,XA
        CLR1    IRQ0            ;CLEAR POWER UP/DOWN FLAG
        SET1    BSB0.0          ;SET POWER DOWN FLAG
        SET1    BSB0.1          ;SET LED ON FLAG
        SET1    BSB0.3          ;ENABLE BUZZER
        EI
        RETI
        END
```

**

(COMMAND FILE :)

SOURCE STATEMENT

```
        NAME    LCDTABL1
;DISPLAY DATA LOOK UP TABLE WRITTEN FOR 75308 BY B.GOUGH 4/87
;*****************************************************************
;NAME            :TABLE1
;DESC            :LOOK UP TABLE
;DEVICE          :75308
;MEMORY BANK     :-
;REGISTER BANK   :-
;USED HARDWARE   :-
;INPUT           :-
;OUTPUT          :-
;DESTROY         :XA
;CALLS           :-
;*****************************************************************

        PUBLIC  TABLE1
;*****************************************************************
CTSEG   CSEG    PAGE
        DB      1H      ;KEY0
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      0H
        DB      1H      ;KEY1
        DB      1H
        DB      0H
        DB      0H
        DB      1H
        DB      0H
        DB      0H
        DB      0H
        DB      1H      ;KEY2
        DB      0H
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      0H
        DB      1H
        DB      1H      ;KEY3
        DB      1H
        DB      1H
        DB      0H
        DB      1H
        DB      1H
```

**

SOURCE STATEMENT

```
        DB      0H
        DB      1H
        DB      1H      ;KEY4
        DB      1H
        DB      0H
        DB      0H
        DB      1H
        DB      0H
        DB      1H
        DB      1H
        DB      1H      ;KEY5
        DB      1H
        DB      1H
        DB      0H
        DB      0H
        DB      1H
        DB      1H
        DB      1H
        DB      1H      ;KEY6
        DB      1H
        DB      1H
        DB      1H
        DB      0H
        DB      1H
        DB      1H
        DB      1H
        DB      1H      ;KEY7
        DB      1H
        DB      0H
        DB      0H
        DB      1H
        DB      1H
        DB      0H
        DB      0H
        DB      1H      ;KEY8
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H      ;KEY9
        DB      1H
        DB      1H
        DB      0H
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H      ;KEYA
        DB      1H
```

**

SOURCE STATEMENT

```
        DB      OH
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      1H      ;KEYB
        DB      1H
        DB      1H
        DB      1H
        DB      OH
        DB      OH
        DB      1H
        DB      1H
        DB      1H      ;KEYC
        DB      OH
        DB      1H
        DB      1H
        DB      OH
        DB      1H
        DB      1H
        DB      OH
        DB      1H      ;KEYD
        DB      1H
        DB      1H
        DB      1H
        DB      1H
        DB      OH
        DB      OH
        DB      1H
        DB      1H      ;KEYE
        DB      OH
        DB      1H
        DB      1H
        DB      OH
        DB      1H
        DB      1H
        DB      1H
        DB      1H      ;KEYF
        DB      OH
        DB      OH
        DB      1H
        DB      OH
        DB      1H
        DB      1H
        DB      1H
;**********MAKE SURE THAT MOVT INSTR IS IN SAME PAGE AS TABLE*********
TABLE1: MOVT    XA,@PCXA        ;LOAD TABLE DATA
        RET
        END
```

Interfacing the CMOS 32-Bit Microprocessor μPD70616 (V60™) to Standard Components

Contents:

Authors: Thomas Müller, Rolf Elter
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Microprocessors and Peripherals	Data Book
Memory Products 1987	Data Book
μPD70616 (V60)	Product Description
μPD70616 (V60)	Data Sheet
μPD71611	Data Sheet
μPD71613	Data Sheet

Related Products

μPD70616	32-bit Microprocessor	CMOS
μPD71611	CMOS Clock Generator	CMOS
μPD71613	CMOS Bus Controller	CMOS
μPD71051	Serial Control Unit	CMOS
μPD71054	Programmable Timer/Counter	CMOS
μPD71055	Parallel Interface Unit	CMOS
μPD43256	256K Bit Static CMOS RAM	CMOS

1. Introduction

The V60 is a 32-bit CMOS microprocessor with 24 address lines and a 16-bit data bus interface with on-chip integrated functions including memory management unit (MMU), V20™, V30™ emulation mode, functional redundancy monitor (FRM), etc. Beside 16-bit and 32-bit related instructions, this microprocessor supports 21 addressing modes for byte addressable data and 18 addressing modes for bit addressable data. This feature gives the designer the opportunity for easy interfacing of the V60 to standard microcomputer devices such as peripherals, RAM's, ROM's, etc., without special hardware.

The aim of this application note is to demonstrate the ease of designing a 32-bit microcomputer system, integrating the CPU μPD70616 (V60) and standard CMOS devices with 8-bit data bus structure. Scope of this document includes the interfacing between various CMOS devices ant the V60, and selecting parts from a single board computer build-up within NEC's application laboratory. On this basis, the following description is only related to the features of most interest, focussing on the general solutions used within the application.

2. Single Board Computer with V60 — Specifications and Features of the SBC-V60

The single board computer SBC-V60 was constructed around the microprocessor μPD70616 (V60) which runs up to 16 MHz clock frequency. The target system which fulfills the common requirements of several general purpose applications may as well be used for evaluation purposes.

From the hardware standpoint, the board can be equipped with up to 1.5 MByte RAM and 256 kByte ROM which are mapped at block addresses of 128 kB/512 kB boundaries. The board supports two serial I/O (RS232) interfaces which allow an operation with an user terminal and a host computer for up/download functions in parallel. To communicate with the board via an external system, any standard program is allowed (KERMIT, for example). Beside the serial interfaces, the board was designed to operate with 48 port lines, controlled by μPD71055. This feature gives the user the opportunity to install, for example, a CENTRONICS® interface. Beside these features, the board was assigned also for other state-of-the-art-functions such as interrupt control, DMA-transfer, slave processing, etc. These features, including all board related special function commands will be discussed in a seperate document, later.

Implemented on the system is a small monitor program with functions like "fill memory", "dump memory", "move memory", "go", etc. These allow the user to get experience with the SBC-V60 at a low level. The monitor program was written in assembler language for fastest execution. It resides at execution level 0, i.e. the highest possible execution level on chip. For examples, please refer to section 4.

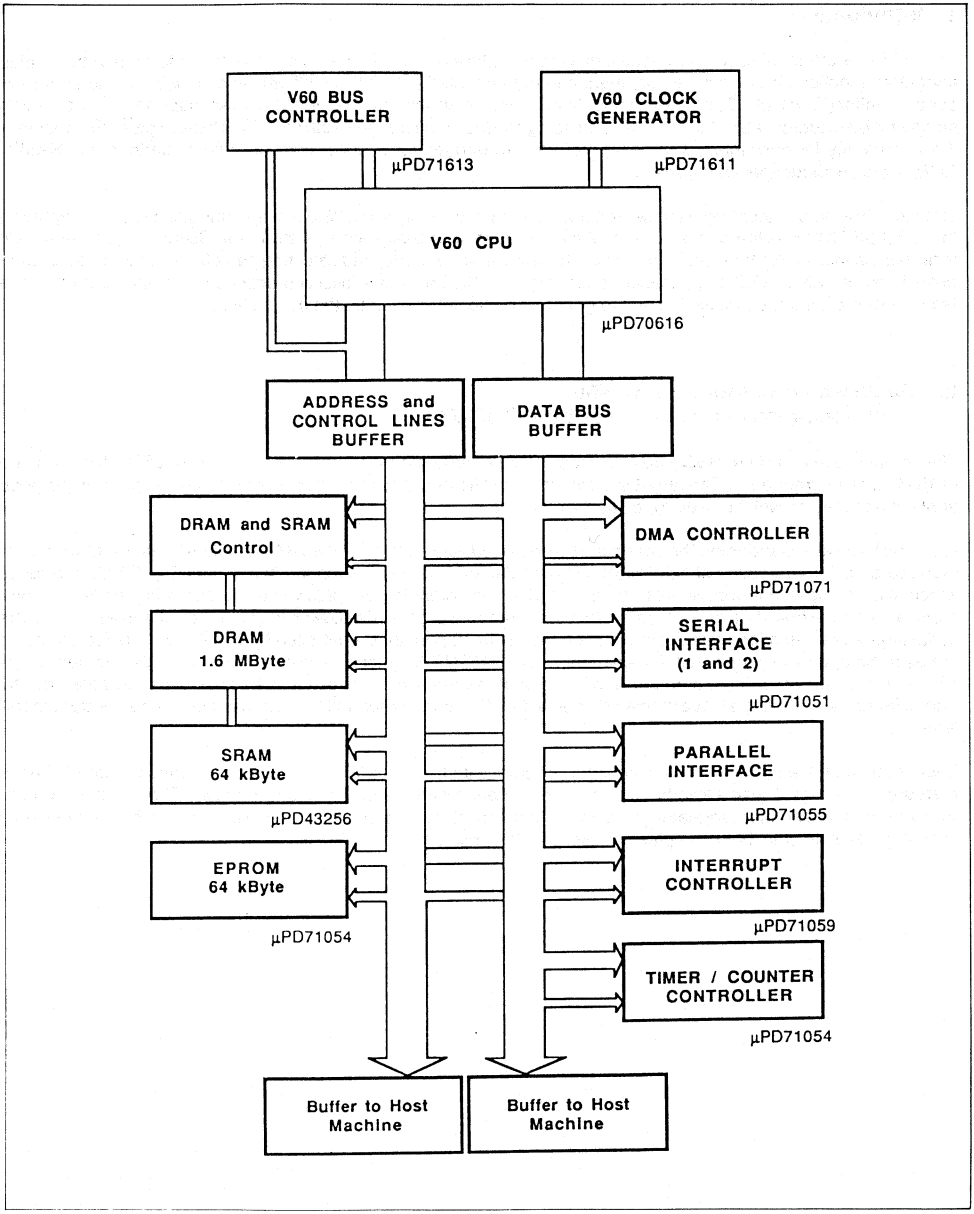


Figure 1. Block Diagram of the SBC-V60

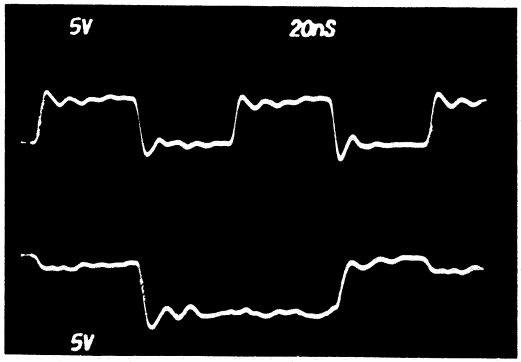
3. Block Description

3.1 CPU Interface

The V60 requires for clock and bus interface two external components; the Clock generator μPD71611 and the Bus controller μPD71613. For detailed description of these devices please refer to the V60 PRODUCT DESCRIPTION, chapter 3 and 4.

- Clock Generator

  The Clock Generator block can be driven with any crystal up to 32 MHz frequency. It outputs a clock rate at the two output pins CLK (input clock frequency devided by two) and PRCLK (devided by four). Selectable is also an external clock input pin for the system clock. The influence of the capacitors used at the crystal inputs is of less importance within a range of 10pF to 20pF. The photo below shows the output signal accuracy of the μPD71611.



Note: a) CLK Output
b) PRCLK Output

Figure 2. Generated Clock Cycles with a 22.198 MHz Crystal

- Reset Control

The reset input must be driven by a true logical level. A Schmitt-Trigger device is necessary to generate a proper reset signal.

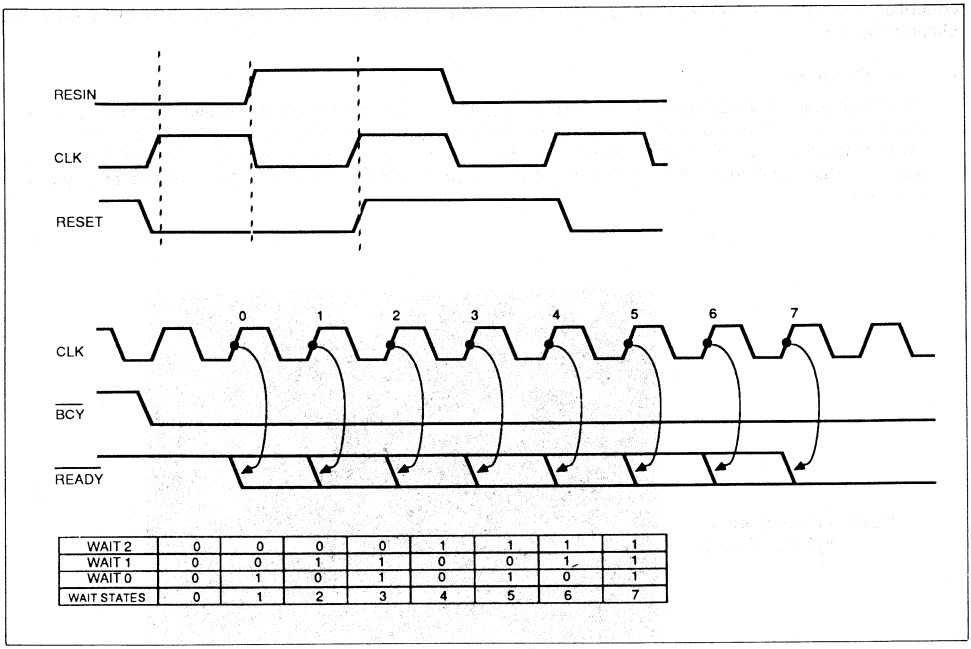
- Wait State Generator

The wait control clock can be used in two ways for generating wait states:

1. Wait states, which are appended to every BCY* signal by the generator.
2. Wait states, which are generated by external logic.

The three inputs for wait control (WAIT0. . .WAIT2) can generate 7 wait cycles at maximum. After this time the READY* output becomes active low again.

But this is not the only way to generate wait cycles for the V60. There are also two ready inputs on the μPD71611. Each one can be enabled by a separate control line. For many applications, the ARDY (Asynchronous Ready) is the most interesting input. Via this input, ready states can be used to synchronize read/write cycles of the CPU and memory, peripherals etc. The external READY* signals may occur at any time the BCY* signal goes active. The CPU tests the signal at the falling edge of the next clock cycle.



WAIT 2	0	0	0	0	1	1	1	1
WAIT 1	0	0	1	1	0	0	1	1
WAIT 0	0	1	0	1	0	1	0	1
WAIT STATES	0	1	2	3	4	5	6	7

Figure 3. Schematic Diagram of Clock States and READY* Signal Generating

- Bus Control Unit

The Bus Controller decodes the status lines of the μPD70616 and generates control signals for address and data bus driver. With the bus interface used on this board, two signals, the DBR*/W and the DBEN* are used for the direction and enable/disable control of the bus drivers 74HCT244/245 respectively. All input and output lines, except CLOCK, RESET and BFREZ of the V60 should be pulled up with resistor values between 5 and 10 kOhms. All Bus Control lines can be directly connected to the devices. The Bus Freeze (BFREZ) is active high and if not used should be pulled down.

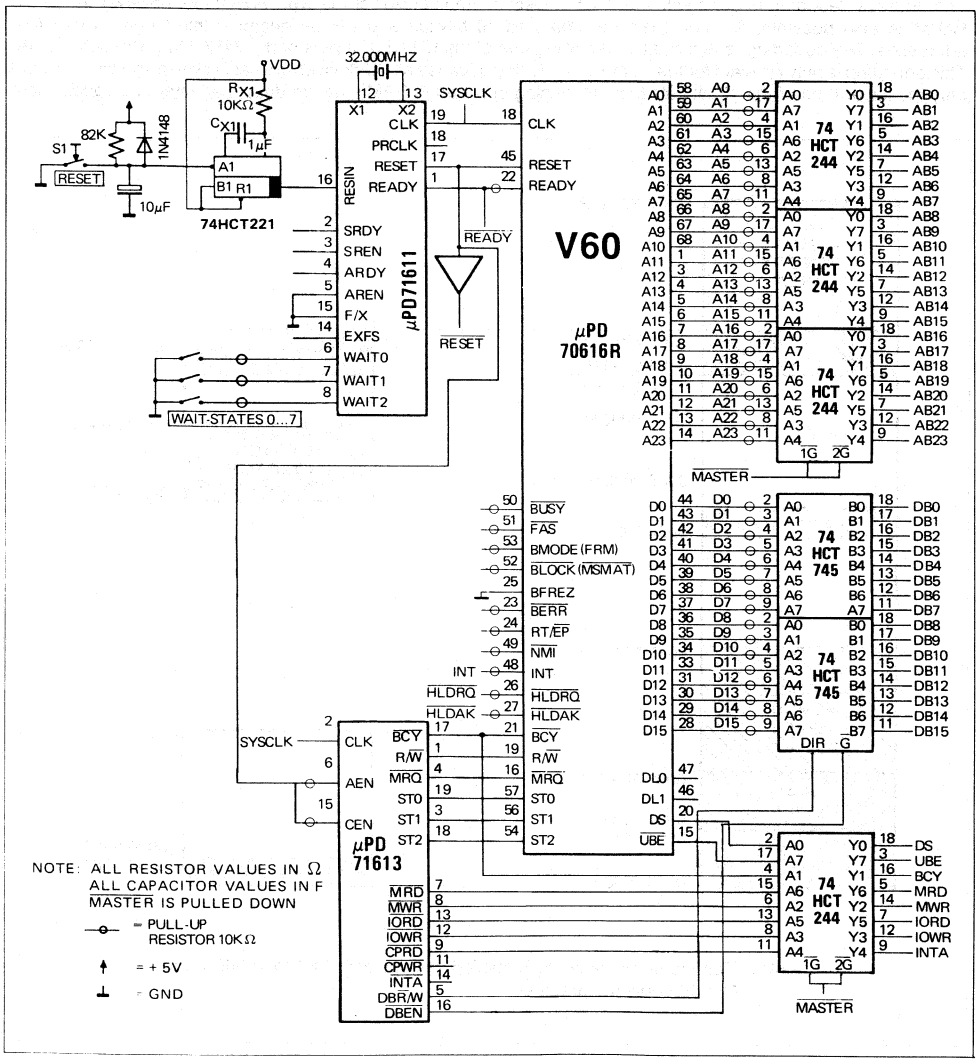


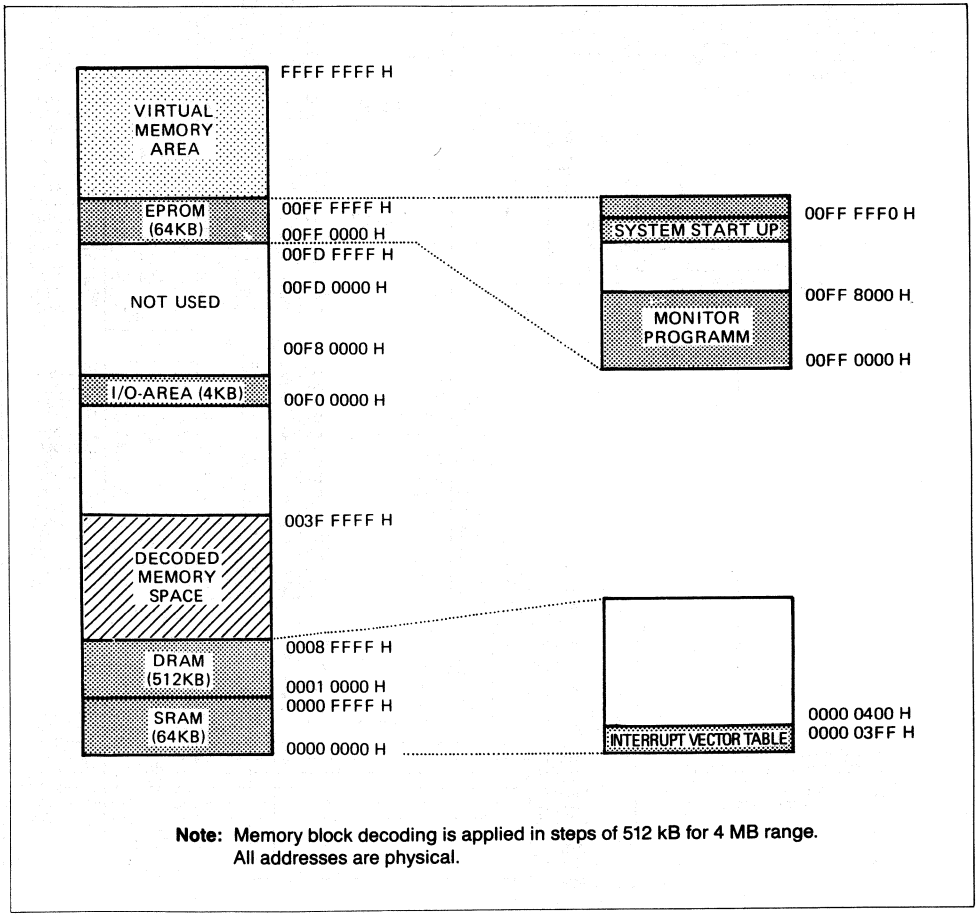
Figure 4. Schematic Circuit Diagram of the CPU Interface

3.2 Memory Interface

The memory is split into three different types:

Static RAM	64 kByte at all
Dynamic RAM	1.5 MByte
EPROM	64 kByte at all

The address decoding is done with four 1-OF-8 Decoder/Multiplexer of the type 74HCT138, however the use of PAL's® is also possible. All memories are organized 16 bitwise and will be decoded interlaced on even/odd addresses. The decoding is realized by controlling one of the ENABLE inputs of the 74HCT138 with UBE* or A0. This decoding is only relevant for the memory interfacing because all I/O devices are set to even addresses and use only the lower 8 bit of the 16 bit data bus. Interfacing of dynamic RAM's will be described later in a special note.



Note: Memory block decoding is applied in steps of 512 kB for 4 MB range.
All addresses are physical.

Figure 5. Address Table of the SBC-V60

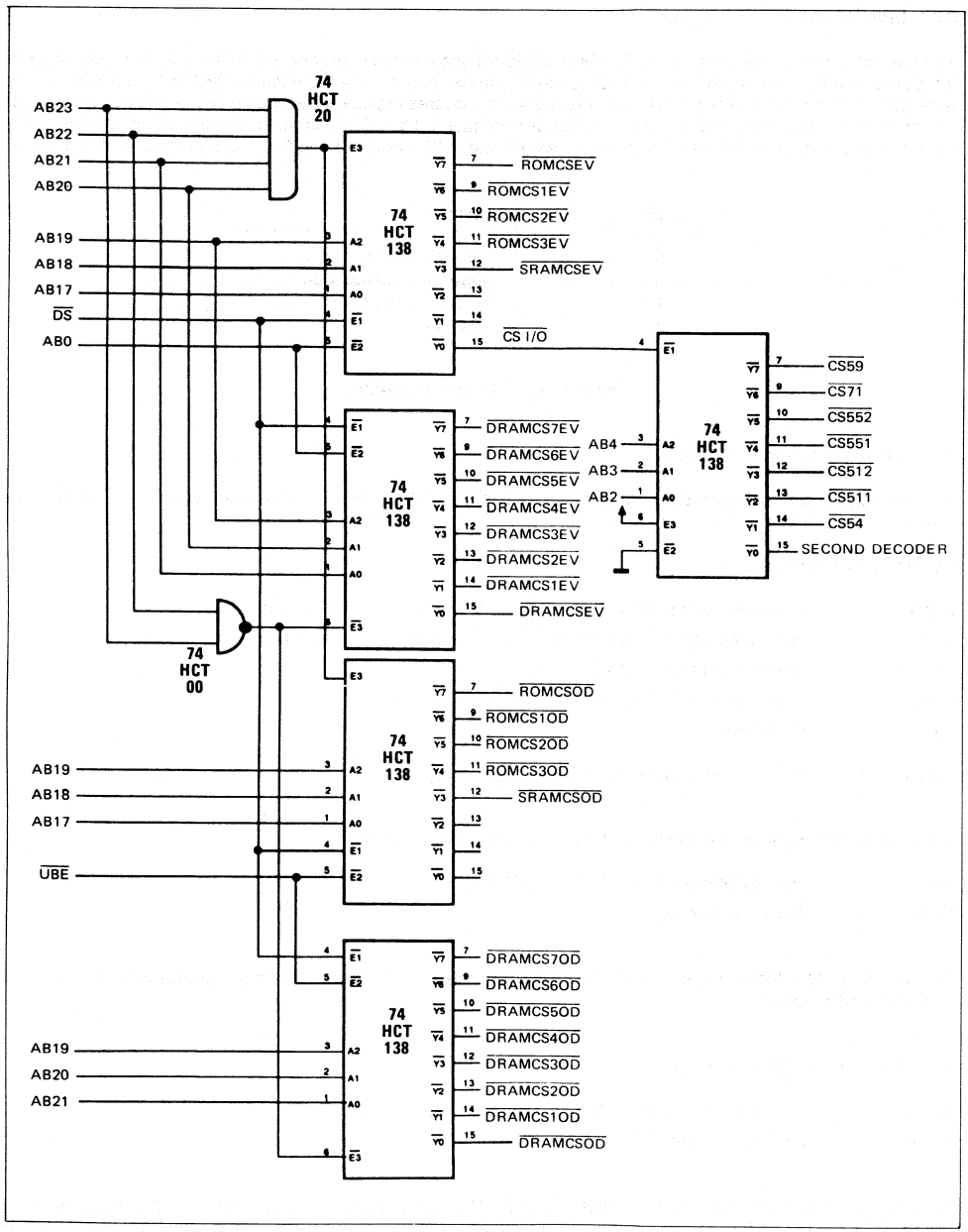


Figure 6. Schematic Diagram of Address Decoding Unit

3.2.1 EPROM and SRAM Interface

The easiest memory types usable are SRAM or EPROM because these devices will not require any complex logic for refresh control. The processor can easily access to these memories without taking READY* signals to wait for during the refresh. As shown within the chapter above, the address space is decoded on even and odd addresses, i.e. the memory space of the EPROMs nd RAMs are interlaced. The V60 controls the memory read or write operations of bytes and halfwords with two control lines; A0 and UBE* (Address line 0 and Upper Byte Enable output).

UBE*	A0	Comment
0	0	halfword memory access
0	1	upper byte memory access
1	0	lower byte memory access
1	1	reserved

Table: Even/Odd Address Access

3.2.2 Timing of Memory Access

With designing any microcomputer system, the required speed for the applicable memories is of interest. The following approximation should be helpful.

Declarations of timing data:

t_{DDEC}	delay time of the decoder from DS* active	max. 30 ns
t_{DKDS}	delay time from falling edge of T1 to DS* active	max. 40 ns
t_{CYK}	clock cycle time 62.5 ns (16 MHz)	
t_{MRD}	delay between DS* active and valid data on bus	
T_W	Wait State	

$t_{MRD} = 1/2 * T1 - t_{DKDS} + T2 + T3 + T_W [-t_{DDEC}]$

This means for t_{MRD}, without taking the buffer delay time into account:

$t_{MRD} = 1/2 * 62.5ns - 40ns + 62.5ns + 62.5ns$

$t_{MRD} = 116.5ns$ (minimal)

For any wait state, which is inserted, one clock cycle must be added to t_{MRD}. When the processor is driven in the fast mode, T3 is not valid.

Memory read with buffer delay time:

$t_{MRD} = 1/2 * 62.5ns - 40ns + 62.5ns + 62.5ns - 30ns$

$t_{MRD} = 86.25ns$ (worst case)

The decoder delay time with 30ns is very high. Typically HCT device types have a delay time of 17ns. Advantageous will be ACT, AS, or PAL® decoders, which have a typical delay time of approx. 13ns. Based on this, the following approximation holds true:

t_{MRD} = 1/2 * 62.5ns —40ns + 62.5ns +62.5ns — 13ns

t_{MRD} = 103.25ns (typical)

The write access time is not critically compared to the read timing, because the data will be valid for more than 120 nS and will be written to the memory with the rising edge of CS*.

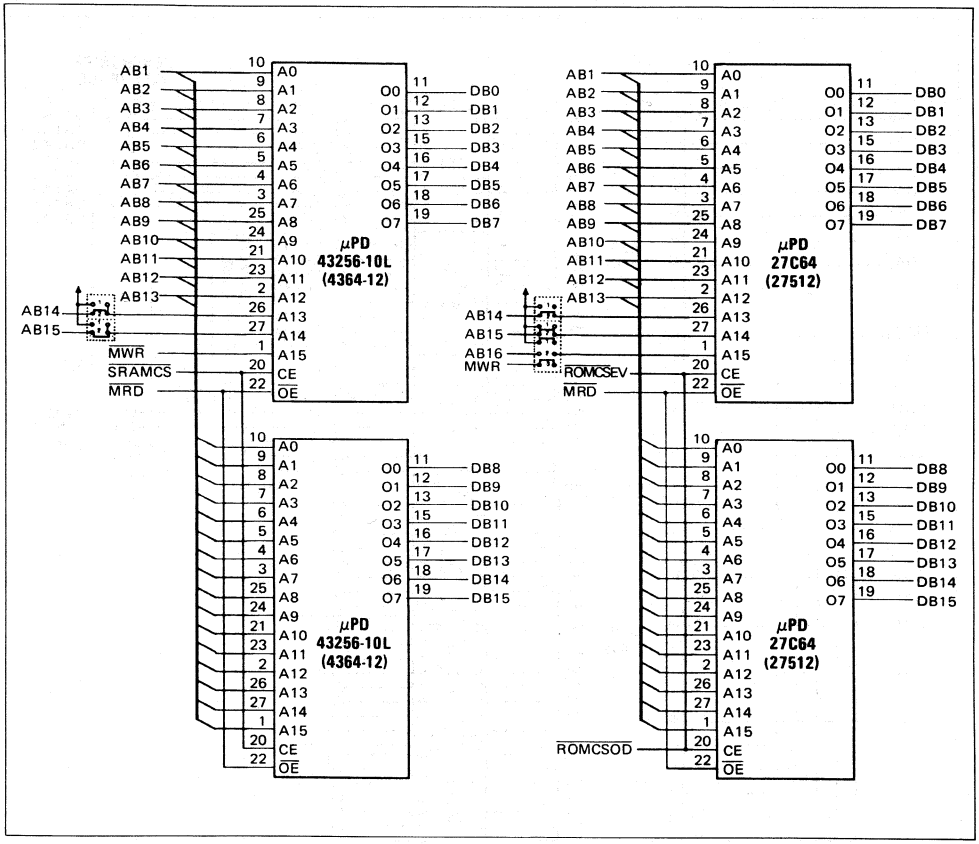


Figure 7. Schematic Diagram of RAM/EPROM Circuit

3.3 Interfacing of Peripherals

For this board only, the standard interface units were designed to demonstrate the ease of handling within 32 bit systems. The address decoding is established with one, 1-OF-8 Decoder/Multiplexer type 74HCT138. The base address is given by decoding the chip select, regardless of even or odd addresses (as address decoding was shown within chapter 2.2). Because of the feature μPD71613 Bus Controller, which supports two signals to handle the I/O operations, no other control logic is necessary. The output pins IORD* and IOWR* are connected to the Bus Controller pins or the same names.

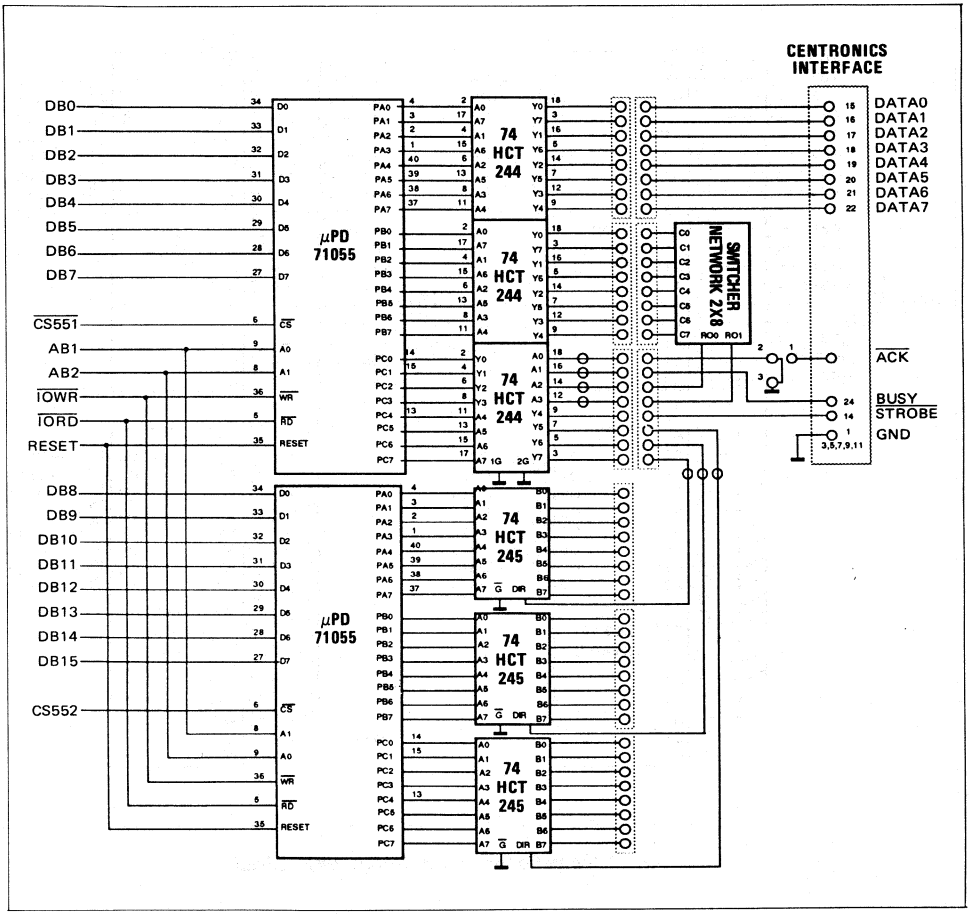


Figure 8. Schematic Diagram of the Parallel Interface

4. Initialization of the Peripherals

This chapter describes the I/O handling of the devices. It may be used as an example for programming the different peripherals and how to serve the controller. It will not be a description of the devices, but rather a short example on how to handle the software of the V60. The example below, initializes the I/O addresses and reflects them to the memory space. This program module is needed for all further programs, which use the listed devices.

Example:

```
            .globl   tcu_count0      -- global definitions of I/O addresses
            .globl   tcu_count1
            .globl   tcu_count2
            .globl   tcu_modreg      -- TCU Control Register
            .globl   sio_data
            .globl   sio_ctrl        -- SCU Control Register
            .globl   pio_data0
            .globl   pio_data1
            .globl   pio_data2
            .globl   pio_ctrl        -- PIU Control Register

            .base    absolute        -- absolute values

# Specification of the I/O base address

            .data    "io_dev"(RW) > 0xfff00000

# The entry name is "io_dev", the section can be read and written by a programm
# Real entry address is 0FFF00000H.

            .space   0x8             -- reserves 8 bytes
tcu_count0: .space   2               -- 2 byte for channel 0
tcu_count1: .space   2               --     --- " ---      1
tcu_count3: .space   2               --     --- " ---      2
tcu_modreg: .space   2               -- control register timer/counter
sio_data:   .space   2               -- location of data register
sio_ctrl:   .space   2               -- control register serial control unit
            .space   24
pio_ctrl:   .space   2               -- control register parallel interface
pio_port0:  .space   2
pio_port1:  .space   2
pio_port2:  .space   2

# end
```

4.1 Initialization of the Serial Control Unit μPD71051

The program is formatted like a subroutine in small model. It needs no parameters and will give none back to main routine.

Example:

```
            .file    "sio_init"                -- giving the file a name
            .base    absolute
#
# First define some variables (local variables)
#
            .set     soft_reset, 0x40          -- software reset SIO
            .set     ctrl_byte, 0xce           -- control word
            .set     cmd_byte, 0x37            -- command word
#
# Starting the initialization of the uPD71051
#
            .globl   _sio_init

_sio_init:
            movz.bw  #0, r0
            out.h    r0, /sio_ctrl             -- be sure to get SIO
            out.h    r0, /sio_ctrl
            out.h    r0, /sio_ctrl

            movz.bw  #soft_reset, r0           -- giving a software reset
            out.h    r0, /sio_ctrl

            movz.bw  #ctrl_byte, r0            -- send and receive, 8 bit/char
            out.h    r0, /sio_ctrl             -- no parity, 2 stop bit

            movz.bw  #cmd_byte, r0
            out.h    r0, /sio_ctrl
            rsr                                -- back to main programm

# end of subroutine
```

It is also possible to write the program without absolute address locations and use the displacement relative to the PC (program counter). The address label changes from "/sio__ctrl" to "sio__ctrl". At the beginning of the program, the ".base absolute" command must be changed to ".base pc". Every general purpose register can be specified as the base address of the program.

4.2 Initialization of the Timer Counter Unit μPD71054

In this program, the TCU is initialized in the same style as the program modules before.

```
            .file     "tcu_init"
            .base     absolute
#
# Define the variables
#
            .set      tcu_mod3, 0xb6          -- TCU mode 3
            .set      tcu_hibyte, 0           -- Set divisor(upper byte)
            .set      tcu_lobyte, 0x20        -- set divisor(lower byte)
#
# Subroutine start
#
            .globl    _tcu_init

_tcu_init:
            movz.bw   #tcu_mod3, r0           -- get ready for tcu mod 3
            out.h     r0, /tcu_modreg         -- baud rate generation

            movz.bw   #tcu_lobyte             -- low byte divisor
            out.h     r0, /tcu_count2         -- set divisor

            movz.bw   #tcu_hibyte, r0         -- high byte divisor
            out.h     r0, /tcu_count2         -- complete

            rsr                               -- back to main programm

# end of subroutine
```

In this same way, it is possible to initialize the TCU channels with other modes.

In the following, a complete initialization routine for sending a message to the SIO is described. The program demonstrates how to operate using the serial interface.

Exemple:

```
            .file    "start_up"
            .base    absolute
            .text    "all_start"(X) > 0xfffffff0

#
# This small programm starts with the reset vector
#

allstart:
            jmp      /_init_dev

            .text    "rom_start"(X) > 0xffff0000
init_dev:
            mov.w    #0x800, sp                  -- setting the stack pointer
            mov.w    #0x1000, fp                 -- setting the frame pointer
            jsr      init_tcu                    -- init the timer contol unit
            jsr      sio_init                    -- init the sio
            movea.w  /_outline, r3               -- pointer to text
            jsr      _bprints                    -- send string to sio
            movea.w  /_buffer, r3                -- set pointer to input buffer
            jsr      _bgets                      -- read terminal keys
            halt                                 -- end of test programm

_outline:
            .byte    0xd, 0xa                    -- <CR> and <Lf>
            .strz    "CONTROOLED OUTPUT TO SIO"

#
# this subroutine send a character string (r3 points to the buffer) to
# the SIO. The subroutine ends, when end of string (\0) is reached
#

_bprints:
            test.b   [r3]                        -- end of string reached ?
            je       ret_bprint                  -- EOF means 00h
_wait_sio:
            in.h     /sio_ctrl, r0               -- sio ready to send ?
            and.b    #01, r0
            cmp.b    #01, r0
            jne      _wait_sio                   -- wait for a while
            mov.b    [r3], r0                    -- get the character to send
            inc.w    r3                          -- update pointer
            movz.bw  r0, r0                      -- clear upper bytes
            out.h    r0, /sio_data               -- send data
            jr       _bprints                    -- send next char

_ret_brints:
            rsr                                  -- back to main programm
```

```
#
# This subroutine reads a character string, which is sent by the terminal
# from SIO into the defined buffer (r3 points to buffer). The subroutine
# ends by receiving the control character <CR>.
#


_bgets:     in.h      /sio_ctrl, r0          -- ready to receive ?
            and.b     #02, r0
            cmp.b     #02, r0
            jne       _bgets
            in.h      /sio_data, r0          -- read one character
            movz.bw   r0, r0                 -- clear upper bytes
            mov.b     r0, [r3]               -- save to buffer
            inc.w     r3                     -- update buffer pointer
            cmp.b     #0xd, r0               -- <CR> from terminal ?
            jne       _bgets                 -- no - next char to receive
            rsr                              -- back to main programm

# end of testfile
```

This I/O routine cannot be linked to programs, which are written in C, because the C-Compiler cc70616 uses function call's for every subroutine. Following are some examples showing how to link assembler programs in the C language programs.

Example for "getchar()", written in C:

```
char buffer[80] ;     /* maybe changed if you want */
char sio_data ;       /* get a global definition */

getchar()
{
      return(sio_data)     /* read data from address 'sio_data' */
}
```

The C-Compiler generates the assembler source code, which is shown below:

```
                .file     "getchar.c"
                .text
                .align    4
                .globl    _getchar
   _getchar:
                pushm     #.L14
                pushm     #.F14
                movs.bw   _sio_data, r0    -- *** line to be substituted ***
                dispose
                popm      #.L14
                ret       #0               --1
                .set      .L14, 0x0
                .set      .F14, 0x0
                .data
                .globl    _buffer
                .comm     _buffer, 80
                .globl    _sio_data
                .comm     _sio_data, 1
                .data
```

This source file must be converted to hardware specific software. The converted file "__getlop" shown below is substituted for the line marked with I/O instructions. Definitions of "__buffer" and "__sio__data" have to be deleted and can be defined in the include file "stdio.h".

```
_getlop:
            in.h      /sio_ctrl, r0    -- SIO ready to receive
            movz.bw   r0, r0
            and.b     #02, r0
            cmp.b     #02, r0
            jne       _getlop          -- no:-->wait a little bit
            in.h      /sio_data, r0    -- o.k. read character
            movz.bw   r0, r0           -- be sure to return only one character
```

The next example shows a simple C program to print out character strings to screen.

Example:

```
char sio_data ;       /* pseodo define of sio data register */
char buffer[80] ;     /* change lenght, if you want */

putstr(buffer_ptr)
char *buffer_ptr ;
{
   while((*buffer_ptr)!='\0')
      putchar(*buffer_ptr++) ;       /* only if character not 00h */
}
putchar(a)
char a ;
{
   sio_data = a
}
```

The generated source coce below is not changed and not commented.

```
                .file    "put.c"
                .text
                .align   4
                .globl   _putstr
_putstr:
                pushm    #.L14
                prepare  #.F14
                jr       .L17
.L18:
#       line 7, file "exput.c"
                mov.w    [ap],r0
                inc.w    [ap]
                movs.bw  [r0],[-sp]
                call     _putchar,[sp]
                add.w    #4,sp
.L17:
#       line 7, file "exput.c"
                test.b   [[ap]]
                jne      .L18
```

```
.L16:
            dispose
            popm      #.L14
            ret       #0                --0
            .set      .L14,0x0
            .set      .F14,0x0
            .text
            .align    4
            .globl    _putchar
_putchar:
            pushm     #.L20
            prepare   #.F20
#       line 12, file "exput.c"
            mov.b     [ap],_sio_data  -- *** line to be substituted ***
            dispose
            popm      #.L20
            ret       #0                --0
            .set      .L20,0x0
            .set      .F20,0x0
            .data
            .globl    _sio_data
            .comm     _sio_data,1
            .globl    _buffer
            .comm     _buffer,80
            .data
```

Now the code has to be substituted with "_putlop" shown below again at the marked line. Only the labels have been changed to readable names to provide better visibility of the function.

```
            .file     "putstr"
            .text
            .globl    _putstr
#
# ap points to the stack location, where the buffer pointer is located
#

_putstr:
            pushm     #.L13           -- save working register
            prepare   #.F13           -- enter a new stack frame
            jr        _test_char
_send_char:
            mov.w     [ap],r0         -- r0 points to current buffer address
            inc.w     [ap]            -- point to next location
            mov.b     [r0],-0x1[fp]   -- save character to stack
            movs.bw   -0x1[fp],[-sp]  -- save character location to stack
            call      _putc,[sp]      -- print one character
            add.w     #4,sp           -- correction of the stack pointer
 _test_char:
            test.b    [[ap]]          -- end of string ?
            jne       _send_char      -- no
 _ret_putstr:
            dispose                   -- back to the old stack frame
            popm      #.L13           -- get back old register
            ret       #0
            .set      .L13,0x0
            .set      .F13,0x4
```

```
#
# This function send one character to the sio
# ap points to the address, where the character is located
#
            .text
            .globl   _putc
_putc:
            pushm    #.L19          -- save again all matching register
            prepare  #.F19          -- new stack frame is not required
_putlop:
            in.h     /sio_ctrl, r0  -- SIO ready to send ?
            movz.bw  r0, r0
            and.b    #1, r0
            cmp.b    #1, r0
            jne      _putlop        -- no:--> wait a little bit
            out.b    [ap],sio_data  -- send character
            dispose
            popm     #.L19
            ret      #0             -- return with current stack location
            .set     .L19,0x0
            .set     .F19,0x0
            .data
```

4.3 Initialization of the Parallel Interface Unit μPD71055

The initialization of the PIU is very simple, therefore an output routine to the printer is described in the same subroutine.

Example:

```
            .file    "piu_out"
            .base    absolute
#
# First some Definitions
#
            .set     ctrl_cmd, 0xa8          -- control command for PIU

#
# start of printer routine
# Parameters:   r3 points to the output buffer location
# the subroutine ends, when "\0" is reached
#
            .globl   send_prn

send_prn:
            push     r1                      -- r1 is needed, so save it
            movz.bw  #ctrl_cmd, r0           -- set mode to:
            out.h    r0, /piu_ctrl           -- group0 = output
                                             -- group1 = input/output

prn_lop:
            test.b   [r3]                    -- read character from buffer
            je       prn_ret                 -- end reached
```

```
wait_prn:
            in.h      /piu_port2, r1          -- output buffer empty
            movz.bw   r1, r1                  -- clear all other positions
            mov.w     r1, r0                  -- set to r0 too
            and.b     #0x80, r1
            cmp.b     #0x80, r1
            jne       wait_prn                -- wait if output not ready
            and.b     #0x20, r0               -- can printer accept data ?
            cmp.b     #0x20, r0
            jne       wait_prn                -- if no: wait
            movz.hb   [r3], r0                -- take character from buffer
            inc.w     r3                      -- update pointer
            out.h     r0, /piu_port0          -- send character to printer
            jr        prn_lop                 -- next char to send

prn_ret:    pop r1                            -- get old value of r1
            rsr                               -- back to main programm

# end of subroutine
```

Stepper Motor Control Using a μPD78C11

Contents:

Author: E. Goldberg
NEC Electronics NATICK/USA

Persons to contact: R. Cherrington, W. Knippschild
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Products Description
μCOM87 Family

Related Products

μPD78C10/C11 8-Bit Single Chip Microcomputers CMOS
μPD78C12/C14

1. Controlling a Stepper Motor

Microcomputers have envolved into very powerful devices since their introduction approximately 10 years ago. The instruction sets have become very sophisticated and the internal architectures are commonly 16 bits. In addition they now contain many on-chip peripherals and are fabricated in CMOS. The net result of this progress is that more functionality can be achieved with less hardware.

The NEC μPD78C11 is one of the type of CMOS microcomputers which has all these features. As a result, it can be used in a wide variety of applications. One very good application is in the control of stepping motors. This application is ideal since motors generate lots of noise and CMOS has high inherent noise immunity. By using the newest generation of power MOSFETS with a 78C11, a stepping motor can be controlled with a 78C11 and 10 external components. One of the unique on-chip peripherals which make this possible is an 8-channel 8-bit A/D converter. The motor control is accomplished using the on-chip A/D converter, the timer/event counter, one of the two interval timers, the 4k ROM and the 10 external components.

In previous years, the motor control circuitry would have taken a full board of MSI and SSI IC's. Using one of the earlier microcomputers would have reduced the IC count significantly, but an external A/D converter, level shifting MOSFETS and the 10 parts listed above would still have been required. There are stepping motor controller IC's on the market, but at least two of these IC's along with 4 diodes, 6 resistors, and two capacitors would be required. Also these controller IC's cannot drive more than a 1.5 amp motor winding and they do not generate the velocity ramping functions. Additionally, input pulses to the motor controller with a specific phase relationship must be provided to step the motor and control the direction. Thus, it is likely that a microcomputer or microprocessor would be required in the design. This being the case, why not chose a micro which can do the entire job.

Figure 1 shows how the 78C11 is used for a typical motor control application. The power space and number of components external to the 78C11 is significantly reduced. This configuration consists of 4 MOSFETS (Q1—Q4), 4 protection diodes (D1—D4) and 2 sense resisters (Rs). The inductors W1—W4 are the four motor windings of the stepper motor and they are driven by MOSFETS Q1—Q4 which are controlled by outputs PA0—PA3 of the 78C11. MOSFETS Q5 & Q6, diode D5 and resister RL are not required to control the motor and will be discused later in this application note. For the present, assume Q5 is on and Q6 is off. Because outputs PA0—PA3 drive a MOSFET gate requiring virtually no current, their output voltage at the gate is 4.5 volts. This allows a power MOSFET to be directly driven by the 78C11 since the newer power MOSFETS require a Vgs of 3—4 volts. One such MOSFET is the NEC 2SK591 which can supply 15 amps at Vgs = 3.0 volts. This immediately saves four transistors which were previously required in older designs as level shifters. These shifters were located between the port outputs PA0—PA3 and their associated power MOSFETS.

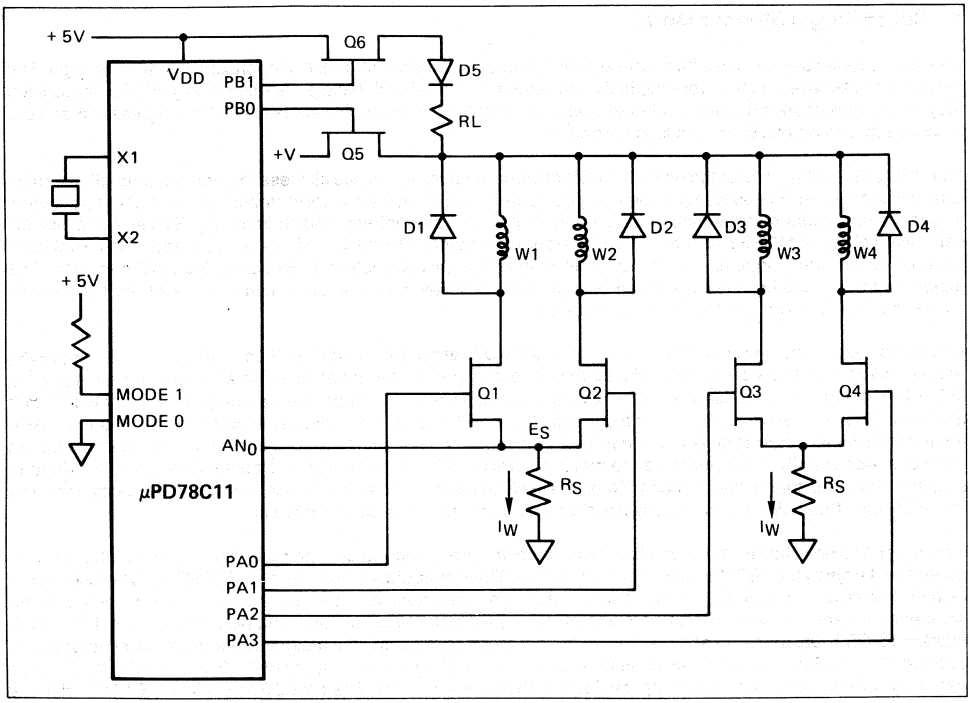


Figure 1. Schematic of μPD78C11 Controlling a Stepper Motor

In the design shown in Figure 1, the motor stepping rate and the motor winding current Iw are controlled by the 78C11. The control of the motor stepping rate (measured in motor steps/sec) is accomplished using the timer/event counter in the 78C11. The firmware in the 78C11 can be written so that the motor can be controlled to accelerate to a given velocity, slew at that velocity for a programmable distance and then decelerate to a stop at a specific position. This procedure, known as 'ramping', is required in many stepping motor applications because the inertial load on the motor shaft is high enough such that the motor cannot be driven at its maximum speed from a standing start. Also, the motor cannot stop instantly when running at maximum speed. The typical recomendation of stepping motor manufactures is to accelerate and decelerate the motor at a constant rate. The velocity profile can be determined by the following equations:

$$a = k$$

$$v = \int_0^t a\,dt = \int_0^t k\,dt = kt$$

This profile is shown in Figure 2a. The 78C11 can then be programmed to drive the motor using a velocity profile, which is approximated by a series of ramp steps as shown in Figure 2b. Virtually any velocity profile can be emulated by the 78C11 simply by tailoring the ramp step table. Multiple profiles can be easily created. When the 78C11 is used with a 15 MHz crystal, the timer/event counter's resolution is 0.8 μsec. This small resolution combined with many ramp steps can be used to create very accurate velocity profiles. These tables can then be combined with motor commands to generate motor motion with very specific profiles traversing predetermined distances. The associated firmware which couples the motor commands with the velocity profiles is stored in the 78C11's 4k byte on-chip ROM along with the user defined velocity profiles.

The velocity profile tables are constructed after a diagram such as shown in Figure 2a is drawn. The number of entries in the table depends upon how accurately the velocity approximation profile (Figure 2b) must be generated. The three ramp tables ('RMPTBL1, RMPTBL2 or RMPTBL3') shown in the listings are for 4, 5, or 6 step ramps. These approximation profiles and their idealized profiles are shown in Figures 3, 4 and 5. Each ramp step shown in Figure 2b corresponds to a table entry. A table entry consists of a divider number for the timer/event counter and the number of motor steps the motor will be driven at that rate. First the velocity in Figure 2b must be converted to motor shaft revolutions/sec. The time scale in Figure 2b shows the time for each ramp step Trs. Each table entry can be calculated the following two equations (see section IV for derivations) when the motor velocity is expressed in revolutions/second:

$$Drs = \frac{360 \times Vm \times Trs}{Dm} \quad \frac{\text{Motor steps}}{\text{Ramp Step}}$$

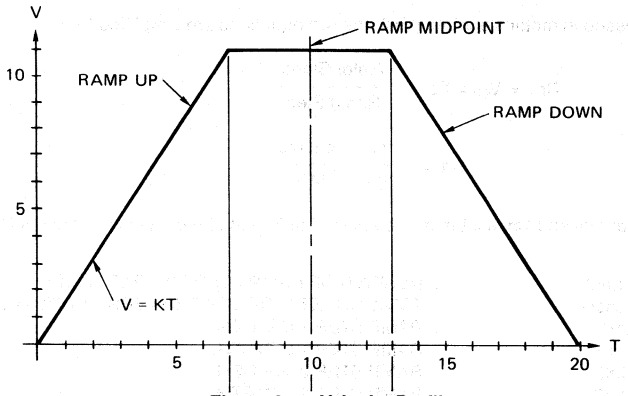


Figure 2a. Velocity Profile

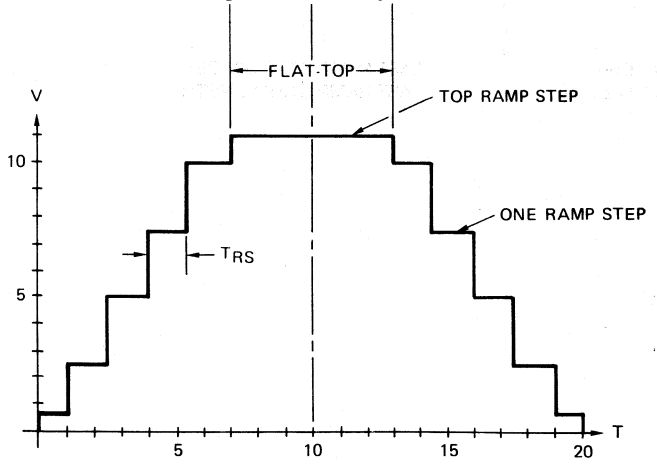


Figure 2b. Velocity Approximation Profile

Figure 2. Velocity Profiles

$$Cn = \frac{Dm \times fc}{360 \times Vm} \quad \frac{counts}{MS}$$

Where:

Drs = distance, in motor steps.
Vm = motor speed, in revolutions/second.
Trs = time of a ramp step, in seconds.
DM = motor resolution, in degrees/motor step.
Cn = velocity counter divider number.
fc = frequency of timer/event counter clock.

If the velocity is expressed in motor steps/sec (Vss), the two equations are simplified to:

$$Drs = Vss \times Trs \quad \frac{\text{Motor Steps}}{\text{Ramp Step}}$$

$$Cn = \frac{fc}{Vss} \quad \frac{counts}{MS}$$

Using the equations for Drs and Cn, a table may be built. The format of the table would be as follows:

```
VEL1:   DW   nnnnH          ; NUMBER OF RAMP STEPS IN THIS RAMP.
        DW   ssssH          ; TOTAL NUMBER OF MOTOR STEPS IN THIS RAMP.
        DW   Cn1            ; RAMP STEP 1 DIVIDER.
        DW   Drs1           ; RAMP STEP 1 LENGTH.
        DW   Cn2            ; RAMP STEP 2 DIVIDER.
        DW   Drs2           ; RAMP STEP 2 LENGTH.
                 .
                 .
                 .
        DW   Cnn            ; LAST RAMP STEP DIVIDER.
        DW   Drsn           ; LAST RAMP STEP LENGTH.
```

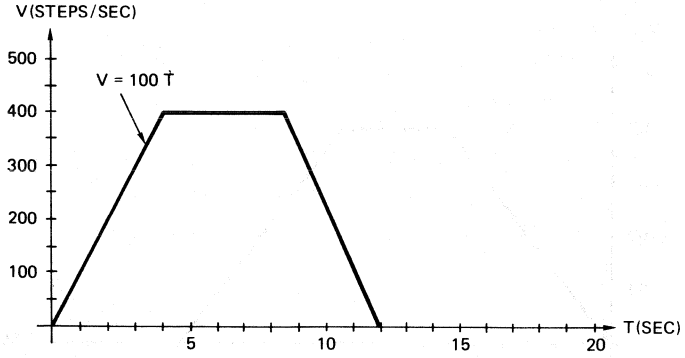


Figure 3a. Velocity Profile with v = 100 t

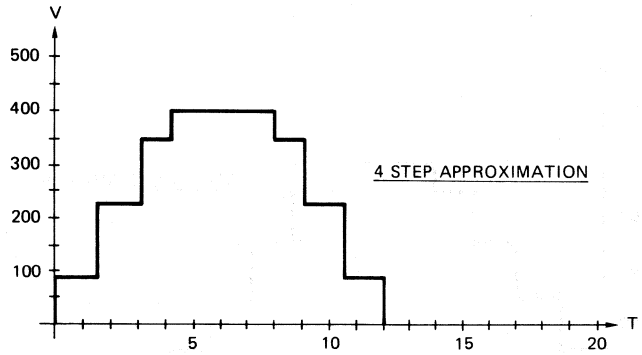


Figure 3b. Velocity Approximation Profile

Figure 3. Low Inertial Load Velocity Profile

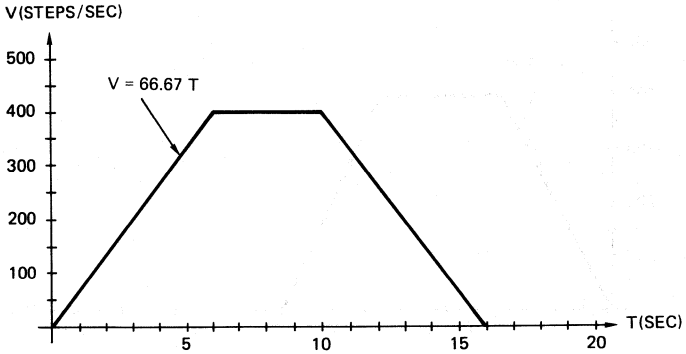


Figure 4a. Velocity Profile with v = 66.67 t

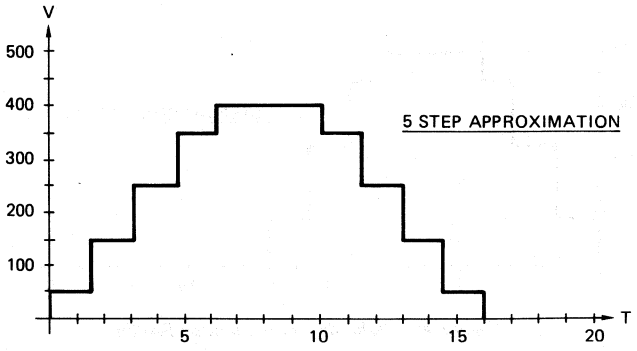


Figure 4b. Velocity Approximation Profile

Figure 4. Medium Inertial Load Velocity Profile

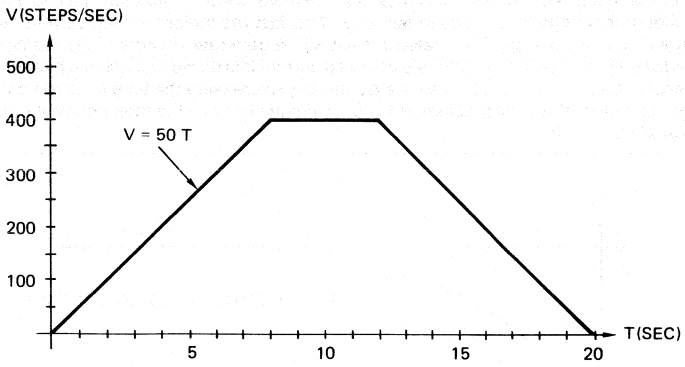


Figure 5a. Velocity Profile with v = 50 t

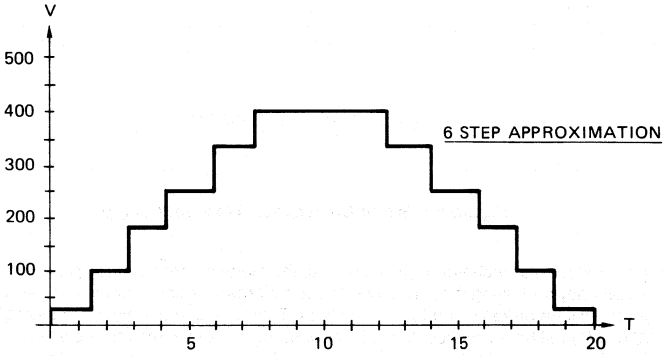


Figure 5b. Velocity Approximation Profile

Figure 5. Large Inertial Load Velocity Profile

High stepping motor rates are achieved using a motor winding voltage V which exceeds the rated motor winding voltage Vr as shown in Figure 6. The maximum rated operating current of the motor windings is Ir. The maximum stepping rate of the motor is dependent upon how fast the motor's electromagnetic field is generated. The time to generate the electromagnetic field is dependent upon how fast the motor winding current can be generated. As shown in Figure 6, at rated voltage Vr, the rated current will be generated in time t1. By raising the motor voltage V above the rated voltage Vr, the time to generate the rated current Ir is t2. Hence, using a higher voltage on the motor winding decreases the time to reach Ir by a time Δt, thereby decreasing the time to build up the field. This results in higher stepping motor rates. The problem with this technique is that if the motor current Iw is not controlled, the motor windings will burn out.

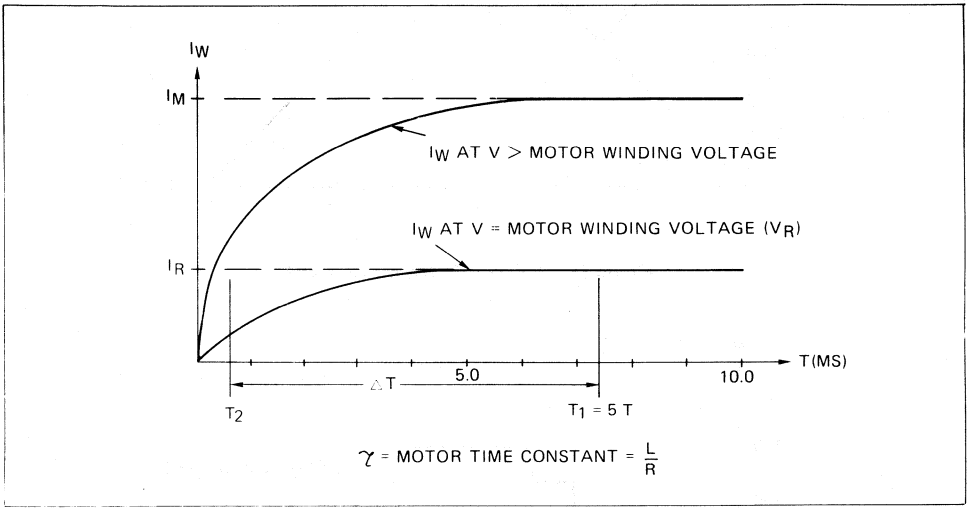


Figure 6. Motor Current vs. Winding Voltage

Current chopping is one popular technique which controls the motor current thereby protecting the motor windings. The current Iw can be chopped using the A/D converter, some firmware and outputs PA0—PA3 of the 78C11. In this application, the motor will be driven using full motor steps. This means that only one winding in each winding pair (W1 or W2 and W3 or W4) will be on. Hence, the current Iw being sensed in resistor Rs is the current in one winding. The sensed voltage Es is input to AN0 of the A/D converter. A firmware routine polls input AN0 and if the value Es exceeds a preprogrammed threshold value 'IMOT', all four MOSFETS are turned off. The input AN0 continues to be polled and when its value is less than or equal to the threshold value, the MOSFETS which were previously turned off are again turned on. In this manner the stepping motor current is chopped (turned on and off) and controlled to operate at a predetermined average value. Figure 7 shows the waveform of Ir when it is chopped. Although the current reaches a value Ip which is greater than Ir, the motor winding will not burn out if the RMS value of Iw (irms) is less than Ir. The value 'IMOT' can be chosen so that Irms $<$ Ir.

Another consideration in Figure 1 is the sense resistor Rs. The power rating of the resistor is calculated by the equation:

$$P = (Irms)^2 \times Rs$$

If Irms = 1 amp, then a 1 ohm 2 watt resistor may be used which will not require much room or power. If, however, Irms = 5 amps, then a 1 ohm resister would dissapate 25 watts and a 40 watt resistor would be required. This of

course is not desirable since it uses a lot of power, requires a lot of space and is an expensive component. A better solution to the problem would be to use a 0.1 ohm resistor. Here the power consumed is 2.5 watts and five watt resister could be used.

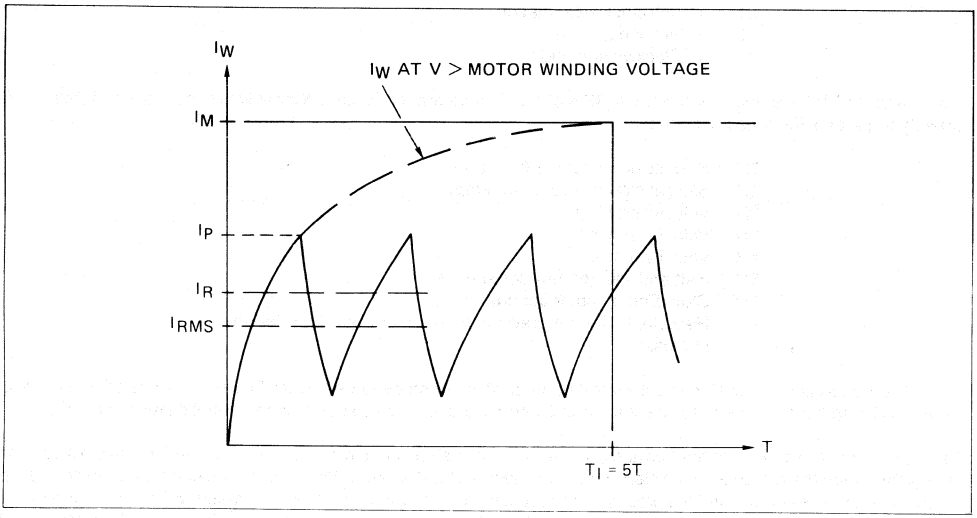


Figure 7. Waveform of I_W chopped

2. Multitasking the 78C11

If the 78C11 is constantly polling its A/D converter, its time is completely consumed and it has no time to perform other functions. However, with the addition of MOSFETS Q5 and Q6, one diode D5, and one current limiting resistor RL (see figure 1), the 78C11 could be used in a multitasking environment.

Here when the motor is idle, Q6 would be on and Q5 would be off. The motor holding current would be supplied by the 78C11's +5 volt power supply. Port B bits PB0 and PB1 would control Q5 and Q6. Resistor RL would be used to keep the motor holding current at or below Ir. With the motor current limited by RL, the 78C11 can now stop polling the A/D converter and is available to perform many other user definable functions. When the 78C11 receives a command to drive the motor, Q5 would be turned on, Q6 would be turned off, the A/D polling would start, and the motor control would begin. When a firmware flag indicates that the motor has reached its final position, Q5 is turned off, Q6 is turned on, A/D polling ceases and the 78C11 is free to take on new tasks.

3. Motor Control Firmware

The motor control firmware fetches commands which are in the motor command buffer and uses them to control the motor. The commands are put into the buffer by the main program and a jump is made to the control firmware. The firmware fetches and processes the first command, switches the motor from the holding to the active mode, and executes the command. When the command has been executed, the next motor command is fetched and executed. The sequence of fetching and executing commands continues until the STOP command is detected. When this happens, the motor is put in the hold mode and the CPU jumps from the motor control firmware to the main program where other tasks are performed.

Each motor command consists of two words. Nibble N0 bits 3 and 2 of the first word specify the motor direction as follows:

00 = Stop motor rotation
01 = Anti clockwise rotation
10 = Not used
11 = Clockwise rotation

Bits 1 and 0 of N0 are used for the two MSB's of the total command length. Nibble N1 of the first word (bits 4—7) specify the motor functins as follows:

0H Slew at slow speed 1 (no ramp)
1H Slew at slow speed 2 (no ramp)
2H Velocity profile 1
3H Velocity profile 2
4H Velocity profile 3
5H Halt and wait for new commands
6H Stop. End of motor commands
7H Recycle — Repeat previous commands (Used for debug)
8H—FH Not used

Word two contains the low 16-bits of the total number of motor steps to be traveled for this command. If the 18-bit command length is not adequate, the high order 8-bits of word one can be used for additional distance specification.

The ramp firmware algorithm fetches and executes the ramp steps inthe ramp tables. It gets the first ramp step, and drives the motor the distance of the step at the rate specified for that step. When the step is complete, the next ramp step is taken from the table and the process is repeated until the top step of the ramp is reached. At the top, the motor is driven the length of the ramp flat-top (see figure 2b) at the ramp step rate specified in the table. When the end of the flat-top is reached, the firmware starts ramping down (decelerating) until the bottom of the ramp is reached. When the bottom step is completed, the ramp is finished. The slow speed 1 and 2 commands, which drive the motor at a constant speed, are handled by the firmware as one step ramps. The firmware is designed such that a motor command can specify a ramp whose length is less than the total length of the ramp in the table. When this occurs, the ramp may never reach the top ramp step. In this situation, the velocity profile looks like a pyramid.

Figure 8 shows a flowchart of the motor control firmware. The initialization (starting at 'START') performs such tasks as clearing the timer/event counter, selecting its clock source and specifying what conditions clear the counter. The V register is set to point to the base address of the working registers for the motor control routines. The voltage threshold value IMOT is set and all interrupts except the timer/event counter are masked. The motor commands are moved from an input buffer ('MOTCMD') to a working buffer ('DATBUF') by the 'LOAD' routine. All motor motion is always initiated by the 'SETGO' routine.

The firmware fetches the function nibble N1 of the first command using the RLD instruction in the 'GETCMD' routine. It decodes the nibble and jumps to the subroutine for that command using the TABLE and JB instruction. If the firmware decodes the command to be a ramp ('RAMP1, RAMP2, or RAMP3'), the velocity profiles are generated using the timer/event counter and the motor state tables. The divider value in the timer/event counter changes with each new ramp step. The firmware initializes and evaluates the ramp before the chopper is turned on. See figure 8 and the 'RAMP' routine listing. The first initialization step is to set the firmware to ramp step 0 and set the ramp step increment to +1. The routine then gets the ramp length from word 2 of the motor command. The half length of the command is calculated and saved in a working register. The motor direction is then taken from nibble N0 of word 1 and is also saved in a working register.

The first step in evaluating the ramp and ramp step (routine 'RMPSTP') is to turn off all four MOSFETS Q1—Q4. This is done to protect the motor windings from being damaged since the 78C11 cannot simultaneously evaluate the ramp and use the A/D to do the polling. A check is made to determine if the velocity profile has been completed. If it has, then the firmware will fetch the next command by jumping to 'GETCMD'. If the profile is not finished, the firmware must check to find out what part of the ramp is being emulated. A check is made to determine if the ramp top has been reached. If it has, the top step counter divider Cn is loaded into the timer/event counter and the firmware

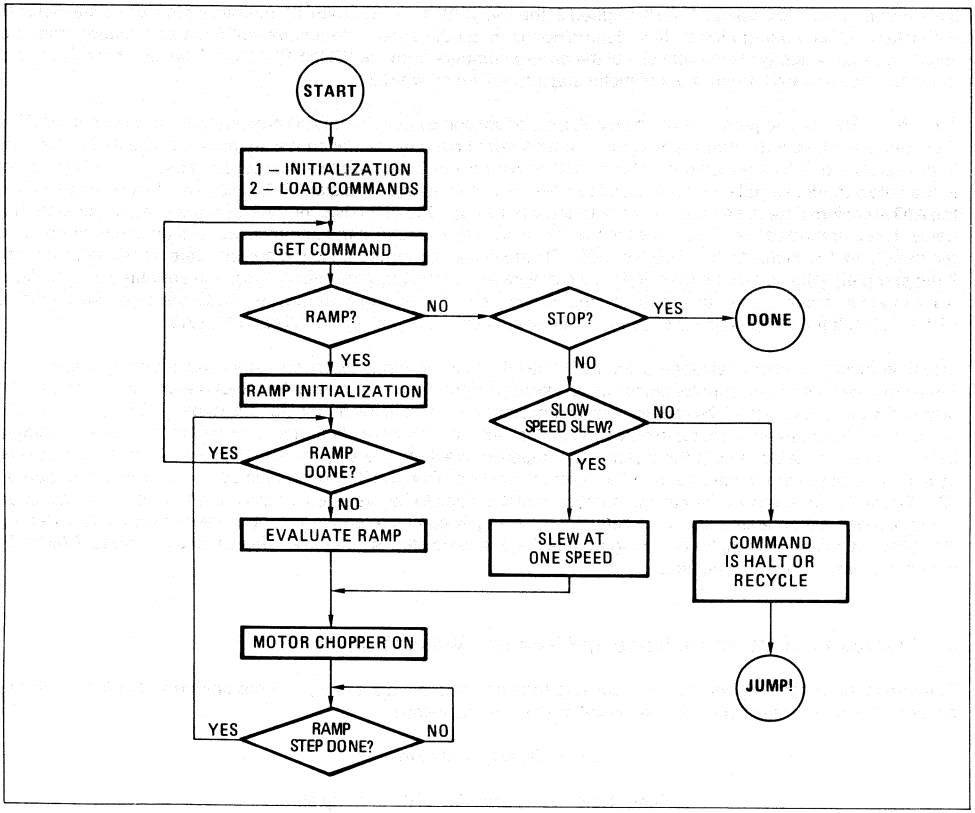


Figure 8. Flowchart of Motor Control Firmware

jumps to the 'CALCTOP' routine which calculates the length of the 'flat top' (see figure 2b). Depending upon the distance specified in word 2 of the motor command, the flat-top length may be less than, equal to, or greater than the top step length specified in the ramp table. When the calculation is finished, the firmware will drive the motor that distance at the fastest ramp step velocity in the table using the 'SETGO' routine.

If the current step is not the ramp top, a jump is made to the 'STEPOK' routine. The corresponding divider Cn is loaded into the counter and a check is made to determine if the motor is being decelerated (ramping down). If it is, the length of the current ramp step is taken from the velocity table and the motor is driven that distance using the ramp down routine ('RAMPDNY'). If the motor is being accelerated (ramping up), a check is made by the 'CHKTOP' routine to determine if the length of the current ramp step will cause the ramp to be driven to or beyond the ramp midpoint. If it does, the firmware will jump to the 'CALCTOP' routine, calculate the flat-top distance and cause the motor to be driven that distance at the current ramp step velocity. If the motor is ramping up, and the ramp step distance will not drive the motor beyond or to the ramp step midpoint, then the ramp step distance is subtracted from the remaining ramp half length ('HALFSEG') and this value is saved. This ramp step will then be executed. Each time a ramp step is completed, the firmware will return to the ramp step routine ('RMPSTP') and this process will be repeated.

If the command is slow speed 1 or slow speed 2, the 'SLOWSP1' or 'SLOWSP2' routines simply drive the motor at a constant velocity using nibble N0 to determine the motor direction. The timer/event counter is loaded from the stepping motor velocity tables with one of the divider numbers from the 'CONSPD' table. The motor is driven until the motor has traveled the number of motor steps specified by word 2.

The 'POLLAD' routine performs the motor chopping function by polling the A/D converter in the select mode. The first step is to disable the interrupts so that the timer/event counter will not cause an interrupt. The A/D is then set to the select mode by executing the MVI ANM,01H instruction. Executing this instruction causes the A/D to be set to the select mode, sample input AN0, and put the converted result in register CR0. A 50 μsec delay is used to give the A/D sufficient time to sample and convert the input value. One of the two interval timers is used to generate the delay. The sampled value is compared to the threshold value 'IMOT'. If the sampled value is less than or equal to the threshold, the motor MOSFET drivers Q1—Q4 are left on, or, turned on to the current state if they had been off. If the sampled value is greater than 'IMOT', the drivers are turned off. The interrupts are then enabled and the firmware checks to see if the current ramp step is complete. If it is, the firmware jumps to the ramp step routine ('RMPSTP'). If the ramp step is not done, the firmware continues polling the A/D at 'POLLAD'.

The timer/event counter interrupt routine 'INTX' has the task of outputting motor states and counting motor steps. Each time the timer/event counter times out, an interrupt is generated indicating the motor has completed one motor step at the current velocity. The routine gets the next motor state from the motor stepping table ('FSTP') and outputs it to port A. This causes the motor to move one motor step. The firmware continuously cycles through the four states in the motor state table. To run the motor in the opposite direction, the sequence is the same except the firmware cycles through the motor state table in the reverse direction. The 'INTX' routine also decrements working register 'CTSTP', which contains the remaining number of motor steps to be executed for the current ramp step. The value in the register is then tested and if it is zero, the ramp step is complete and register B is loaded with a 00H. Setting B to 00H indicates to the motor chopping routine that the ramp step is complete. This causes a jump to 'RMPSTP' where the ramp is again evaluated.

4. Derivation of Ramp Distance and Velocity Divider Equations

The derivations of the equations for calculating the ramp step distance (Drs) and the timer/event counter velocity divider (Cn) for velocities specified in revolutions/sec are as follows:

$$\text{Motor Velocity} = Vm, \text{ rev/sec}$$

$$\text{Ramp Time} = Trs, \text{ sec/RS} \quad ; RS = \text{ramp step}$$

The ramp step distance in motor steps Drs is calculated as follows:

$$d = v \times t = Vm \times Trs = Vm \frac{rev}{sec} \times Trs \frac{sec}{RS} = Vm \times Trs \frac{rev}{RS}$$

$$d \text{ in degrees} = Dd = Vm \times Trs \frac{rev}{RS} \times 360 \frac{deg}{rev} = 360 \times Vm \times Trs \frac{deg}{RS}$$

The stepping motor resolution = Dm deg/MS is specified in the motor manufacture's specification.

$$Drs = Dd/Dm = 360 \times Vm \times Trs \frac{deg}{RS} \times \frac{1}{Dm \text{ deg/Ms}}$$

$$Drs = \frac{360 \times Vm \times Trs}{Dm} \frac{\text{Motor steps}}{\text{Ramp step}}$$

The ramp step divider Cn which is put into the timer/event counter, is calculated by first calculating the number of MS/rev (MSr):

$$MSr = \frac{1}{Dm \text{ deg/MS}} \times 360 \frac{\text{deg}}{\text{rev}} = \frac{360}{Dm} \frac{\text{MS}}{\text{rev}}$$

The velocity in motor steps/sec Vss is:

$$Vss = MSr \times Vm = \frac{360}{Dm} \frac{\text{MS}}{\text{rev}} \times Vm \frac{\text{rev}}{\text{sec}} = \frac{360 \times Vm}{Dm} \frac{\text{MS}}{\text{sec}}$$

The time for each motor step Tms is:

$$Tms = \frac{1}{Vss} = \frac{Dm}{360 \times Vm} \frac{\text{sec}}{\text{MS}}$$

The timer/event counter input clock fc will increment the counter by one count per input clock. Hence the counter divider Cn for a motor step is:

$$Cn = Tms \times fc = \frac{Dm}{360 \times Vm} \frac{\text{sec}}{\text{MS}} \times fc \frac{\text{counts}}{\text{sec}}$$

$$Cn = \frac{Dm \times fc}{360 \times Vm} \frac{\text{counts}}{\text{MS}}$$

```
UCOM-87 FAMILY ASSEMBLER V2.0

SOURCE FILE: MOTCTL.ASM
OBJECT FILE: MOTCTL.REL
COMMAND    : MOTCTL.ASM XR DA(4/1/87)

      ASSEMBLE LIST

  STNO ADRS R OBJECT       M      SOURCE STATEMENT

     1
     2
     3
     4                          $TITLE(STEPPING MOTOR CONTROL USING A uPD78C11)
     5                          ;***
     6                          ;       ************************************************************************************
     7                          ;       *                FIRMWARE TO CONTROL A STEPPING MOTOR USING A uPD78C11             *
     8                          ;       *                                                                                  *
     9                          ;       *       THE MOTOR WILL BE CONTROLLED USING FOUR POWER MOSFETS AND TWO 1.0 OHM SENSE *
    10                          ;       *       RESISTORS. EACH SENSE RESISTOR SENSES THE CURRENT IN ONE WINDING OF A WINDING *
    11                          ;       *       PAIR. THE SENSED VOLTAGE(Es) OF WINDING PAIR W1 & W2 IS THEN SAMPLED BY THE ANO *
    12                          ;       *       INPUT OF THE 78C11's A/D CONVERTER. THE 78C11's FIRMWARE USES THE SAMPLED *
    13                          ;       *       VALUE TO CONTROL THE MOTOR  COIL CURRENT CURRENT. THE STEPPING OF THE MOTOR, *
    14                          ;       *       AND THE MOTOR SPEED ARE CONTROLLED BY THE 78C11.                          *
    15                          ;       *                                                                                 *
    16                          ;       *       THE A/D CONVERTER HAS A REFERENCE OF 5 VOLTS AND A RESOLUTION OF OFFH(256). *
    17                          ;       *       THE MOTOR BEING USED IS A IMC/HANSEN SIZE 23 WITH A RESOLUTION OF 7.5 DEG/STEP *
    18                          ;       *       (48 STEPS/REV). THE MOTOR MAXIMUM OPERATING CURRENT IS 0.7 AMPS. THE A/D  *
    19                          ;       *       THRESHOLD  VOLTAGE Es IS:                                                 *
    20                          ;       *                                                                                 *
    21                          ;       *                          Es = 1.0 ohms x 0.7 amps = 0.7 volts.                  *
    22                          ;       *                                                                                 *
    23                          ;       *       THE VALUE FROM THE A/D CONVERTER WHICH CORRESPONDS TO Es IS 'IMOT' AND IS  *
    24                          ;       *       CALCULATED AS FOLLOWS:                                                    *
    25                          ;       *                                  0.7                                            *
    26                          ;       *                          IMOT =  --- x 256 = 35.84 = 24H                        *
    27                          ;       *                                  5.0                                            *
    28                          ;       *                                                                                 *
    29                          ;       ************************************************************************************
    30                          ;
    31                          ;
    32                          ;
    33                          ;                       NEC  Electronics Inc. thanks  John  Anderson
    34                          ;                       of IMC/HANSEN for his technical consultation
    35                          ;                       on stepping motors for this application note.
    36                          ;
    37                          $EJECT
    38                          ;           PUBLIC
    39                          ;           ENTRY
    40                          ;
    41                          ;***                    ******************************************
    42                          ;***                    *       INTERRUPT VECTORS                *
    43                          ;***                    ******************************************
    44
    45
    46 ----                     INTE1   CSEG    AT 0018H        ;INTERRUPT FROM TIMER/EVC      - INTE1
```

```
 47 0018 R 54E201                       JMP     INTX            ;TIMER SERVICE ROUTINE
 48                                     ;
 49                                     ;
 50                                     ;***                    ************************************************
 51                                     ;***                    *              MOTOR CONTROL MAIN PROGRAM      *
 52                                     ;***                    *   INITIALIZATION ROUTINES ('SETUP' & 'ZRAM')  *
 53                                     ;***                    ************************************************
 54
 55
 56 ----                                MOTR7811  CSEG    PAGE          ;MAIN CONTROL SEGMENT.
 57 0000   BA                           SETUP:  DI                      ;INITIAL ENTRY POINT.
 58 0001   6909                                 MVI     A,09H           ;SET PORT D TO OUTPUT,PORT F FOR
 59 0003   4D00                                 MOV     MM,A            ;       PORT MODE AND ENABLE INTERNAL RAM.
 60 0005   040000                               LXI     SP,0000H        ;TOP ADDRESS OF RAM PLUS ONE.
 61 0008   440000                               LXI     EA,0000H        ;                                       SET 256 BYTES
 62 000B   6A7F                                 MVI     B,7FH           ;WORD COUNT = 7FH.                      OF INTERNAL RAM
 63 000D   B4                           ZRAM:   PUSH    EA              ;LOOP TO ZERO RAM.                       TO ALL 0'S
 64 000E   52                                   DCR     B               ;COUNT WORDS:  SKIP IF BORROW.
 65 000F   FD                                   JR      ZRAM            ;MORE TO GO.
 66 0010   040000                               LXI     SP,0000H        ;SET SP TO TOP OF STACK.
 67 0013   640000                               MVI     PA,00H          ;SET PORT A BITS LOW WHICH SETS MOTOR WINDING CURRENT OFF.
 68 0016   09                                   MOV     A,EAL           ;ZERO ACCUMULATOR.
 69 0017   4D02                                 MOV     MA,A            ;SET PORT A FOR OUTPUT.
 70 0019   4D03                                 MOV     MB,A            ;SET PORT B FOR OUTPUT.
 71 001B   4D01                                 MOV     MCC,A           ;SET PORT C TO PORT MODE.
 72 001D   4D04                                 MOV     MC,A            ;SET PORT C FOR OUTPUT.
 73 001F   4DC7                                 MOV     MF,A            ;SET PORT F FOR OUTPUT.
 74                                     $EJECT
 75                                     ;***
 76                                     ;***    IN A MULTITASKING APPLICATION, THE ABOVE 'SETUP' AND 'ZRAM' ROUTINES  WOULD
 77                                     ;***    HAVE BEEN PERFORMED IN THE MAIN PROGRAM INITIALIZATION ROUTINE. ALSO, THE
 78                                     ;***    MOTOR WOULD HAVE BEEN PUT IN THE HOLD MODE BY CIRCUITRY WHICH PROTECTED THE
 79                                     ;***    MOTOR COILS AT POWER ON AND BY A ROUTINE SUCH AS:
 80                                     ;***
 81                                     ;***                    MVI     A,0H    ;CLEAR ACCUMULATOR.
 82                                     ;***                    MOV     MA,A    ;SET PORT A FOR OUTPUT.
 83                                     ;***                    MOV     MB,A    ;SET PORT B FOR OUTPUT.
 84                                     ;***                    MVI     PB,02H  ;TURN Q6 ON AND C5 OFF;THIS SUPPLIES MOTOR HOLDING CURRENT.
 85                                     ;***                    MVI     PA,0AH  ;SET MOTOR STATE TO 'STATE 0'.
 86                                     ;***
 87                                     ;***
 88                                     ;***    IN A MULTITASKING APPLICATION OF THE 78C11,THE ENTRY POINT FOR THE MAIN
 89                                     ;***    PROGRAM TO THE MOTOR CONTROL PORTION WOULD BEGIN HERE.
 90                                     ;***
 91 0021   BA                           START:  DI                      ;INTERRUPTS OFF.
 92 0022   6900                                 MVI     A,0H            ;CLEAR ACCUMULATOR.
 93 0024   4DCC                                 MOV     ETMM,A          ;CLEAR TIMER/EVENT COUNTER.
 94 0026   44FFFF                               LXI     EA,0FFFFH       ;SET EA = FFFFH.
 95 0029   09                                   MOV     A,EAL           ;
 96 002A   48D2                                 DMOV    ETM0,EA         ;SET ETM0 & ETM1 TO FFFFH (= IDLE SPEED).
 97 002C   4903                                 DMOV    ETM1,EA ;
 98 002E   4DCD                                 MOV     TMM,A           ;DISABLE INTERVAL TIMER.
 99 0030   6BFF                                 MVI     V,0FFH          ;POINT TO BASE OF RAM.
100 0032   695C                                 MVI     A,5CH           ;CONTROL WORD FOR TIMER/EVENT COUNTER SET TO:
101 0034   4DCC                                 MOV     ETMM,A          ;       012 INPUT CLOCK,CLEAR ECNT WHEN = TO ETM1,
102 0036   648300                               MVI     EOM,00H         ;               AND DISABLE CO0 & CO1 OUTPUTS.
103 0039   10                                   EXA                     ;STORE EA=0FFFFH FOR INTERRUPT SERVICE ROUTINES IN EA'.
104 003A   240000                             LXI     D,0000H         ;ZERO MOTOR STEP TABLE POINTER REG D'E'.
```

```
105 003D   340000                    LXI     H,0000H         ;ZERO MOTION CONTROL BYTE (REG L').
106 0040   11                        EXX                     ;STORE FOR INTERRUPT SERVICE ROUTINES IN D',E',H' AND L'.
107 0041   68FF                      MVI     V,0FFH          ;SET V REGISTER TO BEGINING OF INTERNAL RAM
108 0043   4866                      SKNIT   FE1             ;RESET FLAG FOR TIMER/EVENT COUNTER 1.
109 0045   00                        NOP                     ;NEED NOP FOR SKNIT INSTRUCTION.
110 0046   6407BF                    MVI     MKL,0BFH        ;UNMASK INTE1 AND
111 0049   6406FF                    MVI     MKH,0FFH        ;       MASK ALL OTHER INTERRUPTS.
112 004C   710924                    MVIW    IMOT,24H        ;SET MOTOR CURRENT TO 0.7 AMPS.
113                          $EJECT
114
115
116                          ;***
117                          ;***    ********************************************************************************
118                          ;***    *                       MAIN MOTOR CONTROL PROGRAM ('LOAD' ROUTINE)            *
119                          ;***    *                                                                              *
120                          ;***    *       THE 'LOAD' ROUTINE MOVES COMMANDS FROM LOCATION 'MOTCMD' TO 'DATBUF'.  *
121                          ;***    ********************************************************************************
122 004F   345004            LOAD:   LXI     H,MOTCMD        ;LOAD REG HL WITH STARTING ADDRESS OF 'MOTCMD'.
123 0052   2416FF                    LXI     D,DATBUF        ;LOAD REG DE WITH STARTING ADDRESS OF DATBUF.
124 0055   6817                      MVI     C,17H           ;SET REG C WITH A COUNT OF 24 TO TRANSFER 24 BYTES.
125 0057   31                        BLOCK                   ;TRANSFER 'TESTDATA' TO DATBUF.
126
127
128                          ;***    ********************************************************************************
129                          ;***    *               MAIN MOTOR CONTROL PROGRAM ('GETCMD' ROUTINE )                 *
130                          ;***    *                                                                              *
131                          ;***    *       ROUTINE 'GETCMD' GETS AND DECODES THE NEXT MOTOR COMMAND FUNCTION.     *
132                          ;***    *       WHEN ROUTINE 'GETCMD' IS EXITED BY THE JB INSTRUCTION,THE REGISTERS    *
133                          ;***    *       CONTAIN:                                                               *
134                          ;***    *               DE:POINTS TO CURRENT DATA BUFFER LOCATION ('DABUFPT').         *
135                          ;***    *               HL:POINTS TO TEMP WHICH CONTAINS BYTE 1 OF THE MOTOR COMMAND.  *
136                          ;***    *               EA:CONTAINS WORD 1 OF THE CURRENT MOTOR COMMAND.               *
137                          ;***    ********************************************************************************
138                          ;***
139 0058   2416FF            RDATBU: LXI     D,DATBUF        ;REG DE POINTS TO ORIGIN OF DATA BUFFER.
140 005B   702E0CFF                  SDED    DABUFPT         ;ADDRESS OF CURRENT COMMAND PUT IN
141                                                          ;       DATA BUFFER ADDRESS POINTER ('DABUFPT').
142 005F   702F0CFF          GETCMD: LDED    DABUFPT         ;CURRENT DATA BUFFER ADDRESS TO REG DE.
143 0063   340AFF                    LXI     H,TEMP          ;REG HL POINTS TO TEMP.
144 0066   4884                      LDEAX   D++             ;WORD 1 INTO REG EA. REG DE INCREMENTED BY 2.
145                                                          ;               REG DE POINTS TO WORD 2 OF COMMAND.
146 0068   4893                      STEAX   H               ;WORD 1 PLACED IN TEMP.
147 006A   6091                      XRA     A,A             ;CLEAR ACCUMULATOR.
148 006C   4838                      RLD                     ;FUNCTION NIBBLE N1 TO ACCUMULATOR.
149 006E   4825                      SLL     A               ;MULTIPLY ACCUMULATOR BY 2 BECAUSE WORD LENGTHS USED IN TABLE.
150 0070   48A8                      TABLE                   ;VECTOR TO COMMAND ROUTINE:(C)<--(PC+3+A),(B)<--(PC+3+A+1).
151 0072   21                        JB                      ;JUMP TO LOCATION SPECIFIED BY REG BC.
152 0073 R 8800                      DW      SLOWSP1         ;00H: DRIVE AT SLOW SPEED 1(NO RAMP).
153 0075 R 8300                      DW      SLOWSP2         ;01H: DRIVE AT SLOW SPEED 2(NO RAMP).
154 0077 R A800                      DW      RAMP1           ;02H: VELOCITY PROFILE 1.
155 0079 R AB00                      DW      RAMP2           ;03H: VELOCITY PROFILE 2.
156 007B R AE00                      DW      RAMP3           ;04H: VELOCITY PROFILE 3.
157 007D R 8801                      DW      HALT            ;05H: HALT AND WAIT FOR NEW COMMANDS.
158 007F R D301                      DW      STOP            ;06H: STOP, END OF MOTOR COMMANDS.
159 0081 R 5800                      DW      RDATBU          ;07H: RE-CYCLE CONTINUOUSLY REPEATING PREVIOUS COMMANDS.
160                                                          ;               THIS COMMAND IS USED FOR DEBUG OF FIRMWARE.
161                                                          ;08H - 0FH ARE NOT USED.
162                          $EJECT
```

```
STNO ADRS R OBJECT      M   SOURCE STATEMENT

 163                        ;***  ********************************************************************************
 164                        ;***  *                SLOW SPEED ROUTINES ('SLOWSPD2','SLOWSPD1', & 'SLOWSPD')      *
 165                        ;***  *                                                                               *
 166                        ;***  *       THE 'SLOWSPD2', 'SLOWSPD1', AND 'SLOWSPD' ROUTINES LOAD THE             *
 167                        ;***  *       TIMER/EVENT COUNTER WITH THE PROPER DIVIDER AND PUT THE LENGTH OF        *
 168                        ;***  *       THE NEW SEGMENT INTO 'CTSTP'.                                           *
 169                        ;***  ********************************************************************************
 170                        ;***
 171 0083   701F0404        SLOWSP2:LBCD    RMPTBL1-4       ;LOAD REG BC WITH SLOW SPEED 2 COUNTER DIVIDER.
 172 0087   C4                      JR      SLOWSPD         ;
 173 0088   701F0604        SLOWSP1:LBCD    RMPTBL1-2       ;LOAD REG BC WITH SLOW SPEED 1 COUNTER DIVIDER.
 174 008C   A5              SLOWSPD:DMOV    EA,B            ;TIMER DIVIDER OF CURRENT COMMAND MOVED
 175 008D   48D3                    DMOV    ETM1,EA         ;       INTO TIMER/EVENT COUNTER 1.
 176 008F   6901                    MVI     A,01H           ;
 177 0091   630E                    STAW    RAMPT           ;SET RAMP POINTER TO STEP 1.
 178 0093   483A                    NEGA                    ;SET RAMP INCREMENT (RAMPINC) TO -1.
 179 0095   630F                    STAW    RAMPINC         ;
 180 0097 R 406701                  CALL    SEGEVAL         ;GET CURRENT MOTOR COMMAND LENGTH & FUNCTION.
 181 009A   2400FF                  LXI     D,CTSEG         ;LOAD 'TOTAL LENGTH' ADDRESS TO REG DE.
 182 009D   3403FF                  LXI     H,CTSTP         ;LOAD 'LENGTH OF STEP' ADDRESS TO REG HL.
 183 00A0   4884                    LDEAX   D++             ;TRANSFER THE 'TOTAL SEGMENT LENGTH' IN CTSEG
 184 00A2   4895                    STEAX   H++             ;       INTO THE 'CURRENT STEP (CTSTP)'. FOR SLOW
 185 00A4   2C                      LDAX    D+              ;               SPEEDS 1 & 2, THE TOTAL LENGTH AND THE
 186 00A5   30                      STAX    H+              ;                       STEPS ARE THE SAME (NO RAMP).
 187 00A6   4E80                    JRE     SETGO           ;PREPARE TO DRIVE AT SLOW SPEED.
 188                        $EJECT
 189                        ;***  ********************************************************************************
 190                        ;***  *                                  RAMP ROUTINE                                 *
 191                        ;***  *                                                                               *
 192                        ;***  *       THE 'RAMP' ROUTINE IS EXECUTED ONCE JUST BEFORE THE FIRST STEP OF       *
 193                        ;***  *       EACH RAMP. THE RAMPTBL1, 2,OR 3 ORIGINS ARE STORED IN RAMPTAB. 'RAMPT'  *
 194                        ;***  *       IS SET TO STEP 0 AND THE RAMP INCREMENT ('RAMPINC') TO +1. THE SEGMENT  *
 195                        ;***  *       LENGTH IS PUT IN 'CTSEG' AND THE MOTOR DIRECTION IS PUT IN 'MOTDIR'.    *
 196                        ;***  *       THE SEGMENT HALF-LENGTH IS CALCULATED AND PUT IN 'HALFSEG'.             *
 197                        ;***  ********************************************************************************
 198                        ;***                                                  ---------------------------------
 199 00A8   340804  RAMP1:  LXI     H,RMPTBL1       ;LOAD REG HL WITH                                |
 200 00AB   341C04  RAMP2:  LXI     H,RMPTBL2       ;       THE ORIGIN OF THE                USE OF OVERLAY INSTRUCTIONS.
 201 00AE   343404  RAMP3:  LXI     H,RMPTBL3       ;               RAMP TABLE BEING USED.    ------------|--------
 202 00B1   703E14FF RAMP:  SHLD    RAMPTAB         ;STORE RAMP TABLE ORIGIN.
 203 00B5   6091            XRA     A,A             ;CLEAR ACCUMULATOR.
 204 00B7   630E            STAW    RAMPT           ;ZERO RAMP STEP POINTER ('RAMPT').
 205 00B9   41              INR     A               ;SET 'RAMPINC' TO +1,WHERE 'RAMPINC'
 206 00BA   630F            STAW    RAMPINC         ;               IS THE INCREMENT VALUE OF'RAMPT'.
 207 00BC   340AFF          LXI     H,TEMP          ;POINT TO LOW BYTE OF WORD 1.
 208 00BF R 406701          CALL    SEGEVAL         ;GET THE CURRENT MOTOR COMMAND LENGTH AND FUNCTION.
 209 00C2   2406FF          LXI     D,HALFSEG       ;ADDRESS FOR HALF-LENGTH OF SEGMENT TO REG DE.
 210 00C5   3400FF          LXI     H,CTSEG         ;
 211 00C8   4885            LDEAX   H++             ;PUT THE LOW 16-BITS OF THE SEGMENT LENGTH INTO REG EA.
 212 00CA   28              LDAX    H               ;PUT THE HIGH 2-BITS OF THE SEGMENT LENGTH INTO REG A.
 213 00CB   4821            SLR     A               ;DIVIDE THE SEGMENT LENGTH
 214 00CD   48A0            DSLR    EA              ;     BY TWO & STORE THE HALF-
 215 00CF   4894            STEAX   D++             ;               LENGTH IN 'HALFSEG'.
 216 00D1   3A              STAX    D               ;
 217                        $EJECT
```

```
MCS-87 FAMILY ASSEMBLER V2.0    STEPPING MOTOR CONTROL USING A UPD78C11          DATE 4/1/87

 STMT ADRS R OBJECT      M    SOURCE STATEMENT

  218                          ;***
  219                          ;***
  220                          ;***          ********************************************************************************
  221                          ;***          *                        RAMP STEP ROUTINE ('RMPSTP')                          *
  222                          ;***          *                                                                              *
  223                          ;***          *       THE  'RMPSTP' ROUTINE FIRST SHUTS OFF THE CURRENT TO THE MOTOR         *
  224                          ;***          *       WINDINGS. IT THEN CHECKS THE VALUE OF 'RAMPT'. IF 'RAMPT' IS = 0,      *
  225                          ;***          *       THE RAMP IS FINISHED AND THE PROGRAM FETCHES THE NEXT MOTOR COMMAND    *
  226                          ;***          *       VIA A JUMP TO 'GETCMD'. IF THE RAMP IS NOT FINISHED,THEN THE           *
  227                          ;***          *       ROUTINE CHECKS FOR THE TOP OF THE RAMP. IF THE TOP HAS NOT BEEN        *
  228                          ;***          *       REACHED, THE PROGRAM JUMPS TO 'STEPOK'. IF THE TOP HAS BEEN REACHED,   *
  229                          ;***          *       THE NEW DIVIDER IS PUT INTO THE TIMER/EVENT COUNTER AND THE PROGRAM    *
  230                          ;***          *       JUMPS TO 'CALCTOP'.                                                    *
  231                          ;***          ********************************************************************************
  232                          ;***
  233                          ;***
  234 00D2   640000            RMPSTP: MVI     PA,0H           ;SHUT OFF MOTOR WINDINGS FOR PROTECTION.
  235 00D5   703F14FF                  LHLD    RAMPTAB         ;RAMP TABLE ORIGIN ADDRESS TO REG HL.
  236 00D9   010E                      LDAW    RAMPT           ;MOVE PREVIOUS RAMP STEP NUMBER TO REG A.
  237 00DB   74C00F                    ADDW    RAMPINC         ;INCREMENT (OR DECREMENT IF 'RAMPINC' = -1) REG A TO NEXT STEP.
  238 00DE   7700                      EQI     A,0H            ;IF THE STEP = 0,THE RAMP IS FINISHED
  239                                                          ;       AND THE NEXT INSTRUCTION IS SKIPPED.
W 240 00E0 R 54E500                  JMP     TESTTOP         ;NOT BOTTOM,GO TEST FOR TOP STEP OF RAMP.
  241 00E3   4F7A                    JRE     GETCMD          ;YES, LOOK FOR A NEW MOTOR COMMAND.
  242 00E5   70FB            TESTTOP:EQAX    H               ;AT TOP STEP OF RAMP? (IS VALUE IN REG A = TOP STEP).
  243 00E7   CC                      JR      STEPOK          ;NOT TOP STEP, SO USE THE STEP.
  244 00E8   630E                    STAW    RAMPT           ;YES, BUT SAVE POINTER ANYWAY.
  245 00EA   4825                    SLL     A               ;MULTIPLY 'RAMPT' BY 4 TO GET DISPLACEMENT
  246 00EC   4825                    SLL     A               ;                       INTO THE RAMP TABLE OF THE NEW DIVIDER.
  247 00EE   488C                    LDEAX   H+A             ;GET NEW DIVIDER AND PUT IN REG EA.
  248 00F0   48D3                    DMOV    ETM1,EA         ;LOAD TIMER /EVENT COUNTER 1 WITH DIVIDER.
  249 00F2   4E5E                    JRE     CALCTOP         ;CALCULATE LENGTH OF TOP OF SEGMENT.
  250                          $EJECT
  251                          ;***          ********************************************************************************
  252                          ;***          *                                 STEP OK ROUTINE                              *
  253                          ;***          *                                                                              *
  254                          ;***          *       THE 'STEPOK' ROUTINE SETS THE TIMER/EVENT COUNTER WITH THE NEW STEP    *
  255                          ;***          *       RATE. IT THEN CHECKS TO SEE IF THE NEW STEP LENGTH IS GREATER THAN     *
  256                          ;***          *       'HALFSEG'. IF IT IS,THE PROGRAM JUMPS TO 'CALCTOP'. IF IT IS NOT,THE   *
  257                          ;***          *       NEW STEP LENGTH IS SUBTRACTED FROM 'HALFSEG' AND THE PROGRAM CONTINUES *
  258                          ;***          *       AT 'SETGO'.                                                            *
  259                          ;***          ********************************************************************************
  260                          ;***
  261                          ;***
  262 00F4   630E            STEPOK: STAW    RAMPT           ;SAVE NEW POINTER.
  263 00F6   4825                    SLL     A               ;MULTIPLY 'RAMPT' BY 4 TO GET DISPLACEMENT
  264 00F8   4825                    SLL     A               ;                       INTO THE RAMP TABLE OF THE NEW DIVIDER.
  265 00FA   488C                    LDEAX   H+A             ;GET NEW TIMER DIVIDER.
  266 00FC   1A                      MOV     B,A             ;SAVE THE NEW STEP DISPLACEMENT POINTER.
  267 00FD   48D3                    DMOV    ETM1,EA         ;LOAD TIMER/EVENT COUNTER 1 WITH DIVIDER.
  268 00FF   42                      INR     B               ;POINT TO NEW
  269 0100   42                      INR     B               ;       STEP LENGTH IN RAMP TABLE
  270 0101   488D                    LDEAX   H+B             ;                       AND PUT IT INTO REG EA.
  271 0103   6091                    XRA     A,A             ;REG A = HIGH BYTE OF LENGTH AND = 0
```

```
272
273 0105   750FFF                EQIW    RAMPINC,0FFH    ;TEST IF RAMPING DOWN.
W 274 0108 R 540F01              JMP     CHKTOP          ;NO,GO AND CHECK THE RAMP TOP LENGTH.
275 010B   86                    DMOV    D,EA            ;IF RAMPING DOWN,PUT THE CURRENT STEP LENGTH
W 276 010C R 542201              JMP     RAMPDN          ;IN REG DE AND JUMP TO RAMP DOWN ('RAMPDN').
277 010F   3408FF          CHKTOP: LXI     H,HALFSEG+2     ;ADDRESS OF HIGH BYTE OF HALF-LENGTH TO REG HL.
278 0112 R 408701                  CALL    COMP38R         ;COMPARES REMAINING HALF- LENGTH OF
279                                                        ;       CURRENT COMMAND TO CURRENT RAMP STEP LENGTH.
280 0115   4E38                    JRE     CALCTOP         ;IF STEP GOES BEYOND MIDDLE OF RAMP (A > 8), CALCULATE TOP.
281 0117   4895                    STEAX   H++             ;A = 8,HENCE 'HALFSEG' = 2 x CURRENT STEP. STORE
282 0119   38                      STAX    H               ;       LOW 16-BITS OF CURRENT RAMP STEP IN 'HALFSEG',STORE
283 011A   4E36                    JRE     CALCTOP         ;               UPPER 8-BITS IN 'HALFSEG+2' AND JUMP TO 'CALCTOP.
284 011C   86                    DMOV    D,EA            ;NOT AT TOP YET (A < 8). MOVE THE LOW 16-BITS OF THE RAMP
285 011D   18                    MOV     C,A             ;       STEP TO REG DE AND THE HIGH BYTE OF THE RAMP STEP TO REG C.
286 011E R 40AF01                  CALL    SUB38Y          ;SUBTRACT STEP FROM HALF-LENGTH.
287 0121   08                    MOV     A,C             ;RESTORE HIGH BYTE OF LENGTH TO REG A.
288 0122   702E03FF        RAMPDN: SDED    CTSTP           ;SAVE THE
289 0126   6305                    STAW    CTSTP+2         ;  STEP LENGTH.
290                        $EJECT
291                        ;***    ************************************************************************************
292                        ;**     *                       SET GO & WAIT LOOP ROUTINES ('SETGO' & 'WAITLP')           *
293                        ;***    *                                                                                  *
294                        ;***    *       ROUTINE 'SETGO (SET GO)' STARTS THE MOTOR MOVING BY PUTTING A MOTOR        *
295                        ;***    *       DIRECTION IN REG L'. THE 'POLLAD'(POLL A/D CONVERTER) ROUTINE PERFORMS     *
296                        ;***    *       THE MOTOR CURRENT CHOPPING FUNCTION. IF THE SENSED VOLTAGE Es AT ANO       *
297                        ;***    *       IS > IMOT, THE MOTOR CURRENT IS TURNED OFF. IF Es IS < OR = TO IMOT,       *
298                        ;***    *       THE CURRENT IS TURNED ON. ROUTINE 'RSTPDN (RAMP STEP DONE)' CONTINUALLY    *
299                        ;***    *       TESTS THE MOTOR DIRECTION (BY TESTING REG L') TO DETERMINE IF THE          *
300                        ;***    *       CURRENT RAMP STEP IS COMPLETE. IF REG L' DOES NOT = 0 (STOP),THE           *
301                        ;***    *       ROUTINE CONTINUES TO LOOP. WHEN REG L' = 0,THE RAMP STEP IS DONE AND       *
302                        ;***    *       A JUMP IS MADE TO GET THE NEXT STEP AT 'RMPSTP'.                           *
303                        ;***    ************************************************************************************
304                        ;***
305 0128   8A              SETGO:  DI                      ;INTERRUPT OFF.
306 0129   50                     EXH                     ;GET REG H'L'.
307 012A   703F10FF               LHLD    MOTDIR          ;PUT 'MOTDIR' IN REG L'. THIS GIVES  A/D  SUBROUTINE
308                                                        ;                       A MOTOR DIRECTION  AND  STARTS  MOTOR MOVEMENT.
309 012E   50                     EXH                     ;NOW PUT REG HL BACK.
310 012F   8A              POLLAD: DI                      ;INTERRUPTS OFF.
311 0130   648001                 MVI     ANM,01H         ;SET A/D TO SELECT MODE,INPUT ANO.
312 0133 R 400802                 CALL    DEL50           ;DELAY 50US FOR A/D SAMPLE.
313 0136   4CE0                   MOV     A,CR0           ;READ A/D INPUT ANO.
314 0138   74A809                 GTAW    IMOT            ;SKIP NEXT IF REG A > IMOT.
W 315 013B   4E05                  JRE     MOTON           ;IF REG < OR = IMOT, JUMP TO MOTOR ON.
316 013D   640000                 MVI     PA,00H          ;SHUT OFF MOTOR WINDINGS.
W 317 0140   4E04                  JRE     RSTPDN          ;JUMP AND CHECK IF RAMP STEP DONE.
318 0142   0112              MOTON:  LDAW    MSTATE          ;TURN ON MOTOR
319 0144   4DC0                   MOV     PA,A            ;       WINDINGS.
320 0146   50              RSTPDN: EXH                     ;GET REG H'L'.
321 0147   0F                     MOV     A,L             ;PUT REG L' IN REG A.
322 0148   50                     EXH                     ;PUT REG H'L' BACK.
323 0149   AA                     EI                      ;INTERRUPTS ON.
324 014A   2700                   GTI     A,0H            ;SKIP NEXT IF A > 0 :-->NOT DONE.
W 325 014C R 540200                JMP     RMPSTP          ;IF RAMP STEP DONE,JUMP.
W 326 014F R 542F01                JMP     POLLAD          ;RAMP STEP NOT DONE,CONTINUE POLLING.
327
328                        $EJECT
```

```
 STNO ADRS R OBJECT       M     SOURCE STATEMENT

  329                           ;***  ************************************************************************************
  330                           ;***  *                        CALCULATE TOP ROUTINE ('CALCTOP')                         *
  331                           ;***  *                                                                                  *
  332                           ;***  *       THE CALCULATE FLAT TOP ROUTINE ('CALCTOP') CALCULATES THE LENGTH OF        *
  333                           ;***  *       THE FLAT TOP OF A COMMAND. AT THE END OF THE ROUTINE IT SETS 'RAMPINC'      *
  334                           ;***  *       TO -1 (OFFH) AND CAUSES A JUMP TO THE ROUTINE WHICH GIVES THE COMMAND      *
  335                           ;***  *       TO START THE MOTOR. 'CALCTOP' CALCULATES THE FLAT TOP LENGTH BY            *
  336                           ;***  *       MULTIPLYING THE 'HALFSEG' BY 2. THERE ARE TWO TYPES OF RAMPS WHICH         *
  337                           ;***  *       OCCUR AS A RESULT OF A COMMAND. THE FIRST TYPE IS LONG ENOUGHT TO          *
  338                           ;***  *       REACH THE TOP STEP  OF THE RAMP AND THE SECOND TYPE IS NOT LONG ENOUGH     *
  339                           ;***  *       TO REACH THE TOP STEP OF THE RAMP. THE TWO TYPES ARE AS SHOWN:             *
  340                           ;***  *                                                                                  *
  341                           ;***  *                                FLAT TOP = 'HALFSEG' x 2                          *
  342                           ;***  *                                                                                  *
  343                           ;***  *                                                                                  *
  344                           ;***  *                                |<------FLAT TOP-------->|                        *
  345                           ;***  *                                |                        |                        *
  346                           ;***  *                                |   STEP 4(TOP STEP)     |                        *
  347                           ;***  *                                -------------------------                        *
  348                           ;***  *                          STEP 3 |                        |                        *
  349                           ;***  *                          ---------          |           ---------                 *
  350                           ;***  *                   STEP 2 |       |<---------->|<---------->|       |             *
  351                           ;***  *                          -------  | 'HALFSEG'  | 'HALFSEG'  |       -------       *
  352                           ;***  *                 STEP 1 |          |            |                        |      *
  353                           ;***  *                          ----      |  RAMP CENTER                        ----    *
  354                           ;***  *              BOTTOM  |            |                                        |     *
  355                           ;***  *              ----------           |                                        -----  *
  356                           ;***  *                                   COMMAND REACHES TOP STEP OF RAMP                *
  357                           ;***  *                                                                                  *
  358                           ;***  *                                                                                  *
  359                           ;***  *                                   |                                              *
  360                           ;***  *                                   |                                              *
  361                           ;***  *                          'HALFSEG'---->|<--->|<--->|<----'HALFSEG'               *
  362                           ;***  *                                         |     |                                  *
  363                           ;***  *                                         | STEP 3 |                               *
  364                           ;***  *                                         -------------                            *
  365                           ;***  *                                   STEP 2 |     |     |                          *
  366                           ;***  *                                   -------  |     |     |                          *
  367                           ;***  *                                   STEP 1 |  |     |<---------->|     |             *
  368                           ;***  *                                          ----      FLAT TOP      ----            *
  369                           ;***  *                                BOTTOM     |                                |     *
  370                           ;***  *                                ----------------                          ----------*
  371                           ;***  *                                   COMMAND DOES NOT REACH TOP STEP OF RAMP         *
  372                           ;***  *                                      ('HALFSEG' < LENGTH OF STEP 3)               *
  373                           ;***  *                                                                                  *
  374                           ;***  ************************************************************************************
  375 0152   2406FF   CALCTOP:LXI     D,HALFSEG       ;WHAT'S LEFT OF HALF-LENGTH OF COMMAND.
  376 0155   4884             LDEAX    D++             ;LOW BYTES TO REG EA
  377 0157   2C               LDAX     D+              ;       AND HIGH BYTE TO REG A.
  378 0158   48A4             DSLL     EA              ;MULTIPLY
  379 015A   4835             RLL      A               ;       BY TWO.
  380 015C   2403FF           LXI      D,CTSTP         ;STORE FLAT
  381 015F   4894             STEAX    D++             ;       TOP LENGTH
```

```
382 0161   3C                        STAX    D+              ;                IN 'CTSTP'.
383 0162   710FFF                    MVIW    RAMPINC,0FFH    ;SET 'RAMPINC' = -1.
384 0165   4FC1                      JRE     SETGO           ;NOW USE IT.
385                          $EJECT
386                          ;***
387                          ;***    ********************************************************************************
388                          ;***    *                    SEGMENT EVALUATION SUBROUTINE ('SEGEVAL')                 *
389                          ;***    *                                                                              *
390                          ;***    *                                                                              *
391                          ;***    *       FOR THE CURRENT COMMAND,THE 'SEGEVAL' SUBROUTINE GETS THE              *
392                          ;***    *       DIRECTION OF THE MOTOR (CODES ARE 00H = STOP MOTOR,01H = CW ROTATION,   *
393                          ;***    *       AND FFH = -1= CCW ROTATION) AND PUTS THE VALUE IN LOCATION 'MOTDIR'     *
394                          ;***    *       (MOTOR DIRECTION).IT PUTS BITS 17 & 18 OF THE STEP LENGTH IN LOCATION   *
395                          ;***    *       'CTSEG+2' AND THE LOW 16 BITS OF LENGTH IN LOCATIONS 'CTSEG' AND        *
396                          ;***    *       'CTSEG+1'. WHEN DONE, REG DE POINTS TO THE NEXT MOTOR COMMAND. THE      *
397                          ;***    *       SUBROUTINE IS ENTERED WITH:                                            *
398                          ;***    *                                                                              *
399                          ;***    *               NIBBLE NO OF THE COMMAND WORD IS IN BITS 4-7 OF THE LOCATION    *
400                          ;***    *               POINTED TO BY REG HL. NIBBLES N2 & N3 ARE IN THE LOCATION      *
401                          ;***    *               POINTED TO BY REG HL+1. REG DE IS POINTING TO WORD 2.          *
402                          ;***    *                                                                              *
403                          ;***    ********************************************************************************
404
405 0167   6091                SEGEVAL:XRA     A,A             ;CLEAR ACCUMULATOR.
406 0169   4838                        RLD     A               ;NIBBLE NO TO ACCUMULATOR BITS 0 - 3.
407 016B   1A                          MOV     B,A             ;SAVE NIBBLE NO IN REG B.
408 016C   4821                        SLR     A               ;SHIFT RIGHT TWICE TO GET RID
409 016E   4821                        SLR     A               ;       OF BITS 0 & 1 OF NIBBLE NO.
410 0170   3702                        LTI     A,02H           ;NEGATIVE MOTION (> OR = 02H)?
411 0172   69FF                        MVI     A,0FFH          ;IF YES,MAKE IT -1.
412 0174   6310                        STAW    MOTDIR          ;STORE IN MOTOR DIRECTION FIELD (1 BYTE).
413 0176   0A                          MOV     A,B             ;BRING BACK ORIGINAL.
414 0177   0703                        ANI     A,03H           ;SELECT LOW TWO BITS.
415 0179   6302                        STAW    CTSEG+2         ;HIGH BYTE OF LENGTH COMPONENT IN CTSEG+2.
416 017B   3400FF                      LXI     H,CTSEG         ;PUT ADDRESS OF CTSEG IN REG HL.
417 017E   4884                        LDEAX   D++             ;LOAD EA WITH WORD 2 AND POINT TO NEXT MOTOR COMMAND.
418 0180   4893                        STEAX   H               ;STORE LOW 16 BITS OF LENGTH IN CTSEG.
419 0182   702E0CFF                    SDED    DABUFPT         ;SAVE THE DATA POINTER.
420 0186   B8                          RET                     ;FINI
421                          $EJECT
```

```
UCOM-87 FAMILY ASSEMBLER V2.0   STEPPING MOTOR CONTROL USING A uPD78C11          DATE 4/1/87

 STNS ADRS R OBJECT      M    SOURCE STATEMENT

  422                          ;***  ********************************************************************************
  423                          ;***  *                    COMPARE THREE BYTE SUBROUTINE ('COMP3BR')                 *
  424                          ;***  *                                                                              *
  425                          ;***  *       THIS SUBROUTINE COMPARES THE CURRENT RAMP STEP LENGTH WITH 'HALFSEG'    *
  426                          ;***  *       (THE REMAINING DISTANCE TO THE CENTER OF THE CURRENT RAMP). THE        *
  427                          ;***  *       SUBROUTINE IS ENTERED WITH REG HL POINTING TO 'HALFSEG+2',THE HIGH     *
  428                          ;***  *       BYTE OF THE CURRENT STEP LENGTH IN REG A (LENGTH = 0 FOR THIS APPLI-   *
  429                          ;***  *       CATION NOTE),AND THE LOW 16-BITS OF THE CURRENT STEP LENGTH IN REG EA. *
  430                          ;***  *       FOR THIS SUBROUTINE:                                                   *
  431                          ;***  *                                                                              *
  432                          ;***  *                    A = THE CURRENT RAMP STEP LENGTH                          *
  433                          ;***  *                    B = 'HALFSEG'                                             *
  434                          ;***  *                                                                              *
  435                          ;***  *       THE SUBROUTINE WILL RETURN TO THE PROGRAM AS FOLLOWS:                  *
  436                          ;***  *                                                                              *
  437                          ;***  *                    IF A > B:  RETURN TO PC                                   *
  438                          ;***  *                    IF A = B:  RETURN TO PC+2                                 *
  439                          ;***  *                    IF A < B:  RETURN TO PC+7                                 *
  440                          ;***  *                                                                              *
  441                          ;***  ********************************************************************************
  442                          ;***
  443 0187   6A00              COMP3BR:MVI     B,00H           ;CLEAR REG B.
  444 0189   1C                        MOV     D,A             ;SAVE HIGH BYTE OF CURRENT STEP (=00H) IN REG D.
  445 018A   7087                      SUBNBX  H-              ;TEST HIGH BYTES. SET REG HL TO POINT TO MIDDLE BYTE OF 'HALFSEG'.
  446 018C   DE                        JR      COMPBGT         ;IF A < B.
  447 018D   480C                      SK      Z               ;IF A HIGH BYTE = B HIGH BYTE (Z=1),SKIP NEXT INSTRUCTION.
  448 018F   D4                        JR      COMPAGT         ;IF A > B.
  449 0190   08                        MOV     A,EAH           ;GET MIDDLE BYTE SINCE HIGH BYTE OF A = HIGH BYTE OF B.
  450 0191   7087                      SUBNBX  H-              ;TEST MIDDLE BYTES. SET REG HL TO POINT TO LOW BYTE OF 'HALFSEG'.
  451 0193   D7                        JR      COMPBGT         ;IF A < B.
  452 0194   480C                      SK      Z               ;IF A MIDDLE BYTE = B MIDDLE BYTE,SKIP NEXT INSTRUCTION.
  453 0196   CD                        JR      COMPAGT         ;IF A > B.
  454 0197   09                        MOV     A,EAL           ;GET LOW BYTE.
  455 0198   7083                      SUBNBX  H               ;TEST LOW BYTE. REG HL POINTS TO LOW BYTE.
  456 019A   D0                        JR      COMPBGT         ;IF A < B.
  457 019B   480C                      SK      Z               ;IF Z = 0,A = B AND SKIP THE NEXT INSTRUCTION.
W 458 019D   4E05                      JRE     COMPAGT         ;A > B.
W 459 019F   4E00                      JRE     COMPEQU         ;A = B.
  460 01A1   744202            COMPEQU:ADI     B,02H           ;A = B,RETURN TO PC+2.
  461 01A4   0A                COMPAGT:MOV     A,B             ;DISPLACEMENT FOR PC IN REG A.
  462 01A5   A1                        POP     B               ;CALLER'S PC TO REG BC.
  463 01A6   6023                      ADDNC   C,A             ;ADD DISPLACEMENT (00 FOR A > B,02 FOR A = B,07 FOR A < B)
  464                                                          ;       AND SKIP NEXT INSTRUCTION IF NO CARRY.
  465 01A8   42                        INR     B               ;USE THE CARRY.
  466 01A9   0C                        MOV     A,D             ;RESTORE HIGH BYTE OF CURRENT STEP LENGTH  FOR ENTRY.
  467 01AA   21                        JB                      ;RETURN.
  468 01AB   744207            COMPBGT:ADI     B,07H           ;PC+7 IF A < B.
  469 01AE   F5                        JR      COMPAGT         ;
  470                          $EJECT
```

```
STNO ADRS R OBJECT      M   SOURCE STATEMENT

 471
 472                        ;***          ********************************************************************************
 473                        ;***          *                SUBTRACT THREE BYTE ROUTINE ('SUB3BY')                        *
 474                        ;***          *                                                                              *
 475                        ;***          *       THE SUBTRACT 3 BYTE ROUTINE ('SUB3BY') SUBTRACTS TWO 3 BYTE POSITIVE   *
 476                        ;***          *       NUMBERS (M,N):                                                         *
 477                        ;***          *                                                                              *
 478                        ;***          *               'SUB3BY' ----> M - N                                           *
 479                        ;***          *                                                                              *
 480                        ;***          *       THE TWO NUMBERS ARE:                                                   *
 481                        ;***          *                                                                              *
 482                        ;***          *               M = REMAINING LENGTH OF HALF-LENGTH ('HALFSEG')                *
 483                        ;***          *               N = NEW RAMP STEP LENGTH                                       *
 484                        ;***          *                                                                              *
 485                        ;***          *       THE SUBROUTINE IS ENTERED WITH:                                        *
 486                        ;***          *                                                                              *
 487                        ;***          *               REG HL:POINTING TO LOW BYTE OF 'HALFSEG'                       *
 488                        ;***          *               REG DE:CONTAINS LOW 16-BITS OF RAMP STEP LENGTH                *
 489                        ;***          *               REG C :CONTAINS HIGH BYTE OF RAMP STEP LENGTH                  *
 490                        ;***          ********************************************************************************
 491                        ;***
 492 01AF    4885           SU93BY: LDEAX   H++             ;LOW 2 BYTES OF M TO REG EA
 493 01B1    28                     LDAX    H               ;       AND HIGH BYTE OF M TO REG A.
 494 01B2    74E6                   DSUB    EA,D            ;SUBTRACT LOW 2 AND
 495 01B4    60F3                   SBB     A,C             ;       HIGH WITH BORROW.
 496 01B6    3F                     STAX    H-              ;STORE HIGH BYTE OF RESULT IN
 497 01B7    33                     DCX     H               ;       'HALFSEG+2' AND STORE
 498 01B8    4893                   STEAX   H               ;               THE REST IN 'HALFSEG'.
 499 01BA    88                     RET                     ;FINI
 500                        $EJECT

 501                        ;***
 502                        ;***          ********************************************************************
 503                        ;***          *                          HALT ROUTINE                            *
 504                        ;***          *                                                                  *
 505                        ;***          *       THE HALT ROUTINE SETS THE MOTOR TO THE HOLD MODE AND SETS THE   *
 506                        ;***          *       COMMAND COMPLETE BYTE TO ALL COMMANDS COMPLETE ('NEWCMD' = 0).  *
 507                        ;***          *       THE HALT ROUTINE CONTINUALLY TESTS FOR NEW COMMANDS (i.e.       *
 508                        ;***          *       WORKING REGISTER 'NEWCMD' = 1 ). WHEN NEW COMMANDS BECOME       *
 509                        ;***          *       AVAILABLE, THE ROUTINE THEN JUMPS TO 'LOAD' AND EXECUTES THE    *
 510                        ;***          *       COMMANDS.                                                       *
 511                        ;***          ********************************************************************
 512                        ;***
 513 01BB    BA             HALT:   DI                      ;INTERRUPTS OFF.
 514 01BC    6407FF                 MVI     MKL,0FFH        ;MASK ALL INTERRUPTS
 515 01BF    6406FD                 MVI     MKH,0FDH        ;       EXCEPT COMMUNICATIONS RECEIVER.
 516 01C2    640102                 MVI     PB,02H          ;TURN Q6 ON AND Q5 OFF:-->SUPPLIES MOTOR WITH A HOLDING CURRENT.
 517 01C5    0112                   LDAW    MSTATE          ;TURN ON MOTOR WINDINGS
 518 01C7    40C0                   MOV     PA,A            ;       TO THEIR LAST ACTIVE STATE.
 519 01C9    AA                     EI                      ;INTERRUPTS ON.
 520 01CA    751301         CKNEW:  EQIW    NEWCMD,01H      ;CHECK IF NEW MOTOR COMMANDS PRESENT.
W 521 01CD R 54CA01                 JMP     CKNEW           ;NO, KEEP CHECKING.
  522 01D0 R 544F00                 JMP     LOAD            ;YES, EXECUTE NEW COMMANDS.
```

```
                                ;***
                                ;***
                                ;***    ************************************************************************
                                ;***    *                          STOP ROUTINE                                *
                                ;***    *                                                                      *
                                ;***    *       THE STOP ROUTINE PUTS THE MOTOR WINDINGS INTO THE HOLD         *
                                ;***    *       MODE,STOPS THE TIMER/EVENT COUNTER AND CAUSES THE              *
                                ;***    *       FIRMWARE TO JUMP TO A SUBROUTINE (NOT IMPLEMENTED HERE)        *
                                ;***    *       WHICH WOULD ALLOW THE 78C11 TO PERFORM TASKS OTHER THAN        *
                                ;***    *       STEPPING MOTOR CONTROL.                                        *
                                ;***    ************************************************************************
                                ;***
 01D3   BA              STOP:   DI              ;INTERRUPTS OFF.
 01D4   6950                    MVI     A,050H  ;TIMER/EVENT
 01D6   4DCC                    MOV     ETMM,A  ;       COUNTER OFF.
 01D8   640102                  MVI     PB,02H  ;TURN Q6 ON AND Q5 OFF; THIS SUPPLIES MOTOR WITH HOLDING CURRENT.
 01DB   0112                    LDAW    MSTATE  ;TURN ON MOTOR WINDINGS
 01DD   4DC0                    MOV     PA,A    ;       TO THEIR LAST ACTIVE STATE.
W 01DF R 540301                JMP     STOP    ;IN AN ACTUAL APPLICATION PROGRAM, THIS INSTRUCTION
                                                ;       WOULD NOT BE HERE, AND THE JUMP TO 'DONE' WOULD BE.
                                ;***    JMP     DONE    ;SUBROUTINE 'DONE' RETURNS 78C11 FOR OTHER USER TASKS  ***
                                ;***                    ;       AND IS NOT SHOW, BUT LEFT TO THE USER TO IMPLEMENT.***
                                $EJECT


                                ;***    ************************************************************************
                                ;***    *                TIMER/EVENT COUNTER INTERRUPT ROUTINE ('INTX')        *
                                ;***    *                                                                      *
                                ;***    *       THE 'INTX' ROUTINE IS INTERRUPT DRIVEN BY THE FREE RUNNING TIMER/EVENT *
                                ;***    *       COUNTER. IT IS USED TO DRIVE THE MOTOR AND COUNT MOTOR STEPS. THE      *
                                ;***    *       ROUTINE IS ENTERED FROM THE 'WAITLP' ROUTINE. THE 'INTX' ROUTINE WILL  *
                                ;***    *       DRIVE THE MOTOR UNTIL THE STEP OF THE CURRENT COMMAND IS COMPLETED.    *
                                ;***    *       EACH TIME'INTX' IS ENTERED, IT WILL DRIVE THE MOTOR ONE MOTOR STEP.    *
                                ;***    *       THE MOTOR DIRECTION ('MOTDIR') SPECIFIES THE MOTOR TO RUN CW, CCW,OR   *
                                ;***    *       STOP. WHEN THE COMMAND STEP IS FINISHED,REG L' (MOTOR DIRECTION) IS    *
                                ;***    *       SET TO STOP. THE ROUTINE IS WRITTEN FOR FULL STEP MOTOR DRIVE,BUT CAN  *
                                ;***    *       BE CONVERTED TO HALF STEP MOTOR DRIVE AS INDICATED BY THE COMMENTS IN  *
                                ;***    *       THE ROUTINE. THE ROUTINE OPERATES WITH INTERRUPTS DISABLED AND IT      *
                                ;***    *       IS ENTERED WITH:                                               *
                                ;***    *                                                                      *
                                ;***    *                       MOTOR STATE STORED IN 'MSTATE'                 *
                                ;***    *                       MOTOR STATE TABLE POINTER STORED IN REG E'     *
                                ;***    *                       'MOTDIR' CONTROL BYTE STORED IN REG L'         *
                                ;***    ************************************************************************
                                ;***
 01E2   10              INTX:   EXA             ;SAVE REGISTERS. SWAPS REG V,A,AND EA FOR V',A',& EA'.
 01E3   11                      EXX             ;SWAPS REG B,C,D,E,H,& L FOR B',C',D',E',H',& L'.
 01E4   0D                      MOV     A,E     ;MOTOR STATE TABLE POINTER TO REG A.
 01E5   60C7                    ADD     A,L     ;ADD 'MOTDIR' TO REG A (ADDS 0,+1,OR -1).
 01E7   0703                    ANI     A,03H   ;CLEAR BITS 2-7 OF REG A.
 01E9   140004                  LXI     B,FSTP  ;FULL STEP TABLE ADDRESS TO REG BC.
 01EC   6043                    ADD     C,A     ;DEVOLOP MOTOR STEP TABLE ADDRESS:ADD STEP TABLE
                                                ;       OFFSET (REG A) TO ORIGIN OF FSTB TABLE.
 01EE   1D                      MOV     E,A     ;SAVE NEW FSTB TABLE POINTER.
 01EF   0112                    LDAW    MSTATE  ;CURRENT STATE OF MOTOR TO REG A.
 01F1   07F0                    ANI     A,0F0H  ;ERASE LOW NIBBLE.
 01F3   7099                    ORAX    B       ;NEW STATE FROM TABLE (REG A ORED WITH REG C).
```

```
578 01F5  4DC0                  MOV     PA,A            ;OUTPUT NEW MOTOR STATE TO PORT A.
579 01F7  6312                  STAW    MSTATE          ;SAVE NEW STATE.
580 01F9  3003                  DCRW    CTSTP           ;DECREMENT AND SKIP NEXT INSTRUCTION IF BORROW IS GENERATED.
581 01FB  C8                    JR      XRET            ;COUNT MOTOR
582 01FC  3004                  DCRW    CTSTP+1         ;       PULSES AND
583 01FE  C5                    JR      XRET            ;            STOP WHEN
584 01FF  3005                  DCRW    CTSTP+2         ;                 COUNT
585 0201  C2                    JR      XRET            ;                   REACHES -1.
586 0202  6F00                  MVI     L,00H           ;STOP MOTOR BY SETTING REG L' = 0.
587 0204  11            XRET:   EXX                     ;RESTORE PRIMARY
588 0205  10                    EXA                     ;            REGISTERS.
589 0206  AA                    EI                      ;INTERRUPT BACK ON
590 0207  62                    RETI                    ;            AND RETURN.
591                     $EJECT


592                     ;***
593                     ;***
594                     ;***    ********************************************************************************
595                     ;***    *                                                                              *
596                     ;***    *       THE DELAY 50 USEC ('DEL50') ROUTINE GENERATES A 50 USEC DELAY.         *
597                     ;***    *                                                                              *
598                     ;***    ********************************************************************************
599                     ;***
600                     ;***
601 0208  6485FF        DEL50:  MVI     TMM,0FFH        ;RESET AND STOP INTERVAL TIMER.
602 0208  4841                  SKIT    FTO             ;RESET FTO
603 020D  00                    NOP                     ;       FLAG.
604 020E  B0                    PUSH    V               ;SAVE REG V & A.
605 020F  6919                  MVI     A,25            ;SET TIMER
606 0211  4DDA                  MOV     TM0,A           ;       INTERVAL TO 25.
607 0213  5983                  MVI     A,83H           ;SET INTERVAL TIMER FOR 012 CLOCK,
608 0215  4DCD                  MOV     TMM,A           ;       COUNT UP,AND RESET TIMER F/F & TIMER1.
609 0217  4841          TESTTO: SKIT    FTO             ;SKIP NEXT INSTRUCTION IF TIMER TIMES OUT.1
610 0219  4FFC                  JRE     TESTTO          ;LOOP UNTIL TIMER TIMES OUT.
611 021B  6485FF                MVI     TMM,0FFH        ;RESET AND STOP INTERVAL TIMER.
612 021E  A0                    POP     V               ;GET REG V & A BACK.
613 021F  38                    RET                     ;FINISHED.
614                     ;***
615                     ;***
616                     ;***    ********************************************************************************
617                     ;***    *                       MOTOR STATE TABLES ('FSTP')                            *
618                     ;***    *                                                                              *
619                     ;***    *       THE STEPPING MOTOR FULL STEP STATE TABLES 'FSTP' ARE SHOWN BELOW.      *
620                     ;***    *       THE STATE OF EACH STEP IS IN THE LOW NIBBLE OF THE BYTE. THE STEP       *
621                     ;***    *       TABLES ARE FOR DRIVERS WHICH ENERGIZES THE WINDING WHEN THE PORT        *
622                     ;***    *       BIT IS LOW. THE ORIGIN OF THE TABLES MUST BE LOCATED ON 8-BIT           *
623                     ;***    *       BOUNDARIES AND NO SINGLE TABLE MAY CROSS A PAGE BOUNDRY.               *
624                     ;***    *                                                                              *
625                     ;***    ********************************************************************************
626                     ;***
627                     ;***
628 ----        STATETB CSEG    AT 0400H        ;
629 0400  0A            FSTP:   DB      00AH            ;MOTOR STATE IN FULL STEPS:     STATE 0.
630 0401  09                    DB      009H            ;                               STATE 1.
631 0402  05                    DB      005H            ;                               STATE 2.
632 0403  06                    DB      006H            ;                               STATE 3.
633                     $EJECT
```

```
UCOM-87 FAMILY ASSEMBLER V2.0    STEPPING MOTOR CONTROL USING A uPD78C11          DATE 4/1/87

 STNO ADRS R OBJECT      M    SOURCE STATEMENT

  534                          ;***    ********************************************************************************
  635                          ;***    *                        STEPPING MOTOR VELOCITY TABLES                        *
  636                          ;***    *                                                                              *
  637                          ;***    *       THE 'CONSPD' TABLE CONTAINS VALUES TO DRIVE A MOTOR AT A CONSTANT      *
  638                          ;***    *       SPEED OF 100 OR 200 STEPS/SEC. THE 'RMPTBL1, RMPTBL2,AND 'RMPTBL3'    *
  639                          ;***    *       TABLES WILL DRIVE THE MOTOR WITH A MULTI-STEP CONSTANT ACCELERATION    *
  640                          ;***    *       AND DECELERATION RAMP. ALL VALUES SHOWN ARE IN DECIMAL.                *
  641                          ;***    ********************************************************************************
  642                          ;***
  643                          ;***
  644 0404   8813              CONSPD: DW      5000            ;'SLOWSP2':DIVIDER = 200 STEPS/SEC = 4.166 REV/SEC.
  645 0406   1027                      DW      10000           ;'SLOWSP1':DIVIDER = 100 STEPS/SEC = 2.083 REV/SEC.
  646 0408   0400              RMPTBL1:DW      0004            ;4 STEP RAMP.
  647 040A   800C                      DW      3200            ;TOTAL RAMP LENGTH = 66.66 REV.
  648 040C   1534                      DW      13333           ;STEP 1: 75  STEPS/SEC (1.56 REV/SEC).
  649 040E   7000                      DW      112             ;        2.33 REV.
  650 0410   5C11                      DW      4444            ;STEP 2: 225 STEPS/SEC (4.69 REV/SEC).
  651 0412   5201                      DW      338             ;        7.04 REV.
  652 0414   2908                      DW      2857            ;STEP 3: 350 STEPS/SEC (7.29 REV/SEC).
  653 0416   5E01                      DW      350             ;        7.29 REV.
  654 0418   C409                      DW      2500            ;STEP 4: 400 STEPS/SEC (8.33 REV/SEC).
  655 041A   4006                      DW      1600            ;        33.33 REV.
  656 041C   0500              RMPTBL2:DW      0005            ;5 STEP RAMP.
  657 041E   A00F                      DW      4000            ;TOTAL RAMP LENGTH = 83.33 REV.
  658 0420   204E                      DW      20000           ;STEP 1: 50 STEPS/SEC (1.04 REV/SEC).
  659 0422   4B00                      DW      75              ;        1.56 REV.
  660 0424   0A1A                      DW      6666            ;STEP 2: 150 STEPS/SEC (3.13 REV/SEC).
  661 0426   E100                      DW      225             ;        4.69 REV.
  662 0428   A00F                      DW      4000            ;STEP 3: 250 STEPS/SEC (5.21 REV/SEC).
  663 042A   7701                      DW      375             ;        7.81 REV.
  664 042C   2908                      DW      2857            ;STEP 4: 350 STEPS/SEC (7.29 REV/SEC).
  665 042E   0D02                      DW      525             ;        10.94 REV.
  666 0430   C409                      DW      2500            ;STEP 5: 400 STEPS/SEC (8.33 REV/SEC).
  667 0432   4006                      DW      1600            ;        33.33 REV.
  668 0434   0600              RMPTBL3:DW      0006            ;6 STEP RAMP
  669 0436   6A12                      DW      4714            ;TOTAL RAMP LENGTH = 98.21 REV.
  670 0438   409C                      DW      40000           ;STEP 1: 25.0 STEPS/SEC (0.52 REV/SEC).
  671 043A   2500                      DW      37              ;        0.77 REV.
  672 043C   1027                      DW      10000           ;STEP 2: 100 STEPS/SEC (2.08 REV/SEC).
  673 043E   7D00                      DW      125             ;        2.60 REV.
  674 0440   5216                      DW      5714            ;STEP 3: 175 STEPS/SEC (3.65 REV/SEC).
  675 0442   0701                      DW      263             ;        5.48 REV.
  676 0444   A00F                      DW      4000            ;STEP 4: 250 STEPS/SEC (5.21 REV/SEC).
  677 0446   B501                      DW      437             ;        9.10 REV.
  678 0448   0B0B                      DW      3030            ;STEP 5: 330 STEPS/SEC (6.88 REV/SEC).
  679 044A   EE01                      DW      494             ;        10.29 REV.
  680 044C   C409                      DW      2500            ;STEP 6: 400 STEPS/SEC (8.33 REV/SEC).
  681 044E   D007                      DW      2000            ;        41.66 REV.
  682                          $EJECT
```

```
 STNO ADRS R OBJECT      M    SOURCE STATEMENT

  683                        ;***  ************************************************************************************
  684                        ;***  *                                                                                  *
  685                        ;***  *                              MOTOR COMMANDS                                      *
  686                        ;***  *                                                                                  *
  687                        ;***  *       EACH MOTOR COMMAND CONSISTS OF TWO WORDS. THE FORMAT OF THE WORDS ARE:     *
  688                        ;***  *                                                                                  *
  689                        ;***  *                          ------------------------------                          *
  690                        ;***  *                WORD 1: |  N3  |  N2  |  N1  |  N0  |                         *
  691                        ;***  *                          ------------------------------                          *
  692                        ;***  *                BITS--->15      11     7      3     0                           *
  693                        ;***  *                                                                                  *
  694                        ;***  *                NIBBLE N0:MOTOR LENGTH AND DIRECTION:                             *
  695                        ;***  *                                                                                  *
  696                        ;***  *                          BITS 1,0:UPPER 2 BITS (17 & 16) OF MOTOR STEP TRAVEL.   *
  697                        ;***  *                          BITS 3 & 2:MOTOR DIRECTION                              *
  698                        ;***  *                                                                                  *
  699                        ;***  *                                00 = STOP MOTOR                                   *
  700                        ;***  *                                01 = CCW ROTATION                                 *
  701                        ;***  *                                10 = NOT USED                                     *
  702                        ;***  *                                11 = CW ROTATION                                  *
  703                        ;***  *                                                                                  *
  704                        ;***  *                NIBBLE N1:FUNCTIONS ARE:                                          *
  705                        ;***  *                                                                                  *
  706                        ;***  *                                 0H = SLOW SPEED 1 (NO RAMP)                      *
  707                        ;***  *                                 1H = SLOW SPEED 2 (NO RAMP)                      *
  708                        ;***  *                                 2H = VEL1                                        *
  709                        ;***  *                                 3H = VEL2                                        *
  710                        ;***  *                                 4H = VEL3                                        *
  711                        ;***  *                                 5H = HALT & WAIT FOR NEW COMMANDS                *
  712                        ;***  *                                 6H = STOP, END OF MOTOR COMMANDS                 *
  713                        ;***  *                                 7H = RECYCLE, REPEAT PREVIOUS COMMANDS           *
  714                        ;***  *                            8H - FH = NOT USED                                    *
  715                        ;***  *                                                                                  *
  716                        ;***  *                NIBBLE N2:RFU (RESERVED FOR FUTURE USE)                           *
  717                        ;***  *                NIBBLE N3:RFU                                                     *
  718                        ;***  *                                                                                  *
  719                        ;***  *                          ------------------------------                          *
  720                        ;***  *                WORD 2: |  MID BYTE   |  LOW BYTE   |                          *
  721                        ;***  *                          ------------------------------                          *
  722                        ;***  *                BITS--->15           8 7           0                             *
  723                        ;***  *                                                                                  *
  724                        ;***  *                WORD 2 = MIDDLE & LOW BYTE (BITS 15-0) OF MOTOR STEPS TO TRAVEL   *
  725                        ;***  *                                                                                  *
  726                        ;***  ************************************************************************************
  727                        $EJECT
  728                        ;***
  729                        ;***  THE FOLLOWING BUFFER ('MOTCMD') IS USED TO HOLD MOTOR COMMANDS. IT HAS BEEN FILLED
  730                        ;***  BY THE APPLICATION PROGRAM WITH THESE COMMANDS PRIOR TO THE FIRMWARE ENTERING THE
  731                        ;***  MOTOR CONTROL FIRMWARE AT 'START'. ALSO, THE NEW COMMAND BYTE ('NEWCMD') HAS BEEN
  732                        ;***  PREVIOUSLY SET TO A 01H BY THE PROGRAM.
  733                        ;***
  734 0450   2400            MOTCMD: DW      0024H           ;4 STEP RAMP    (WORD 1)
  735 0452   4008                    DW      2980            ;  60 REV CCW. (WORD 2)
```

```
736 0454  0C00                    DW      000CH           ;'SLOWSP1'      (WORD 1)
737 0456  6000                    DW      096             ;   2 REV CW.   (WORD 2)
738 0458  1400                    DW      0014H           ;'SLOWSP2'      (WORD 1)
739 045A  6000                    DW      096             ;   2 REV CCW.  (WORD 2)
740 045C  2C00                    DW      002CH           ;4 STEP RAMP    (WORD 1)
741 045E  4008                    DW      2880            ;   60 REV CW.  (WORD 2)
742 0460  7000                    DW      0070H           ;RE-CYCLE.      (WORD 1)
743 0462  0000                    DW      0000H           ;               (WORD 2)
744
745                       ;***
746                       ;***
747                       ;***      WORKING REGISTERS IN ON-CHIP RAM
748                       ;***
749                       ;***
750 ----                  RAM     DSEG    AT OFF00H       ;WORKING STORAGE: ON-CHIP RAM
751 FF00                  CTSEG:  DS      3               ;COUNT FOR CURRENT SEGMENT
752 FF03                  CTSTP:  DS      3               ;COUNT FOR CURRENT STEP
753 FF06                  HALFSEG:DS      3               ;REMAINING DISTANCE TO CENTER OF CURRENT RAMP
754 FF09                  IMOT:   DS      1               ;OPERATING VALUE OF WINDING CURRENT OF MOTOR
755 FF0A                  TEMP:   DS      2               ;HOLDS OPERAND FOR RLD INSTRUCTIONS
756 FF0C                  DABUFPT:DS      2               ;DATA BUFFER CURRENT ADDRESS
757 FF0E                  RAMPT:  DS      1               ;STEP NUMBER OF CURRENT RAMP STEP
758 FF0F                  RAMPINC:DS      1               ;CURRENT RAMPING DIRECTION  (+/- 1)
759 FF10                  MOTDIR: DS      2               ;MOTOR DIRECTION.
760 FF12                  MSTATE: DS      1               ;CURRENT MOTOR STATE.
761 FF13                  NEWCMD: DS      1               ;NEW COMMAND BYTE:
762                                                       ;       1 = NEW COMMANDS PRESENT, 0 = PREVIOUS COMMANDS COMPLETE.
763 FF14                  RAMPTAB:DS      2               ;BUFFER CONTAINING RAMP ORIGIN ADDRESS.
764 FF16                  DATBUF: DS      4*22            ;BUFFER FOR 22 COMMANDS.
765                               END                     ;
```

```
UCOM-87 FAMILY ASSEMBLER V2.0     STEPPING MOTOR CONTROL USING A UPD78C11

SYMBOL TABLE XREF LIST

SYMBOL     TYPE        VALUE          ATTRIBUTES AND XREF

MOTCTL       ----      ----           MODULE NAME
CALCTOP    C ADDR      0152H   R      SEG= MOTR7811    249 280 283 375#
CHKTOP     C ADDR      010FH   R      SEG= MOTR7811    274 277#
CKNEW      C ADDR      01CAH   R      SEG= MOTR7811    520# 521
COMP38R    C ADDR      0187H   R      SEG= MOTR7811    278 443#
COMPAGT    C ADDR      01A4H   R      SEG= MOTR7811    448 453 458 461# 469
COMPBGT    C ADDR      01ABH   R      SEG= MOTR7811    446 451 456 468#
COMPEQU    C ADDR      01A1H   R      SEG= MOTR7811    459 460#
CONSPD     C ADDR      0404H   A         644#
CTSEG      D ADDR      FF00H   A         181 210 415 416 751#
CTSTP      D ADDR      FF03H   A         182 288 289 380 580 582 584 752#
DABUFPT    D ADDR      FF0CH   A         140 142 419 756#
DATBUF     D ADDR      FF16H   A         123 139 764#
DEL50      C ADDR      0208H   R      SEG= MOTR7811    312 601#
FSTP       C ADDR      0400H   A         571 629#
GETCMD     C ADDR      005FH   R      SEG= MOTR7811    142# 241
HALFSEG    D ADDR      FF06H   A         209 277 375 753#
HALT       C ADDR      01BBH   R      SEG= MOTR7811    157 513#
IMOT       D ADDR      FF09H   A         112 314 754#
INTE1      C SEG       0003H          ABS    46#
INTX       C ADDR      01E2H   R      SEG= MOTR7811    47 566#
LOAD       C ADDR      004FH   R      SEG= MOTR7811    122# 522
MOTCMD     C ADDR      0450H   A         122 734#
MOTDIR     D ADDR      FF10H   A         307 412 759#
MOTON      C ADDR      0142H   R      SEG= MOTR7811    315 318#
MOTR7811   C SEG       0220H          REL= PAGE COMPLETE    56#
MSTATE     D ADDR      FF12H   A         318 517 539 575 579 760#
NEWCMD     D ADDR      FF13H   A         520 761#
POLLAD     C ADDR      012FH   R      SEG= MOTR7811    310# 326
RAM        D SEG       006EH          ABS    750#
RAMP       C ADDR      0081H   R      SEG= MOTR7811    202#
RAMP1      C ADDR      00A8H   R      SEG= MOTR7811    154 199#
RAMP2      C ADDR      00ABH   R      SEG= MOTR7811    155 200#
RAMP3      C ADDR      00AEH   R      SEG= MOTR7811    156 201#
RAMPDN     C ADDR      0122H   R      SEG= MOTR7811    276 288#
RAMPINC    D ADDR      FF0FH   A         179 206 237 273 383 758#
RAMPT      D ADDR      FF0EH   A         177 204 236 244 262 757#
RAMPTAB    D ADDR      FF14H   A         202 235 763#
RDATBU     C ADDR      0058H   R      SEG= MOTR7811    139# 159
RMPSTP     C ADDR      00D2H   R      SEG= MOTR7811    234# 325
RMPTBL1    C ADDR      0408H   A         171 173 199 646#
RMPTBL2    C ADDR      041CH   A         200 656#
RMPTBL3    C ADDR      0434H   A         201 668#
RSTPDN     C ADDR      0146H   R      SEG= MOTR7811    317 320#
SEGEVAL    C ADDR      0167H   R      SEG= MOTR7811    180 208 405#
SETGO      C ADDR      0128H   R      SEG= MOTR7811    187 305# 384
SETUP      C ADDR      0000H   R      SEG= MOTR7811    57#
SLOWSP1    C ADDR      0088H   R      SEG= MOTR7811    152 173#
SLOWSP2    C ADDR      0083H   R      SEG= MOTR7811    153 171#
SLOWSPD    C ADDR      008CH   R      SEG= MOTR7811    172 174#
START      C ADDR      0021H   R      SEG= MOTR7811    91#
STATETB    C SEG       0064H          ABS    628#
STEPOK     C ADDR      00F4H   R      SEG= MOTR7811    243 262#
STOP       C ADDR      01D3H   R      SEG= MOTR7811    154 535# 541
SUB3BY     C ADDR      01AFH   R      SEG= MOTR7811    296 492#
TEMP       D ADDR      FF0AH   A         143 207 755#
TESTTO     C ADDR      0217H   R      SEG= MOTR7811    609# 610
TESTTOP    C ADDR      00E5H   R      SEG= MOTR7811    240 242#
XRET       C ADDR      0204H   R      SEG= MOTR7811    581 583 585 587#
ZRAM       C ADDR      000DH   R      SEG= MOTR7811    63# 65

TARGET CHIP:UPD7811

ASSEMBLY COMPLETE,  NO ERROR FOUND
```

Expansion of μPD75308 LCD Control/Driver in the 4 Time Division and 1/3 Bias Method by 8 Segments

Contents:

Author: B. Gough
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μCOM75X Family Product Description

Related Products

μPD75328 4-Bit Microcomputer

1. Introduction

This Application Note is used to expand the LCD driver of the 75308 by 8 segments, when using the driver/controller with 4 time divisions and 1/3 bias method. The 75308 already has an LCD driver/controller of 32 segments and 4 commons (32 x 4 = 128 max. picture elements). This will give a total of 136 picture elements.

Ports 6 and 3 must be used so that the 3 state I/O bits can also be used. These ports set the LCD drive voltage via a resistor ladder; hence a simple D/A converter.

Timing synchronization of the frame frequency is via SYNC output, which is input into (INT1) interrupt. The segment synchronization per frame is via the LCDCL output, which is input into (INT4) interrupt.

2. Hardware

As described in the Product Description, the 75308 LCD controller/driver has four modes of operation which are as follows:

Bias method	Time division	COM signal used	Maximum number of picture elements
Static	—	COM0	32 (32 segments x 1 common) *1
1/2	2	COM0, 1	64 (32 segments x 2 commons) *2
	3	COM0, 1, 2	96 (32 segments x 3 commons) *3
1/3	3		
	4	COM0, 1, 2, 3	128 (32 segments x 4 commons) *4

Notes: *1: 8 type LCD panel: 8 segment signals/digit x 4 digits
*2: 8 type LCD panel: 4 segment signals/digit x 8 digits
*3: 8 type LCD panel: 3 segment signals/digit x 10 digits
*4: 8 type LCD panel: 2 segment signals/digit x 16 digits

In this application, a 4 time divisions 1/3 bias method is used with a miximum of 128 picture elements. Eight additional picture elements are produced from ports 6 and 3 with a simple D/A converter which produce the analog segment voltages at SX and SY. The resistors in the D/A converter (figure 1) must be of 1 % tolerance and the analog voltages produced for the controller/driver must have resistors of 1 % tolerance (do not use mask option). The reason for such resistors is to reduce D. C. voltage on the LCD to less than 10 mV.

Ports 6 and 3 must be used because they can be set bitwise, tri-stage. Standby condition is controlled by the bias pin, which is connected to the resistor ladders (see figure 1).

3. Timing Methods

The two timing methods which can be used are:

A) Using only interrupts
B) Using a timer

A) Interrupt Method

As shown in the circuit diagram (figure 1), the interrupt control method uses two interrupt inputs; from the SYNC (Frame Synchronization signal) and from the LCDCL (LCD segment clock). The relationship between two signals and the timing of the commons is shown in figure 2. Figure 3 shows the SX and SY output waveforms for displaying the digit 3., where SX and SY are the segment analog outputs. The port logic for SX and SY is as follows:

VSS LEVEL

Logic Level	Pins
O	P60/P63
X	P61/P32
X	P62/P33

VLC2 LEVEL

Logic Level	Pins
X	P60/P63
O	P61/P32
X	P62/P33

VLC1 LEVEL

Logic Level	Pins
X	P60/P63
X	P61/P32
O	P62/P33

VLC0 LEVEL

Logic Level	Pins
X	P60/P63
X	P61/P32
X	P62/P33

Note: where X = High Impedance
O = Logic Level 0 Volts

The advantage of this method is that no timer is required, however, the method is slow because there is a time offset before putting data on the driver port. This offset delay time (interrupt response time) is approx. 15 μs per interrupt (with 4 MHz external crystal). A timer may be used as shown in figure 4. The timer can reduce the offset time to around 1 μs per interrupt, or 8 μs per frame. To synchronize the timer, the SYNC signal is used as an interrupt input.

4. Circuit Diagram (Interrupt Method)

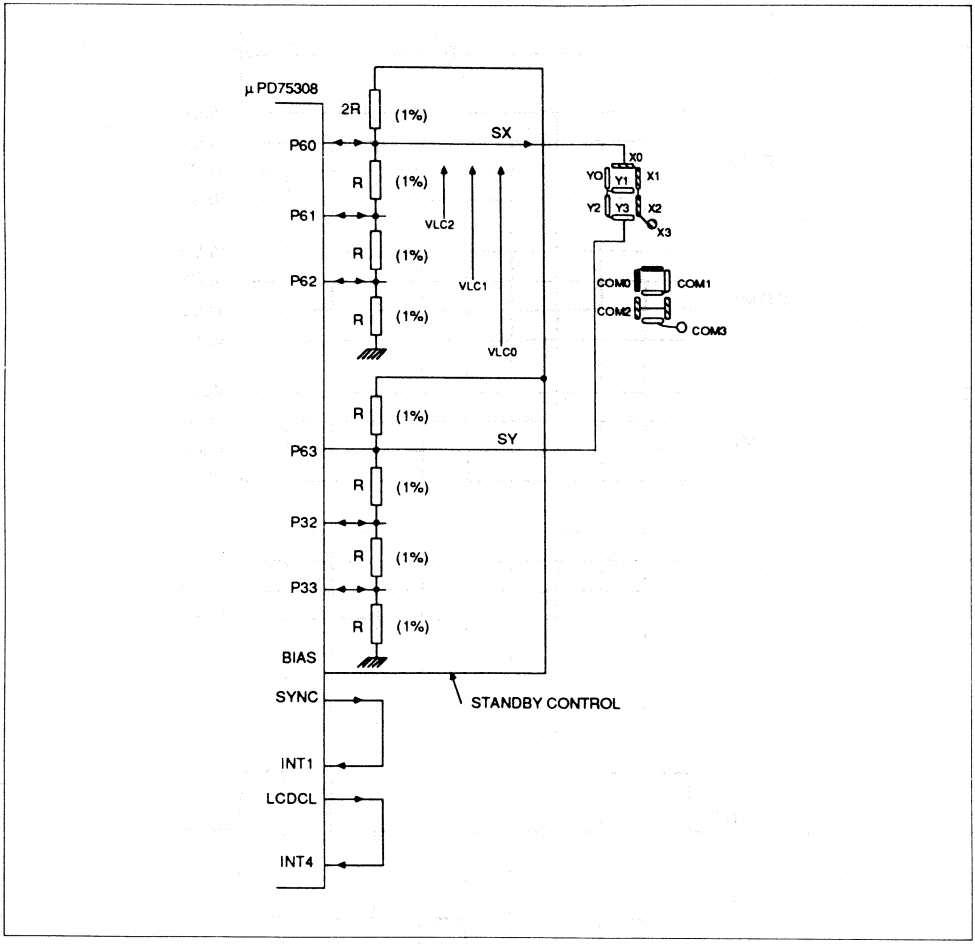


Figure 1.

5. Timing of Commons and SYNC, LCDCL

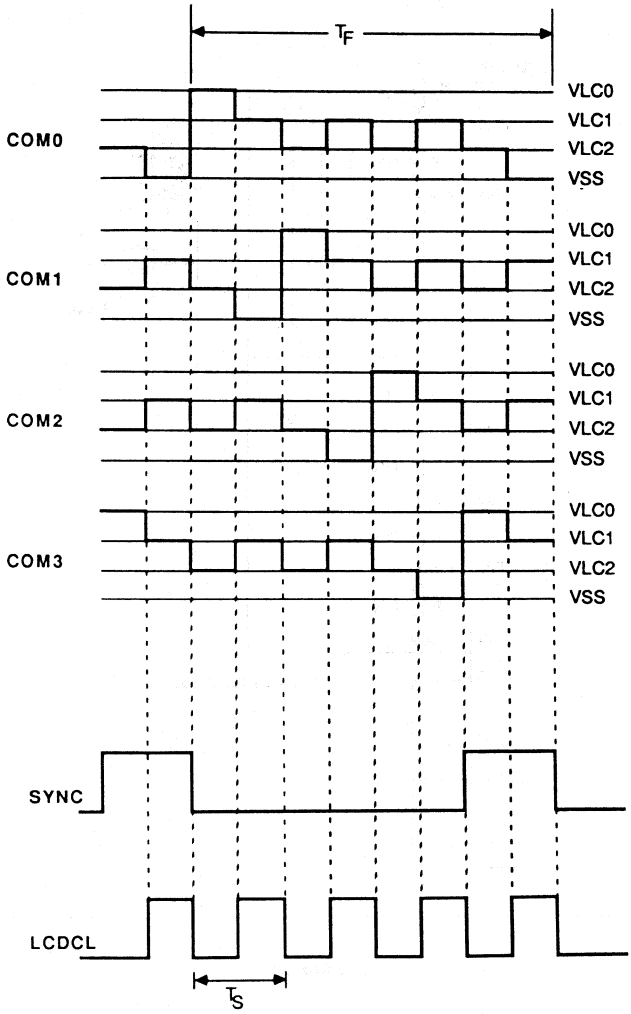


Figure 2.

6. For Example, Display the Digit 3. Interrupt Method

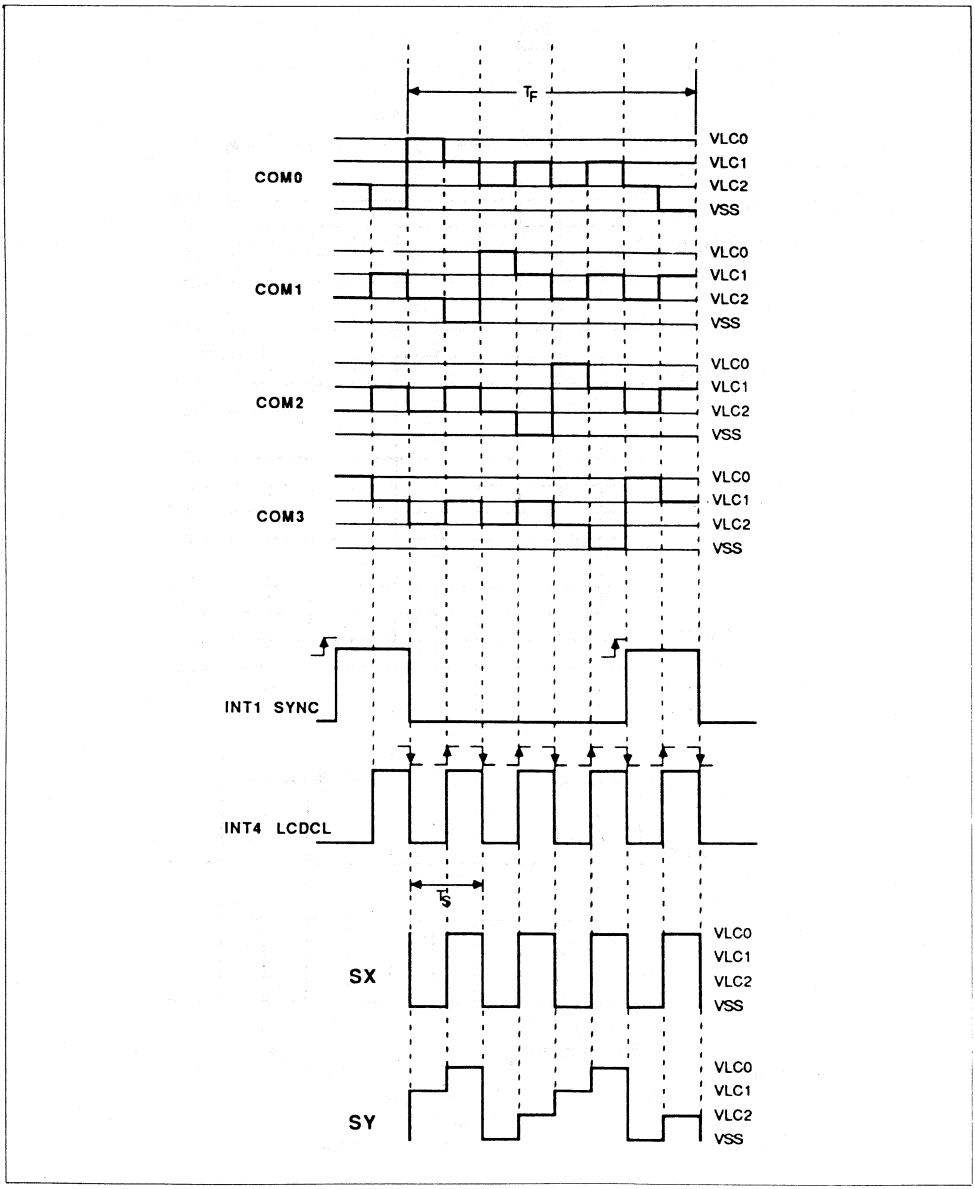


Figure 3.

7. Timing for Timer Method

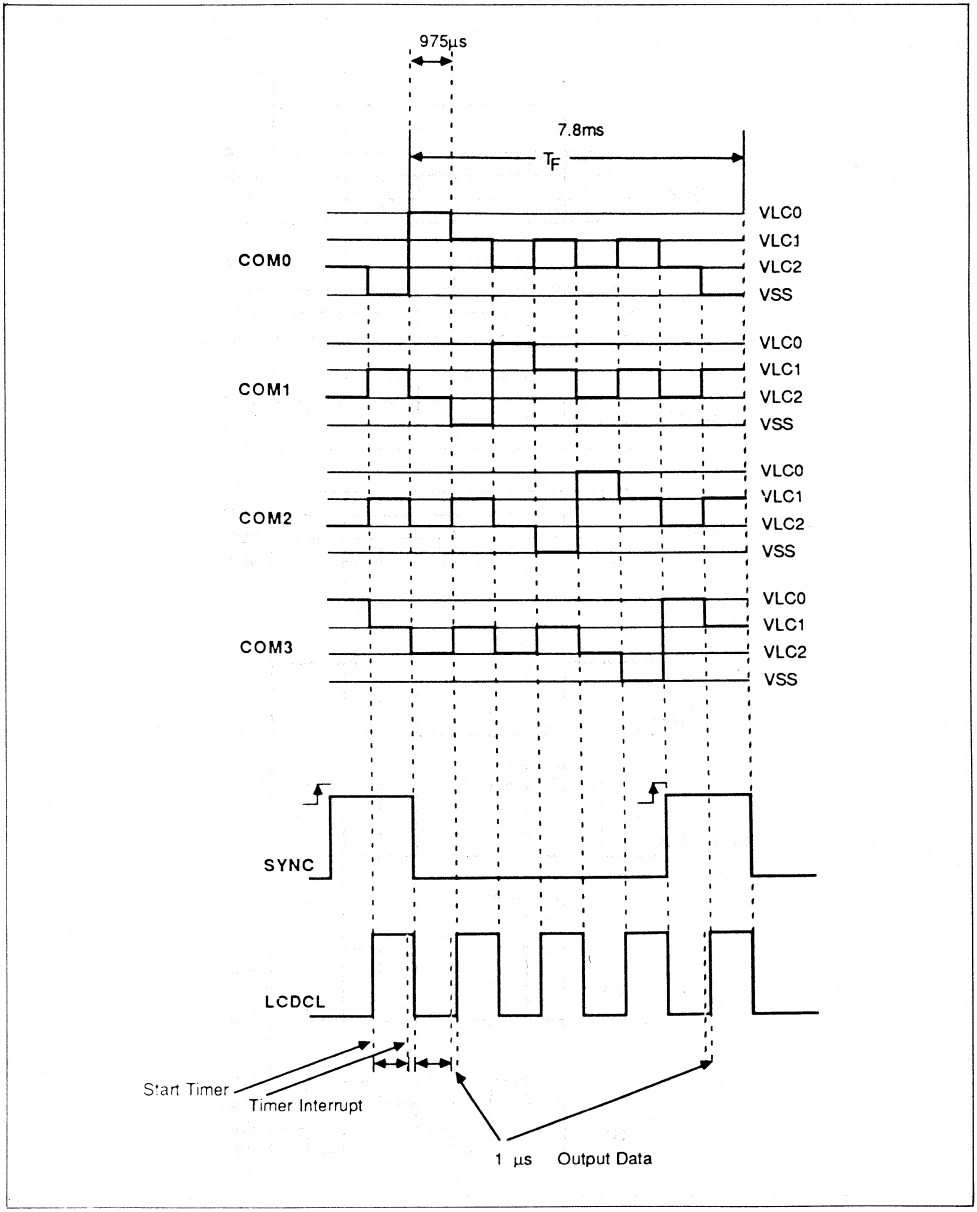
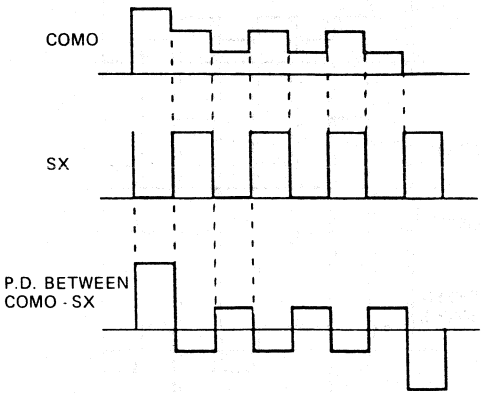


Figure 4.

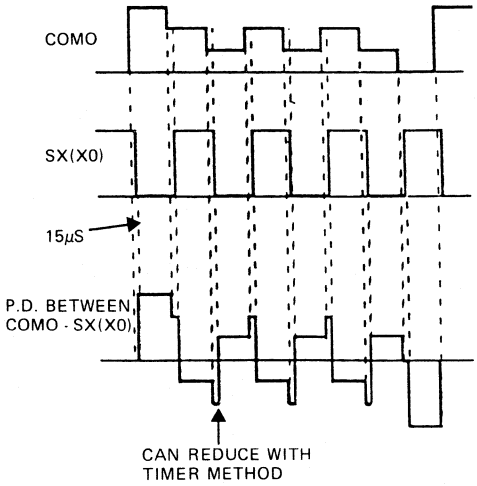
8. P. D. between COMO and SX/SY



Note:
The waveform is symmetrical about the horizontal centre line. The average area is zero, hence the D. C. level is zero.

Figure 5.

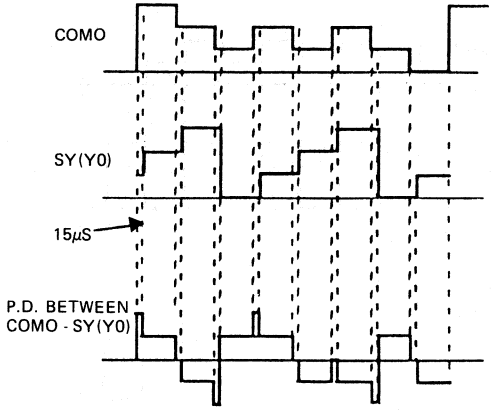
An Example of 15 μs Off Set, Interrupt Method. Display the Digit 3. (X0). Segment on.



Note:
The waveform is symmetrical about the horizontal centre line. The average area is zero, hence the D. C. level is zero.

Figure 6.

An Example of 15μs Off Set, Interrupt Method. Display the Digit 3. (Y0). Segment off.



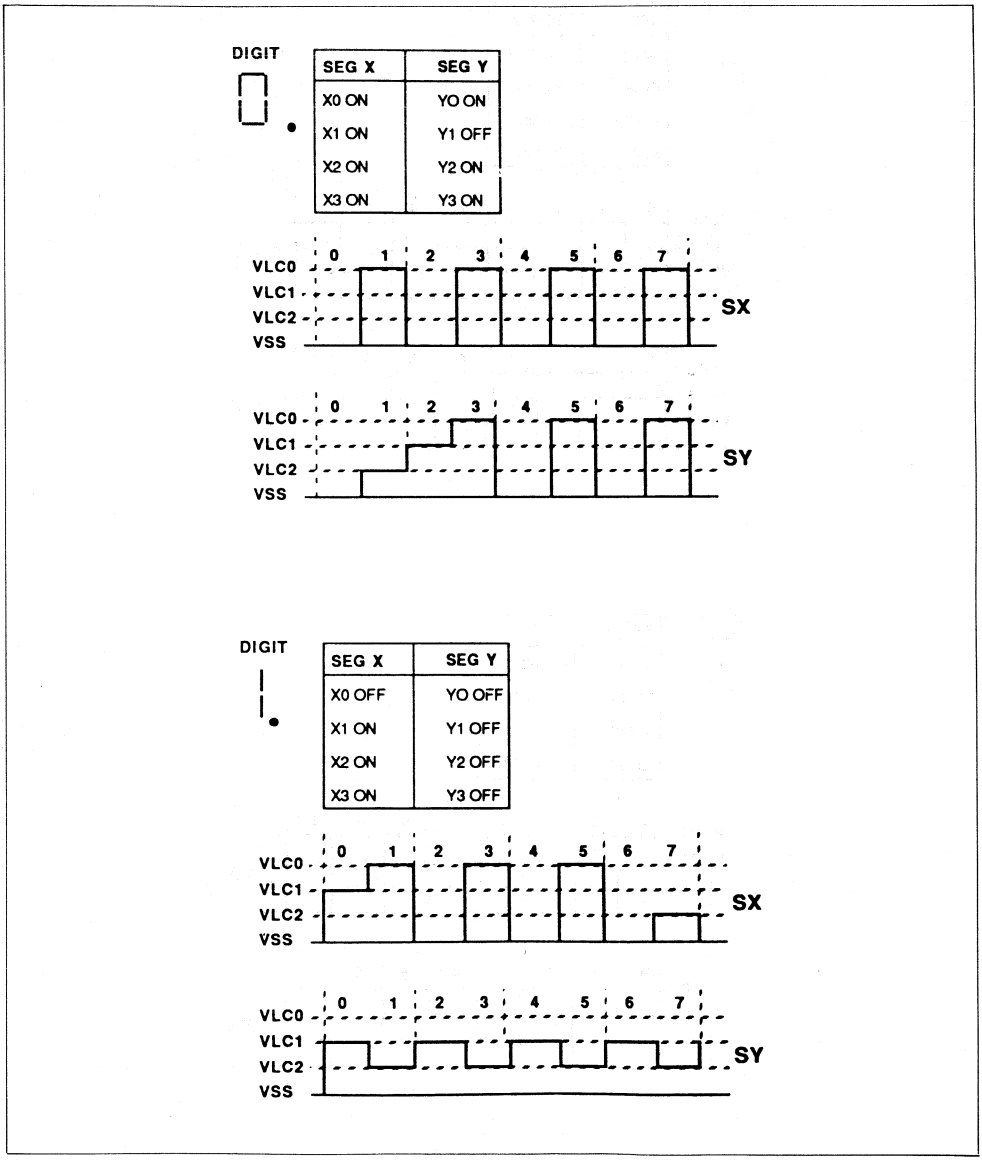
Note:
The waveform is symmetrical about the horizontal centre line. The average area is zero, hence the D. C. level is zero.

Figure 7.

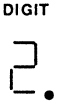
The 15 μs offset will increase the (display off) and decrease the (display on) R. M. S. value by 1 %. This will have no real effect on the viewing angel of the LCD glass.

9. SX and SY Segment Waveforms

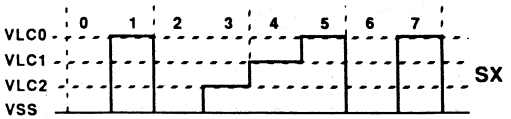
The following shows the waveforms required to produce the digits 0 to F HEX when the 75308 is using the driver/-controller with 4 time divisions and 1(3 bias method.

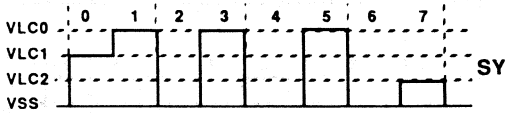


DIGIT



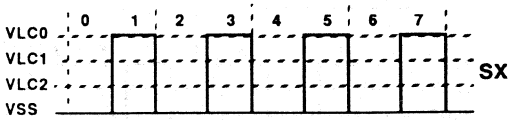
SEG X	SEG Y
X0 ON	YO OFF
X1 ON	Y1 ON
X2 OFF	Y2 ON
X3 ON	Y3 ON

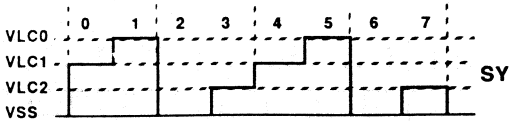




DIGIT

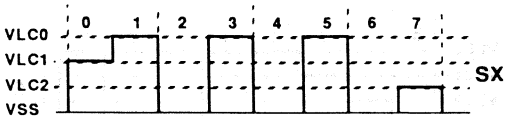
SEG X	SEG Y
X0 ON	YO OFF
X1 ON	Y1 ON
X2 ON	Y2 OFF
X3 ON	Y3 ON

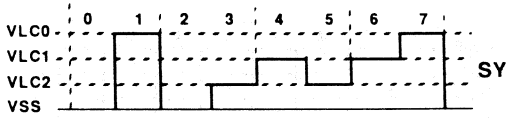




DIGIT

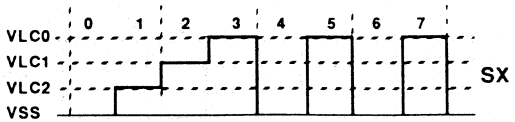
SEG X	SEG Y
X0 OFF	YO ON
X1 ON	Y1 ON
X2 ON	Y2 OFF
X3 ON	Y3 OFF

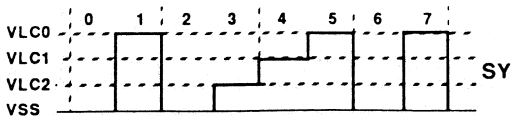




DIGIT

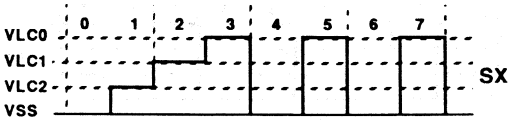
SEG X	SEG Y
X0 ON	YO ON
X1 OFF	Y1 ON
X2 ON	Y2 OFF
X3 ON	Y3 ON

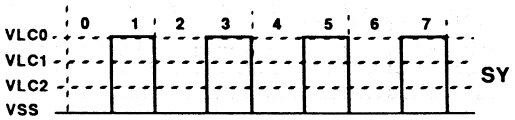




DIGIT

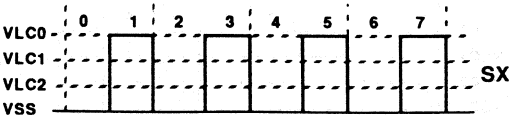
SEG X	SEG Y
X0 ON	YO ON
X1 OFF	Y1 ON
X2 ON	Y2 ON
X3 ON	Y3 ON

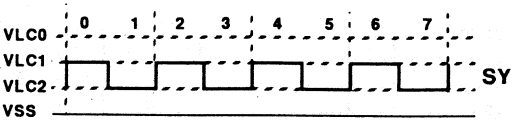




DIGIT

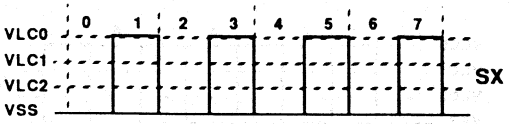
SEG X	SEG Y
X0 ON	YO OFF
X1 ON	Y1 OFF
X2 ON	Y2 OFF
X3 ON	Y3 OFF

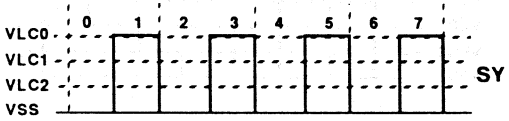




DIGIT 8.

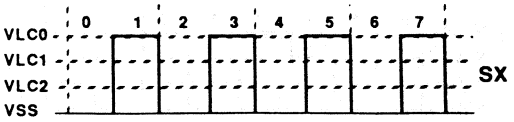
SEG X	SEG Y
X0 ON	YO ON
X1 ON	Y1 ON
X2 ON	Y2 ON
X3 ON	Y3 ON

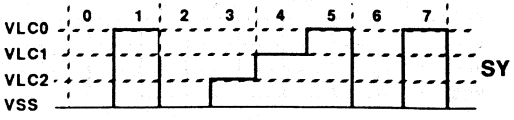




DIGIT 9.

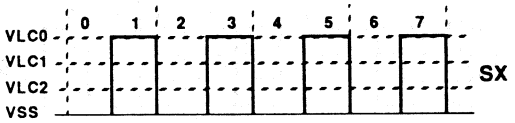
SEG X	SEG Y
X0 ON	YO ON
X1 ON	Y1 ON
X2 ON	Y2 OFF
X3 ON	Y3 ON

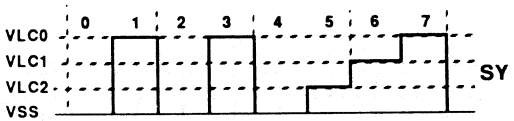




DIGIT

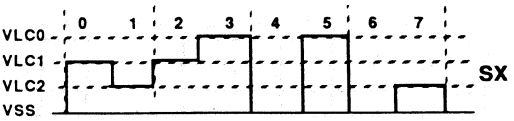
SEG X	SEG Y
X0 ON	YO ON
X1 ON	Y1 OFF
X2 ON	Y2 ON
X3 ON	Y3 OFF

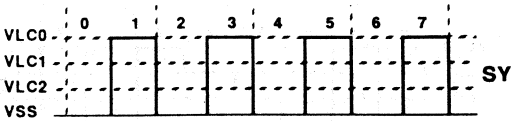




DIGIT

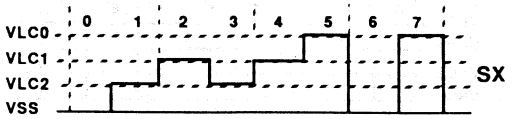
SEG X	SEG Y
X0 OFF	YO ON
X1 OFF	Y1 ON
X2 ON	Y2 ON
X3 ON	Y3 ON

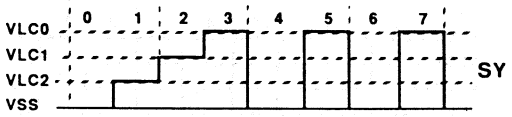




DIGIT

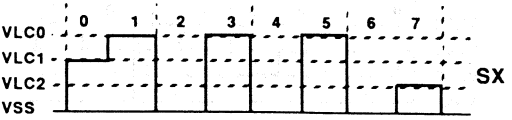
SEG X	SEG Y
X0 ON	YO ON
X1 OFF	Y1 OFF
X2 OFF	Y2 ON
X3 ON	Y3 ON

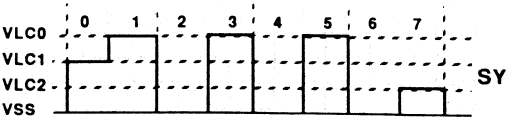




DIGIT

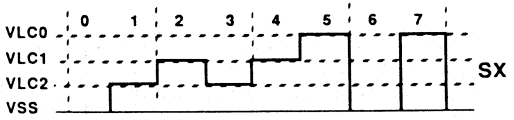
SEG X	SEG Y
X0 OFF	YO OFF
X1 ON	Y1 ON
X2 ON	Y2 ON
X3 ON	Y3 ON

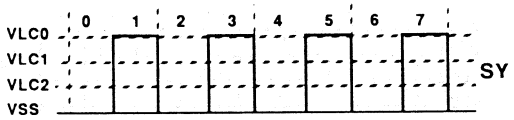




DIGIT

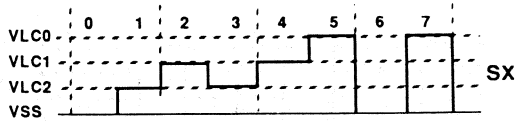
SEG X	SEG Y
X0 ON	YO ON
X1 OFF	Y1 ON
X2 OFF	Y2 ON
X3 ON	Y3 ON

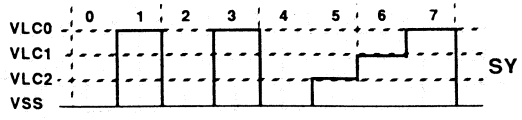




DIGIT

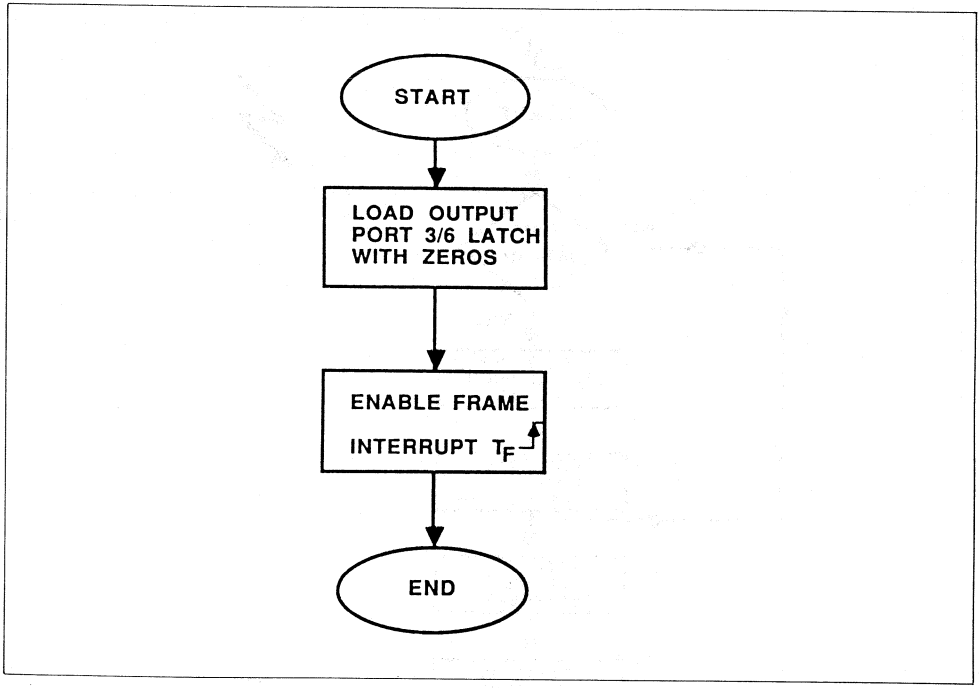
SEG X	SEG Y
X0 ON	YO ON
X1 OFF	Y1 ON
X2 OFF	Y2 ON
X3 ON	Y3 OFF



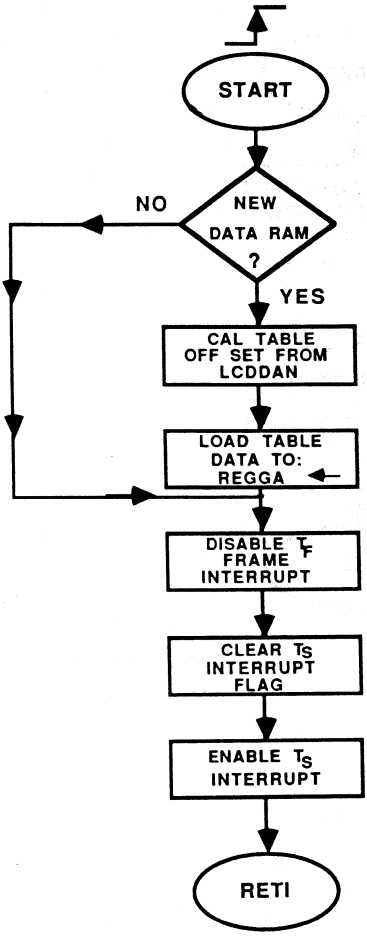


10. Software Interrupt Method

A) Set Up

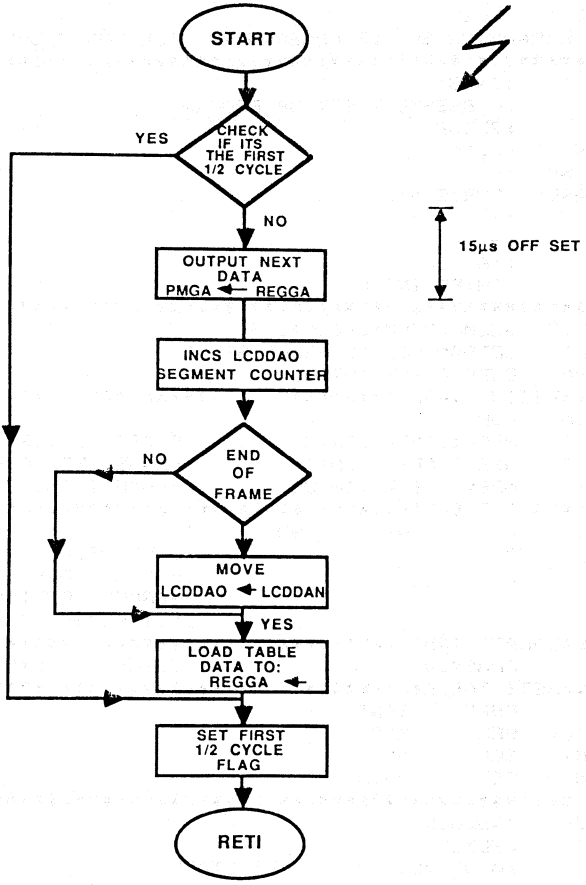


B) T_F Frame Interrupt



Where: LCDDAN = LCD User RAM Data
REGGA = Segment Port Data Reg.
T_S = Segment Interrupt

C) T_S Segment Interrupt



Where: LCDDAN = LCD User RAM Data
LCDDAO = LCD Internal RAM Data

11. Software Listing

SOURCE STATEMENT

```
        NAME    SEGIN
;LCDSEGMENT EXPANSION SET UP PROGRAM WRITTEN FOR 75308 BY B.GOUGH 8/87
;*************************************************************************
;NAME            :INTST
;DESC            :LCDSEGMENT SET UP PROGRAM
;DEVICE          :75308
;MEMORY BANK     :0,15
;REGISTER BANK   :-
;USED HARDWARE   :PORTS 6/3
;INPUT           :-
;OUTPUT          :-
;DESTROY         :XA
;CALLS           :INTFR,INTSG
;*************************************************************************
        PUBLIC  SELM0,SELM1,SELM15
        PUBLIC  LCDDAN,LCDDAO,REGGA,SFLAG
        EXTRN   CODE(INTFR,INTSG)
;************VECTOR TABLE*****************************************************
        STKLN   40H
        VENT0   MBE=1,RBE=0,INTST       ;START VECTOR
        VENT2   MBE=0,RBE=0,INTFR       ;FRAME INTERRUPT VECTOR
        VENT1   MBE=1,RBE=0,INTSG       ;SEGMENT INTERRUPT VECTOR
;************DATA TABLE*******************************************************
D1SEG   DSEG    0       AT      8H
LCDDAN: DS      2                       ;USER DISPLAY RAM
LCDDAO: DS      2                       ;PROGRAM DISPLAY RAM
REGGA:  DS      2                       ;SEGMENT PORT DATA REG
FLAGA:  DS      1                       ;BIT FLAGS
;************EQUATE TABLE*****************************************************
SFLAG   EQU     FLAGA.0                 ;DELAY 1/2 CYCLE FLAG
;************GETI TABLE*******************************************************
        GET1    CSEG    IENT
        SELM0:  SEL     MB0
        SELM1:  SEL     MB1
        SELM15: SEL     MB15
;*************************************************************************
C1SEG   CSEG    INBLOCK
INTST:  GETI    SELM15
        MOV     XA,#STACK       ;SET STACK
        MOV     SP,XA
        MOV     A,#0011B        ;SET HIGH SPEED SYSTEM CLOCK
        MOV     PCC,A
;**********SET LCD BASE FREQUENCE FROM WATCH*********************************
        MOV     XA,#07H         ;FROM SUBSYSTEM OSC 32KHZ
        MOV     WM,XA
;**********SET UP EXPANSION SEGMENT PORTS*************************************
        MOV     A,#0H           ;LOAD LCD SEGMENT LATCH WITH ZEROS
```

**

SOURCE STATEMENT

```
        OUT     PORT3,A
        OUT     PORT6,A
;**********SET UP LCDCL,SYNC,CONTROLLER/DRIVER **********************
        MOV     XA,#00000011B   ;SET P30,P31 AS OUTPUTS FOR LCDCL,SYNC
        MOV     PMGA,XA
        MOV     XA,#00111000B   ;ENABLES LCD DRIVER 128KHZ,1/4,32SEG
        MOV     LCDM,XA
        MOV     A,#4H           ;ENABLES LCDCL AND SYNC
        MOV     LCDC,A
;**********ENABLE FRAME INTERRUPT**************************************
        DI
        MOV     A,#0H
        MOV     IM0,A           ;ENABLE RISING EDGE (Tf)FRAME INTERRUPT
        EI      IE0
        EI
;**********TEST PROGRAM SOFTWARE***************************************
WAIT1:  NOP
        BR      WAIT1
        END
```

SOURCE STATEMENT

```
         NAME    SEGSG
;SEGMENT INTERRUPT PROGRAM WRITTEN FOR 75308 BY B.GOUGH
;8/87
;*************************************************************
;NAME              :INTSG
;DESC              :SEGMENT INTERRUPT PROGRAM
;DEVICE            :75308
;MEMORY BANK       :0
;REGISTER BANK     :-
;USED HARDWARE     :-
;INPUT             :REGGA(SEGMENT PORT DATA REG)
;INPUT             :LCDDAO(PROGRAM DISPLAY RAM)
;OUTPUT            :-
;DESTROY           :XA
;CALLS             :TABLE1
;*************************************************************
         PUBLIC  INTSG
         EXTRN   DATA(LCDDAO,LCDDAN,REGGA)
         EXTRN   BIT(SFLAG)
         EXTRN   CODE(TABLE1)

CSEG     CSEG    INBLOCK
INTSG:   SEL     MB0                ;15US OFF SET TO OUTPUT SEGMENT DATA
         SKF     SFLAG              ;CHECK IF ITS THE FIRST 1/2 CYCLE
         BR      NOSEG
         MOV     XA,REGGA
         SEL     MB15
         MOV     PMGA,XA
         MOV     A,#5H              ;PUT POWER ON LCD GLASS
         MOV     LCDC,A
;
         SEL     MB0                ;CAL NEW OUTPUT SEGMENT DATA
         INCS    LCDDAO             ;CAL NEW LOOK UP TABLE OFF SET
         NOP                        ;MAKE SURE NO SKIP CON
         MOV     XA,LCDDAO
         SKE     A,#0H              ;CHECK FOR END OF FRAME
         BR      NOEU1
         BR      YSEU1
NOEU1:   SKE     A,#8H
         BR      NOEU2
YSEU1:   MOV     XA,LCDDAN
         MOV     LCDDAO,XA
NOEU2:   CALL    !TABLE1
         MOV     REGGA,XA           ;STORE NEXT PORT6/3 SEGMENT OUTPUT DATA
;
NOSEG:   CLR1    SFLAG
         RETI
         END
```

SOURCE STATEMENT

```
        NAME    SEGFR
;LCD FRAME INTERRUPT PROGRAM WRITTEN FOR 75308
;BY B.GOUGH 8/87
;*************************************************************
;NAME            :INTFR
;DESC            :FRAME INTERRUPT
;DEVICE          :75308
;MEMORY BANK     :0
;REGISTER BANK   :-
;USED HARDWARE   :-
;INPUT           :LCDDAN (USER DISPLAY RAM)
;OUTPUT          :REGGA  (SEGMENT PORT DATA REG)
;DESTROY         :XA
;CALLS           :TABLE1
;*************************************************************
        PUBLIC  INTFR
        EXTRN   DATA(LCDDAN,LCDDA0,REGGA)
        EXTRN   BIT(SFLAG)
        EXTRN   CODE(SELM0,SELM1,SELM15)
        EXTRN   CODE(TABLE1)

C2SEG   CSEG    INBLOCK
INTFR:  MOV     XA,LCDDAN       ;LOAD ACC WITH USER DISPLAY REG
        MOV     LCDDA0,XA       ;RESET PROGRAM DISPLAY REG
        CALL    !TABLE1         ;GET PORT6/3 D/A SEGMENT DATA
        MOV     REGGA,XA        ;STORE SEGMENT PORT DATA
        SET1    SFLAG           ;DELAY 1/2 CYCLE TO SYNRONIZE SEGS WITH
        DI                                                      FRAME
        DI      IE0             ;DISABLE FRAME SYNC INTERRUPT
        CLR1    IRQ4
        EI      IE4             ;ENABLE SEGMENT INTERRUPT BOTH EDGES
        EI
        RETI
        END
```

SOURCE STATEMENT

```
        NAME    SEGTABL1
;PORT6/3 D/A LOOK-UP TABLE DATA WRITTEN FOR 75308
;BY B.GOUGH 8/87
;*****************************************************
;NAME            :TABLE1
;DESC            :D/A LOOK-UP TABLE
;DEVICE          :75308
;MEMORY BANK     :-
;REGISTER BANK   :-
;USED HARDWARE   :-
;INPUT           :XA
;OUTPUT          :XA
;DESTROY         :XA,PSW
;CALLS           :-
;*****************************************************
        PUBLIC TABLE1

C4SEG   CSEG    PAGE
;*****************************************************
        DB      10010011B       ;DIGIT 0. OFFSET 00H
        DB      00000111B
        DB      00011011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
;*****************************************************
        DB      01001011B       ;DIGIT 1. OFFSET 08H
        DB      00000111B
        DB      00011011B
        DB      00000111B
        DB      00011011B
        DB      00000111B
        DB      00011011B
        DB      00100111B
;*****************************************************
        DB      00011011B       ;DIGIT 2. OFFSET 10H
        DB      00000011B
        DB      10010011B
        DB      00100011B
        DB      11000011B
        DB      00000011B
        DB      10010011B
        DB      00000111B
;*****************************************************
        DB      00011011B       ;DIGIT 3. OFFSET 18H
        DB      00000011B
```

```
SOURCE STATEMENT

        DB      10010011B
        DB      00000111B
        DB      00011011B
        DB      00000011B
        DB      10010011B
        DB      00000111B
;*************************************************************
        DB      11000011B       ;DIGIT 4. OFFSET 20H
        DB      00000011B
        DB      10010011B
        DB      00000111B
        DB      00011011B
        DB      00000111B
        DB      00011011B
        DB      00100011B
;*************************************************************
        DB      10010011B       ;DIGIT 5. OFFSET 28H
        DB      00100011B
        DB      11000011B
        DB      00000111B
        DB      00011011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
;*************************************************************
        DB      10010011B       ;DIGIT 6. OFFSET 30H
        DB      00100011B
        DB      11000011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
;*************************************************************
        DB      00011011B       ;DIGIT 7. OFFSET 38H
        DB      00000111B
        DB      00011011B
        DB      00000111B
        DB      00011011B
        DB      00000111B
        DB      00011011B
        DB      00000111B
;*************************************************************
        DB      10010011B       ;DIGIT 8. OFFSET 40H
        DB      00000011B
        DB      10010011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
;*************************************************************
        DB      10010011B       ;DIGIT 9. OFFSET 48H
        DB      00000011B
        DB      10010011B
```

```
SOURCE STATEMENT

        DB      00000111B
        DB      00011011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
;*****************************************************
        DB      10010011B       ;DIGIT A. OFFSET 50H
        DB      00000011B
        DB      10010011B
        DB      00000011B
        DB      10010011B
        DB      00000111B
        DB      00011011B
        DB      00000011B
;*****************************************************
        DB      11000011B       ;DIGIT B. OFFSET 58H
        DB      00100011B
        DB      11000011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
        DB      10010011B
        DB      00100011B
;*****************************************************
        DB      10010011B       ;DIGIT C. OFFSET 60H
        DB      00100111B
        DB      01001011B
        DB      00100011B
        DB      11000011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
;*****************************************************
        DB      01001011B       ;DIGIT D. OFFSET 68H
        DB      00000011B
        DB      10010011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
        DB      10010011B
        DB      00100111B
;*****************************************************
        DB      10010011B       ;DIGIT E. OFFSET 70H
        DB      00100011B
        DB      11000011B
        DB      00100011B
        DB      11000011B
        DB      00000011B
        DB      10010011B
        DB      00000011B
;*****************************************************
        DB      10010011B       ;DIGIT F. OFFSET 78H
        DB      00100011B
        DB      11000011B
        DB      00100011B
```

```
SOURCE STATEMENT

        DB      11000011B
        DB      00000111B
        DB      00011011B
        DB      00000011B
;*****************************************************
;
;*********MAKE SURE THAT MOVT INSTR IS IN SAME PAGE AS TABLE**********
TABLE1: MOVT    XA,@PCXA        ;LOAD TABLE DATA
        RET
        END
```

SBI Bus to I²C Bus Conversion on the μCOM75X Family Single Chip Microcomputers (μPD75308/008/328)

Contents:

Author: B. Gough
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μCOM75X Family Product Description

Note: I²C is a trademark of PHILIPS

Related Products

μPD75X 4-Bit Microcomputers Family

1. Introduction

This application note converts any μCOM75X Family microcomputer which has SBI bus, to I²C bus. At present there are many IC's on the market with an I²C interface, for example: DTMF, EEPROM, RAM and LCD Driver/Controllers. In this application note, the EEPROM PCB8582 from Valvo with I²C bus is interfaced on the SBI bus.

2. Hardware

In this application note full I²C bus protocol is not implemented; the microcomputer is a master only and the EEPROM is a slave only. SBI bus and I²C are both open drain bus systems, the μCOM75X Family microcomputers have on-chip programmable pull-up resistors on the serial input/output and on the clock line. SBI bus hardware can be utilized to perform I²C bus protocol. As shown in figure 1, the serial interface can be configured in the three different modes: 3 wire I/O mode, 2 wire I/O mode and 2 wire SBI bus mode.

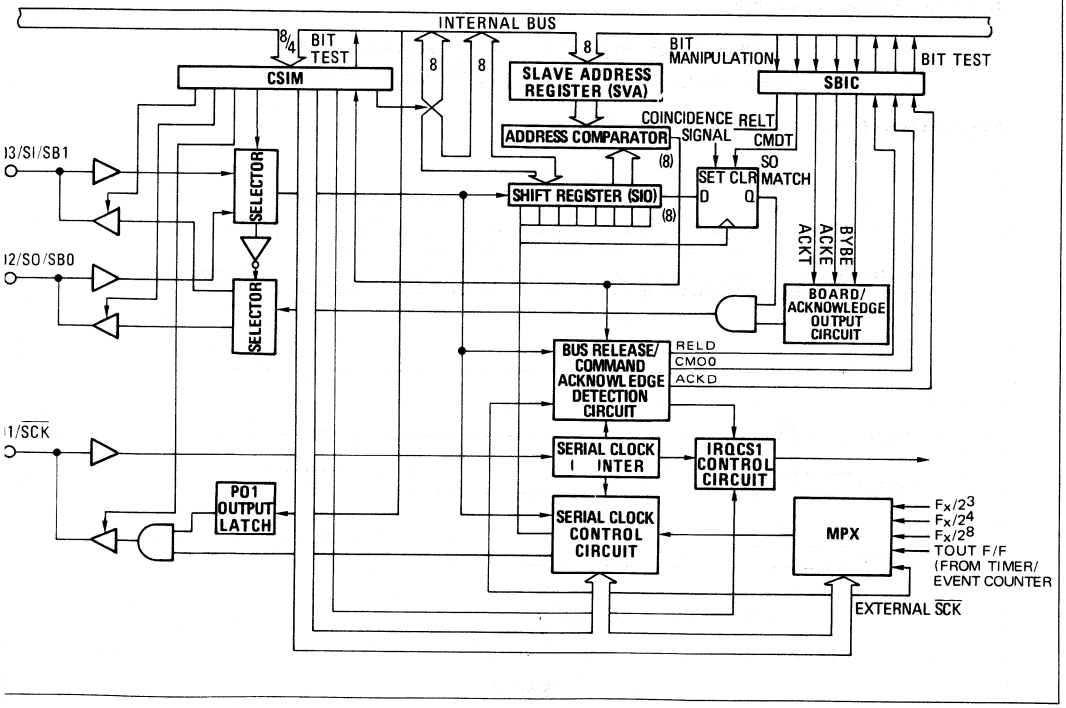


Figure 1.

SBI bus is 5 times faster (500 KHz) than I²C bus with a minimum selectable clock speed of 262 KHz at (4.19 MHz) external crystal. Hence, an external clock source is required to the run the bus. The μPD75308/008/328 all have an external 65.5 KHz clock source which can be used. As shown in section 3, P22 provides this clock source. Figure 2 shows the protocol requirements of I²C on the EEPROM.

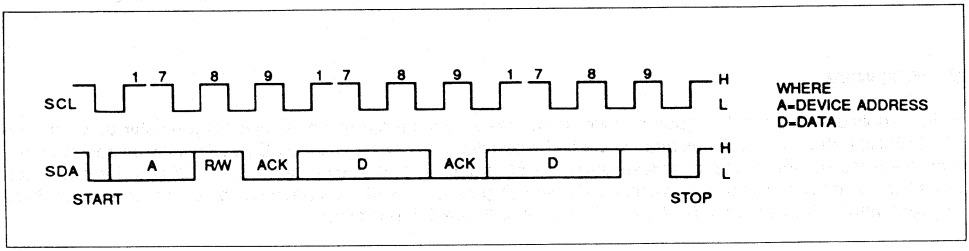


Figure 2.

There are three basic protocols to consider: start condition (from master), acknowledge (from slave) and stop condition (from master). To produce the required protocols the user can set and clear the serial output latch by software. The software to do this is called:

(bus release trigger) RELT −> SET output latch high
(bus command trigger) CMDT −> SET output latch low

Figure 3 shows how a start condition is set with SBI bus.

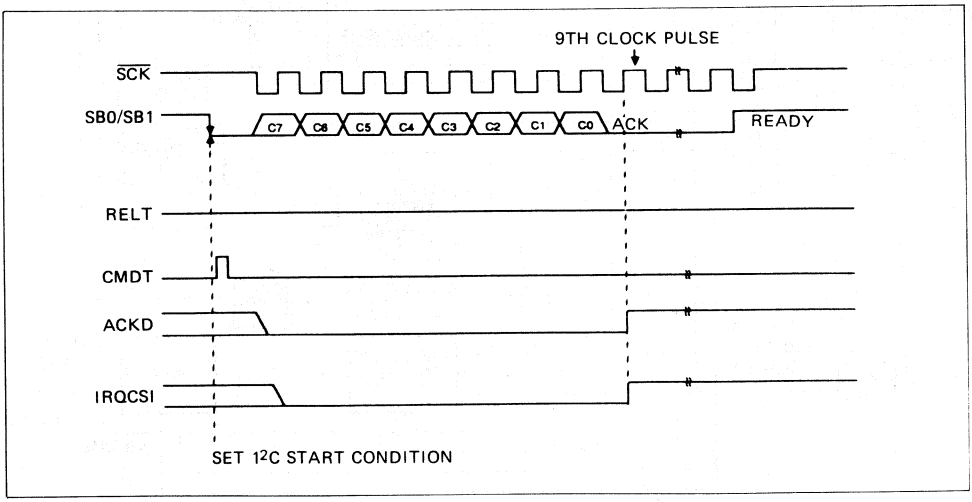


Figure 3.

The acknowledge signal from the EEPROM is clocked in on the 9th clock pulse (rising edge). The NEC microcomputers have an acknowledge test flag called ACKD, which can be tested with a bit test instruction (for example, SKT ACKD).

The stop condition is set by setting the serial I/O line low with CMDT then setting the clock high via P22 and followed by setting serial I/O line high with RELT. The diode (D1) and P21 pin allows the clock to be stop at the EEPROM. The reason for stopping the clock is to allow the microcomputer time to reload the serial register.

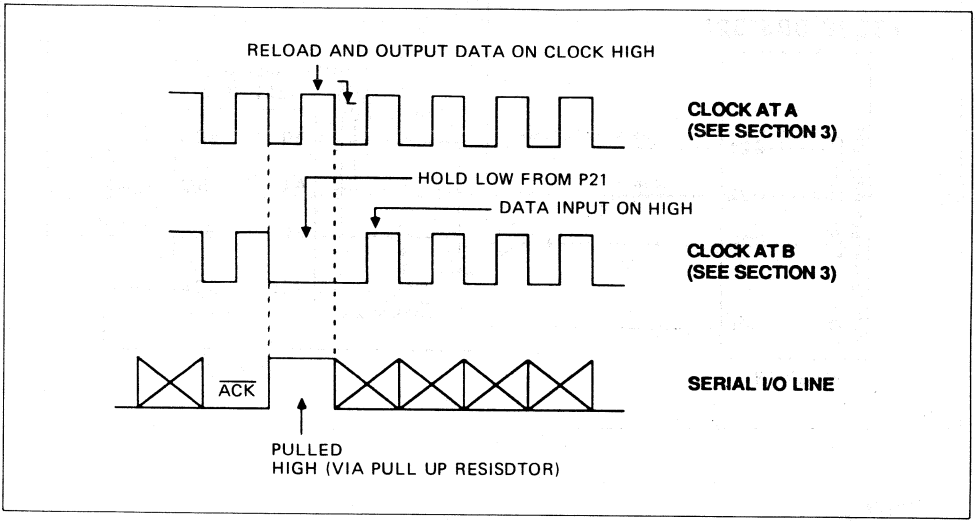
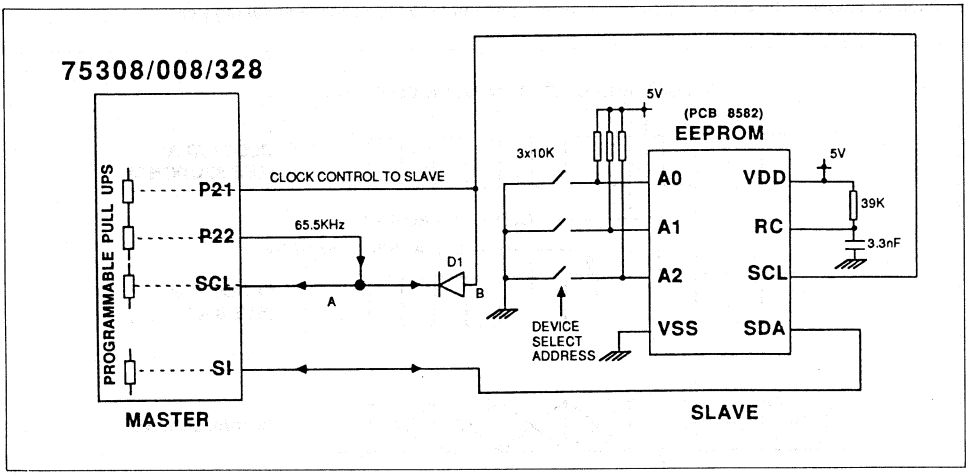


Figure 4.

Data is clocked out on the falling edge of the clock in SBI bus mode and input on I²C bus with the clock line high. Hence using P22 to set clock line low would cause problems when enabling the clock, because invalid data would be clock into the EEPROM when clock the line is high (please see figure 4).

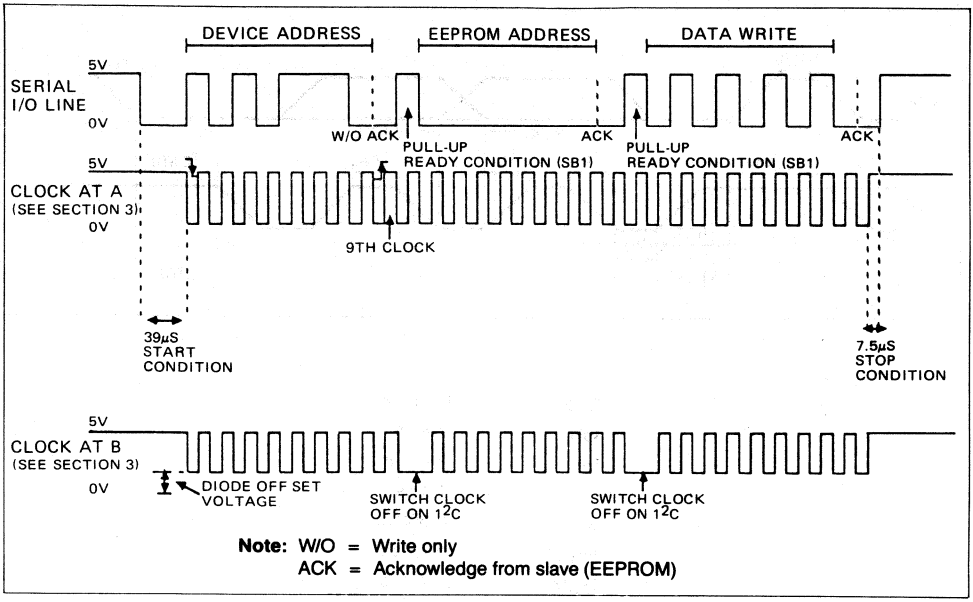
3. Circuit for SBI to I²C Conversion



4. Timing

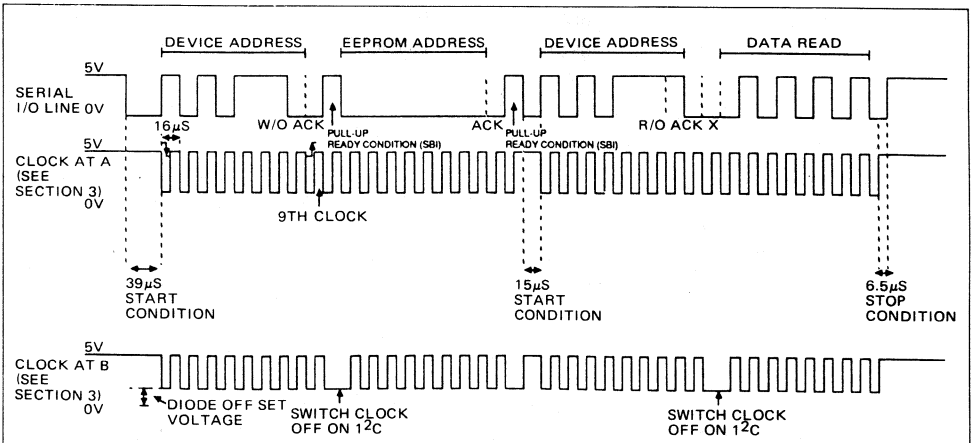
Timing on the I²C bus is split into two parts; write only and read only. If we consider the timing diagram in section A4 Write Only, data is written into the EEPROM. In this example the device address is AEHEX, the EEPROM address is OOHEX, and data to be stored is 55HEX. Two clock positions are shown at 'A' (into microcomputer) and at 'B' (into EEPROM — please see section 3). The clock at 'B' is chopped by the diode offset voltage, but this has no effect on the timing to the EEPROM (please refer PCB-8582 spec.). The start condition is set such that the serial I/O line goes from the microcomputer and clocked into the EEPROM on a high level. The first data transmitted to the EEPROM is the device address, the first 4 bits are preset by the EEPROM, the second 3 bits are set by setting external logic levels on pins A0, A1, A2. The next bit is '0' ('if write only'), and '1' ('if read only'). After every byte transmitted to the EEPROM an acknowledge signal is transmitted from the EEPROM. This is clocked in on the rising edge of the 9th clock pulse. The bus is now in a SBI ready condition, here also the clock is disabled to the EEPROM. In the SBI ready condition the bus is pulled high. The next two bytes are the EEPROM address and data to be stored. The stop condition is set by the serial I/O line going from low to high with the clock set high. Section 4B shows the timing diagram for read only. Before reading data out of the EEPROM, the EEPROM address must be written into EEPROM. After this, data can be read out of the EEPROM address location. Section 4B shows the point X where I²C bus (from EEPROM) pulls the bus low, hence the SBI bus can not get a ready condition. At this point data is polled into the microcomputer.

4.1 Timing Diagram Write Only



Note: W/O = Write only
ACK = Acknowledge from slave (EEPROM)

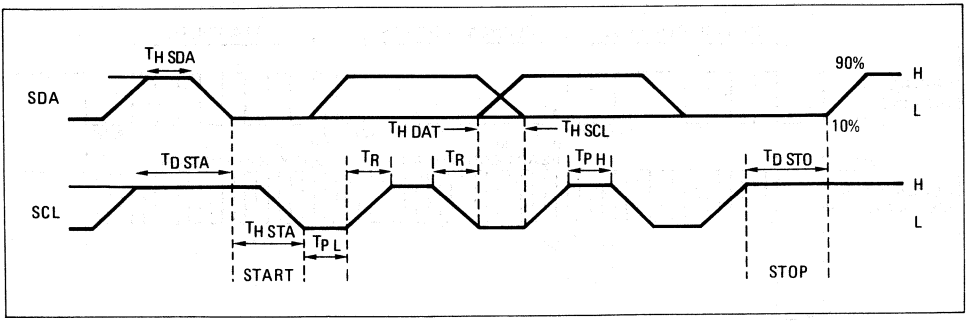
4.2 Timing Diagram Read Only



Notes: 1) At Point 'X' I²C bus pulls the bus low, hence no ready condition on SBI.

2) W/O = Write only
R/O = Read only
ACK = Acknowledge from slave (EEPROM)

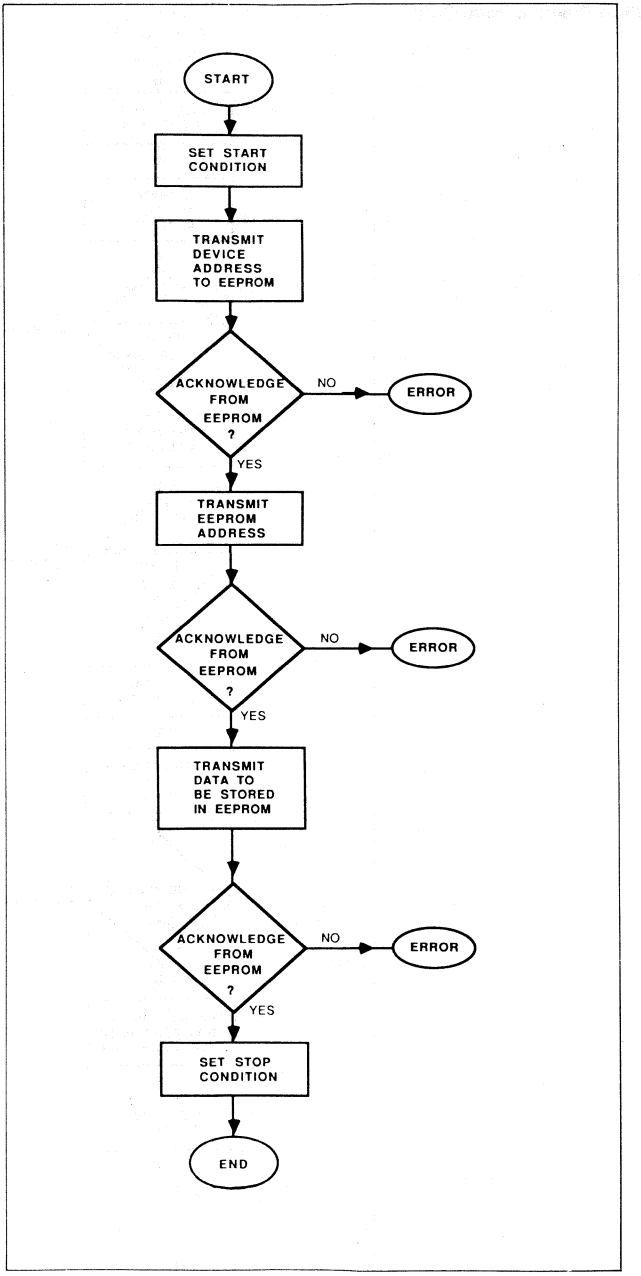
4.3 EEPROM Timing



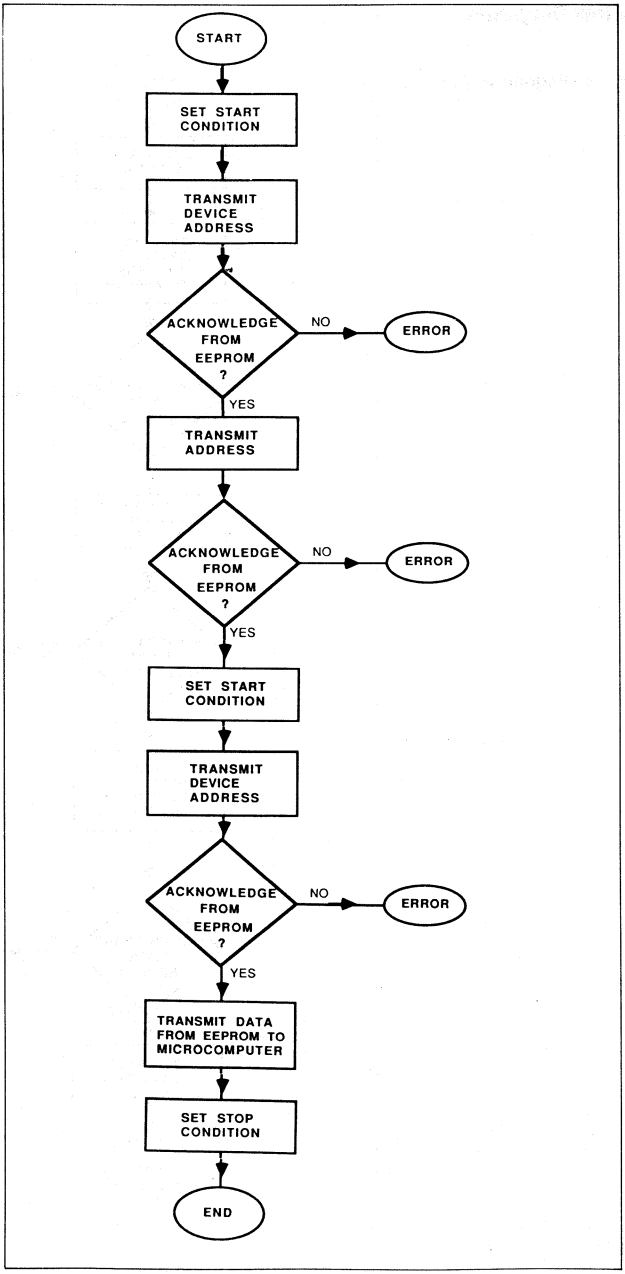
f_{SCL}	100 KHz
t_{pH}	$> 4\ \mu s$
t_{pL}	$> 4.7\ \mu s$
t_r	$< 1\ \mu s$
t_f	< 300 ns
t_{hSDA}	$> 4.7\ \mu s$
t_{dSTA}	$> 4.7\ \mu s$
t_{hSTA}	$> 4\ \mu s$
t_{dSTO}	$> 4.7\ \mu s$
t_{hDT}	> 0
t_{dSCL}	$> 0.25\ \mu s$

5. Flow Diagrams

5.1 Flow Diagram Write Only



5.2 Flow Diagram Read Only



6. Demo Software / Software Listing

There are two programs:

— one for writing
and
— one for reading.

Write Program

REG

BC = DEVICE ADDRESS
DE = EEPROM ADDRESS
HL = DATA WRITE

Read Program

REG

BC = DEVICE ADDRESS
DE = EEPROM ADDRESS
HL = DATA READ

Software Listing Write Program

```
                NAME    IICWR
;IIC BUS WRITE ONLY PROGRAM WRITTEN FOR 75308 BY B.GOUGH 8/87
;*************************************************************************
;NAME            :IICSTW
;DESC            :IIC BUS WRITE ONLY PROGRAM
;DEVICE          :75308
;MEMORY BANK     :15
;REGISTER BANK   :-
;USED HARDWARE   :SERIAL PORT
;INPUT           :BC(DEVICE ADDRESS),DE(EEPROM ADDRESS),HL(DATA WRITE)
;OUTPUT          :-
;DESTROY         :XA
;CALLS           :-
;************VECTOR TABLE*************************************************
        STKLN   40H
        VENTO   MBE=1,RBE=0,IICSTW      ;START VECTOR
;************GETI TABLE***************************************************
        GET1    CSEG    IENT
        SELM15: SEL     MB15
;************SET UP PROGRAM ***********************************************
C1SEG   CSEG    INBLOCK
IICSTW: GETI    SELM15
        MOV     XA,#STACK       ;SET STACK
        MOV     SP,XA
        MOV     A,#0011B        ;SET HIGH SPEED SYSTEM CLOCK
        MOV     PCC,A
;
        MOV     XA,#05H         ;SET PULL UPS BECAUSE OF OPEN DRAIN BUS
        MOV     POGA,XA
;
        SET1    PORT2.1         ;SET CLOCK CONTROL LINE HIGH TO EEPROM
        SET1    PORT2.2         ;SET CLOCK LINE HIGH
        MOV     XA,#04H         ;SET IIC TIMER PORT
        MOV     PMGB,XA
;
        MOV     A,#0FH          ;SET IIC CLOCK SOURCE
        MOV     CLOM,A
;
        MOV     XA,#0DH         ;RESET TRANSMIT COUNTER
        PUSH    XA              ;SAVE COUNTER ON STACK
;*
;***************START CONDITION*********************************
;*
        SET1    RELT            ;SET SERIAL LINE HIGH
;
        MOV     XA,#10011000B   ;SET CONTROL REG(SBI EXTERNAL CLOCK)
        MOV     CSIM,XA
;
        CLR1    PORT2.1         ;DISABLE EEPROM CLOCK
```

```
SOURCE STATEMENT

        NOP
        NOP
        NOP
        CLR1    PORT2.2         ;ENABLE CLOCK
;
        MOV     XA,#0FFH        ;DISABLE SERIAL INTERFACE OUTPUT
        MOV     SIO,XA                                  TRANSISTORS
;
        MOV     A,#0EH          ;SYNC IIC CLOCK
SYN1:   SKF     PORT0.1
        BR      SYN1
SYN2:   SKT     PORT0.1
        BR      SYN2
        INCS    A
        BR      SYN1
;
        SET1    PORT2.2         ;DISABLE CLOCK
        MOV     A,#8H
DEL0:   INCS    A
        BR      DEL0
        SET1    PORT2.1         ;ENABLE EEPROM CLOCK
;
        MOV     A,#8H
DEL:    INCS    A
        BR      DEL
        SET1    CMDT            ;SET SERIAL I/O LOW
;
        MOV     A,#04H          ;SYNC DELAY
DEL1:   INCS    A
        BR      DEL1
        NOP                     ;FINE SYNC DELAY
;
        CLR1    PORT2.2         ;ENABLE CLOCK (START CON)
;*
;***************MAIN TRANSMIT*************************************
;*
TRAN1:  XCH     XA,BC           ;LOAD TRANSMIT DATA
        MOV     SIO,XA          ;START TRNSMIT
;
        MOV     A,#0CH          ;DELAY WHEN ON CLOCK TO EEPROM
DEL2:   INCS    A
        BR      DEL2
        SET1    PORT2.1         ;CONTROL LINE HIGH ENABLE CLOCK TO EEPROM
;
        XCH     XA,DE           ;MOVE DE TO BC
        XCH     XA,BC
        XCH     XA,HL           ;MOVE HL TO DE
        XCH     XA,DE
;
        POP     XA
        INCS    A               ;INCS TRANSMIT COUNTER
        BR      WAIT0
        BR      WAIT2
;
WAIT0:  PUSH    XA              ;SAVE TRANSMIT COUNTER
```

SOURCE STATEMENT

```
WAIT1:  SKTCLR  IRQCSI          ;TEST NEXT TRANSMIT FLAG
        BR      WAIT1
        SKT     ACKD
        BR      WAIT3           ;ERROR NO ACKNOWLEDGE
;
        CLR1    PORT2.1         ;CLOCK CONTROL LINE LOW (NO CLOCK TO
        BR      TRAN1                                          EEPROM)
;
WAIT2:  SKTCLR  IRQCSI
        BR      WAIT2
;
        SKT     ACKD            ;CHECK FOR ACKNOWLEDGE BIT
        BR      WAIT3           ;NO ACKNOWLEDGE FROM EEPROM
;*
;***************SET STOP CONDITION*****************************
;*
        SET1    CMDT            ;SET SO LOW FOR STOP CON
        MOV     A,#0FH          ;DELAY >4.7US
DEL3:   INCS    A
        BR      DEL3
        SET1    PORT2.2         ;SET CLOCK HIGH
;
        MOV     A,#0FH          ;DELAY >4.7US
DEL4:   INCS    A
        BR      DEL4
        SET1    RELT            ;SET SERIAL I/O LINE HIGH
;
WAIT3:  NOP
        BR      WAIT3
        END
```

Software Listing Read Program

SOURCE STATEMENT

```
                NAME    IICRD
;IIC BUS READ ONLY PROGRAM WRITTEN FOR 75308 BY B.GOUGH 8/87
;*************************************************************************
;NAME            :IICSTR
;DESC            :IIC BUS READ PROGRAM
;DEVICE          :75308
;MEMORY BANK     :15
;REGISTER BANK   :-
;USED HARDWARE   :SERIAL PORT
;INPUT           :BC(DEVICE ADDRESS),DE(EEPROM ADDRESS)
;OUTPUT          :HL(DATA READ)
;DESTROY         :XA
;CALLS           :-
;************VECTOR TABLE*************************************************
        STKLN   40H
        VENTO   MBE=1,RBE=0,IICSTR      ;START VECTOR
;************GETI TABLE***************************************************
        GET1    CSEG    IENT
        SELM15: SEL     MB15
;************SET UP PROGRAM ***********************************************
C1SEG   CSEG    INBLOCK
IICSTR: GETI    SELM15
        MOV     XA,#STACK       ;SET STACK
        MOV     SP,XA
        MOV     A,#0011B        ;SET HIGH SPEED SYSTEM CLOCK
        MOV     PCC,A
;
        MOV     XA,#05H         ;SET PULL UPS BECAUSE OF OPEN DRAIN BUS
        MOV     POGA,XA
;
        SET1    PORT2.1         ;SET CLOCK CONTROL LINE HIGH
        SET1    PORT2.2         ;SET CLOCK LINE HIGH
        MOV     XA,#04H         ;SET IIC TIMER PORT
        MOV     PMGB,XA
;
        MOV     A,#0FH          ;SET IIC TIMER 65.5KHZ
        MOV     CLOM,A
;
        MOV     XA,BC           ;SAVE DEVICE ADDRESS READ(2)
        PUSH    BC
        AND     A,#1110B
        XCH     XA,BC           ;DEVICE ADDRESS READ(1)
;
        MOV     XA,#0EH         ;RESET TRANSMIT COUNTER
        PUSH    XA
;*
;****************START CONDITION(1)*************************************
;*
        SET1    RELT            ;SET SERIAL LINE HIGH


        MOV     XA,#10011000B   ;SET CONTROL REG(SBI EXTERNAL CLOCK)
        MOV     CSIM,XA
```

```
;
        CLR1    PORT2.1         ;DISABLE EEPROM CLOCK
        NOP
        NOP
        NOP
        CLR1    PORT2.2         ;ENABLE CLOCK
;
        MOV     XA,#0FFH        ;DISABLE SERIAL INTERFACE OUTPUT
        MOV     SIO,XA                                  TRANSISTORS
;
        MOV     A,#0EH          ;SYNC IIC CLOCK
SYN1:   SKF     PORT0.1
        BR      SYN1
SYN2:   SKT     PORT0.1
        BR      SYN2
        INCS    A
        BR      SYN1
;
        SET1    PORT2.2         ;DISABLE CLOCK
        MOV     A,#8H
DEL0:   INCS    A
        BR      DEL0
        SET1    PORT2.1         ;ENABLE EEPROM CLOCK
;
        MOV     A,#8H
DEL:    INCS    A
        BR      DEL
        SET1    CMDT            ;SET SERIAL I/O LOW
;
        MOV     A,#4H           ;SYNC DELAY
DEL1:   INCS    A
        BR      DEL1
        NOP                     ;FINE SYNC DELAY
;
        CLR1    PORT2.2         ;ENABLE CLOCK (START CON)
;*
;*****************PROGRAM TO READ FROM EEPROM(1)*************************
;*
        XCH     XA,BC
TRAN1:  MOV     SIO,XA          ;START TRANSMIT
;
        MOV     A,#0CH          ;DELAY WHEN NO CLOCK ON EEPROM
DEL2:   INCS    A
        BR      DEL2
        SET1    PORT2.1         ;CONTROL LINE HIGH ENABLE CLOCK TO EEPROM
;
        POP     XA
        INCS    A               ;INCS TRANSMIT COUNTER
        BR      WAIT0
        BR      WAIT2
;
WAIT0:  PUSH    XA              ;SAVE TRANSMIT COUNTER
```

```
SOURCE STATEMENT

WAIT1:  SKTCLR  IRQCSI          ;TEST NEXT TRANSMIT FLAG
        BR      WAIT1
        SKT     ACKD
        BR      WAIT3           ;NO ACKNOWLEDGE FROM EEPROM
;
        CLR1    PORT2.1         ;CLOCK LINE LOW (NO CLOCK TO EEPROM)
        XCH     XA,DE           ;TRANSMIT DATA(EEPROM ADDRESS)
        BR      TRAN1
;
WAIT2:  POP     BC              ;RECOVER READ ADDRESS(2)
        PUSH    XA              ;SAVE TRANSMIT COUNTER
WAIT4:  SKTCLR  IRQCSI          ;TEST NEXT TRANSMIT FLAG
        BR      WAIT4
        SKT     ACKD            ;CHECK ACKNOWLEDGE BIT FROM EEPROM
        BR      WAIT3           ;NO ACKNOWLEDGE FROM EEPROM
;*
;***************START CONDITION (2)*******************************
;*
        CLR1    PORT2.1         ;DISABLE CLOCK ON EEPROM
        SET1    PORT2.2         ;CLOCK LINE HIGH
        NOP
        NOP
        SET1    RELT            ;DATA LINE HIGH
        NOP
        NOP
        SET1    PORT2.1         ;ENABLE CLOCK ON EEPROM
        NOP
        NOP
;
        SET1    CMDT            ;SET CONTROL REG BIT CMDT(START CON)
;
        MOV     A,#09H          ;SYNC DELAY
DEL3:   INCS    A
        BR      DEL3
        CLR1    PORT2.2         ;ENABLE CLOCK (START CON)
;*
;***************PROGRAM TO READ FROM EEPROM (2)*********************
;*
        XCH     XA,BC           ;LOAD TRANSMIT DATA
        MOV     SIO,XA          ;START TRANSMIT
WAIT5:  SKTCLR  IRQCSI          ;TEST NEXT TRANSMIT FLAG
        BR      WAIT5
        SKT     ACKD            ;CHECK ACKNOWLEDGE BIT FROM EEPROM
        BR      WAIT3           ;NO ACKNOWLEDGE FROM EEPROM
;
        CLR1    PORT2.1         ;STOP CLOCK TO EEPROM
        SET1    PORT2.2         ;CLOCK HIGH AT MICRO(SET INPUT MODE)
        MOV     XA,#0FFH        ;TURN OFF N CHANNEL TRANSISTORS
        MOV     SIO,XA
        MOV     XA,#00000000B   ;DISABLE SERIAL INPUT
        MOV     CSIM,XA
        MOV     L,#7H           ;SET SERIAL INPUT COUNTER
        CLR1    PORT2.2         ;ENABLE CLOCK
        SET1    PORT2.1         ;ENABLE CLOCK ON EEPROM
```

SOURCE STATEMENT

```
;*NOTE :SERIAL INPUT BY POLING BECAUSE THE SBI BUS CAN NOT INPUT IIC
;*                                                              FORMAT
POLE1:  SKT     PORT0.1         ;POLE CLOCK INPUT
        BR      POLE1
        SKT     PORT0.3         ;INPUT DATA
        BR      POLE2
        SET1    BSB0.3L         ;DATA=1
        DECS    L
        BR      POLE1
        BR      OUT1
POLE2:  CLR1    BSB0.3L         ;DATA=0
        DECS    L
        BR      POLE1
;
OUT1:   MOV     XA,BSB0
        XCH     XA,HL           ;RECEIVE DATA IN HL
;*
;***************SET STOP CONDITION*************************************
;*
        MOV     XA,#10011000B   ;RESET SBI BUS MODE TO USE CMDT
        MOV     CSIM,XA
;
        SET1    CMDT            ;SET SO LOW FOR STOP CON
        MOV     A,#0FH          ;DELAY >4.7US
DEL4:   INCS    A
        BR      DEL4
        SET1    PORT2.2         ;SET CLOCK HIGH
;
        MOV     A,#0FH          ;DELAY >4.7US
DEL5:   INCS    A
        BR      DEL5
        SET1    RELT            ;SET CONTROL REG BIT RELT (STOP CON)
WAIT3:  NOP                     ;WAIT HERE IF NO ACKNOWLEDGE
        BR      WAIT3
        END
```

Interfacing the CMOS 32-Bit Microprocessor V60 to Dynamic RAM

Contents:

Author: Thomas Müller
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Microprocessors and Peripherals	Data Book
Memory Products 1987	Data Book
μPD70616 (V60)	Data Sheet
μPD71611	Data Sheet
μPD71613	Data Sheet

Related Products

μPD70616	32-bit Microprocessor	CMOS
μPD71611	Clock Generator	CMOS
μPD71613	Bus Controller	CMOS
μPD41256	Dynamic RAM	NMOS

1. Introduction

The purpose of this application note is to show an interface between the V60 CPU and dynamic RAM's μPD41256.

As μPD41256 support the CAS before RAS (CBR) refresh mode, it eases the interface because no external refresh addresses need to be generated externally for their refresh.

The dynamic RAM control application example has been built on a single board computer with V60 CPU described in application note no. μcom 31.

The DRAM controller is built around a delay line which generates the time critical RAS — MUX — and CAS — signals for dynamic RAM array.

The refresh signal is generated by a simple RC oscillator. A simple arbitration logic governs the read/write access from the CPU and the refresh request from the RC oscillator.

Although this may not be the most elegant solution, it is easy to understand and can be implemented with off the shelf components.

2. Dynamic RAM Access

The DRAM's used in this application are of type μPD42256 (256 k *1 bit), which can be used in different modes to read or write data. The used mode is the EARLY WRITE mode, which means all bits are written with a complete access of RAS*/CAS* generation and full addresses. The data is strobed into the DRAM by the CAS* signal.

The timing diagram of the DRAM's is shown in figure 2.2. For more information, please refer to the MEMORY DATA BOOK 1987.

The μPD42256 DRAM has only a 9-bit address, but has RAS* and CAS* signal inputs to split the addresses in rows and columns. Each row — and column — address use an 9-bit address input to achieve the full 18-bit address, therefore it is necessary to multiplex the address bus. With the falling edge of the RAS* signal, the DRAM receives the row address, and then the address multiplexer switches to the lower address bits, which are read 30 ns to 60 ns later with the falling edge of the CAS* signal. During this time the operation modes of read or write access differ. The data will be written to the DRAM's as soon as the address is complete.

When a read access is performed, the memory gives the data to the bus 60 ns later, however, the V60 CPU reads the data only with the falling edge of the T4 clock cycle. This may occur earlier than the data becomes valid (for example 16 MHz, 3 clock bus cycle), or later. In the first case, the CPU has to wait until the access is completed, or the DRAM has to serve the CPU for longer than 60 ns. To get a dynamically driven DRAM, the read/write access is finished with the rising edge of the BCY* signal, and for the second case the CPU inserts WAIT states for a minimum time of 60 ns.

There are two ways to refresh the DRAM — RAS — ONLY and CAS — Before — RAS (CBR) refresh mode. To use the RAS — ONLY refresh, it is first necessary to generate a 8 bit refresh address everytime the refresh is needed.

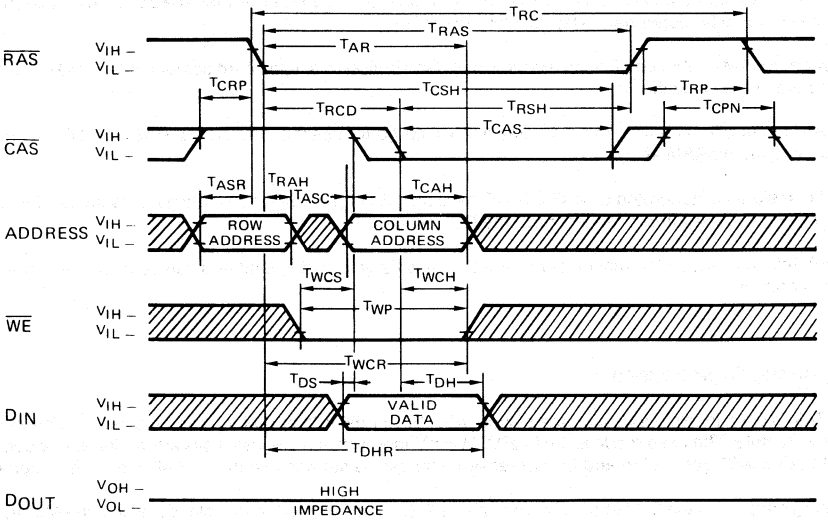
In this application the CBR refresh mode is used, because it is simple to handle and does not need any refresh address. The RAS — ONLY refresh mode requires additional hardware to generate the refresh addresses. The refresh is realized with the same controller which generates RAS* or CAS* signals with the only difference being that the RAS*, CAS* signal generation sequence is reversed by a multiplexer. For the controller it is a simple read access.

To generate these three signals; RAS*, CAS* and MUX (multiplexer signal), a delay line is used. In this application a line of 200 ns delay is used, which has an impedance of 100 Ω.

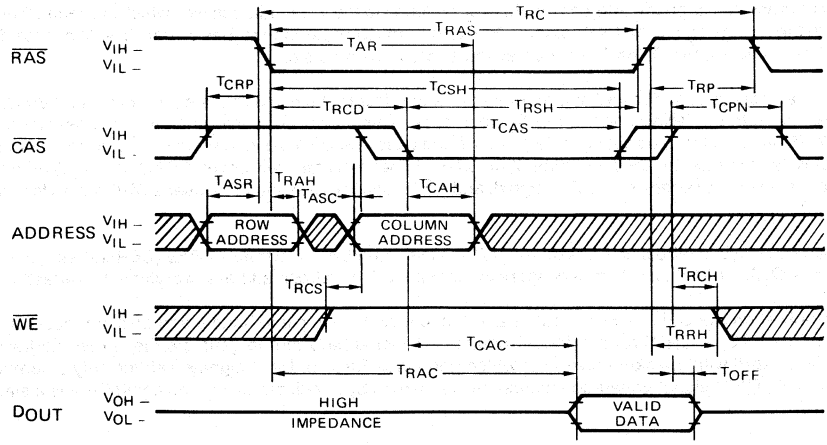
The following pages show the timing diagrams of DRAM and V60 CPU access.

Note: "*" = active low

WRITE CYCLE (EARLY WRITE)



READ CYCLE



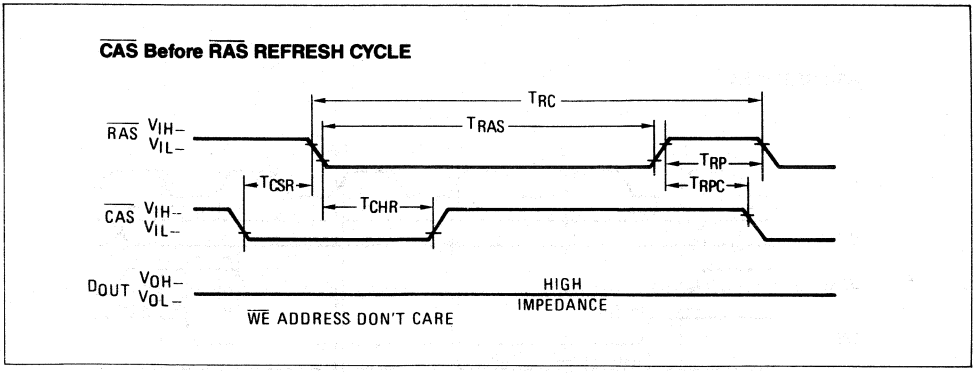
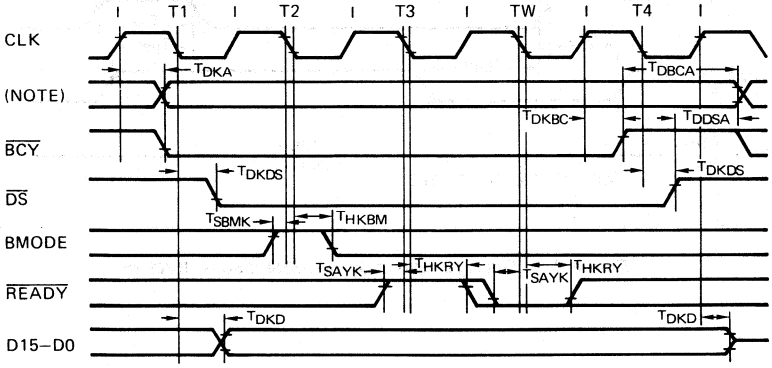


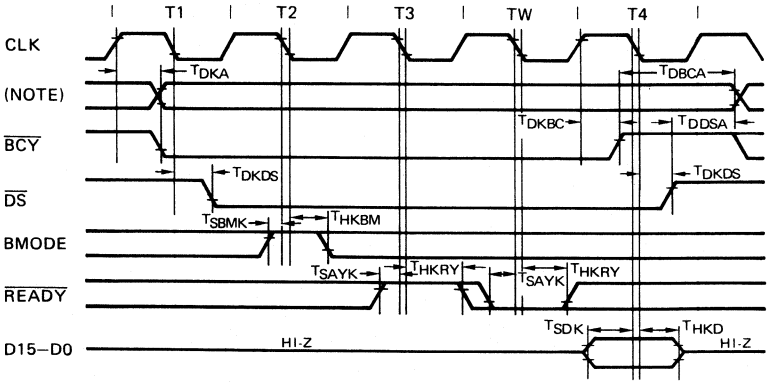
Figure 2.1 Timing Diagram of the μPD42256 — 10/12/15

WRITE TIMING



NOTE – A23–A0, $\overline{UBE}$, R/$\overline{W}$, ST2–ST0, $\overline{MRQ}$, DL1–DL0, $\overline{FAS}$

READ TIMING



NOTE – A23–A0, $\overline{UBE}$, R/$\overline{W}$, ST2–ST0, $\overline{MRQ}$, DL1–DL0, $\overline{FAS}$

Figure 2.2 Timing Diagram of the Read/Write Cycles of the V60

3. DRAM Controller

3.1 Function Overview

The controller consists of five different blocks:

- DRAM Access Arbitration Logic
  This block arbitrates between the refresh request and the DRAM access request from the CPU.
- Delay Line
  This unit generates the DRAM control signals like RAS, CAS, multiplex (MUX) and the WAIT signal for the CPU.
- Signal Generator
  This unit generates the RAS* and CAS* signals for the DRAM access and is served by the Delay Line. This unit is also responsible for reversing the RAS* and CAS* signal generation sequence during CBR refresh mode.
- Refresh Generator
  The generator is a 64 kHz RC oscillator and generates the refresh request signal for the DRAM.
- DRAM Interface
  Interface to the DRAM, which selects the even and odd address banks of the DRAM. Further, it multiplexes the address lines and controls so that both address banks are selected simultaneously during a refresh cycle.

The DRAM Interface is realized with a PAL (type 16L8), which splits the RAS and CAS signal into even and odd address banks. The boolean equations are shown below.

```
RASEV   =  !( !RAS & REFRESH  #  !A0 & !DARAM & !RAS)
RASOD   =  !( !RAS & REFRESH  #  !DRAM & !RAS & !UBE)

CASEV   =  !( !CAS & REFRESH  #  !A0 & !DRAM & !CAS)
CASOD   =  !C !CAS& REFRESH  #  !UBE & !DRAM & !CAS)
```

Definitions:

= equal to . . .
! NEGATION
& logical AND
# logical OR

RASEV, RASOD	Outputs, RAS for even and odd address banks.
CASEV, CASOD	Outputs, CAS for even and odd address banks.
RAS, CAS	Inputs from Switching Unit of the DRAM Controller.
REFRESH	Input from Refresh Generator.
A0, UBE	Inputs, from VC60 CPU to select address banks for byte or word operations.
DRAM	Input, from Address Decoding Unit of the V60 SBC.

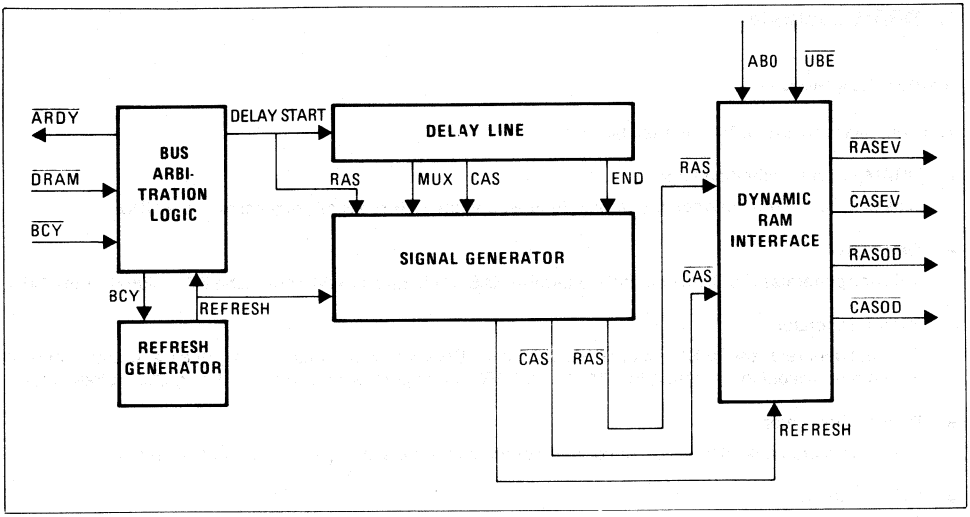


Figure 3.1 Block Diagram of the DRAM Controller

3.2 RAS/CAS Generation

The RAS/CAS generation is built up with a delay line which has a total delay of 200 ns, parted into 10 single delays each with 20 ns ± 3 % delay time. The delay line generates only the switching signals for the signal generator starting with a refresh cycle or DRAM access request. A simplified function diagaram is shown below.

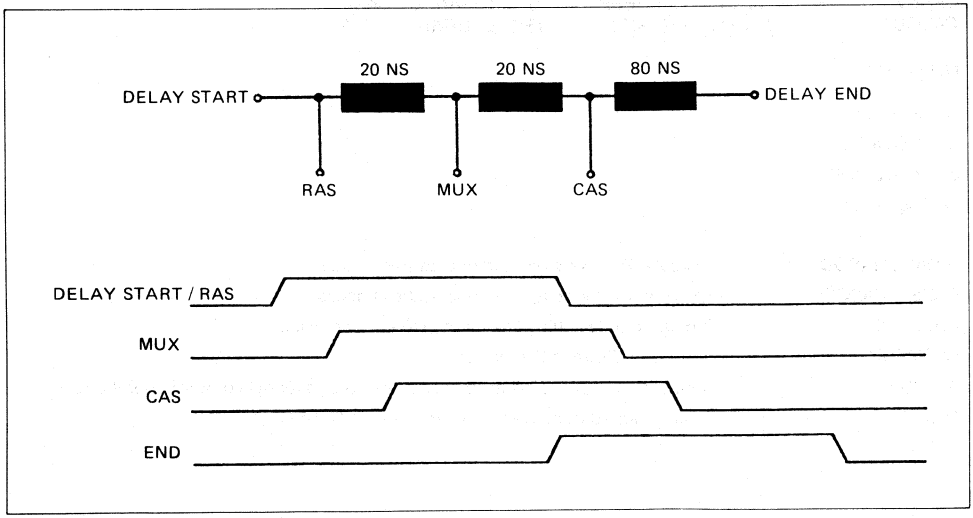


Figure 3.2 Simplified Function Diagram of the Delay Line

An active high pulse is input to the delay line which generates the RAS signal, and then is formed to RAS*. After 20 ns the wave reaches the point MUX, which switches the address multiplexers to low address lines. Further, 20 ns later, the point CAS becomes active high, and after another 60 ns later, the end of the line is reached. Now all points of the line are active high.

These signals are used to drive the signal generator. This generator consists of D-Flip-Flop's and generates active low signals. The Flip-Flop's are driven through the C-input and switch the logical 1 at the D-input to the output Q. At the same time Q* becomes active low and will stay in this state until a reset is given to them. At this point, when the processor has read or written the data, the Flip-Flop's must be reset.

Below a simplified schematic diagram of the RAS* and CAS* generation is shown. The Flip-Flop's were set by the signals of the delay line which starts with a DRAM request as active high.

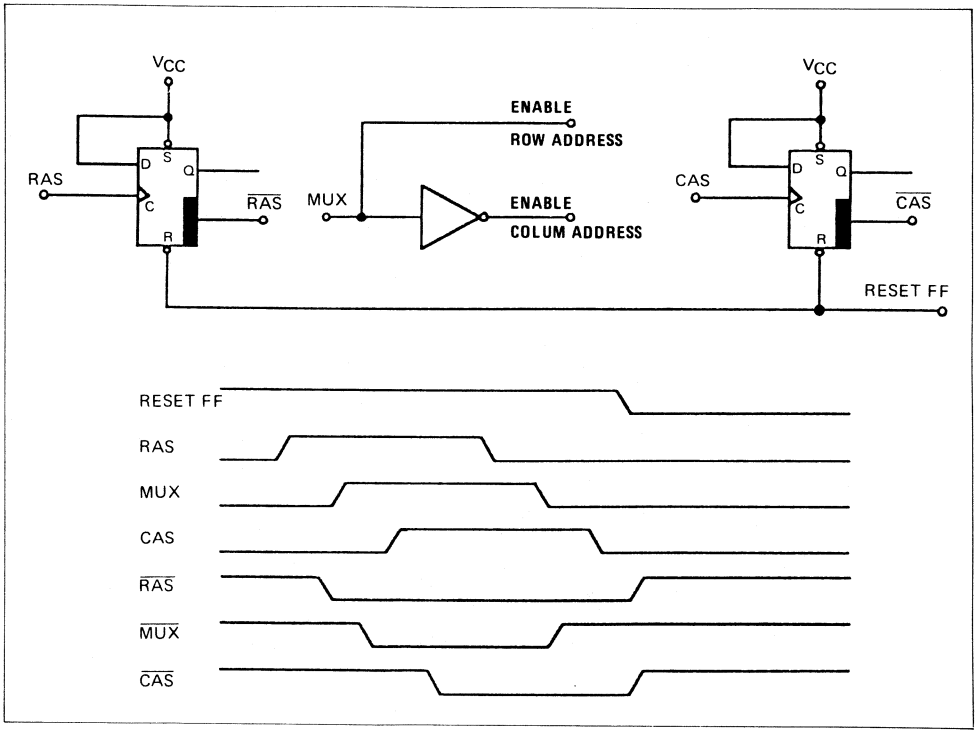


Figure 3.3 Generation of RAS*/CAS*

To end a cycle, the RAS* and CAS* active signals must be reset when the V60 CPU is ready to read the data. The precharge time of the DRAM is handled by the CPU giving the next DRAM request. This is realized by using the BCY*, which turns to high when the CPU ends the machine cycle. The inverted signal is used to reset the signal generation and RAS*, CAS* and MUX reach passive high. On the next pages, the timing diagrams of the controller are shown for the CPU read — and write — access.

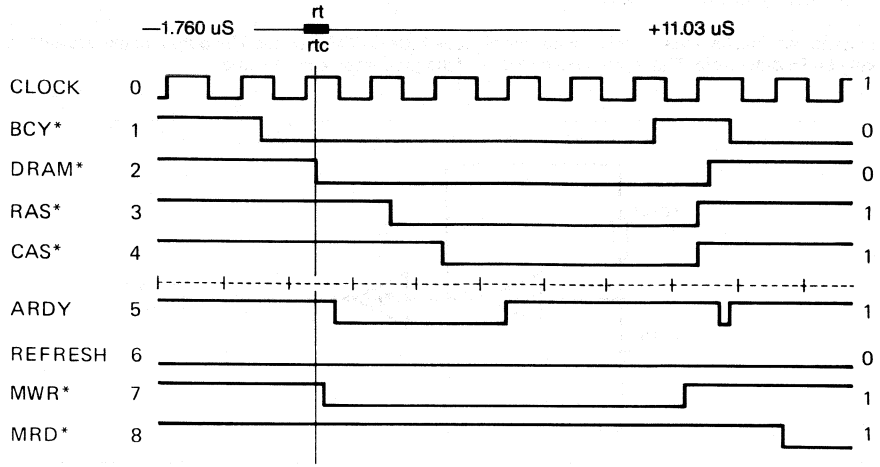
Signal Description:

CLOCK	System Clock 16 MHz
BCY*	Processor Bus Cycle signal
RAS*	Row Address Select, active low
CAS*	Column Address Select, active low
ARDY	Async. Ready Input of V60 clock Controller, active high
REFRESH	Refresh signal, active high
MWR*	Memory Write, active low
MRD*	Memory Read, active low

WRITE ACCESS

TIMING WAVEFORM DISPLAY ITA 2200 ACQ DATA
Timebase: 10.00 nS M pod: +2.50V L pod: +2.5V
Mag: 60.00 nS/div

c to r: 0.000 nS
c to t: 0.000 nS



READ ACCESS

TIMING WAVEFORM DISPLAY ITA 2200 ACQ DATA
Timebase: 10.00 nS M pod: +2.50V L pod: +2.50V
Mag: 60.00 nS/div

c to r: 0.000 nS
c to t: 0.000 ns

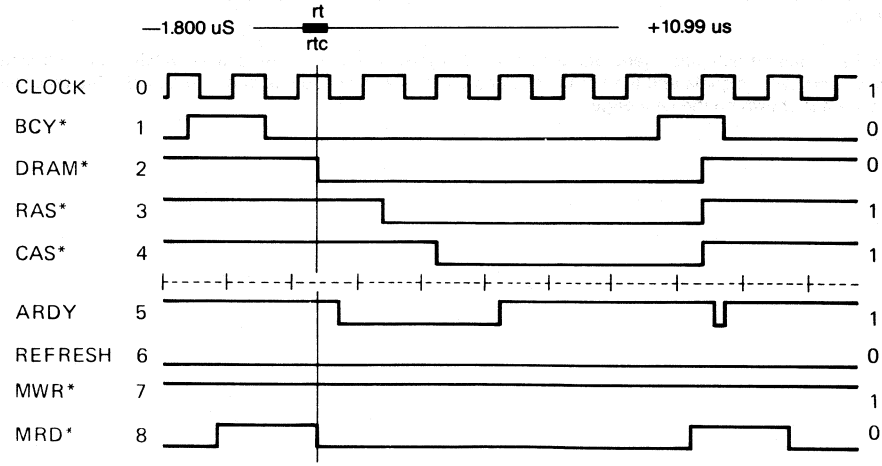


Figure 3.4 Timing Diagram of the DRAM Controller

3.2 Refresh Generator

This generator uses the CBR refresh. It is the simplest way to refresh μPD42256 or equivalents types of DRAM, because no refresh address is required by the DRAM. The refresh cycle time is 256 x 1 refresh address per 4 ms, i.e. 64 refreshs per 1 ms.

- Refresh Cycle Time 15,625 μs

The generator is a low-cost RC-Generator, which uses 7404 CMOS inverter. The output of the oscillator pulse has nearly 50 % duty cycle. The refresh needs 1.6 % of the processor working time.

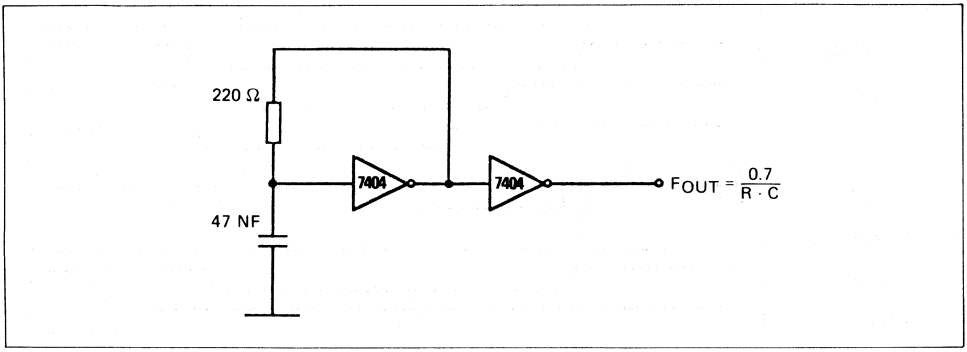


Figure 3.5 Schematic Circuit Diagram of the RC-Oscillator

Timewise, the generated pulse takes much longer than the refresh. Therefore the refresh request is stored in a D-Flip-Flop and synchronized with the rising edge of the BCY signal. During the refresh cycle the WAIT signal is held active low and the CPU inserts WAIT cycles until the end of the refresh cycle is reached. This is done to prevent bus errors.

Internally, the refresh pulse generates a normal read cycle, but the r/w — refresh — switch — unit interchange the RAS* and CAS* signals and the DRAM interface unit enables all address banks to refresh. The recorded timing diagram is shown on the next page.

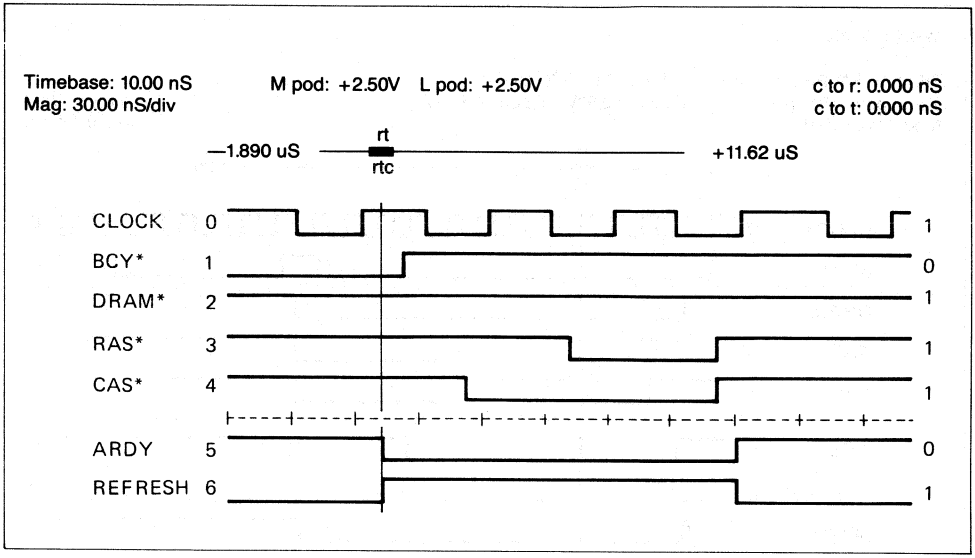


Figure 3.6 Timing Diagram of Refresh

If a DRAM access request and a refresh request start at the same time, first the refresh is performed and the DRAM signal is stored. At the end of a refresh cycle the stored DRAM signal is released and can pass over the delay line like a normal read or write access. The circuit used to store this is shown below. The Refresh is saved in the left Flip-Flop (figure 3.7). At the end of the refresh, when all Flip-Flop's get a reset, the falling edge of the wave which runs through the delay line switches the right Flip-Flop in case the DRAM is active high. The CPU WAIT line is then again enabled and a new request starts.

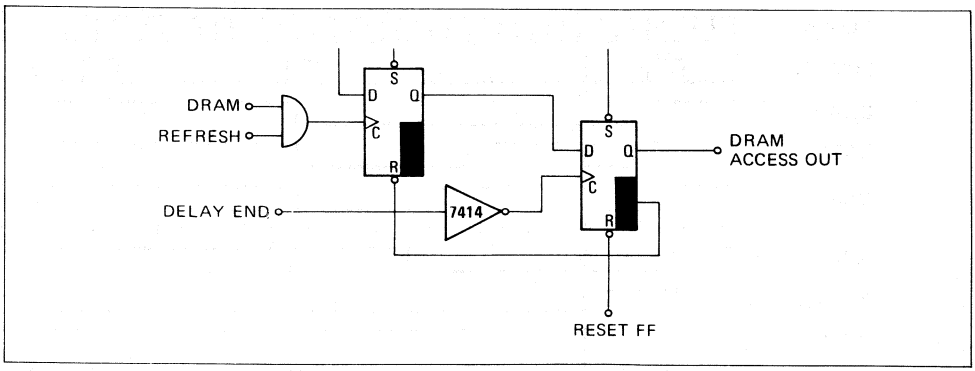
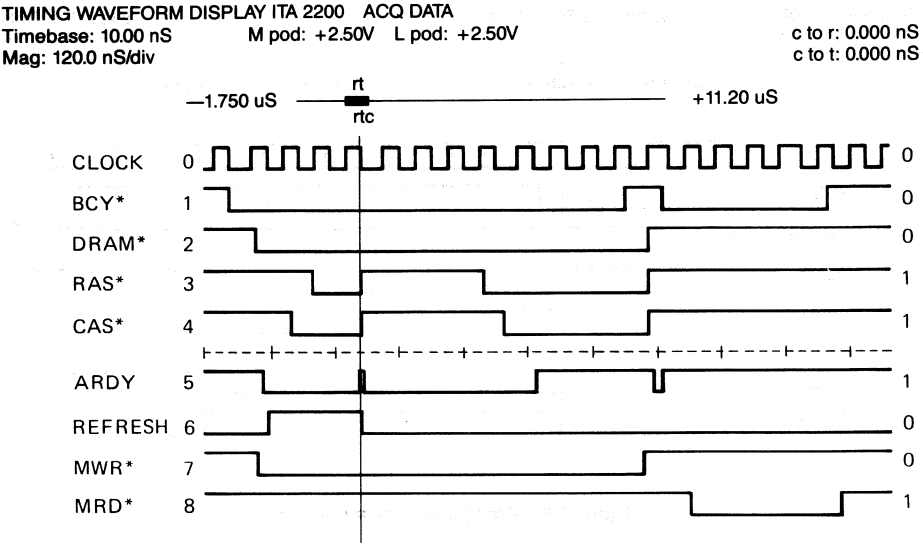


Figure 3.7 Schematic Circuit Diagram of DRAM Restore

WRITE ACCESS



READ ACCESS

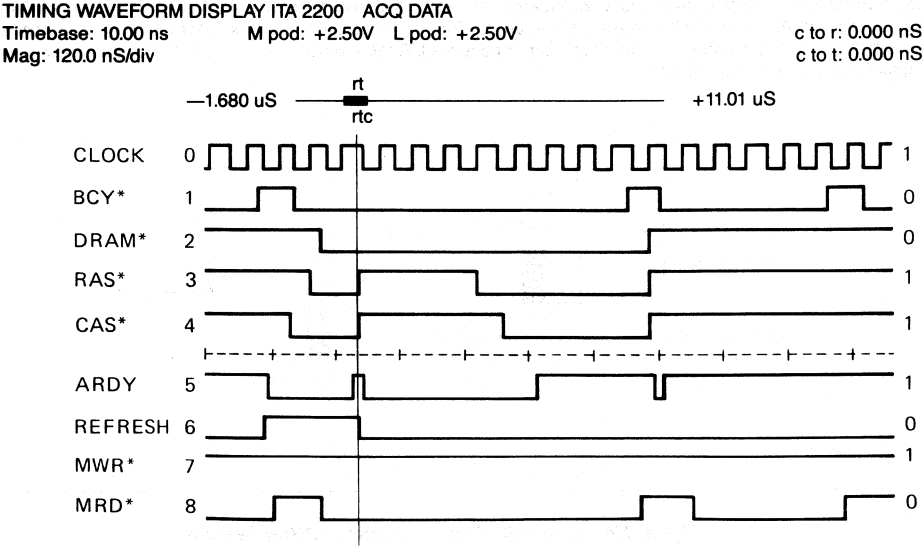


Figure 3.8 Timing Diagram of Refresh corresponding of DRAM acknowledge

3.3 DRAM Access Arbitration Logic

This part of the controller checks the Bus Control Lines BCY*, MRD*, MWR*, DS* and ARDY. The ARDY is an input of the Clock Generator μPD71611 and generates wait cycles for the V60. This line is passive at every DRAM access by the V60, i.e. the WAIT states are dynamically driven by the controller.

WAIT has to be active (WAIT = 0), if:

— DRAM access is requested or
— REFRESH is active

WAIT has to be passive (WAIT = 1), if:

— The minimum time of CAS* to data ready is performed or
— no DRAM access is requested or
— REFRESH is passive

Everytime a DRAM access or refresh cycle starts, the ARDY input of the V60 Clock Controller is driven low to insert WAIT states for the V60 CPU because the V60 is faster than the DRAM can serve the data bus. This ARDY (Asynchronous Ready, active high) is driven high again when the DRAM access pulse (active high) on the delay line reaches the end. The V60 ends its machine cycle and turns BCY* to high.

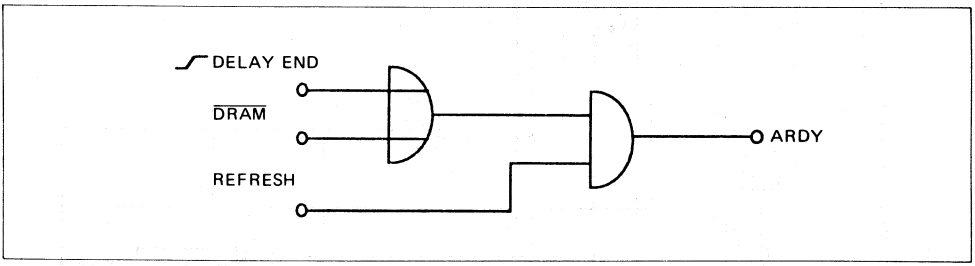


Figure 3.9 WAIT Signal Generation

4. Line Termination

The termination of the delay line must be done with proper line impedance (in this application 100 Ω). In this case a termination with a 180 Ω resistor to Vcc and 200 Ω to Ground was taken. Parallel to the resistor connected to ground, a 150 nF Ceramic capacitor is connected. To reduce the noise on the line a 50 pF Capacitor is connected from the gain of the delay line to ground. Each line of the V60 address — and data — bus is terminated with 220 Ω resistor to Vcc and a 330 Ω resistor to ground.

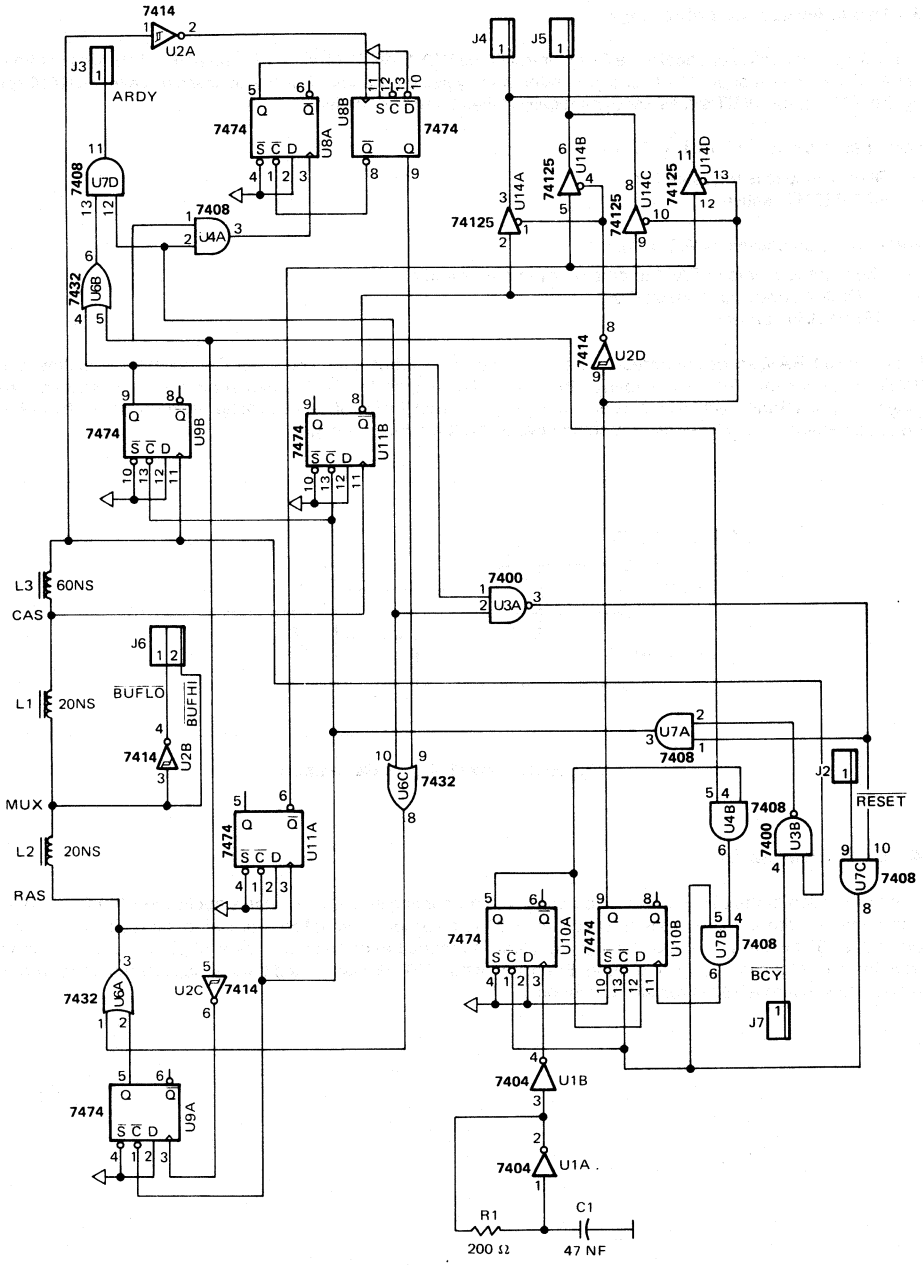


Figure 3.10 Schematic Circuit Diagram of the DRAM Controller

μPD78310 / 78312 Macro Service Function Examples

Contents:

Author: NEC Electronics Inc.
NATICK/USA

Persons to contact: R. Cherrington, W. Knippschild
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

Product Description
μCOM78K I/II/III 8/16 Bit Microcomputer

Related Products

μPD78310/312 8/16 Bit Microcomputer	CMOS

The μPD78312/78310 gives the programmer the option of servicing interrupt requests in any one of three different ways. The first is standard vectored interrupt servicing; the second is Context Switching, in which the program counter and program status word are saved and a new bank of general registers is selected by the hardware; and the third is Macro Service. This application note describes the Macro Service facility.

1. General Description

The Macro Service facility responds to an interrupt request by transferring either a byte or a word of data between any specified special function register and memory in whichever direction is specified. This is done for the specified number of interrupt requests (up to 255), and then a completion routine is called. The transfer is made by hardware during cycles "stolen" from the executing program, but the program counter and program status word are not altered, so the transfer is completely transparent to the executing program. The effect is the same as that of a direct memory access channel, but it is not as fast as a hardware DMA. Of the 15 sources of maskable interrupt requests on the μPD78312, 11 can be used to initiate Macro Service transfers.

Transfers can be made to or from any one of the special function registers, and this includes the I/O ports, all the registers associated with the timers and counters, and all the mode and control registers. The width of the transfer (byte or word) is determined by the width of the special function register and by the specification in the Macro Service control register. Both must be 16 bits to specify a word transfer. The table of special function registers in the μPD78312 user manual specifies which are valid for 16 bit transfers. External circuits interfaced to the External Access Area (FFBOH to FFBFH) become special function registers, and therefore Macro Service transfers can be made to and from these circuits. In this case transfers are made over the external data bus, which is only 8 bits wide, and therefore byte transfers only are possible.

The Macro Service facility is an extremely powerful tool whose uses are limited only by the imagination of the programmer. A typical use of the Macro Service facility would be to receive a line of ASCII characters from the serial communications port. The programmer would specify the number of characters in the line, the location of the buffer in memory, and the address of the serial receive buffer. After loading the necessary control registers, he would enable the serial port and return to the background program. The interrupt service routine would be entered only after the complete line of data had been received. The service routine would then process the data and set up the transfer of the next line.

Another example of its use would be to generate a square wave of arbitrarily varying duty cycle and repetition rate. To do this a timing table would be generated and stored in memory. The table would contain alternating times for high and low output. The Macro Service facility would then be used to re-load the timer modulus register from the memory table at each timer interrupt request. The output would then be available at the timer output pin TO0 or TO1.

A final example is the use of the Macro Service facility to extend the range of the timer. At slow speed the timer counts SCLK/128, or 46875 Hz for a 12 MHz crystal. The timer counter has a capacity of 16 bits, and therefore the longest interval is 1.398 sec. This interval can be extended by using the Macro Service counter as the high order 8 bits of the timer counter, thus creating a 24 bit counter and extending the maximum interval to 357.9 sec. To do this the programmer specifies a dummy transfer (e.g. transfer the contents of any SFR to an unused memory location without incrementing the memory pointer, MSP). He then loads the timer modulus register, MD0 (or MD1) with "m" and the Macro Service counter, MSC, with "n" such that:

$$n\,(m+1)\,t = I$$

where "I" is the desired interval, "t" the timer clock period (6/SCLK or 128/SCLK), and "m" and "n" are integers. Finally he defines the interrupt vector (or context switch) and enables the timer and the interrupt. The interrupt service routine will then be entered at the completion of the specified interval. The resolution of the interval so specified is n x t. It should be noted that the timer counts for one clock period longer than the number loaded into the modulus register.

2. Using the Macro Service Facility

There are 8 macro service channels, and associated with each is a 4 byte storage area in on-chip RAM. Each of these areas contains a 2-byte memory address pointer MSPn, a 1-byte special function register pointer SFRPn, and a 1-byte counter, MSCn, where the channel number, "n", ranges from 0 to 7. MSCn is decremented, and MSPn is optionally incremented after each transfer. The storage for all 8 channels is occupies the area from FEE0H to FEFFH, and shares the space with register banks 0 and 1. It is therefore not possible to use context switching to service interrupts of priority 0 and 1 if all the macro service channels are in use. The location and configuration of the macro service channels is illustrated in figure 2.

Immediately after the macro service counter reaches zero a normal interrupt is executed. This may be either a vectored interrupt or a context switch, as specified in the associated interrupt control register. In the case of priorities 0 and 1 the context switch is dependent upon the availability of the necessary register bank. The routine is used for any processing of data required by the application and to setup the macro service channel for the next transfer block, if one is desired. The macro service pointer, MSP, the macro service counter, MSC, and the MS/INT bit (bit 5) in the associated interrupt control register must all be reinitiailzed by this routine in order to enable the next transfer block.

Before the Macro Service facility can be used, a number of registers must be initialized. These include the mode and/or control registers for the peripheral device being used, the pertinent interrupt control register, the pertinent Macro Sercive control register, and the Macro Service channel. After these registers have been set, the peripheral device is enabled and the IE bit in the PSW is set to one. Refer to the μPD78312/78310 user manual for a description of the mode and control registers for the various peripheral devices. A list of interrupt sources and a description of the interrupt control registers, the Macro Service control registers, and the Macro Service channels follow. It should be noted that the interrupts are divided into groups and that the priority for the entire group is always set in the interrupt control register for the first interrupt in each group. Thus if any interrupt in a group is used, the control register for the first interrupt of the group must always be loaded to set the priority. If the first interrupt is not used, it should be masked by setting bit 6 of its control register to one.

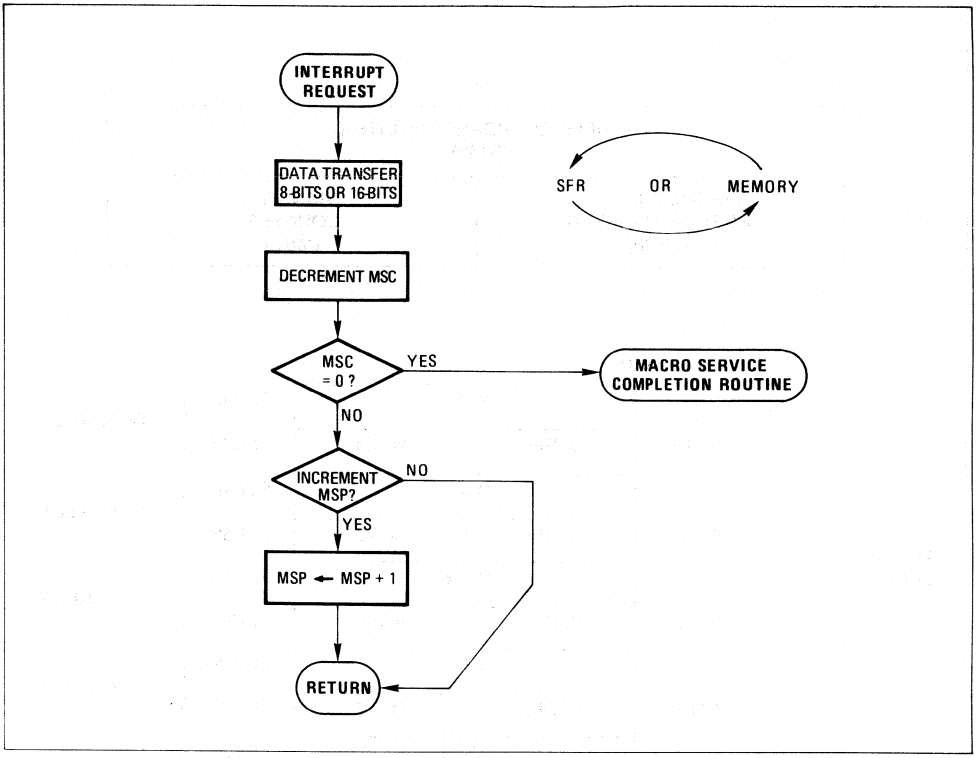
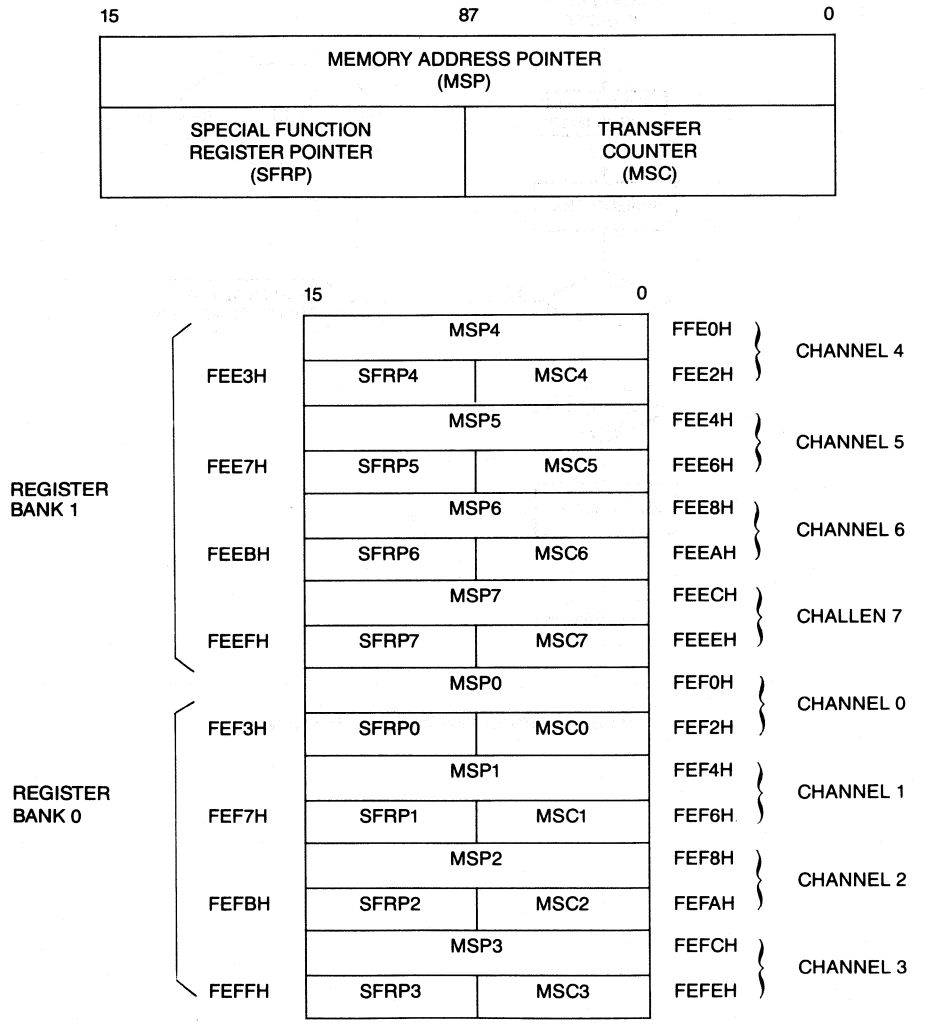


Figure 1. Macro Service Operation Flow



Note: The macro service pointers share storage with register banks 0 and 1.

Figure 2. Macro Service Channels

Table 1. Interrupt Sources and Vector Addresses

	DEFAULT PRIORITY		INTERRUPT SOURCE	MACRO SERVICE	VECTOR ADDRESS
	—	RESET	EXTERNAL RESET LINE	—	0000H
NON-MASKABLE INTERRUPTS	—	NMI	EXTERNAL NON—MASKABLE INT.	N	0002H
	—	WDT	WATCHDOG TIMER	N	000AH
MASKABLE INTERRUPTS	0	CRF00	UP/DOWN COUNTER	Y	001AH
	1	CRF01	UP/DOWN COUNTER	N	001CH
	2	CRF10	UP/DOWN COUNTER	Y	001EH
	3	CRF11	UP/DOWN COUNTER	N	0020H
	4	EXIF0	EXTERNAL INTERRUPT 0	Y	0004H
	5	EXIF1	EXTERNAL INTERRUPT 1	Y	0006H
	6	EXIF2	EXTERNAL INTERRUPT 2	Y	0008H
	7	TIMF0	TIMER FLAG 0	Y	000EH
	8	TIMF1	TIMER FLAG 1	Y	0010H
	9	TIMF2	TIMER FLAG 2	Y	0012H
	10	SEF	SERIAL PORT ERROR	N	0022H
	11	SRF	SERIAL PORT RECEIVE BUFFER	Y	0024H
	12	STF	SERIAL PORT TRANSMIT BUFFER	Y	0026H
	13	ADF	A/D CONVERTER DONE FLAG	Y	0028H
	14	TBF	TIMEBASE COUNTER FLAG	N	000CH
	—	BRK	BREAK INSTRUCTION	N	003EH

Interrupt Control Registers: xxICn

There is an interrupt control register for each of the 15 maskable interrupts. These are 8-bit read/write registers which are set to 47H by external RESET. This setting masks each interrupt and sets each to the lowest priority level. Prioroties are assigned to the interrupts by groups, and the control register for the first interrupt within each group is used to define the priority for the entire group.

Bits 0—2: PR0—PR2

The three low order bits, PR0—PR2, specify the priority of the interrupt. Values range from 0 for the highest priority to 7 for the lowest priority. These bits are used in the first interrupt control register of each group only and specify the priority for the entire group. Priorities within the group (or within any major priority level if more than one group is assigned to the same priority level) are defined by the order in which the interrupts appear in table 1. The within-group priorities, however, are effective only if two or more interrupt requests become active within the same instruction time.

Bit 3: Not Used

Bit 4: ENCS

The ENCS bit enables the context switch. If ENCS is zero, the interrupt is serviced by vectoring. If ENCS is one, the interrupt is serviced by context switch.

Bit 5: MS/INT

Bit 5 is used to specify the macro service function. If bit 5 is zero, the interrupt service routine is entered directly at every interrupt request. If bit 5 is one, a macro service transfer is performed and the macro service counter is decremented. The interrupt service routine is entered only when the counter reaches zero. This bit is cleared to zero by the hardware at the end of each block (when the counter goes to zero) and must be reinitialized to one by software before this interrupt can be used to initiate additional macro sercice transfers.

Bit 6: MASK

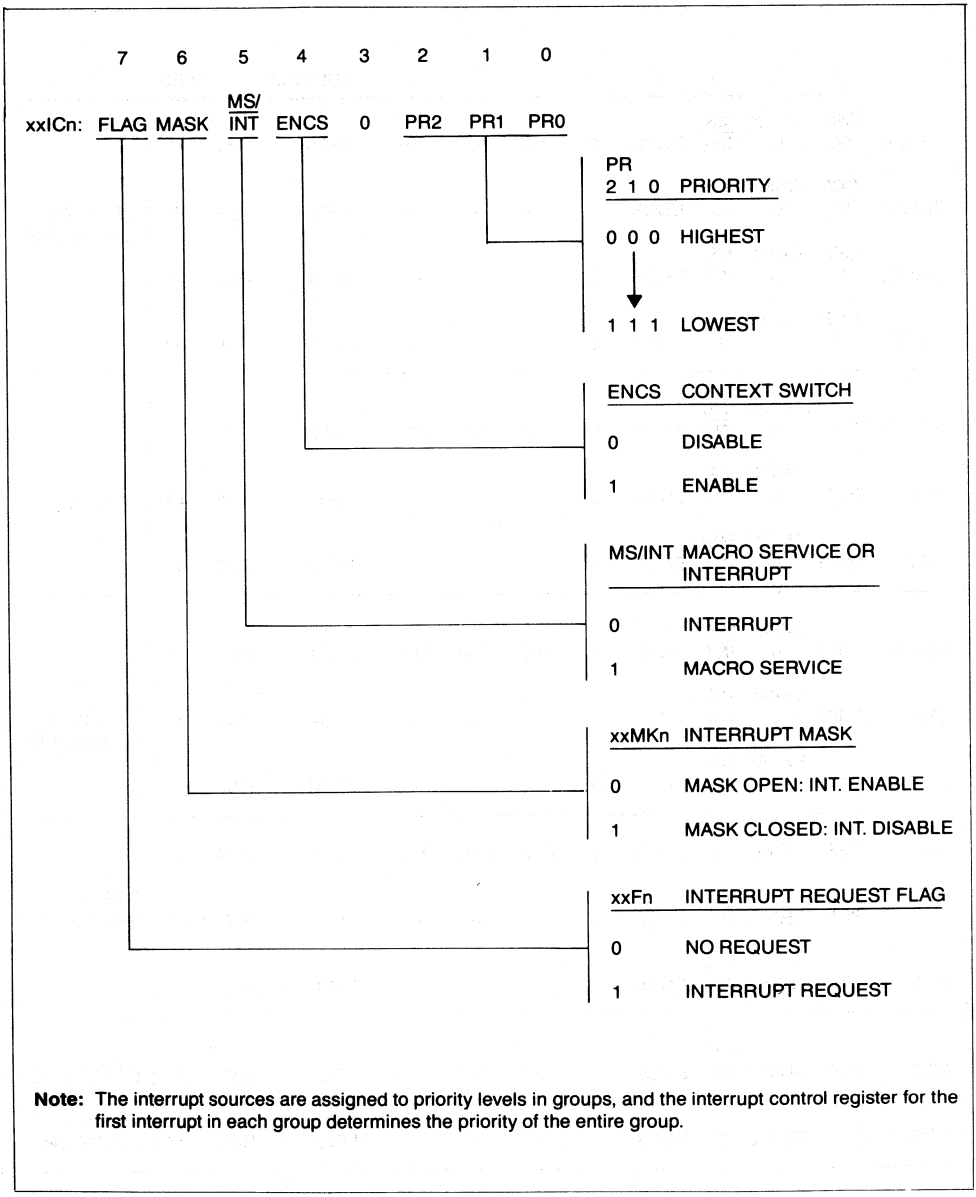
If the MASK bit is zero, the interrupt is enabled. If it is one, the interrupt is disabled.

Bit 7: FLAG

The FLAG bit is set by an interrupt request. If the mask bit is zero, the IE bit in the PSW is one, and if the current priority of the processor is lower than that of the interrupt, the interrupt will be accepted. If these conditions are not all met, the bit will remain set and the interrupt will be pending until the conditions are met, or until the bit is reset by software. The bit is reset to zero at the time the interrupt is accepted.In addition, the FLAG bit can be either set or cleared by software. This means that it is possible for the software to simulate any one of the maskable interrupts by setting the FLAG bit.

	7	6	5	4	3	2	1	0	ADDRESS	RESET		
CRIC00:	CRF 00	CRMK 00	MS/$\overline{\text{INT}}$	ENCS	0	PR2	PR1	PR0	FFC0	R/W	47	UP/DOWN COUNTERS
CRIC01:	CRF 01	CRMK 01	0	ENCS	0	—	—	—	FFC2	R/W	47	
CRIC10:	CRF 10	CRMK 10	MS/$\overline{\text{INT}}$	ENCS	0	—	—	—	FFC4	R/W	47	
CRIC11:	CRF 11	CRMK 11	0	ENCS	0	—	—	—	FFC6	R/W	47	
TMIC0:	TMF0	TMMK 0	MS/$\overline{\text{INT}}$	ENCS	0	PR2	PR1	PR0	FFCE	R/W	47	TIMERS
TMIC1:	TMF1	TMMK 0	MS/$\overline{\text{INT}}$	ENCS	0	—	—	—	FFD0	R/W	47	
TMIC2:	TMF2	TMMK 0	MS/$\overline{\text{INT}}$	ENCS	0	—	—	—	FFD2	R/W	47	
EXIC0:	EXIF0	EXIMK 0	MS/$\overline{\text{INT}}$	ENCS	0	PR2	PR1	PR0	FFC8	R/W	47	EXTERNAL INTERRUPTS
EXIC1:	EXIF1	EXIMK 1	MS/$\overline{\text{INT}}$	ENCS	0	—	—	—	FFCA	R/W	47	
EXIC2:	EXIF2	EXIMK 2	MS/$\overline{\text{INT}}$	ENCS	0	—	—	—	FFCC	R/W	47	
SEIC:	SEF	SEML	0	ENCS	0	PR2	PR1	PR0	FFDA	R/W	47	SERIAL INTERFACE
SRIC:	SRF	SRMC	MS/$\overline{\text{INT}}$	ENCS	0	—	—	—	FFDC	R/W	47	
STIC:	STF	STML	MS/$\overline{\text{INT}}$	ENCS	0	—	—	—	FFDE	R/W	47	
ADIC:	ADF	ADMK	MS/$\overline{\text{INT}}$	ENCS	0	PR2	PR1	PR0	FFE0	F/W	37	A/D CONVERTER
TBIC:	TBF	TBMK	0	ENCS	0	—	—	—	FFE2	R/W	47	TIMBASE COUNTER

Figure 3. Interrupt Control Registers



Note: The interrupt sources are assigned to priority levels in groups, and the interrupt control register for the first interrupt in each group determines the priority of the entire group.

Figure 4. Interrupt Control Registers (Continued)

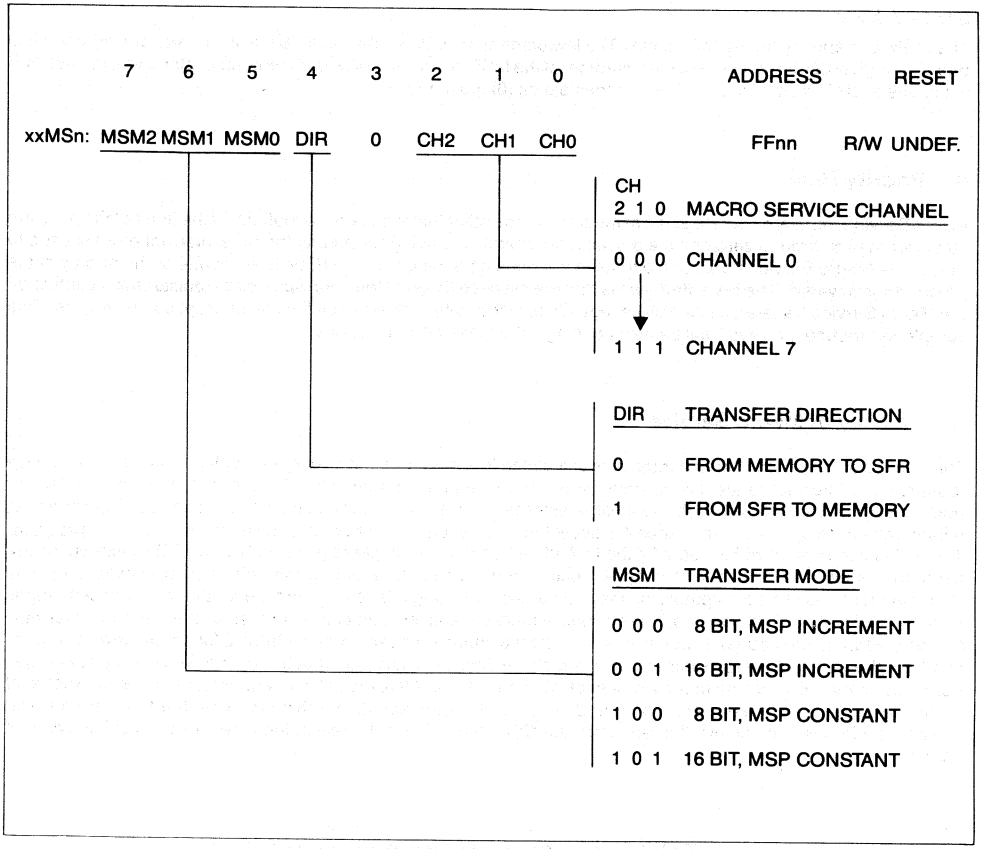


Figure 5. Macro Service Control Registers

Macro Service Control Registers: xxMSn

There are 11 Macro Service Control Registers, one for each of the interrupt sources for which macro service transfers are valid. They are 8-bit read/write registers which are undefined after external RESET.

Bits 0—2: CH0—CH2

The CH bits specify which of the macro service channels applies to the specified interrupt. Values range from 0 to 7.

Bit 3: Not Used

Bit 4: DIR

The DIR bit specifies the direction of transfer. If DIR is zero the transfer is made from memory to the SFR. If DIR is one the transfer ismade from the SFR to memory.

Bits 5—7: MSM

The MSM bits specify the transfer mode. The low order bit (5) defines the width: 8 bits if it is zero and 16 bits if it is one. The high order bit (7) controls the increment of the MSP: if it is zero MSP is incremented after each transfer, and if it is one MSP is held constant. The intermediate bit (6) must be zero.

4. Priority Rule

Macro Service transfers operate at the same priorities as other interrupt service routines. If the IE bit of the program status word is cleared to zero or if the processor is operating at a higher priority than the interrupt request used to trigger the Macro Service transfer, the transfer will remain pending until the IE bit is set to one or the priority of the processor is lowered. The one difference is that the mask bit (bit 6) of the interrupt control register has no effect on the Macro Service transfer; it takes place regardless of the state of the mask bit in the interrupt control register. The completion routine, however, will be held pending if the mask bit is set to one.

5. Latency of Macro Service

The latency of the Macro Service transfer is calculated in a manner similar to that for entry to an interrupt service routine. If the interrupt request is external, time is required (a maximum of 3, minimum of 2 processor states) for noise discrimination. Whether the request is external or internal, a level scan is then executed (maximum 16, minimum of 2 states), and, if the request is accepted, time is required for completion of the instruction in progress. The last is a maximum of 52 states for the DIVUX instruction. If multiple register PUSH and POP instructions are used, they should be limited to four register pairs per instruction to remain within this maximum execution time. String instructions are interruptable, so they do not generate long latencies. The time for the transfer itself ranges from 11 to 20 states. Finally the addressed special function register is an active counter, additional wait states may be required while the counter is being updated. The maximum number of wait states is 2 for the up/down counters, and 5 for the timer counters. Summing all of the above times, we calculate that the Macro Service latency ranges from a minimum of 13 states to a maximum of 95 states. If a 12 MHz crystal is used, processor states execute at 6 MHz, and this becomes a range of from 2.17 to 15.83 microseconds. Of this time only the transfer time and possible added wait states are "stolen" from the CPU. Table 2 lists the execution times for the various types of transfer.

Table 2. Macro Service Transfer Times (States)

	MSP Fixed		MSP Incremented	
Memory Location	Internal	External	Internal	External
Byte Transfer	11	13	12	14
Word Transfer	15	19	16	20

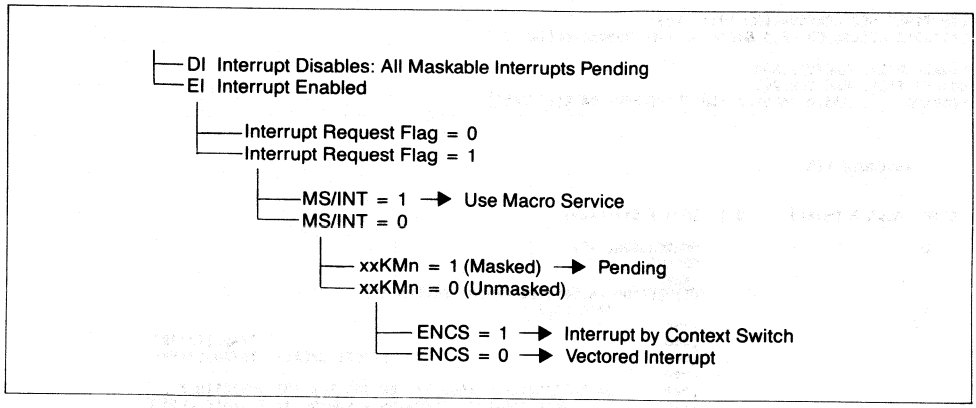


Figure 6. Interrupt Handling Sequence

6. Sample Programs

Two sample programs which illustrate the use of the Macro Service Facility are included. Both can be assembled with the RA310 assembler and run either under the Emulator IE-78310-R or with the Breakout Board. If they are to be used with the Breakout Board, they should be Located with the code segment starting at 8080H so that they can be loaded into the RAM of the Breakout Board. Both examples use Context Switch to service the interrupts. The vector tables of the Breakout Board's monitor program have, however, been copied to 8000H. If the user wishes to incorporate the examples into a larger program, he can patch in his own interrupt vectors and use the upper vector table by setting bit 1 of the CPU Control Word, CCW.

The first sample program sets Up/Down Counter 0 to count the internal clock and then repeatedly reads the contents of the counter. The result is converted to the ASCII representation of the hexadecimal number, and then Macro Service Facility is used to transmit it on the asynchronous serial line. Each block consists of 4 hexadecimal digits followed by CR/LF. The Baud rate is 4800 with no parity, 8 data bits, and 1 stop bit. Any terminal capable of 4800 Baud can be attached to the serial line to monitor the results. The program uses the serial line, so if it is used in the Breakout Board, it is necessary to reset the board in order to return to the Breakout Board's monitor program.

The second sample program generates an arbitrarily varying square wave which is available on the Timer 0 output pin TO0 (Port 3, Bit 6). In this case the Macro Service Facility is used to re-load the timer's modulus register each time the timer counts out. The new value will then control the duration of the second interval after the interrupt request. The values to be loaded into the modulus register are stored in a table in memory, and the interrupt service routine is entered only after the end of the table has been reached. An oscilloscope can be attached to the TO0 pin to monitor the result. A trigger signal for the oscilloscope is also provided each time the table is re-started. The latter signal is available on Port 3 Bit 7. This is the pin which can be used for the TO1 output, but in this case it is used as a simple output port pin. The program illustrates how the timers can be used to create additional pulse-width modulated outputs if more than two are required.

```
COPYRIGHT NEC CORPORATION 1985, 1986
UPD78310 ASSEMBLER V2.3 Macro Service Demonstration 2

SOURCE FILE: MACSVD2.ASM
OBJECT FILE: MACSVD2.REL
COMMAND    : RA310 MACSVD2.ASM PRINT(PRN) DATE(16/7/87)


     ASSEMBLE LIST


 STNO  ADRS R OBJECT     M I  SOURCE STATEMENT

    0                         $PROCESSOR(310)                    ;
                              $DEBUG                             ;
                              $XREF                              ;
                              $TITLE('Macro Service Demonstration 2')
    1                               NAME MACSD2
                              ;***
                              ;***                                          1/April/1987
                              ;***                              LATEST UPDATE  16/July/1987
                              ;***
                              ;***      MACRO SERVICE DEMONSTRATION TWO FOR THE uPD78310/78312
                              ;***           THIS PROGRAM GENERATES A SQUARE WAVE WHOSE PERIOD
                              ;***           AND DUTY CYCLE VARY ACCORDING TO A TABLE STORED
                              ;***           IN MEMORY.
                              ;***
                              ;***      THE OUTPUT WAVEFORM IS AVAILABLE ON PIN TO0.
                              ;***
                              ;***      THE MACRO SERVICE FACILITY IS USED TO CONTROL THE
                              ;***           DURATION OF THE HIGH AND LOW PORTIONS OF THE
                              ;***           WAVEFORM BY REPEATEDLY RELOADING THE TIMER'S
                              ;***           MODULUS REGISTER FROM THE MEMORY TABLE.
                              ;***
                              ;***      DEFINITION OF STORAGE AREAS IN ON-CHIP RAM
                              ;***           NOTE:  IN ON-CHIP RAM DS DIRECTIVES ONLY ARE USED
                              ;***
    2                         ORG       OFE00H          ;origin of on-chip RAM
    3  FE00                   DS        0020H           ;the 20H bytes of RAM for which soddr
                              ;                           addressing cannot be used will hold the stack.
    4  FE20                   OTBUF:    DS        06H   ;output buffer: total of 6 bytes
    5  FE26                   GENRAM:   DS        005AH ;remainder of general RAM
    6  FE80                             DS        10H   ;register bank 7 stored here
    7  FE90                             DS        4     ;AX & BC of RB6
    8  FE94                   PCSV6:    DS        2     ;PC save area of RB6
    9  FE96                   PSWS6:    DS        2     ;PSW save area of RB6
   10  FE98                             DS        8     ;RP4 through RP7 of RB6
   11  FEA0                             DS        4*10H ;RB5 through RB2
   12  FEE0                             DS        3*4   ;Macro Service Channels 4, 5, 6
   13  FEEC                   MSP7:     DS        2     ;Macro Service Pointer 7
   14  FEEE                   MSC7:     DS        1     ;Macro Service Counter 7
   15  FEEF                   SFRP7:    DS        1     ;SFR Pointer for Channel 7
   16  FEF0                   BKGREG:   DS        10H   ;Register Bank 0 will be used for background.
   17                                   ENDS              ;end of absolute segment


                              $select
                              ;***
                              ;***      INITIALIZATION SECTION
                              ;***
   18                                   CSEG              ;main code segment
   19  0000  4A                INIT:    DI                ;
   20  0001  2BFF00                     MOV       PSWH,#00H ;
   21  0004  2BFE00                     MOV       PSWL,#00H ;
   22  0007  2B4E00                     MOV       CCW,#00H  ;
   23  000A  0944FF00                   MOV       STBC,#00H ;go full speed (SCLK = Xtal/2)
   24  000E  2B4191                     MOV       RFM,#91H  ;REFRESH @ 5.33 usec., < (4 msec / 256 rows)
   25  0011  BA80                       MOV       R2,#80H   ;length of vector table
   26  0013  650080                     MOVW      RP6,#8000H ;base of external RAM
   27  0016  670000                     MOVW      RP7,#00H  ;base of external ROM
   28  0019  1520                       MOVBK     [DE+],[HL+] ;vectors copied to high table area
                              ;***
                              ;***      INITIALIZE ALL OF ON-CHIP RAM EXCEPT: CURRENT REGISTER BANK
                              ;***                & LOCATIONS OFE70H THROUGH OFE7FH (BB MONITOR'S AREA)
                              ;***                EACH LOCATION IS SET TO THE LOW BYTE OF IT'S ADDRESS
                              ;***
   29  001B  6500FE                     MOVW      RP6,#0FE00H ;base of on-chip RAM
```

```
30  001E  BA70          ;                MOV     R2,#070H        ;count for first block
31  0020  2400      INLP:MOVW    RP0,RP6         ;A <- 0FEH
32  0022  D8            XCH     A,R0            ;A <- 00H (to begin with)
33  0023  50            MOV     [DE+],A         ;will be low byte of its address
34  0024  8F01          CMP     R0,R1           ;got to FE80 yet?
35  0026  8109          BE      $SETSP          ;yes, at last!
36  0028  32F6          DBNZ    C,$INLP         ;more to go
37  002A  4580FE        MOVW    RP6,#0FE80H     ;start of second block
38  002D  BA9F          MOV     R2,#09FH        ;make it a long count
39  002F  14EF          BR      $INLP           ;go initialize the second block
40  0031  08FC20FE  SETSP:MOVW   SP,#0FE20H      ;put stack in "lost" RAM
                    ;                           ;"lost" RAM is inaccessable to "saddr" addresses

                    $eject
                    ;***
                    ;***    INITIALIZE TIMER 0 AND IT'S ASSOCIATED REGISTERS:
                    ;***            INTERRUPT VECTOR IN RP2 OF RB6
                    ;***            I/O PORT 3 MODE-CONTROL AND MODE REGISTERS
                    ;***            INTERRUPT CONTROL AND MACRO SERVICE CONTROL
                    ;***            SFR POINTER IN MACRO SERVICE CHANNEL
                    ;***            TIMER CONTROL
                    ;***
41  0035  0C945800      ;            MOVW    PCSV6,#TMOINT   ;interrupt vector
42  0039  2B334F        ;            MOV     PMC3,#4FH       ;enable TO0 control signal output
43  003C  2B237F        ;            MOV     PM3,#07FH       ;set port to read internal signal for bit 6
                    ;                                       ;  and write port latch for bit 7
44  003F  3A0300        ;            MOV     P3,#00H         ;P3 latches cleared to 0
45  0042  2BCE36        ;            MOV     TMIC0,#36H      ;interrupt control: level 6,
                    ;                                       ;  macro service, context switch
46  0045  2BCF27        ;            MOV     TMMS0,#27H      ;macro service control: 16 bit transfer
                    ;                                       ;  to SFR, increment MSC, channel 7
47  0048  3AEF8A        ;            MOV     SFRP7,#8AH      ;SFR pointer in macro service channel set to
                    ;                                       ;  modulus register of timer 0 (MD0)
48  004B  285F00        ;            CALL    !MCSET          ;set macro service address & count
49  004E  08BA0800      ;            MOVW    MD0,#0008H      ;load a short interval to get started
50  0052  2B8088        ;            MOV     TMC0,#88H       ;start timer 0 with TO0 output enabled
51  0055  4B            ;            EI                      ;now we can enable the interrupt
52  0056  14FE      SPIN:BR          $SPIN           ;let the interrupt do the rest
                    ;***
                    ;***
                    ;***    TIMER 0 INTERRUPT SERVICE
                    ;***
53  0058  4B        TMOINT:EI                        ;
54  0059  285F00            CALL    !MCSET          ;set macro service address & count
55  005C  295800            RETCS   !TMOINT         ; & go 'way
                    ;
                    ;***
                    ;***    SETUP MACRO SERVICE MEMORY ADDRESS, TRANSFER COUNT,
                    ;***            AND ENABLE BIT IN INTERRUPT CONTROL REGISTER
                    ;***        THIS ROUTINE ALSO ACTIVATES AN OSCILLOSCOPE TRIGGER SIGNAL
                    ;***            WHICH IS AVAILABLE ON PIN 42 (TO1) OF THE 78310 AND ON
                    ;***            J3-20 OF THE BREAKOUT BOARD
                    ;***
56  005F  B703      MCSET:  SET1     P3.7            ;raise scope trigger signal
57  0061  0CEC6E00          MOVW    MSP7,#INTTAB    ;origin of interval table to memory pointer
58  0065  3AEE14            MOV     MSC7,#TABLEN    ;length (in words) of interval table
59  0068  088DCE            SET1    TMIC0.5         ;enable macro service
60  006B  A703              CLR1    P3.7            ;clear scope trigger signal
61  006D  56                RET                     ;ready for next block

                    $eject
                    ;***
                    ;***    TABLE OF TIMER DELAYS:
                    ;***            VALUES IN MICROSECONDS, DECIMAL
                    ;***            TOTAL TIME IS 1.1 msec.
                    ;***            EACH ENTRY IS A WORD BECAUSE MD0 IS A 16 BIT REGISTER
                    ;***
62  006E  0A00A00   INTTAB: DW       10,10           ;
63  0072  140014 00          DW      20,20           ;
64  0076  1E001E00           DW      30,30           ;
65  007A  28002800           DW      40,40           ;
66  007E  32003200           DW      50,50           ;
67  0082  3C003C00           DW      60,60           ;
68  0086  46004600           DW      70,70           ;
69  008A  50005000           DW      80,80           ;
70  008E  5A005A00           DW      90,90           ;
71  0092  64006400           DW      100,100         ;
72        0014      TABLEN  EQU     ((S-INTTAB)/2) ;define length of table in words
```

```
                                ;***
                                ;***    THE FOLLOWING ROUTINE IS USEFUL DURING DEBUGGING AND CAN BE RUN
                                ;***        FROM THE EMULATOR.  IT IS NECESSARY BECAUSE THE EMULATOR
                                ;***        WILL NOT DISPLAY THE CONTENTS OF UPPER ON-CHIP RAM.
                                ;***
                                ;***    COPY DOWN UPPER RAM TO PERMIT DISPLAY
                                ;***
 73  0096   6500FD      COPREG:  MOVW    RP6,#0FD00H    ;destination, external RAM
 74  0099   6700FE               MOVW    RP7,#0FE00H    ;source, on-chip RAM
 75  009C   BAFF                 MOV     R2,#0FFH       ;transfer count
 76  009E   1520                 MOVBK   [DE+],[HL+]    ;move the block
 77  00A0   5D                   MOV     A,[HL]         ;and the
 78  00A1   54                   MOV     [DE],A         ;  last byte
 79  00A2   14FE          COPRF:  BR      $COPRF         ;done, so spin
 80                               ENDS                   ;end of code segment
 81                               END                    ;
```

```
COPYRIGHT NEC CORPORATION 1985, 1986
UPD78310 ASSEMBLER V2.3 Macro Service Demonstration 1

SOURCE FILE: MACSVD1.ASM
OBJECT FILE: MACSVD1.REL
COMMAND    : RA310 MACSVD1.ASM PRINT(PRN) DATE(14/7/87)


     ASSEMBLE LIST

 STNO  ADRS R OBJECT     M I  SOURCE STATEMENT

   0                          $PROCESSOR(310)                ;
                              $DEBUG                         ;
                              $XREF                          ;
                              $TITLE('Macro Service Demonstration 1')
                                   NAME MACSD1
   1                          ;***
                              ;***                                          1/April/1987
                              ;***                                LATEST UPDATE 13/July/1987
                              ;***
                              ;***    MACRO SERVICE DEMONSTRATION ONE FOR THE uPD78310/78312
                              ;***        THIS PROGRAM REPEATEDLY TRANSMITS THE CONTENTS
                              ;***        OF UP/DOWN COUNTER 0 IN HEX OVER THE SERIAL LINE.
                              ;***
                              ;***    ANY TERMINAL CAPABLE OF 4800 BAUD CAN BE USED TO VIEW
                              ;***        THE RESULTS.
                              ;***
                              ;***    THE MACRO SERVICE FACILITY IS USED TO TRANSMIT BLOCKS
                              ;***        OF 6 BYTES, AND THE INTERRUPT SERVICE ROUTINE GETS
                              ;***        NEW DATA AND RESETS THE MACRO SERVICE FACILITY FOR
                              ;***        THE NEXT BLOCK.
                              ;***
                              ;***    DEFINITION OF STORAGE AREAS IN ON-CHIP RAM
                              ;***        NOTE:  IN ON-CHIP RAM DS DIRECTIVES ONLY ARE USED
                              ;***
   2                               ORG     0FE00H         ;origin of on-chip RAM
   3  FE00                         DS      0020H          ;the 20H bytes of RAM for which soddr
                                                           ;  addressing cannot be used will hold the stack.
                              ;
   4  FE20                 OTBUF:  DS      06H            ;output buffer: total of 6 bytes
   5  FE26                 GENRAM: DS      005AH          ;remainder of general RAM
   6  FE80                         DS      10H            ;register bank 7 stored here
   7  FE90                         DS      4              ;AX & BC of RB6
   8  FE94                 PCSV6:  DS      2              ;PC save area of RB6
   9  FE96                 PSWS6:  DS      2              ;PSW save area of RB6
  10  FE98                         DS      8              ;RP4 through RP7 of RB6
  11  FEA0                         DS      4*10H          ;RB5 through RB2
  12  FEE0                         DS      3*4            ;Macro Service Channels 4, 5, 6
  13  FEEC                 MSP7:   DS      2              ;Macro Service Pointer 7
  14  FEEE                 MSC7:   DS      1              ;Macro Service Counter 7
  15  FEEF                 SFRP7:  DS      1              ;SFR Pointer for Channel 7
  16  FEF0                 BKGREG: DS      10H            ;Register Bank 0 will be used for background.
  17                               ENDS                   ;end of absolute segment
```

```
                        $eject
                        ;***
                        ;***    INITIALIZATION SECTION
                        ;***
18
19  0000  4A            INIT:   CSEG                    ;main code segment
20  0001  2BFF00                DI                      ;
21  0004  2BFE00                MOV     PSWH,#00H       ;
22  0007  2B4E00                MOV     PSWL,#00H       ;
23  000A  0944FF00              MOV     CCW,#00H        ;
24  000E  2B4191                MOV     STBC,#00H       ;go full speed (SCLK = Xtal/2)
25  0011  BA80                  MOV     RFM,#91H        ;REFRESH @ 5.33 usec., < (4 msec / 256 rows)
26  0013  650080                MOV     R2,#80H         ;length of vector table
27  0016  670000                MOVW    RP6,#8000H      ;base of external RAM
28  0019  1520                  MOVW    RP7,#00H        ;base of external ROM
                                MOVBK   [DE+],[HL+]     ;vectors copied to high table area
                        ;***
                        ;***    INITIALIZE ALL OF ON-CHIP RAM EXCEPT: CURRENT REGISTER BANK
                        ;***            & LOCATIONS OFE70H THROUGH OFE7FH (BB MONITOR'S AREA)
                        ;***            EACH LOCATION IS SET TO THE LOW BYTE OF IT'S ADDRESS
                        ;***
29  001B  6500FE                MOVW    RP6,#0FE00H     ;base of on-chip RAM
30  001E  BA70                  MOV     R2,#070H        ;count for first block
31  0020  2400          INLP:  MOVW    RP0,RP6         ;A <- OFEH
32  0022  D8                    XCH     A,R0            ;A <- 00H (to begin with)
33  0023  50                    MOV     [DE+],A         ;will be low byte of its address
34  0024  8F01                  CMP     R0,R1           ;got to FE80 yet?
35  0026  8109                  BE      $SETSP          ;yes, at last!
36  0028  32F6                  DBNZ    C,$INLP         ;more to go
37  002A  6580FE                MOVW    RP6,#0FE80H     ;start of second block
38  002D  BA9F                  MOV     R2,#09FH        ;make it a long count
39  002F  14EF                  BR      $INLP           ;go initialize the second block
40  0031  08FC20FE      SETSP: MOVW    SP,#0FE20H      ;put stack in "lost" RAM
                                                        ;"lost" RAM is inaccessable to "saddr" addresses
                        ;***
                        ;***    INITIALIZE COUNTER 0
                        ;***
41  0035  2B7280                MOV     UDCC0,#80H      ;enable for continuous count
42  0038  2B7400                MOV     CRC,#00H        ;all to compare mode
43  003B  2B7008                MOV     CUIM,#08H       ;counter 0 to mode 2
44  003E  2BC047                MOV     CRIC00,#47H     ;no interupts
45  0041  2BC247                MOV     CRIC01,#47H     ;  from counters

                        $eject
                        ;***
                        ;***    INITIALIZE SERIAL TRANSMIT: ASYNCH.
                        ;***
46  0044  2B5000        SPSET: MOV     SCM,#00H        ;disable serial port during setup
47  0047  2B22FF                MOV     PM2,#0FFH       ;port 2 all to input
48  004A  2B32FF                MOV     PMC2,#0FFH      ;port 2 all to control
49  004D  2B539C                MOV     BRG,#156D       ;4800
50  0050  2B5201                MOV     SCC,#01H        ;  baud
51  0053  0C947500              MOVW    OFE94H,#STINT   ;entry to service routine
52  0057  3AEF57                MOV     SFRP7,#57H      ;SFR pointer for macro service
53  005A  287F00                CALL    !RDCT           ;get the first count
54  005D  28A200                CALL    !MCSET          ;setup macro service
55  0060  2BDF07                MOV     STMS,#07H       ;8 bits from memory, channel 7
56  0063  2BDA46                MOV     SEIC,#46H       ;must load it to set priority 6
57  0066  2BDC46                MOV     SRIC,#46H       ;no desire to receive
58  0069  2BDE36                MOV     STIC,#36H       ;macro service & context switch
59  006C  2B5089                MOV     SCM,#89H        ;enable transmit only, no parity
60  006F  2B5700                MOV     TXB,#00H        ;send a null to get started
61  0072  4B                    EI                      ;
62  0073  14FE          SPIN:  BR      $SPIN           ;let the interrupt do the rest!
                        ;***
                        ;***    SERIAL TRANSMIT INTERRUPT SERVICE
                        ;***
63  0075  4B            STINT: EI                      ;
64  0076  287F00                CALL    !RDCT           ;get the count & hex it,
65  0079  28A200                CALL    !MCSET          ;set macro service address & count
66  007C  297500                RETCS   !STINT          ;  & go 'way
                        ;***
                        ;***    TRANSFER COUNTER CONTENTS TO OUTPUT BUFFER
                        ;***
67  007F  6720FE        RDCT:  MOVW    RP7,#OTBUF      ;start of output buffer
68  0082  BA04                  MOV     R2,#04H         ;nibble count
69  0084  1C1C                  MOVW    AX,UDC0         ;get counter
70  0086  24A8                  MOVW    RP5,AX          ;  contents
```

```
71  0088  289400      RDCTL:  CALL    !RDCTC          ;hex converter
72  008B  31E3              SHLW    RP5,4           ;next nibble
73  008D  32F9              DBNZ    C,$RDCTL        ;count 'em
74  008F  0C240A0D          MOVW    OTBUF+4,#0D0AH  ;CR/LF
75  0093  56                RET                     ;
                            $select
                            ;***
                            ;***    HEX CONVERT ROUTINE
                            ;***
76  0094  2408      RDCTC:  MOVW    RP0,RP5         ;all we want is the high nibble
77  0096  30A1              SHR     R1,4            ;so dump the rest of the accumulator
78  0098  AF0A              CMP     A,#0AH          ;>9?
79  009A  8702              BN      $RDCTD          ;no
80  009C  A807              ADD     A,#07H          ;make it a letter
81  009E  A830      RDCTD:  ADD     A,#30H          ;make it ASCII
82  00A0  51                MOV     [HL+],A         ;& put it in output buffer
83  00A1  56                RET                     ;
                            ;
                            ;***
                            ;***    SETUP MACRO SERVICE CHANNEL 7
                            ;***
84  00A2  0CEC20FE  MCSET:  MOVW    MSP7,#OTBUF     ;point to output buffer
85  00A6  3AEE06            MOV     MSC7,#06H       ;total of six characters
86  00A9  088DDE            SETI    STIC.5          ;enable macro service
87  00AC  56                RET                     ;
                            ;
                            ;***
                            ;***    THE FOLLOWING ROUTINE IS USEFUL DURING DEBUGGING AND CAN BE RUN
                            ;***        FROM THE EMULATOR.  IT IS NECESSARY BECAUSE THE EMULATOR
                            ;***        WILL NOT DISPLAY THE CONTENTS OF UPPER ON-CHIP RAM.
                            ;***
                            ;***    COPY DOWN UPPER RAM TO PERMIT DISPLAY
                            ;***
88  00AD  6500FD    COPREG: MOVW    RP6,#0FD00H     ;destination, external RAM
89  00B0  6700FE            MOVW    RP7,#0FE00H     ;source, on-chip RAM
90  00B3  BAFF              MOV     R2,#0FFH        ;transfer count
91  00B5  1520              MOVBK   [DE+],[HL+]     ;move the block
92  00B7  5D                MOV     A,[HL]          ;and the
93  00B8  54                MOV     [DE],A          ;  last byte
94  00B9  14FE      COPRF:  BR      $COPRF          ;done, so spin
95                          ENDS                    ;end of code segment
96                          END                     ;
```

Initialisation Routines for μPD70320 (V25) Microcomputer

Contents:

Author: Alister Greenhill
Application Department
NEC Electronics (U.K.) Ltd.

Related Documentation

μPD70320/322 Product Description
μPD70320/320 Relocatable Assembler User's Manual
μPD70108/116 C-Compiler User's Manual

Related Products

μPD70320/322	16 Bit Microcomputer	CMOS
μPD70330/332	16-Bit Microcomputer	CMOS

1. Introduction

The purpose of this application note is to introduce a suite of definition files for the NEC V-Series C compiler which may be included with any C source. These will enable easy access to the internal registers and data areas of the μPD7032 (V25) microcomputer. An example initialisation programme, using these files, is also discussed.

2. Addressing Structure

The internal registers and RAM of the V25 are memory mapped to an area called the Internal Data Base (IDB). This area may be mapped into any 4K block of the 1 Mbyte address space. Within the IDB not all registers may be addressed in the same manner, they may be one of 8 bit, 16 bit, 16/8 bit or 8/1 bit addressing.

The best way to access this type of area in C is to define a structure made up of data types char (8 bit) and int (16 bit). This has been done and defined as a structure of type sfr in file idb.c. The IDB area is of course located in an absolute area of memory which is defined by the value in the IDB register. The position of the IDB structure must therefore be passed to any C programme in the form of a pointer to a structure of type sfr. This can be done by defining an external variable at the beginning of any C source, for example;

```
extrn struct sfr *IDB.
```

This pointer must exist as a physical location in data memory and be initialised with the correct value by a start-up routine, the details of which will be dealt with later. Within the C source programme it will now be possible to refer to any register with the following construct;

```
IDB -> TMC0 = 0x33;
IDB -> WTC  = 0x2244;
```

with the compiler producing the correct byte or word addressing.

In the example initialisation module begin. c it will be noted that the initialisation procedure init is passed the value of the IDB pointer. This enables the IDB pointer to be re-defined within the init procedure as a register variable. The C compiler uses the index registers IX and IY for register variables therefore extremely efficient code is produced when the IDB area is to be accessed repeatedly as there is no need to re-load an index register with the IDB pointer on each occasion.

3. Bit addressing

The logical way to deal with individual bit addressing would be to define each register as a union of a char variable and bit field variables for example;

```
union {
    char BYTE;
    struct {
        unsigned dum      : 4;
        unsigned ENCS     : 1;
        unsigned MS_INT   : 1;
        unsigned TMMK     : 1;
        unsigned TMF      : 1;
    } BIT;
} TMIC1;
```

A byte write would look like; `IDB -> TMIC1.BYTE = 0x10`
and a bit write would look like; `IDB -> TMIC1.BIT.ENCS = 1`

Unfortunately C defines bit fields to be held within data locations of type int which equate to a 16 bit quantity on the V25. Therefore a union of a char variable and a bit field will allocate two bytes of storage making it impossible to define two consequtive bit addressable 8 bit registers.

The solution is to define three pseudo functions bit_set, bit_clr and bit_tst which are used as;

bit_set (TMIC1, TMF);

the define statements in the file bit.h will cause the pre-processor to produce;

IDB −> TMIC1 = IDB −> TMIC1 ¦ 0x80;

which will force the compiler to manipulate register TMIC1 using byte reads and writes.

To facilitate these operations all bit names have been defined with a value equivalent to an 8 bit mask with a bit set at the appropriate position. For example TMF shown above is converted to 128 = 1000 0000B by the pre-processor.

The bit name used in the V25 product description has been used in all cases except when several bits within one register differ in name by the last numeric character only. For example in the macro-service control registers CH0, CH1, CH2 these have been re-defined as CH_B0, CH_B1, CH_B2, this is to avoid conflicts which occur in some cases with register names.

If an un-named bit is to be specified on, for example port registers, then BIT0, BIT1, — BIT9 may be used.

4. Register initialisation

For each register a set of key words have been defined as bit masks such that the arithmetic sum of the required key words will produce a bit pattern which may be written to the appropriate register. Take for example the serial control register shown below;

```
                                    /* serial channel 1 setup */
pt −> SCM1  =  ASYNC                /* mode; sync or async */
           +  TxEN
           +  RxEN
           +  CHAR_8                /* data bits per character; also char_7 */
           +  NO_PARITY             /* no, zero, odd, even possible */
           +  STOP_1;               /* stop_1, stop_2; 1 or 2 stop bits */
```

This will produce the correct bit pattern because the following substitutions, defined in file sio.h, will be performed by the pre-processor.

```
#define ASYNC       1       /* asynchronous mode */
#define SYNC        0       /* i/o interface mode */
#define TxEN        64      /* receiver enable */
#define Rxafidy     0       /* receiver disable */
#define STOP_1      0       /* 1 stop bit */
#define STOP_2      4       /* 2 stop bits */
#define CHAR_7      0       /* 7 data bits */
#define CHAR_8      8       /* 8 data bits */
#define NO_PARITY           0
#define ZERO_PARITY         16
#define ODD_PARITY          32
#define EVEN_PARITY         48
#define TxEN        128     /* transmitter enable */
```

```
#define TxDIS      0      /* transmitter disable */
#define CLK_INT    1      /* internal clock for i/o mode */
#define CLK_EXT    0      /* external clock for i/o mode */
#define TSK        8      /* trigger serial receive clock bit */
```

5. DMA and Macro-Service Channels

The structure defined as sfr also includes the DMA and Macro-Service channels which are located within memory area xxE00 to xxE3F of the IDB area. A register in Macro-Service channels 2—7 may be accessed as follows;

IDB −> CH3.MSP = 0x1122.

For channels 0 and 1 the use as Macro or DMA must be specified;

IDB −> CH0.MAC.MSC = 0x77;
IDB −> CH1.DMA.TC = 0x4455.

6. Start-Up Routine

Assume that the application is for a dedicated controller with a separate ROM and a RAM area and no operating system. There is therefore a requirement for a short routine to take control after a hardware reset and perform the following functions;

1. Initialise the DS0 and SS segment registers. C does not use DS1 and the far jump loaded at the reset start address will initialise PS.
2. Initialise the stack pointer (SP) to the top of available memory.
3. Define the location of the IDB area in memory by writing to the IDB register which can be accessed always at address 0FFFFFh.
4. Define a location called _IDB which contains a 16 bit value representing the offset between the DS0 address and the IDB base.
5. Call the main C module, usually called _MAIN.

An example assembler routine is contained in file startup.asm.

7. Memory Mapping Considerations

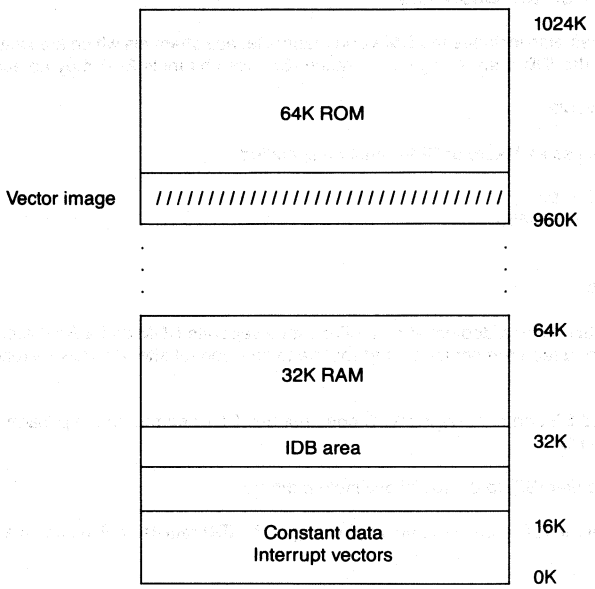
The following applies to the C compiler small model only.

The compiler produces four output segments for the linker, CODE, DATA, CONST, STACK. It assumes that the CONST, DATA and STACK segments all have the same segment base and must therefore all be within the same 64K block. In addition the compiler considers the IDB structure to be in the DATA segment. A typical application requiring constant data to be in ROM must therefore be designed with all RAM, the 512 bytes of the IDB area and the constant area of ROM mapped to the same 64K block. For example consider a μPD70320, a single 27C512 64K x 8 EPROM and a single 43256 32K x 8 RAM. The following decode may then be used;

$\overline{A19.A18.A17.A16.A15}$ = RAM chip select.

$(A19+A18+A17+A16+A15+A14).\overline{\text{RAM chip select}}$ = EPROM chip select.

This will cause the RAM to be located in address range 32K to 64K and the ROM to be located in range 0 to 16K and 64K to 1024 K. If the IDB area is then mapped to the address range 16K to 32K, then up to 16K of ROM, the IDB area and the RAM may all be accessed from the same DS0 segment value of zero. Mapping the ROM to the first 1K of memory also allows the interrupt vector tables to fixed in ROM for stand alone applications. The memory map would be as follows;



To create the above memory map the following linker command should be used;

```
lk70116 startup, %1, cc79116.lib to start st(start_up) bs
add(seg(const(0), data(8000h), code(0F4000h) ) )
ord(seg(const, data, stack, code) )
```

In the above the relocatable startup routine is linked to the users compiled C module %1 and to the standard C library. The result is a file called start. The label start_up in the startup module is defined as the reset start address and the linker is instructed to generate a far jump at the hardware reset address to label start_up. The add control forces the linker to build the const, data and code segments from addresses 0, 8000h and 0F4000h respectively. The code address assumes that the vector table and the const segment use all of the 16K allocated. Due to the mapping arrangement this address could be lowered to use up any ROM not used by the const segment. The ord control is necessary to modify the default ordering of the segments.

8. Example Source File

The file begin.c contains an example C source file which initialises the V25 and then sets up serial channel 1 to output the message 'Hello world' repeatedly while port 0 bit 0 is held low. The transfer is performed using macro-service channel 4.

9. Disk Contents

If this application note is supplied in disk form then the following files should be present;

wtc.h	required if referencing wait control register
pcr.h	required if referencing processor control register
rfm.h	required if referencing refresh mode register
prt.h	required if referencing ports
sio.h	required if referencing serial interface unit
tim.h	required if refeencing timer unit
dma.h	required if referencing DMA control registers
bit.h	required if using bit operations in IDB area
int.h	required if referencing interrupt registers
mac.h	required if referencing macro-service registers
idb.c	required if referencing may locations in IDB area
begin.c	example initialisation C source
startup.asm	example start up routine assembler source
cnote.txt	application report Wordstar format
cnote.prn	application report basic printer format

10. Programme Listing

```
/* C-Compiler definition file for V25 - V1.0 */
/* wtc.h          wait control register */

£define wait_0      0
£define wait_1      1
£define wait_2      2
£define ready       3

£define BLOCK0(a)          a * 1
£define BLOCK1(a)          a * 4
£define BLOCK2(a)          a * 16
£define BLOCK3(a)          a * 64
£define BLOCK4(a)          a * 256
£define BLOCK5(a)          a * 1024
£define BLOCK67(a)         a * 4096
£define IO_SPACE(a)        a * 16384
```

```
/* C-Compiler definition file for V25 - V1.1 */
/* pcr.h          processor control register */

£define RAMEN    0x40     /* enable internal RAM */
£define RAMDIS   0        /* disable internal RAM */

                          /* processor clock period */
£define PCK_2    0        /* fCLK = fx/2 */
£define PCK_4    1        /* fCLK = fx/4 */
£define PCK_8    2        /* fCLK = fx/8 */
£define PCK_DIS  3        /* fCLK = disabled */

                          /* time base interrupt interval */
£define TB_10    0        /* fCLK/2**10 */
£define TB_13    4        /* fCLK/2**13 */
£define TB_16    8        /* fCLK/2**16 */
£define TB_20    12       /* fCLK/2**20 */

£define PCK_B0   1        /* bit masks for PRC register /*
£define PCK_B1   2
£define TB_B0    4
£define TB_B1    8

£define SBF      1        /* standby flag in standby control register */
```

```
/* C-Compiler definition file for U25 - V1.0 */
/* rfm.h          refresh control registers */

                        /* refresh period */
#define RFT_4    0      /* 2**4/fCLK */
#define RFT_5    1      /* 2**5/fCLK */
#define RFT_6    2      /* 2**6/fCLK */
#define RFT_7    3      /* 2**7/fCLK */

                        /* refresh wait states wait_0, wait_1, wait_2 */
#define RFW( a )        ((a)*4)

#define RFEN            16      /* refresh enable */
#define RFDIS           0       /* refresh disable */
#define HLTRFEN         32      /* halt refresh enable */
#define HLTRFDIS        0       /* halt refresh disable */
#define HLDRFEN         64      /* hold refresh enable */
#define HLDRFDIS        0       /* hold refresh disable */
#define RFLV1           128     /* refresh level bit 1 */
#define RFLV0           0       /* refresh level bit 0 */
```

```
/* C-Compiler definition file for U25 - V1.0 */
/* prt.h          port control registers */

/* definitions for port, port mode, port mode control and pmt */
#define         control         1
#define         port            0
#define         output          0
#define         input           1
#define         CLKOUT          1
#define         set             1
#define         reset           0
#define         VTH             16
#define         BIT_0(a)                ((a)*1)
#define         BIT_1(a)                ((a)*2)
#define         BIT_2(a)                ((a)*4)
#define         BIT_3(a)                ((a)*8)
#define         BIT_4(a)                ((a)*16)
#define         BIT_5(a)                ((a)*32)
#define         BIT_6(a)                ((a)*64)
#define         BIT_7(a)                ((a)*128)
#define         VREF            PMT
```

```
/* C-Compiler definition file for V25 - V1.0 */
/* sio.h        definitions for serial I/O unit */

£define ASYNC    1        /* asynchronous mode */
£define SYNC     0        /* i/o interface mode */
£define RxEN     64       /* receiver enable */
£define RxDIS    0        /* receiver disable */
£define STOP_1   0        /* 1 stop bit */
£define STOP_2   4        /* 2 stop bits */
£define CHAR_7   0        /* 7 data bits */
£define CHAR_8   8        /* 8 data bits */
£define NO_PARITY         0
£define ZERO_PARITY       16
£define ODD_PARITY        32
£define EVEN_PARITY       48
£define TxEN     128      /* transmitter enable */
£define TxDIS    0        /* transmitter disable */
£define CLK_INT  1        /* internal clock for i/o mode */
£define CLK_EXT  0        /* external clock for i/o mode */
£define TSK      8        /* trigger serial receive clock bit */

/* ************************************************************** */
/* *  NOTE: The following values assume a 5MHz clock, for other * */
/* *        clocks the constants must be re-calculated.         * */
/* ************************************************************** */

£define n_110    7        /* baud rate generator prescalar value */
£define n_150    7
£define n_300    6
£define n_600    5
£define n_1K2    4
£define n_2K4    3
£define n_4K8    2
£define n_9K6    1
£define n_19K2   0
£define n_38K4   0
£define n_125K   0

£define G_110    178      /* baud rate generator modulo value */
£define G_150    130
£define G_300    130
£define G_600    130
£define G_1K2    130
£define G_2K4    130
£define G_4K8    130
£define G_9K6    130
£define G_19K2   130
£define G_38K4   65
£define G_125K   2

£define TSK      8        /* mode register bit masks */
£define RSCK     4
£define SL       4
£define CL       8
£define PRTY0    16
```

```
£define PRTY1    32
£define RxE      64
£define TxE      128
£define TxRDY    128

£define PRS_B0   1        /* baud rate prescalar bit masks */
£define PRS_B1   2
£define PRS_B2   4
£define PRS_B3   8

£define ER0      1        /* error register bit masks */
£define ERF      2
£define ERP      4
£define RXB      128
£define OVERRUN  1
£define FRAMING  2
£define PARITY   4
£define RX_BIT   128
```

```
/* C-Compiler definition file for U25 - V1.0 */
/* tim.h          definitions for timer mode registers */

£define INTERVAL          0        /* timer 0 interval mode */
£define ONE_SHOT          1        /* timer 0 one shot mode */
£define TOUT_HI           4        /* TOUT fixed high */
£define TOUT_LO           0        /* TOUT fixed low */
£define TOUT_TOG          8        /* square wave at TOUT */
£define MCLK_12           0        /* MD0 clock = fCLK/12 */
£define MCLK_128          16       /* MD0 clock = fCLK/128 */
£define TCLK_6            0        /* TM0 clock = fCLK/6 */
£define TCLK_12           0        /* TMx clock = fCLK/12 */
£define TCLK_128          64       /* TMx clock = fCLK/128 */

£define MOD_B0   1;       /* define flag bit masks */
£define MOD_B1   2;
£define ALV      4;
£define ENTO     8;
£define MCLK     16;
£define MS       32;
£define TCLK     64;
£define TS       128;
```

```
/* C-Compiler definition file for V25 - V1.0 */
/* dma.h          definitions for DMA control registers */

/* bit masks DMAMx */

£define MD_B2    128
£define MD_B1    64
£define MD_B0    32
£define W        16
£define EDMA     8
£define TDMA     4

£define WRD_MOV  16        /* transfer by word */
£define BYT_MOV  0         /* transfer by byte */

£define SSM      0         /* single step mode */
£define DR_IOM   32        /* demand release mode I/O to memory */
£define DR_MIO   64        /* demand release mode memory to I/O */
£define BM       128       /* burst mode */
£define ST_IOM   160       /* single transfer mode I/O to memory */
£define ST_MIO   192       /* single transfer mode memory to I/O */

/* bit masks DMACx */

£define PS_B0    1
£define PS_B1    2
£define PD_B0    16
£define PD_B1    32

£define SRC_INC  1         /* source address increment */
£define SRC_DEC  2         /* source address decrement */
£define DST_INC  16        /* destination address increment */
£define DST_DEC  32        /* destination address decrement */
```

```
/* C-Compiler definition file for V25 - V1.0 */
/* bit.h          define bit operations for 8bit IDB registers */

£define bit_set(a,b)     IDB->a = IDB->a | b
£define bit_clr(a,b)     IDB->a = IDB->a & ~b
£define bit_tst(a,b)     IDB->a & b

£define          BIT0    1         /* define bit masks */
£define          BIT1    2
£define          BIT2    4
£define          BIT3    8
£define          BIT4    16
£define          BIT5    32
£define          BIT6    64
£define          BIT7    128
```

```
/* C-Compiler definition file for U25 - V1.0 */
/* int.h          interrupt control registers */

£define PR_B0    1          /* define bit masks */
£define PR_B1    2
£define PR_B2    4
£define ENCS     16
£define MS_INT   32
£define IMK      64
£define IF       128

£define PR_0     0          /* define priority levels */
£define PR_1     1
£define PR_2     2
£define PR_3     3
£define PR_4     4
£define PR_5     5
£define PR_6     6
£define PR_7     7

£define INT_EN   0          /* interrupt operation enable */
£define MS_EN    32         /* macro service operation enable */
£define CS_EN    16         /* context switch operation enable */

£define MSK_EN   64         /* interrupt mask enable */
£define MSK_DIS  0          /* interrupt mask disable */

£define ESNMI    1          /* bit masks for INTM register */
£define ES_B0    4
£define ES_B1    16
£define ES_B2    64

£define NMI_UP   1          /* interrupt on rising (UP) or */
£define P0_UP    4          /* falling (DN) edges          */
£define P1_UP    16
£define P2_UP    64
£define NMI_DN   0
£define P0_DN    0
£define P1_DN    0
£define P2_DN    0
```

```
/* C-Compiler definition file for V25 - V1.0 */
/* mac.h          macro service control registers */

£define CH_B0    1          /* define bit numbers */
£define CH_B1    2
£define CH_B2    4
£define DIR      16
£define MSM_B0   32
£define MSM_B1   64
£define MSM_B2   128

£define CH_0     0          /* define channel numbers */
£define CH_1     1
£define CH_2     2
£define CH_3     3
£define CH_4     4
£define CH_5     5
£define CH_6     6
£define CH_7     7

£define MEM_SFR  0
£define SFR_MEM  16         /* define macro service direction */

£define NORM_8   0
£define NORM_16  32
£define SEARCH   128        /* macro service operating mode */
```

```
/* C-Compiler definition file for V25 - V1.0 */
/* idb.c     structure of internal data base area */

struct dma {
    int SAR;
    int DAR;
    char    DARH;
    char    SARH;
    int TC;
};

struct mac {
    char    MSC;
    char    SFRP;
    char    SCHR;
    char    RESV;
    int MSP;
    int MSS;
};

struct sfr {                /* special function reg. structure          */
    char dum27[3584];       /* dummy array to offset to addr. xxE00     */
    union dmamac {          /* define DMA and Macro-serv. control       */
        struct dma DMA;     /* blocks in reg. bank 0 RAM area           */
        struct mac MAC;     /* NOTE RAMEN must be set to access these */
    }CH0;                   /* locations as memory addr. will be used */

    union dmamac CH1;
    struct mac CH2;
    struct mac CH3;
    struct mac CH4;
    struct mac CH5;
    struct mac CH6;
    struct mac CH7;
    char dum31[224]; /* dummy array to offset to addr. xxF00 */

    char P0     ;   /* 0xF00 */
    char PM0    ;   /* 0xF01 */
    char PMC0   ;   /* 0xF02 */
    char dum00[05];
    char P1     ;   /* 0xF08 */
    char PM1    ;   /* 0xF09 */
    char PMC1   ;   /* 0xF0A */
    char dum01[05];
    char P2     ;   /* 0xF10 */
    char PM2    ;   /* 0xF11 */
    char PMC2   ;   /* 0xF12 */
    char dum02[37];
    char PT     ;   /* 0xF38 */
    char dum03[02];
    char    PMT;    /* 0xF3B */
    char dum04[04];
    char INTM   ;   /* 0xF40 */
    char dum05[03];
    char EMS0   ;   /* 0xF44 */
```

```
char EMS1   ;   /* 0xF45 */
char EMS2   ;   /* 0xF46 */
char dum06[05];
char EXIC0  ;   /* 0xF4C */
char EXIC1  ;   /* 0xF4D */
char EXIC2  ;   /* 0xF4E */
char dum07[17];
char RxB0   ;   /* 0xF60 */
char dum08[01];
char TxB0   ;   /* 0xF62 */
char dum09[02];
char SRMS0  ;   /* 0xF65 */
char STMS0  ;   /* 0xF66 */
char dum10[01];
char SCM0   ;   /* 0xF68 */
char SCC0   ;   /* 0xF69 */
char BRG0   ;   /* 0xF6A */
char SCE0   ;   /* 0xF6B */
char SEIC0  ;   /* 0xF6C */
char SRIC0  ;   /* 0xF6D */
char STIC0  ;   /* 0xF6E */
char dum12[01];
char RxB1   ;   /* 0xF70 */
char dum13[01];
char TxB1   ;   /* 0xF72 */
char dum14[02];
char SRMS1  ;   /* 0xF75 */
char STMS1  ;   /* 0xF76 */
char dum15[01];
char SCM1   ;   /* 0xF78 */
char SCC1   ;   /* 0xF79 */
char BRG1   ;   /* 0xF7A */
char SCE1   ;   /* 0xF7B */
char SEIC1  ;   /* 0xF7C */
char SRIC1  ;   /* 0xF7D */
char STIC1  ;   /* 0xF7E */
char dum17[01];
int  TM0    ;   /* 0xF80 */
int  MD0    ;   /* 0xF82 */
char dum18[04];
int  TM1    ;   /* 0xF88 */
int  MD1    ;   /* 0xF8A */
char dum19[04];

char TMC0;      /* 0xF90 */

char TMC1;      /* 0xF91 */
char dum20[02];
char TMMS0  ;   /* 0xF94 */
char TMMS1  ;   /* 0xF95 */
char TMMS2  ;   /* 0xF96 */
char dum21[05];
char TMIC0  ;   /* 0xF9C */
char TMIC1  ;   /* 0xF9D */
char TMIC2  ;   /* 0xF9E */
```

```
    char dum22[01];
    char DMAC0   ;    /* 0xFA0 */
    char DMAM0   ;    /* 0xFA1 */
    char DMAC1   ;    /* 0xFA2 */
    char DMAM1   ;    /* 0xFA3 */
    char dum23[8];
    char DIC0    ;    /* 0xFAC */
    char DIC1    ;    /* 0xFAD */
    char dum24[50];
    char STBC    ;    /* 0xFE0 */
    char RFM     ;    /* 0xFE1 */
    char dum25[06];
    int  WTC     ;    /* 0xFE8 */
    char FLAG    ;    /* 0xFEA */
    char PRC     ;    /* 0xFEB */
    char TBIC    ;    /* 0xFEC */
};
```

```
/* Example C-Compiler source for V25 - V1.0 */
/* begin.c  V25 initialisation */

#include "wtc.h"     /* needed if ref. wait cntl. reg. */
#include "pcr.h"     /* needed if ref. proc. cntl. reg. */
#include "rfm.h"     /* needed if ref. refresh mode reg. */
#include "prt.h"     /* needed if ref. ports */
#include "sio.h"     /* needed if ref. serial i/f unit */
#include "tim.h"     /* needed if ref. timer unit */
#include "dma.h"     /* needed if ref. DMA cntl. reg.s */
#include "bit.h"     /* needed if using bit oper. in IDB area */
#include "int.h"     /* needed if ref. int. reg.s */
#include "mac.h"     /* needed if ref. macro-ser. regs */

#include "idb.c"  /* needed if ref. locations in IDB area */

struct sfr *IDB;
char message[] = "Hello world\n\r\0";    /* def. message */

MAIN()
{

init(IDB);
while (1) /* set endless loop which will repeatedly output */
     /* the message 'Hello world' on serial ch. 1       */
     /* for as long as port 0 bit 0 is held high.       */
     /* cntl. is via macro-ser. ch. 4 which             */
     /* is set to transfer char.s from array            */
     /* message to transmit buffer 1 until a null       */
     /* char. is detected. The int. request             */
     /* flag is polled in the loop to determine end     */
     /* of xmt.                                         */
    {
    if (bit_tst(P0,BIT0))
        {                       /* setup macro-ser. ch. 4 */
        IDB->CH4.MSC = 0xff; /* set count to maximum */
        IDB->CH4.MSS = 0;     /* for small model assume data
                                 segment set to zero */

        IDB->CH4.SFRP = &(IDB->TxB1) - &(IDB->P0);
                /* calculate 8bit offset from  */
                /* sfr address F00 (P0) to TxB1*/

        IDB->CH4.MSP = message; /* set ptr. to message */
        bit_set(SCM1,TxRDY);     /* start xmt */

        while (~(bit_tst(SCM1,IF)));
                             /* wait till int. flag set */

        bit_clr(SCM1,TxRDY);              /* disable xmt */
        bit_clr(SCM1,IF);              /* clear int. flag */
        };
    };

};
```

```
init(IDB)
register struct sfr *IDB;
{

        IDB->WTC = BLOCK0( wait_0 ) /* def. wait cntl.   */
         + BLOCK1( wait_0 )       /* valid cntl.s wait_0 */
             + BLOCK2( wait_1 ) /*                wait_1 */
             + BLOCK3( wait_0 ) /*                wait_2 */
             + BLOCK4( wait_0 ) /*                ready  */
             + BLOCK5( wait_0 )
             + BLOCK67( wait_0 )
             + IO_SPACE( wait_2 );

        IDB->PRC = RAMEN          /* def. processor cntl. */
         + PCK_2
         + TB_20;

        IDB->RFM = RFEN           /* def. refresh cntl. */
         + RFW( wait_0 )      /* valid cntl.s as above */
         + RFT_6
         + HLTRFDIS
         + HLDRFEN
         + RFLV0;

       IDB->PMC0 = BIT_7(CLKOUT); /* port 0 mode cntl. reg. */
                                  /* valid cntl.s CLKOUT port */

                    /* port 0 mode reg. */
                 /* valid cntl.s input output */
        IDB->PM0 = BIT_0( input )
         + BIT_1( output )
         + BIT_2( output )
         + BIT_3( output )
         + BIT_4( output )
         + BIT_5( output )
         + BIT_6( output )
         + BIT_7( output );
                          /* port 1 mode cntl. reg. */
                          /* valid cntl.s; cntl.,port */
       IDB->PMC1 = BIT_3( port )       /* cntl. = /INTAK */
         + BIT_4( control )    /* cntl. = INTR */
         + BIT_5( control )    /* cntl. = TOUT */
         + BIT_6( control )    /* cntl. = /SCKO */
         + BIT_7( port );      /* cntl. = READY */

                 /* port 1 mode reg. */
              /* valid cntl.s input output set */
        IDB->PM1 = BIT_0( set )
         + BIT_1( set )
         + BIT_2( set )
         + BIT_3( set )
         + BIT_4( output )
         + BIT_5( output )
         + BIT_6( output )
```

```
    + BIT_7( output );

                        /* port 2 mode cntl. reg. */
                    /* valid cntl.s; cntl. port */
  IDB->PMC2 = BIT_0( control ) /* cntl. = DMARQ0 */
    + BIT_1( control )      /* cntl. = /DAAAK0  */
    + BIT_2( control )      /* cntl. = /TC0   */
    + BIT_3( control )      /* cntl. = DMARQ1   */
    + BIT_4( control )      /* cntl. = /DMAAK1  */
    + BIT_5( control )      /* cntl. = /TC1   */
    + BIT_6( control ) /* cntl. = /HLDAK */
    + BIT_7( control );     /* cntl. = HLDRQ   */

                              /* port 2 mode reg. */
                    /* valid cntl.s input output */
   IDB->PM2 = BIT_0( input )
    + BIT_1( input )
    + BIT_2( input )
    + BIT_3( input )
    + BIT_4( input )
    + BIT_5( input )
    + BIT_6( input )
    + BIT_7( input );

   IDB->PMT = VTH*4/16;     /* def. comp. ref. voltage */

                    /* serial ch. 0 setup */
  IDB->SCM0 = SYNC      /* mode; sync or async */
    + TxEN
    + RxEN
    + CLK_EXT;  /* rcv. clk.; clk_int or clk_ext */

  IDB->SCC0 = n_4K8;      /* def. baud rate gen. prescaler
                     /* value n */

  IDB->BRG0 = G_4K8;    /* def. buad rate gen. value G */

                        /* serial ch. 1 setup */
  IDB->SCM1 = ASYNC     /* mode; sync or async */
    + TxDIS             /* do not start yet */
    + RxDIS             /* RxEN immediately enables rcv.*/
    + CHAR_8            /* data bits/char.; also char_7 */
    + NO_PARITY         /* no,zero,odd,even possible */
    + STOP_1;       /* stop_1,stop_2 ; 1 Or 2 stop bits */

  IDB->SCC1 = n_19K2;   /* def. baud rate gen. prescaler
                        /* value n */

   IDB->BRG1 = G_19K2;  /* def. baud rate gen. value G */

 IDB->STIC1 = MS_EN     /* enable marco-ser. on xmt. ch1 */
    + PR_4;             /* set priority 4 */

 IDB->STMS1 = CH_4      /* assign macro ch. 4 to xmt. ch1 */
    + MEM_SFR           /* def. direction of transfer */
```

```
        + SEARCH;              /* def. search mode */

   IDB->CH4.SCHR = 0;          /* set search char. to null */

                               /* timer 0 setup */
      IDB->TMC0 = ONE_SHOT /* basic mode: also INTERVAL possible */
        + TOUT_TOG /* output behavior: also TOUT_LO TOUT_HI */
        + MCLK_128 /* modulo clock: also MCLK_12 = fCLK/12 */
        + TCLK_6;  /* timer clock: also TCLK_128 or TCLK_12
                      if in interval timer mode */

      IDB->TMC1 = TCLK_128;     /* timer clock is only init.*/

    /* ********************************************************
       bit operations can be performed on the IDB registers
       by using the functions bit_set( register, bit mask )
                  bit_clr( register, bit mask )
                  bit_tst( register, bit mask )
       these must be used to force byte operations.
       ******************************************************** */

    bit_set(TMC1,TS);          /* start timer reg. 1 */

    bit_set(TMC0,TS);          /* start timer reg. 0 */

    bit_set(TMC0,MS);              /* start modulo reg. 0 */

       IDB->TMIC0 = PR_3            /* timer int.s priority 3 */
          + INT_EN             /* normal int. for TM0 */
          + IMK;               /* mask int. for now */

       IDB->TMIC1 = MS_EN      /* macro service for MD0 */
          + IMK;               /* set int. mask */

       IDB->TMIC2 = INT_EN     /* int. with context switch */
          + ENCS               /* for TM1 */
          + IMK;               /* mask int. for now */

       IDB->DMAM0 = DR_MIO; /* DMA ch. 0 set to demand */
                                    /* release mode */

       IDB->DMAM1 = BM;        /* DMA ch. 1 set to burst mode */

};
```

```
;
; Start up module for V25 'C' programs
;
; modified 9/4/87 for use with begin.c
;
RAM_TOP         equ     8000h   ;define top of ram assuming 0 start
IDB             equ     0fh     ;set IDB area to be at top of first
                                ;64k of memory. ie 0fe00 - 0ffff
IDB_BASE        equ     0f000h
IDB_PTR         equ     0fffffh ;IDB register at 0ffffh after reset

        name    start_up
        public  start_up,_IDB

DGROUP  GROUP   CONST,DATA

        ASSUME DS0:DGROUP

CONST   SEGMENT PARA PUBLIC 'CONST'

BUFFER  DBS     2FFFH   ;Dummy vector area to avoid V25 MINI IE
BUF_END DBS     1       ;vector and working area.

CONST   ENDS

DATA    SEGMENT PARA PUBLIC 'DATA'

_IDB    DW      ?

DATA    ENDS

CGROUP  GROUP   CODE

        ASSUME  PS:CGROUP

CODE    SEGMENT PARA PUBLIC 'CODE'

                extrn   _MAIN:near

start_up:       mov     sp,RAM_TOP  ;set stack
                mov     aw,IDB_BASE
                mov     ds1,aw      ;define top 64k of memory
                mov     al,IDB
                mov     ix,IDB_PTR
                mov     ds1:byte ptr [ix],al ;set IDB register
                mov     aw,((IDB and 0fh) shl 12)
                mov     _IDB,aw ;define 16 bit offset to
                                ;IDB area to be available in 'C'
                                ;as _IDB global structure ptr.
                call    _MAIN
CODE    ENDS
        END
```

Serial Bus Interface (SBI) for μPD78310/312

Contents:

Author: W. Knippschild
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μCOM78 I/II/III 8/16 Bit Family	Product Description Microcomputer
μPD78C10/11/12/14	Product Description
μPD70320/322	CMOS 16-Bit Microcomputer

Related Products

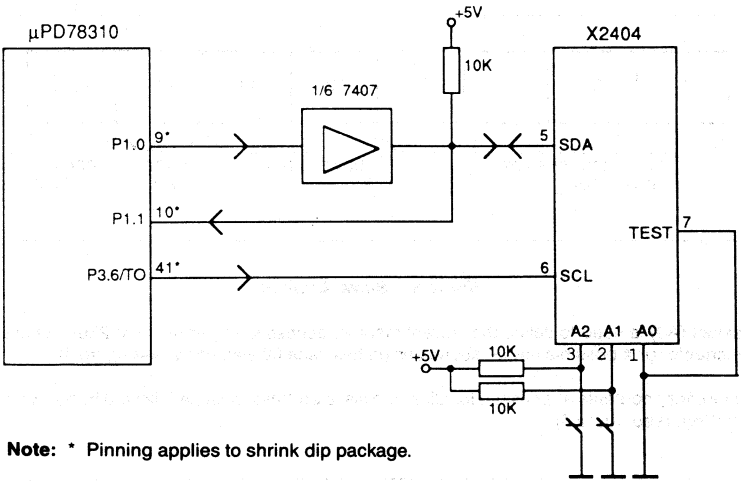
μPD78310/312	8-Bit Microcomputer	CMOS
μPD7807/08/09	8-Bit Microcomputer	NMOS
μPD78C10/11/12/14	8-Bit Microcomputer	CMOS
μPD70320/322	16-Bit Microcomputer	CMOS
μPD70330/332	16-Bit Microcomputer	CMOS

1. Introduction

The Serial Bus Interface (SBI) is a powerful serial communication procedure for microprocessor systems. It forsees wake-up, address, command and data pattern transfers to efficiently support a serial network with one master and up to 256 slave devices, depending on the specification of the attached components. NEC's microcontrollers μPD78210/220/224 and μPD78320/322 support this interface by special on-chip hardware, independent of built-in asynchronous interface (UART). If it is required to interface a μPD78310/312 microcontroller to a SBI-device, the necessary functions must be implemented by software. This application note provides an example of how to establish a serial bus connection between μPD78310/312 and an EEPROM (Xicor 2404).

2. Hardware

Figure 1 shows the hardware used to run the serial bus interface to an EEPROM, tested on an EBIBM-78310 board. A SN74LS07 hex-buffer is connected to the data-line of the microprocessor (P1.0) in order to have the necessary open-collector interface to the EEPROM. Any other open-collector/open-drain buffer should also work. The level of the bidirectional data-line is monitored by P1.1. Timer 0 functions as data clock and outputs a square wave on P3.6. Address definition of the EEPROM X2404 is done by selecting the levels of A1 and A2. A0 and TEST pins must be pulled to ground according to Xicor databook specifications.



Note: * Pinning applies to shrink dip package.

Figure 1. Circuit Diagram

3. Description of Communication Protocol

A communication between master (μPD78310/312) and slave (EEPROM) starts at the negative edge on dataline when the clockline is high (see figure 2).

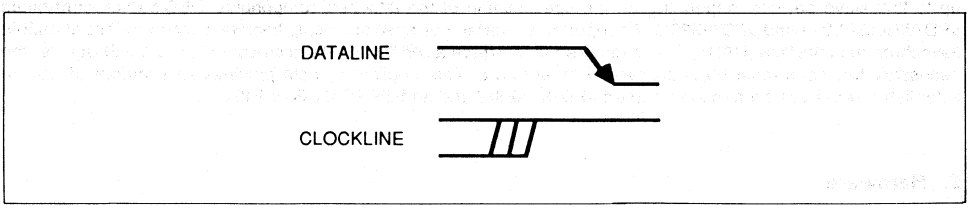


Figure 3. Start Condition

After transmitting an 8 bit slave-address the master device receives an acknowledge (dataline pulled to low) from the slave device at the ninth clock cycle. The data itself, such as slave-address, word-address and data pattern, always change when the clockline gets low and is clocked in at the rising edge of the clockpulse. The slave-address is used to select one of the connected slave-devices (see figure 3). The word address selects the desired location in the EEPROM for either read or write operation.

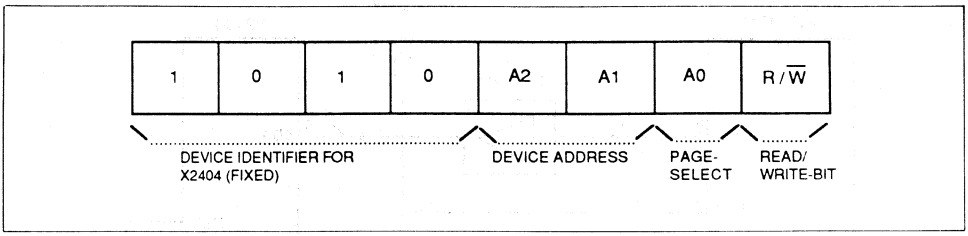


Figure 3. Slave Address

Note: Page select (A0) is used to define the current memory access to upper or lower 256-byte area. A1 and A2 are switchable for 4 possible device addresses (in this case 00 was used, see figure 1).

To change the memory contents at a desired location, the slave address must be followed by the word address and the new data pattern (see figure 4).

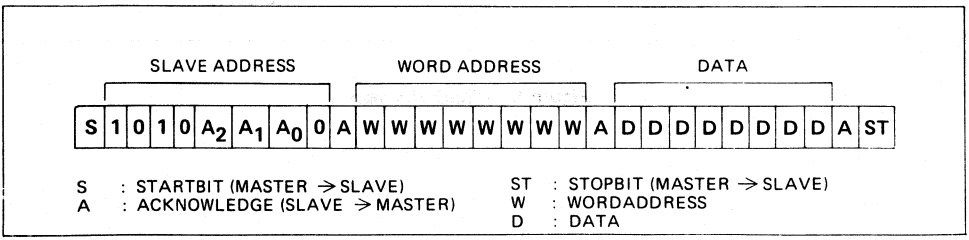


Figure 4. Sequence for Changing a Byte

The reception of slave address, word address and data pattern are acknowledged by the slave if the data transfer was successful (always at 9th clockcycle). Transmission is terminated by sending a stop condition, that is, a positive edge on dataline when clockline is high (see figure 5).

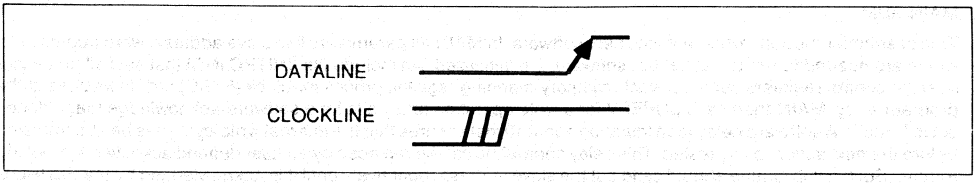


Figure 5. Stop Condition

A read operation always refers to the current word address that is stored in an EEPORM register. This register is maintained internally in the EEPROM. Every read or write cycle automatically increments the register contents by one. So if reading of a specific address is required, the user must insert a dummy write-cycle in order to set the new word address (see figure 6). For other sequences such as sequential read, page write and read at current address please refer to Xicor databook specifications.

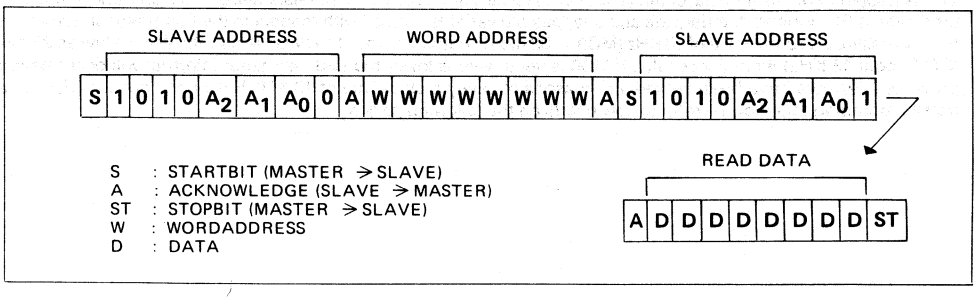


Figure 6. Random Read

4. Software

The software consists of three modules:

MAIN.ASM:

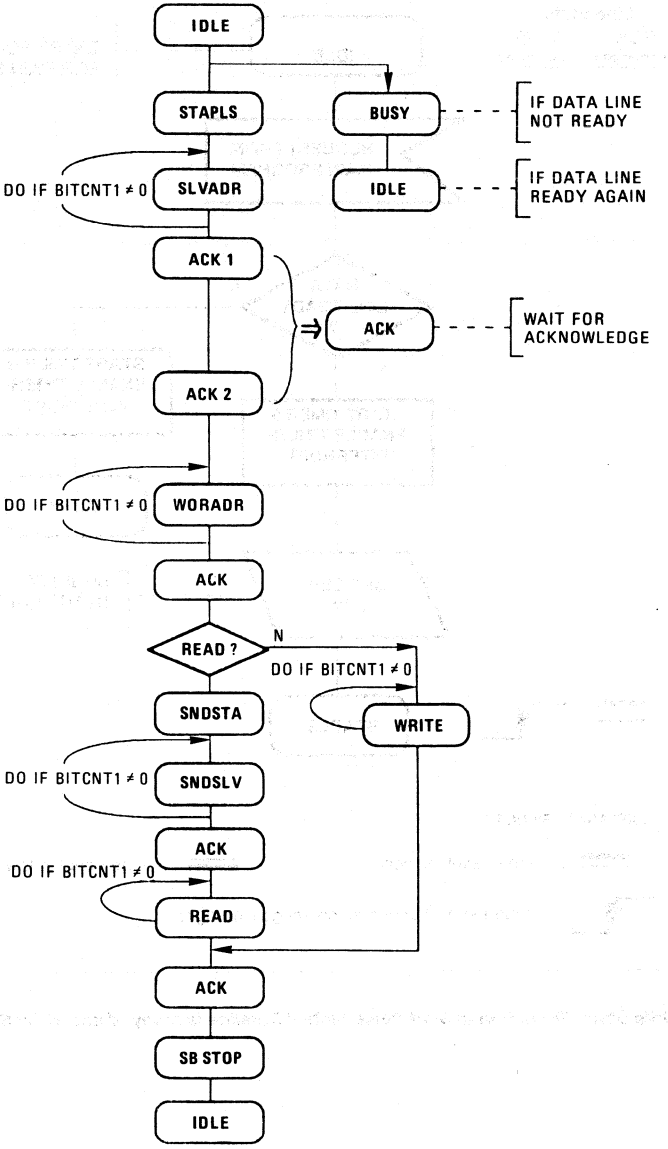
This assembler module represents the user software. In MAIN all parameters like slave address, word address etc. which are needed to run the serial bus software, are initialized. An include file (INITPC.INC) that sets all processor specific control registers such as; stack, memory-mapping-register, processor-clock (STBC) etc., is invoked at the program entry. MAIN then calls SUPERV for a write operation and waits for a software-acknowledge (flag "done" set to "true"). A software delay loop (duration about 10 ms) ensures that the internal write-cycle has been completed before the next action is requested. This delay subroutine can be replaced by an user defined acknowledge-polling routine. Such a subroutine should send out the slave address repetitively until it is acknowledged by the EEPROM at the end of the internal write-cycle (see remark in SUPVIS.ASM, state ACK2: "call error-handler").

SBICOM.ASM:

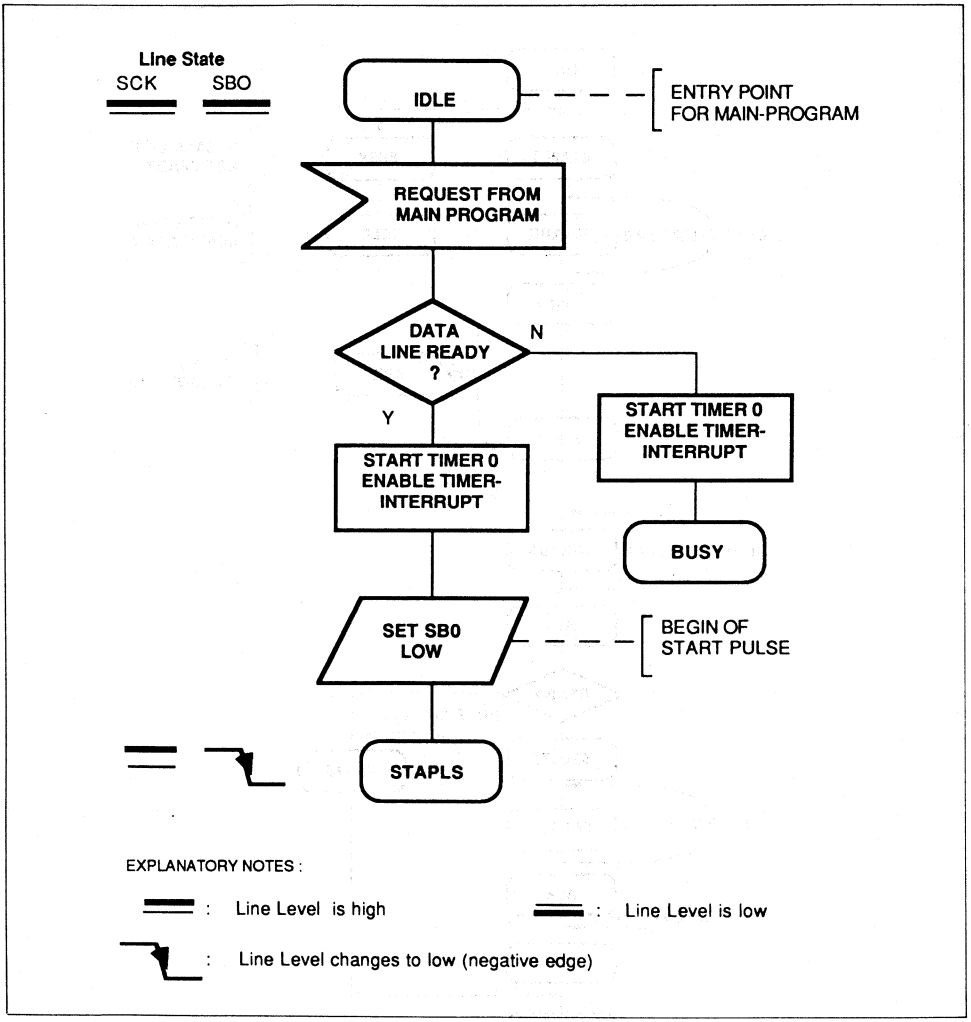
Contains common data used by MAIN and SUPVIS. The data is located in the short address area to ease parameter passing by usage of saddr-addressing.

SUPVIS.ASM:

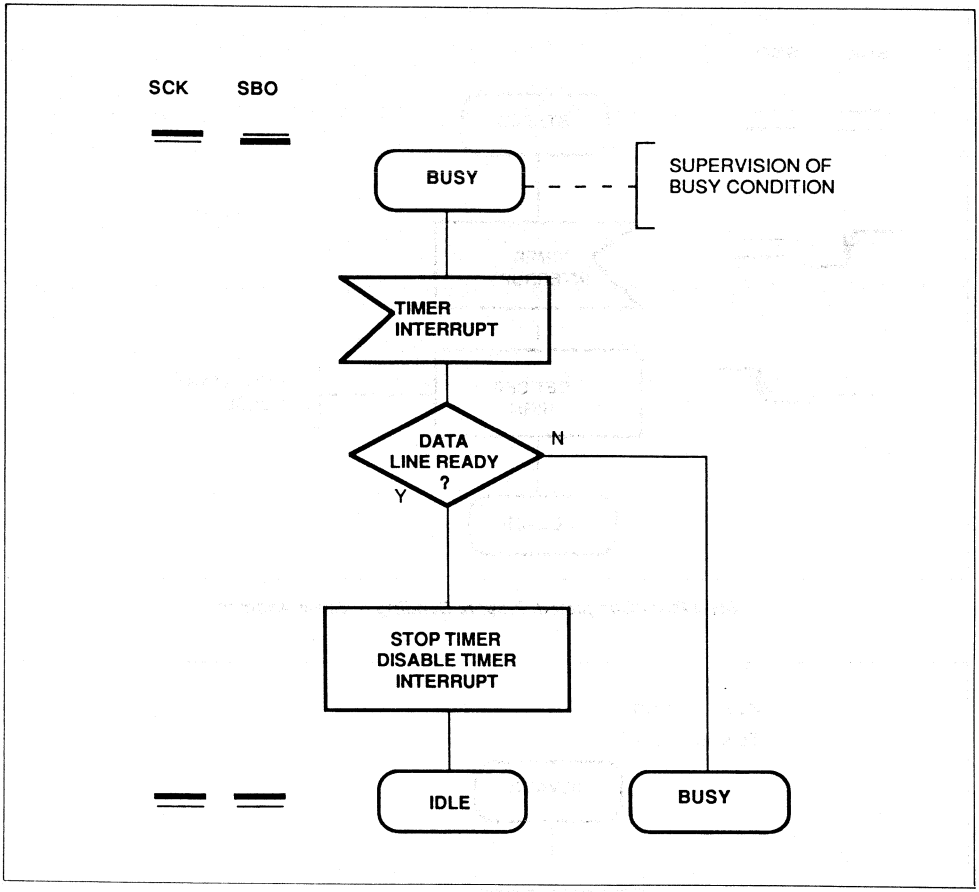
This is a supervisor module that controls all access to the serial bus. The supervisor starts and stops timer 0 and outputs data at P1 monitors the dataline and changes the variable "state" with respect to the bus-protocol. When the timer is active, a square wave of 10 KHz (MD0 = 32 hex) is output to clock the data. Tests with MD0 as low as 26 hex (SCK about 13 KHz) were successful. If MD0 is set to values lower than 26 hex, timer interrupt will occur before previous interrupts have been served. For reasons of safety a clock rate of 10 KHz has been choosen. For more details regarding this module refer to flow-charts on the following pages.



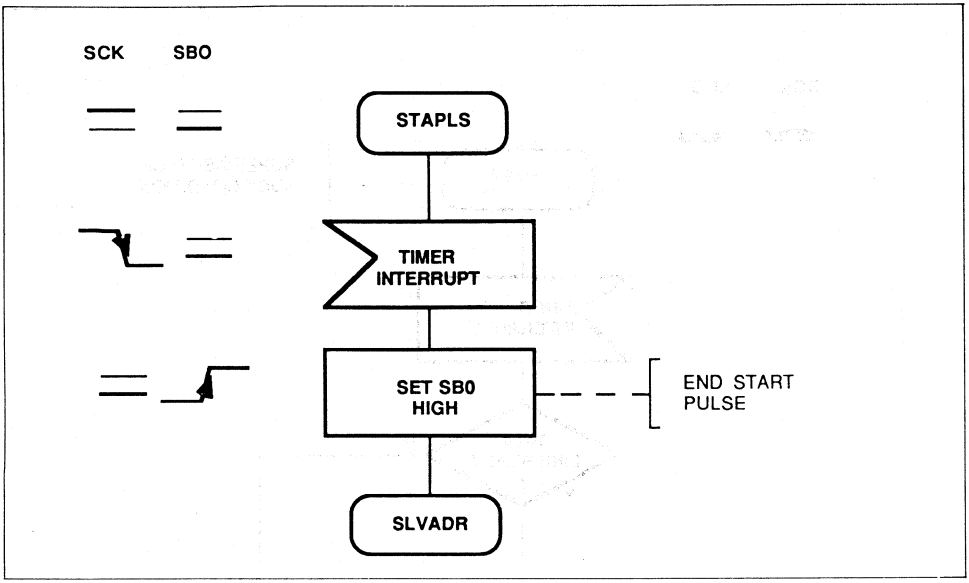
State Transition Table of Module SUPVIS



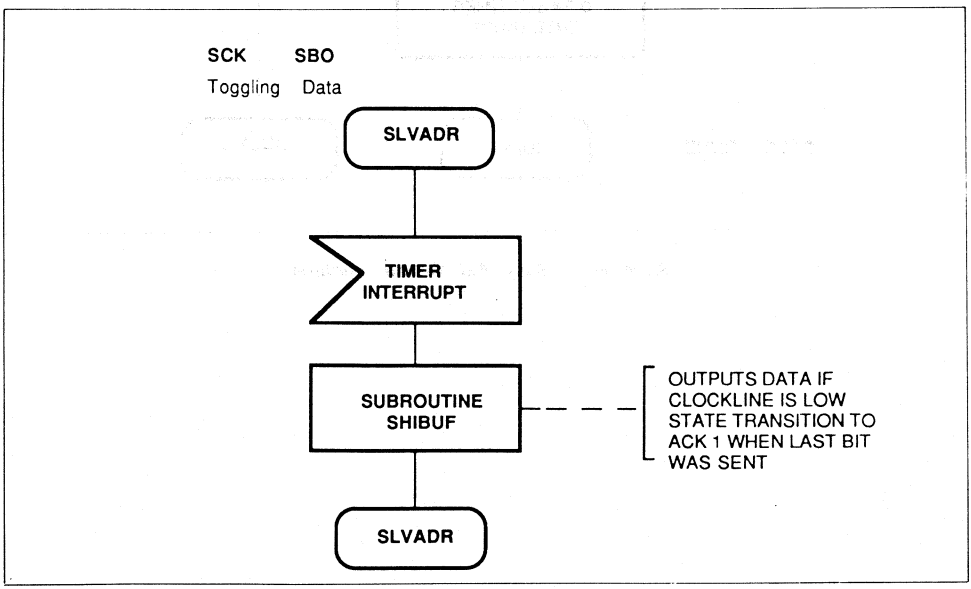
Idle State: Transition to Start Pulse State if Dataline is ready, if not: Busy Supervision



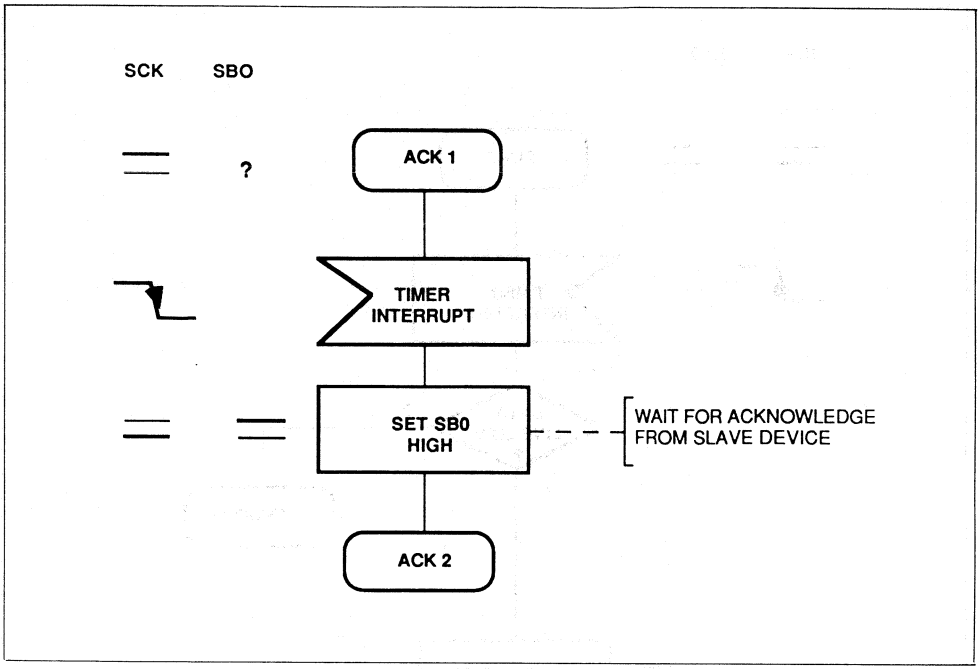
Slave Device Busy: Wait for Idle Condition



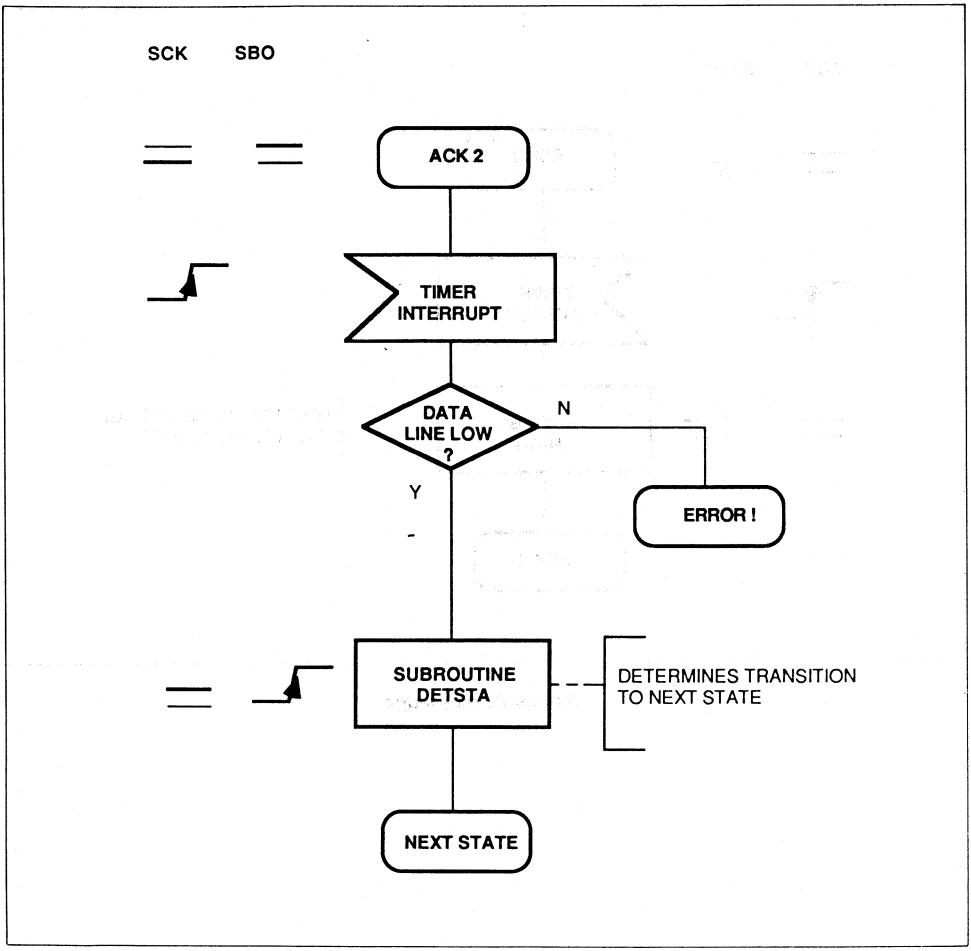
Start Pulse Completed, Prepare Sending of Slave Address



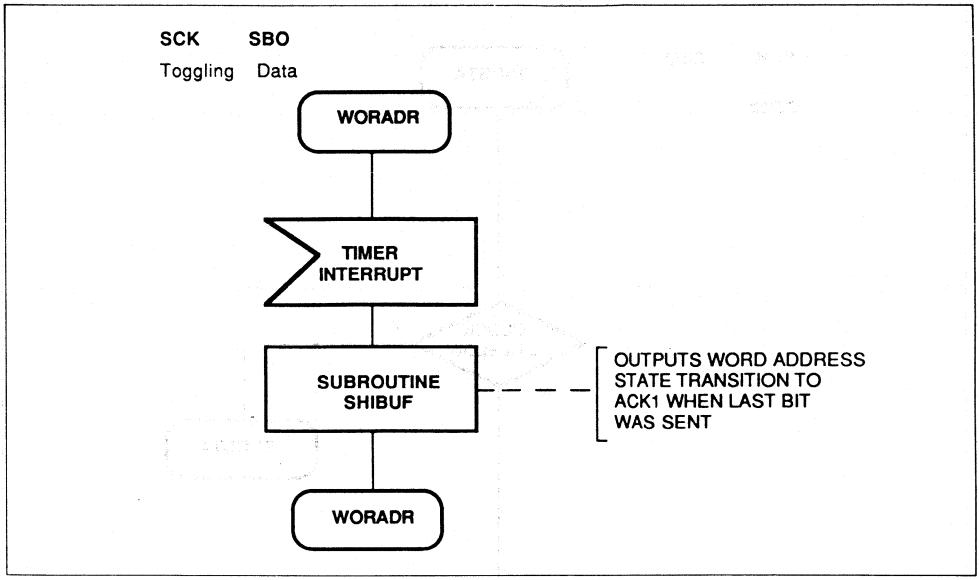
Send Slave Address



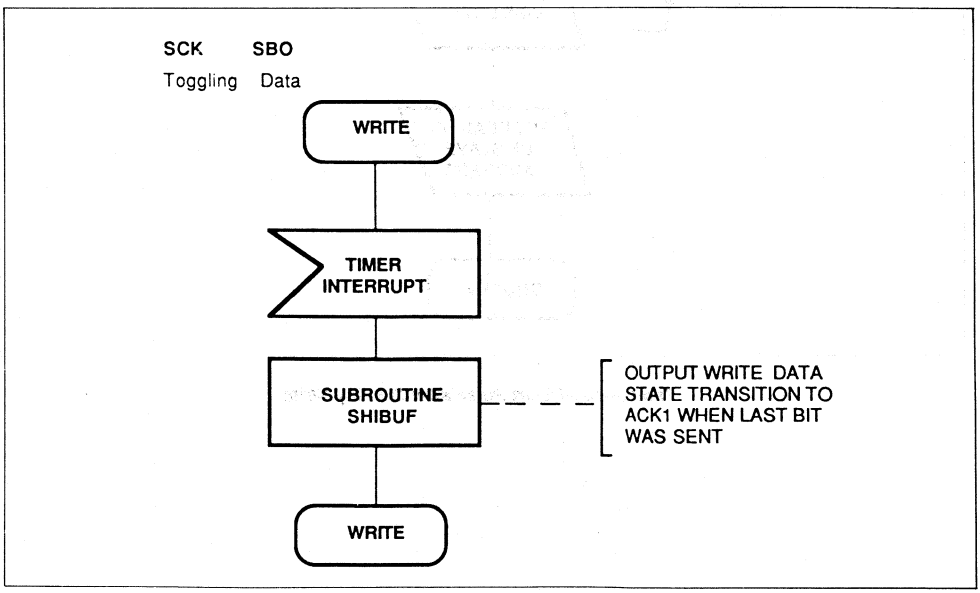
Wait for Acknowledge



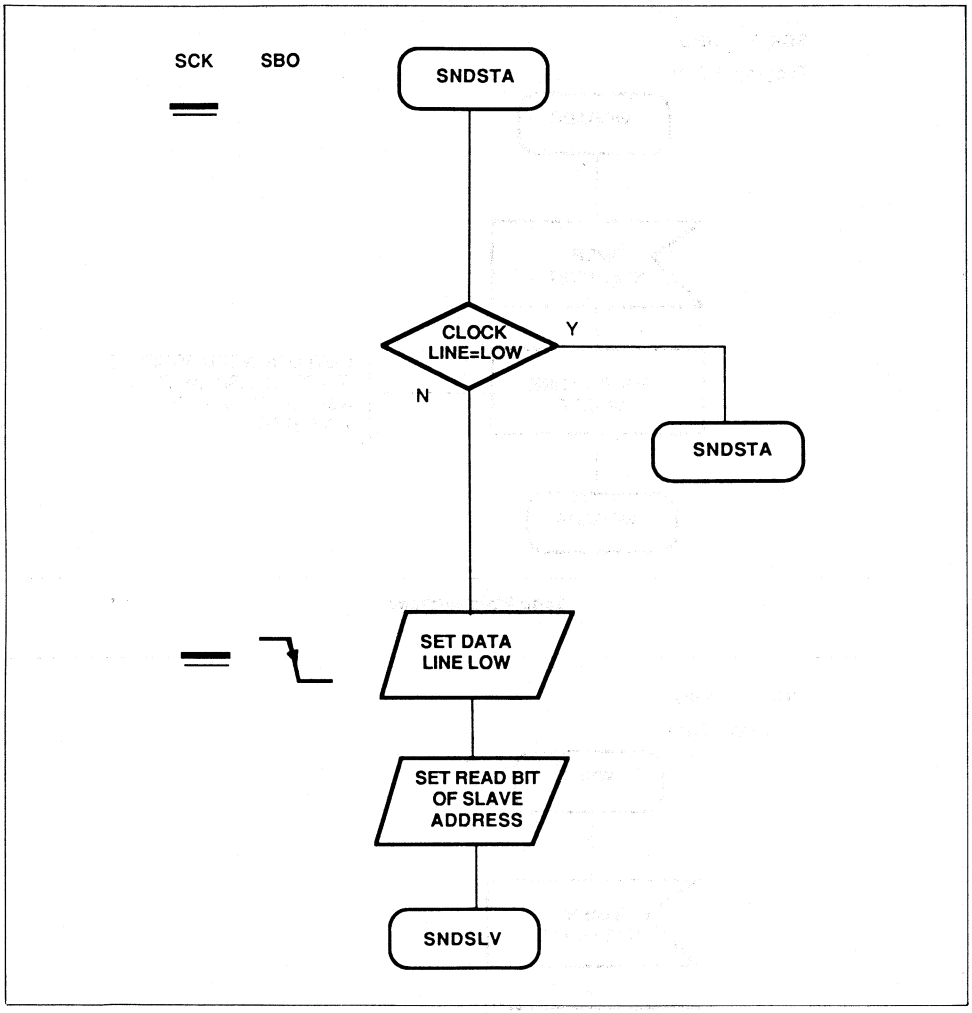
Acknowledge Received



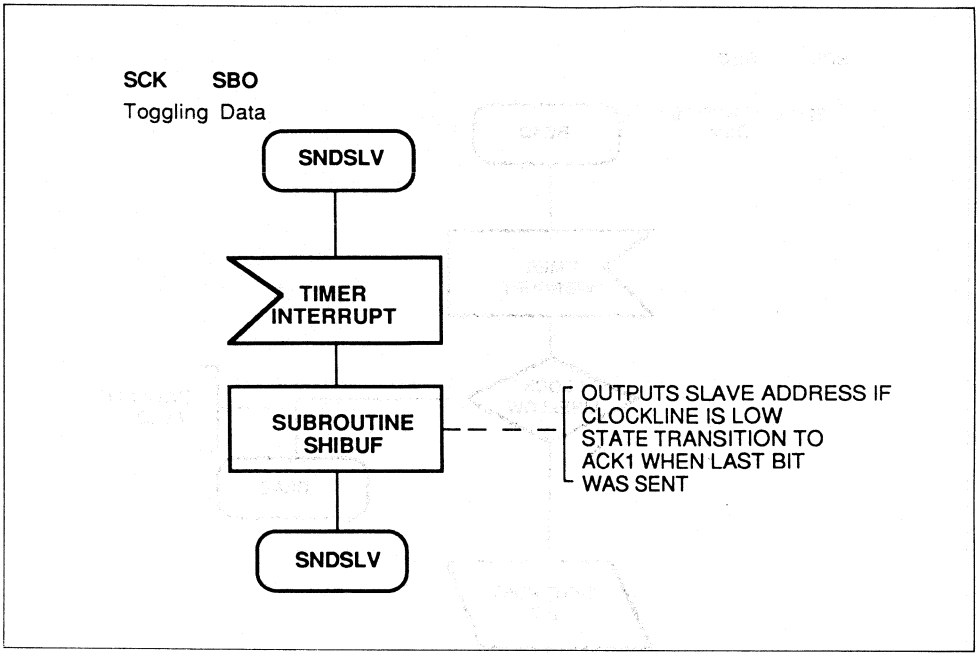
Send Word Address



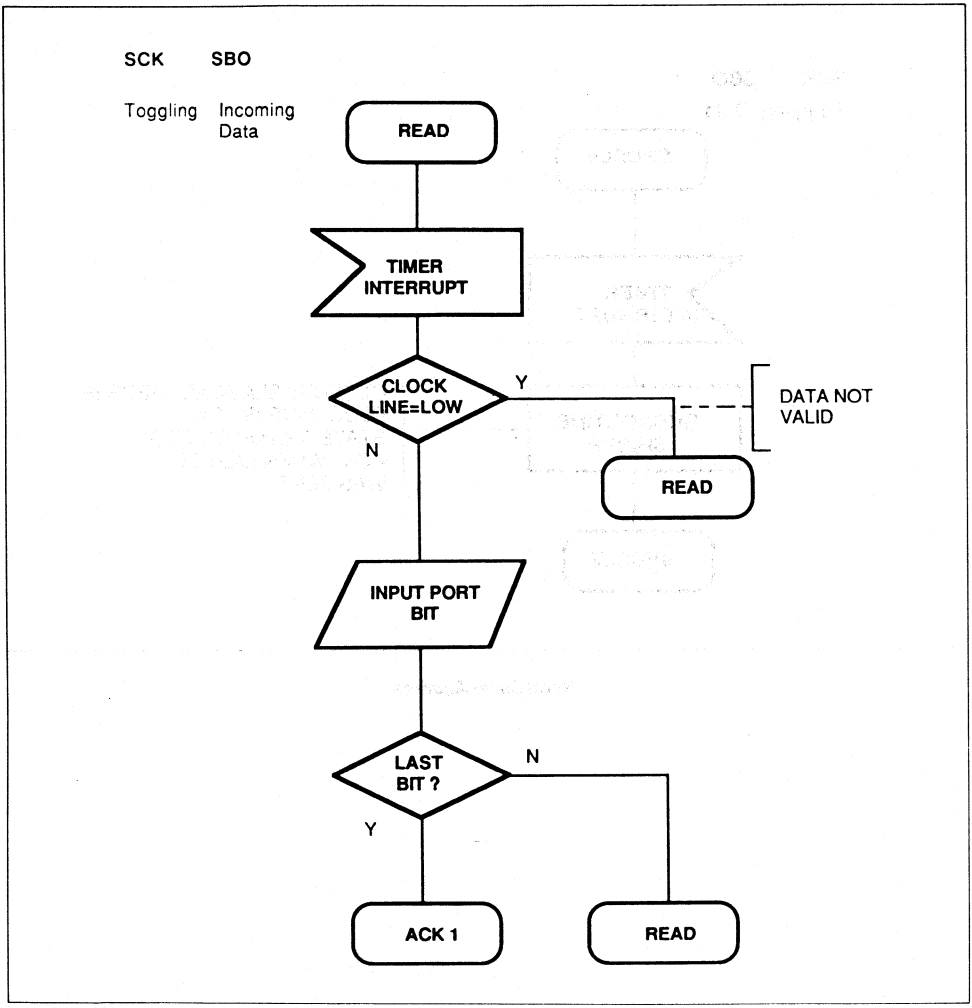
Output Write Data



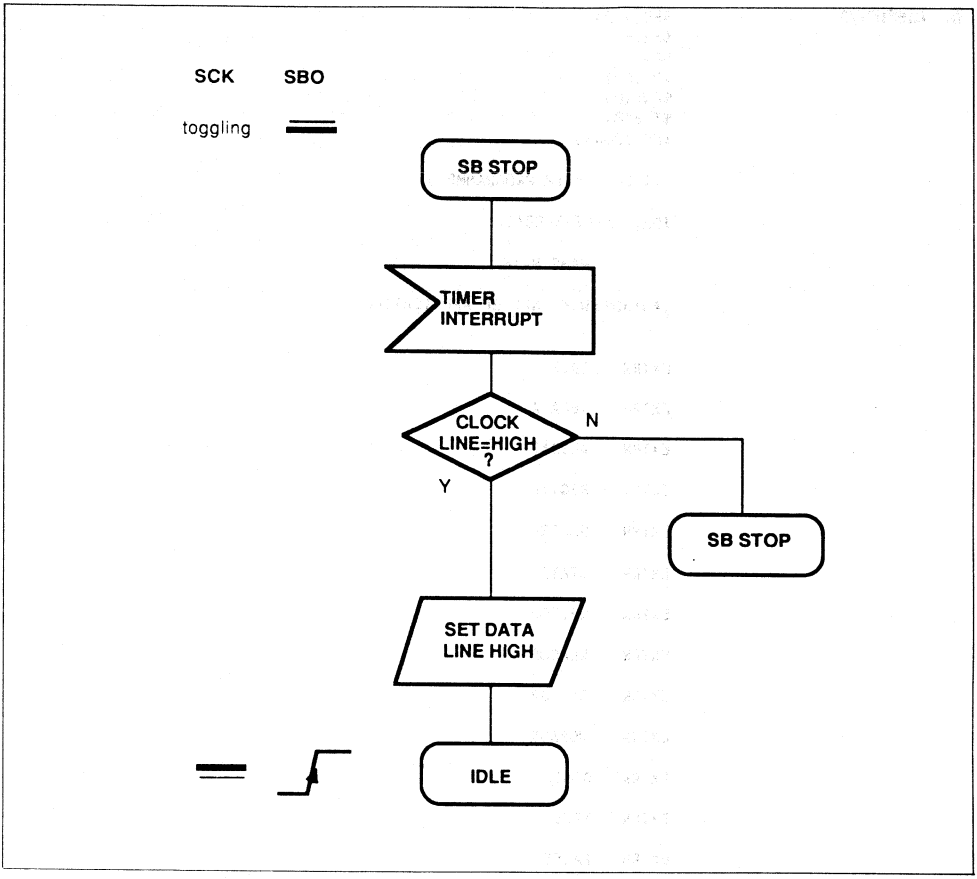
Send Second Start Pulse after Dummy Write



Send Slave Address


SCK
SBO
Toggling
Incoming Data
READ
TIMER INTERRUPT
CLOCK LINE=LOW
Y
DATA NOT VALID
N
READ
INPUT PORT BIT
LAST BIT ?
N
Y
ACK 1
READ

Read Port



Send Stop Pulse

6. Listings

```
$PC(312)
$XREF
$DG
$PL(60)
$PW(100)
$TAB(8)
$EP (CON:)

$TITLE ('MAIN PROGRAMME')

$DATE ('07-3-88')

        NAME MAIN

;AUTHOR:W.K. NEC EE APPLICATION


EXTRN   IDLE

EXTRN   REDATA

EXTRN   WRIDAT

EXTRN   REQTYP

EXTRN   SUPERV

EXTRN   STATE

EXTRN   WRITOP

EXTRN   READOP

EXTRN   SLVADR

EXTRN   WORADR

EXTRN   DONE

EXTRN   TRUE

EXTRN   FALSE


DSEG

        ORG     0000H                   ;RESET VECTOR

RESET:  DW      INIT

        ORG     000EH                   ;INTERRUPT VECTOR TIMER

SUPVIS: DW      SUPERV

ENDS
```

```
        CSEG

        $INCLUDE (A:INITPC.INC)

START:  CMPW    STATE,#IDLE             ;ASK FOR CURRENT SUPERVISOR STATE
        BNZ     $L_END                  ;ACCESS NOT POSSIBLE
        MOV     SLVADR,#10100000B       ;INIT SLAVE-ADDRESS CAUTION: NEVER
                                        ;SET BIT0 (R/W-BIT) TO 1 - THIS IS
                                        ;DONE BY SUPERVISOR PROGRAM IF A
                                        ;READ-OPERATION IS REQUESTED !
        MOV     WORADR,#10100101B       ;INIT WORD-ADDRESS
        MOV     REQTYP,#LOW (WRITOP)    ;INIT REQUEST TYPE
        MOV     WRIDAT,#0FH             ;DATA TO BE WRITTEN
        MOV     DONE,#LOW(FALSE) ;INIT COMPLETION CODE
        CALL    !SUPERV                 ;REQUEST TO SUPERVISOR
L_CMP1: CMP     DONE,#LOW(TRUE)         ;REQUESTED ACTION PERFORMED?
        BNZ     $L_CMP1
        MOV     REQTYP,#LOW(READOP)     ;DONE! NEXT ACTION NOW
        MOV     DONE,#LOW(FALSE) ;RE-INIT COMPLETION-CODE
L_CMP2: CMPW    STATE,#IDLE             ;ACCESS POSSIBLE?
        BNZ     $L_CMP2
        CALL    !DELAY                  ;EEPROM NEEDS UP TO 10MS FOR
                                        ;WRITE CYCLE, SO TO PREVENT NO-
                                        ;ACKNOWLEDGE-SITUATIONS, INSERT
                                        ;DELAY
        CALL    !SUPERV                 ;REQUEST TO SUPERVISOR
L_END:  BR      $$                      ;ENDLESS LOOP


DELAY:  MOVW    RP7,#2B0H               ;THIS SUBROUTINE INSERTS A DELAY OF
                                        ;ABOUT 10MS
D1:     DIVUX   RP0                     ;
        DECW    HL                      ;
        MOVW    AX,HL                   ;
        CMPW    AX,#0                   ;
        BNZ     $D1                     ;
        RET                             ;

        ENDS            SET             ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      STANDARD  INITIALIZATION OF  THE 78310  PROCESSOR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

INSP            EQU     0FE20H              ;STACKPOINTER START ADDRESS
INSTBC          EQU     00001000B           ;SYSTEM CLOCK SCLK=FOSC/2, RUN MODE, STAND BY
                                            ;FLAG RESET
INMM            EQU     00000111B           ;NO WAIT, 64 KBYTE EXTERNALLY

INIT:           MOVW    SP,#INSP
                MOV     PSWH,#00H           ;RESET VALUE
                MOV     PSWL,#00H           ;RESET VALUE
                MOV     CCW,#00H            ;RESET VALUE OF CPU CONTROL WORD
                MOV     STBC,#INSTBC        ;
                MOV     MM,#INMM
                MOV     RFM,#00H            ;NO REFRESH MODE (RESET VALUE IS 10H!!)
                SEL     RB2                 ;RB0,RB1 MAY BE OCCUPIED BY MACRO
                                            ;SERVICE

                RSS     0                   ;RESET THE RSS BIT LOGICALLY (PSEUDO-
                                            ;INSTRUCTION OF ASSEMBLER)
                                            ; (PHYSICALLY RESET IN PSW)

                MOVW    STATE,#IDLE         ;INIT DEVICE-STATE OF SUPERVISOR

                EI

                BR      START

$EJECT
*
```

```
$PC(312)
$XREF
$DG
$PL(60)
$PW(100)
$TAB(8)
$EP (CON:)

$TITLE ('SBI COMMON DATA')

$DATE ('07-3-88')

        NAME SBICOM

;AUTHOR:W.K. NEC EE APPLICATION

PUBLIC  BITCNT

PUBLIC  DATBUF

PUBLIC  NEXSTA

PUBLIC  REDATA

PUBLIC  WRIDAT

PUBLIC  REQTYP

PUBLIC  STATE

PUBLIC  WRITOP

PUBLIC  READOP

PUBLIC  SLVADR

PUBLIC  WORADR

PUBLIC  DONE

PUBLIC  TRUE

PUBLIC  FALSE

DSEG

SADDR   ORG     0FE20H                  ;SHORT ADDRESS AREA


BITCNT: DS (1)                          ;CONTAINS VALUE OF BITPOSITION TO BE
                                        ;IN/OUTPUT NEXT

DATBUF: DS (1)                          ;TEMPORARY BUFFER FOR SHIFT OPERATIONS

NEXSTA: DS (2)                          ;CONTAINS NEXT STATE OF SUPERVISOR
                                        ;MODULE

STATE:  DS (2)                          ;CONTAINS CURRENT STATE OF SUPERVISOR
                                        ;MODULE

REQTYP: DS (1)                          ;CONTAINS REQUEST TYPE (READ OR WRITE
                                        ;OPERATION)
```

```
REDATA: DS (1)          ;CONTAINS VALUE READ FROM SLAVE

WRIDAT: DS (1)          ;CONTAINS VALUE TO BE WRITTEN INTO
                        ;SLAVEDEVICE

SLVADR: DS (1)          ;CONTAINS SLAVE-ADDRESS

WORADR: DS (1)          ;CONTAINS WORD ADDRESS

DONE:   DS (1)          ;CONTAINS COMPLETION CODE (TRUE/FALSE)
                        ;OF REQUIRED ACTION

ENDS



READOP  EQU     00      ;ELEMENT OF REQUEST TYPE

WRITOP  EQU     01      ;ELEMENT OF REQUEST TYPE

TRUE    EQU     01

FALSE   EQU     00

END
```

```
$PC(312)
$XREF
$DG
$PL(60)
$PW(100)
$OP(5)
$TAB(8)
$EP (CON:)

$TITLE (' SUPERVISOR MODULE SBI ')

$DATE ('07-3-88')

        NAME SUPVIS

;AUTHOR:W.K. NEC EE APPLICATION


EXTRN   STATE

EXTRN   NEXSTA

EXTRN   DATBUF

EXTRN   BITCNT

EXTRN   REQTYP

EXTRN   READOP

EXTRN   SLVADR

EXTRN   WORADR

EXTRN   WRITOP

EXTRN   REDATA

EXTRN   WRIDAT

EXTRN   DONE

EXTRN   TRUE

PUBLIC  IDLE

PUBLIC  SUPERV
```

```
        CSEG                            ;RELOC. CODESEGMENT FOR SUPERVISOR

        ;MODULE SPECIFIC BITNAMES
        ;************************

SBOMON  EQU     P1.1                    ;SBO MONITOR: READ DATA-LINE

SBO     EQU     P1.0                    ;SBO: DATA-LINE (OUTPUT)

SCK     EQU     P3.6                    ;CLOCK LINE

;*****************************************************************************
;*                                                                           *
;*  SUBROUTINE SETIDL: SET IDLE CONDITION FOR SERIAL-BUS-INTERFACE:          *
;*                     DISABLE CLOCK AND SET CLOCK-LINE HIGH, SET DATA-      *
;*                      LINE HIGH; THIS SUBROUTINE IS ALSO USED FOR INITIALI-*
;*                      SATION OF SERIAL-BUS-INTERFACE                       *
;*                                                                           *
;*****************************************************************************

SETIDL: MOV     PM1,#02                 ;SET PORT1.0 TO OUTPUT, P1.1 TO INPUT

        SET1    SBO                     ;SET DATA-LINE TO IDLE-CONDITION

        SET1    PMC3.6                  ;SET CONTROL-MODE OF TOO-PIN

        SET1    P3.6                    ;SET CLOCK-LINE TO HIGH


        SET1    PM3.6                   ;ENABLE OUTPUT OF CLOCK-SIGNAL

        MOV     TMC0,#00        ;INIT TIMER0-CONTROL-REGISTER:
                                ;       COUNT DISABLED,
                                ;       FCKL/6 = TIMER SOURCE
                                ;       OUTPUT TOO FIXED TO HIGH

        MOVW    MD0,#32H        ;SCK = 10KHZ

        RET

;*****************************************************************************
;*                                                                           *
;*  SUBROUTINE DETSTA: DETERMINE NEXT STATE                                  *
;*                                                                           *
;*****************************************************************************

DETSTA: CMPW    STATE,#READ             ;DETERMINE NEXT STATE

        BZ      $L_DET0

        CMPW    STATE,#WRITE

        BNZ     $L_DET3
```

```
L_DET0: MOVW    NEXSTA,#SBSTOP          ;IF STATE = WRITE NEXSTA BECOMES
                                        ;SBSTOP (SEND STOP-PULSE)

        MOV     DATBUF,WRIDAT           ;INIT DATABUFFER FOR WRITE-OPERATION

        BR      L_DET2                  ;END SUBROUTINE

L_DET3: CMPW    STATE,#WORDAD           ;IF STATE CHANGES TO WORDAD, NEXSTA
                                        ;HAS TO BE CHANGED ACCORDING TO
                                        ;REQUEST-TYPE (READ OR WRITE-OPERATION)

        BNZ     $L_DET2                 ;BRANCH TO END IF STATE /= WORDAD

        CMP     REQTYP,# LOW (READOP)   ;READ-OPERATION REQUESTED?

        BNZ     $L_DET1                 ;NO!

        MOVW    NEXSTA,#SNDSTA          ;INIT NEXT STATE FOR ACTIONS AFTER
                                        ;ACKNOWLEDGE OF WORD ADDRESS

        MOV     DATBUF,WORADR           ;WORD-ADDRESS IS OUTPUT NEXT

        BR      L_DET2

L_DET1: MOVW    NEXSTA,#WRITE           ;INIT NEXT STATE FOR ACTIONS AFTER
                                        ;ACKNOWLEDGE

        MOV     DATBUF,WORADR           ;INIT DATABUFFER

L_DET2: RET

;*****************************************************************************
;*                                                                           *
;*  SUBROUTINE SHIBUF: OUTPUT BIT ON DATA-LINE                               *
;*                                                                           *
;*****************************************************************************

SHIBUF: BT      SCK,$L_RET              ;READ STATUS OF CLOCK-LINE, IF CLOCK
                                        ;=HIGH -> DON'T CHANGE DATA ON SBO

        MOV     A,DATBUF        ;CLOCK=LOW -> OUTPUT NEXT BIT ON SBO

        SHL     A,1                     ;PUT MSB INTO CARRY

        MOV1    SBO,CY                  ;OUTPUT ON PORT

        MOV     DATBUF,A        ;RESTORE DATA-BUFFER

        DEC     BITCNT                  ;DECREMENT BITCOUNTER

        BNZ     $L_RET                  ;LAST BIT SENT?

        MOVW    STATE,#ACK1             ;LAST BIT SENT! CHANGE STATE (WAIT
                                        ;FOR ACKNOWLEDGE)

        MOV     BITCNT,#08              ;RE-INIT BITCOUNTER

L_RET:  RET
```

```
;*****************************************************************************
;*                                                                           *
;*  SUPERVISOR MODULE: SUPERVISION OF SERIAL-BUS-INTERFACE ACTIVITIES, SUCH  *
;*  AS READ/WRITE-CONDITION, IDLE OR BUSY STATE. TIMER INTERRUPTS (TIMER0)   *
;*  ARE PROCESSED ACCORDING TO PRESENT STATE.                                *
;*                                                                           *
;*****************************************************************************

SUPERV: MOVW    AX,STATE        ;RETRIEVE PREVIOUS SUPERVISOR-STATE

        BR      AX              ;BRANCH TO STATE-DEPENDING LABEL


;*******************************************
;*                                         *
;* STATE = IDLE: CHECK AVAILABILITY OF DATA- *
;*               LINE                      *
;*                                         *
;*******************************************

IDLE:   BT      SBOMON,$L_ID    ;DATA-LINE READY? IF READY JUMP TO L_ID

        MOVW    STATE,#BUSY     ;DATA-LINE BUSY: CHANGE STATE TO BUSY

        SET1    TMC0.7          ;START TIMER0 (PURPOSE: SUPERVISION OF
                                ;BUSY-CONDITION)

        CLR1    TMIC0.6         ;ENABLE TIMER0 INTERRUPT

        RET                     ;SUPERVISOR WAS INVOKED BY MAINPRO-
                                ;GRAMME, THEREFORE RET STATEMENT
                                ;REQUIRED

L_ID:   CALL    !SETIDL         ;SET PROPER HW-CONDITION

        MOVW    STATE,#STAPLS   ;CHANGE STATE TO 'START-PULSE'

        OR      TMC0,#88H       ;START TIMER 0 AND ENABLE TOGGLE-
                                ;MECHANISM FOR TO-OUTPUT

        CLR1    TMIC0.6         ;ENABLE TIMER-INTERRUPT

        CLR1    SBO             ;OUTPUT LOW LEVEL = STARTPULSE

        RET                     ;SEE REMARK AT L_ID
```

```
;*********************************************
;*                                           *
;* STATE = BUSY: ENABLE TIMERINTERRUPT,      *
;*               WAIT FOR IDLE               *
;*                                           *
;*********************************************

BUSY:   BF      SBOMON,$L_BU            ;FETCH STATUS OF DATA-LINE, IF NOT
                                        ;READY: RETURN

        MOVW    STATE,#IDLE             ;UPDATE STATE

        CLR1    TMC0.7                  ;STOP TIMER

        SET1    TMIC0.6                 ;DISABLE TIMER-INTERRUPT

L_BU:   RETI


;*********************************************
;*                                           *
;* STATE = STAPLS: START-PULSE WAS SENT TO   *
;*                 SLAVE-DEVICE              *
;*                                           *
;*********************************************

STAPLS: SET1    SBO                     ;TIMER EXPIRED FIRST TIME: END
                                        ;STARTPULSE

        MOVW    STATE,#SLAVAD           ;CHANGE STATE

        MOVW    NEXSTA,#WORDAD          ;SAVE NEXT STATE FOR ACTIONS AFTER
                                        ;ACKNOWLEDGE

        MOV     BITCNT,#08              ;INIT BITCOUNTER

        MOV     DATBUF,SLVADR           ;SLVADR HAS TO BE OUTPUT

        CALL    !SHIBUF                 ;CALL OUTPUT-ROUTINE

        RETI


;*********************************************
;*                                           *
;* STATE = SLAVAD: START-PULSE WAS SENT TO   *
;*                 SLAVE-DEVICE              *
;*                                           *
;*********************************************

SLAVAD: CALL    !SHIBUF                 ;OUTPUT DATA

        RETI
```

```
;*********************************************
;*                                           *
;* STATE = ACK1: RELEASE DATA-LINE TO ENABLE *
;*               OUTPUT OF ACK-SIGNAL FROM   *
;*               SLAVE-DEVICE                *
;*                                           *
;*********************************************

ACK1:   BT      SCK,$L_ACR              ;CLOCK-LINE HIGH? IF HIGH: RETURN

        SET1    SBO                     ;SET DATA-LINE HIGH BECAUSE CLOCK-LINE
                                        ;IS LOW (IF SBO IS SET TO HIGH WHEN SCK
                                        ;IS HIGH THIS WOULD BE A STOP COND.)

        MOVW    STATE,#ACK2             ;WAIT FOR NEXT PHASE OF ACK-SEQUENCE

L_ACR:  RETI

;*********************************************
;*                                           *
;* STATE = ACK2: WAIT FOR LOW SIGNAL ON      *
;*               DATA-LINE AT 9TH CLOCK      *
;*               CYCKLE                      *
;*                                           *
;*********************************************

ACK2:   BF      SCK,$L_AC2              ;BRANCH IF LOW SIGNAL ON CLOCK-
                                        ;LINE: WAIT FOR POSITIVE EDGE OF
                                        ;9TH CLOCK-CYCKLE

        BF      SBOMON,$L_AC1           ;IS DATA LINE PULLED TO LOW AFTER
                                        ;OCCURENCE OF 9TH CLOCK-PULSE?

        ; CALL ERROR-HANDLER (NO ACKNOWLEDGE RECEIVED)
        ; THE USER SHOULD CODE THIS SUBROUTINE ACCORDING
        ; TO HIS APPLICATION (E.G. DO A RETRY)

        RETI


L_AC1:  MOVW    STATE,NEXSTA            ;SET STATE ACCORDINGLY

        CALL    !DETSTA                 ;DETERMINE NEXT STATE

L_AC2:  RETI


;*********************************************
;*                                           *
;* STATE = WORDAD: OUTPUT WORD-ADDRESS TO    *
;*                 SLAVE-DEVICE              *
;*                                           *
;*********************************************

WORDAD: CALL    !SHIBUF                 ;OUTPUT DATA

        RETI
```

```
;*********************************************
;*                                           *
;* STATE = SNDSTA: SEND SECOND STARTPULSE    *
;*                 TO SLAVE-DEVICE AFTER     *
;*                 DUMMY WRITE CYCLE         *
;*                                           *
;*********************************************

SNDSTA: BF      SCK,$L_SND1             ;CHECK STATUS OF CLOCK-LINE

        CLR1    SBO                     ;POSITIVE EDGE OF 10TH CLOCK-CYCLE
                                        ;DETECTED: SET SBO LOW = START-PULSE

        MOVW    STATE,#SNDSLV           ;PREPARE TRANSMISSION OF SLAVE-ADDRESS
                                        ;(2ND TIME)

        MOVW    NEXSTA,#READ            ;

        MOV     DATBUF,SLVADR           ;PREPARE DATABUFFER

        INC     DATBUF                  ;SET WRITE-BIT

L_SND1: RETI

;*********************************************
;*                                           *
;* STATE = SNDSLV: SEND SAVE-ADDRESS SECOND  *
;*                 TIME AFTER DUMMY WRITE    *
;*                  CYCLE                    *
;*                                           *
;*********************************************

SNDSLV: CALL    !SHIBUF                 ;OUTPUT 2ND SLAVE-ADDRESS

                RETI

;*********************************************
;*                                           *
;* STATE = WRITE: OUTPUT WRITE DATA TO       *
;*                SLAVE-DEVICE               *
;*                                           *
;*********************************************

WRITE:  CALL    !SHIBUF                 ;OUTPUT DATA

        RETI
```

```
;*********************************************
;*                                           *
;* STATE = READ: FETCH DATA SENT FROM        *
;*                 SLAVE-DEVICE              *
;*                                           *
;*********************************************

READ:   BF      SCK,$L_RE               ;CHECK CLOCK-STATUS; CLOCK = LOW ->

                                        ;DATA NOT VALID

        MOV     A,LOW (REDATA)          ;

        MOV1    CY,SBOMON               ;READ DATA-LINE

        ROLC    A,1                     ;SHIFT CARRYBIT INTO ACCU BIT 0

        MOV     LOW (REDATA),A          ;STORE READ DATA

        DEC     BITCNT                  ;

        BNZ     $L_RE                   ;

        MOVW    STATE,#SBSTOP           ;

L_RE:   RETI

;*********************************************
;*                                           *
;* STATE = SBSTOP: SENDS STOP-PULSE TO       *
;*                 SLAVE-DEVICE              *
;*                                           *
;*********************************************

SBSTOP: CLR1    SBO

        BF      SCK,$L_STP              ;CHECK CLOCK STATUS

        SET1    SBO                     ;SET DATA-LINE HIGH DURING CLOCK
                                        ;HIGH -> THAT IS STOP CONDITION

        MOV     DONE,#LOW(TRUE)         ;REQUESTED ACTION PERFORMED,SO SET
                                        ;COMPLETION CODE TO TRUE

        CALL    !SETIDL                 ;STATE TRANSITION TO IDLE

        MOVW    STATE,#IDLE

L_STP:  RETI

ENDS

END
```

7. Waveform Plot

The figures 7 and 8 show a data analyzer plot that comes out when the demo program is executed.

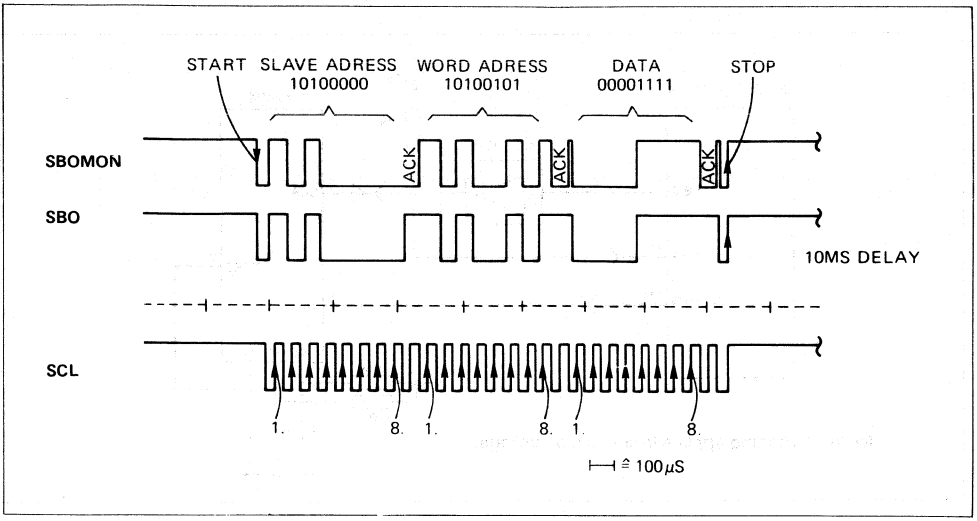


Figure 7. Write Cycle

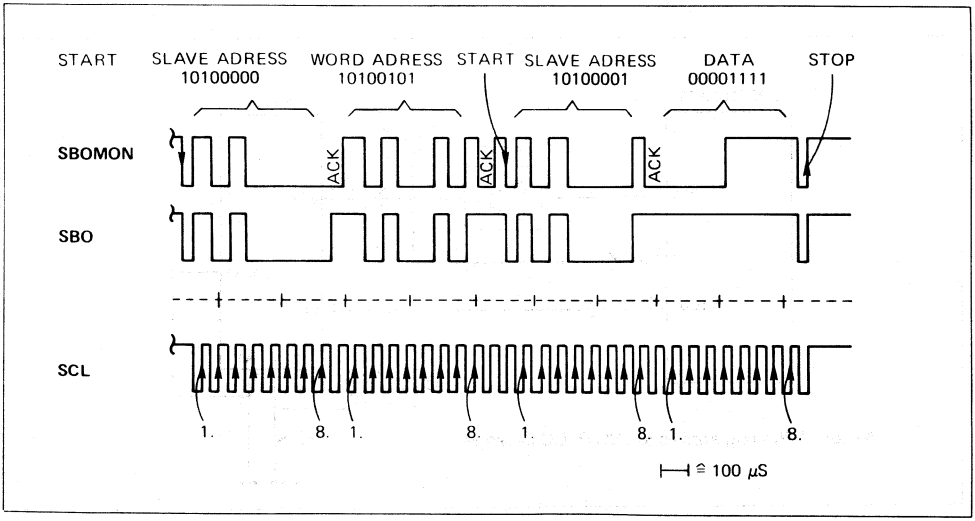


Figure 8. Read Cycle

8. SBI for μCOM87 Family and V25/V35 Microcomputers

A Serial Bus Interface can also be implemented in μCOM87 and V25 systems using the same software structure as previously described. The coding of course will be different. Figures 9 and 10 show possible hardware solutions for μCOM87 and V25.

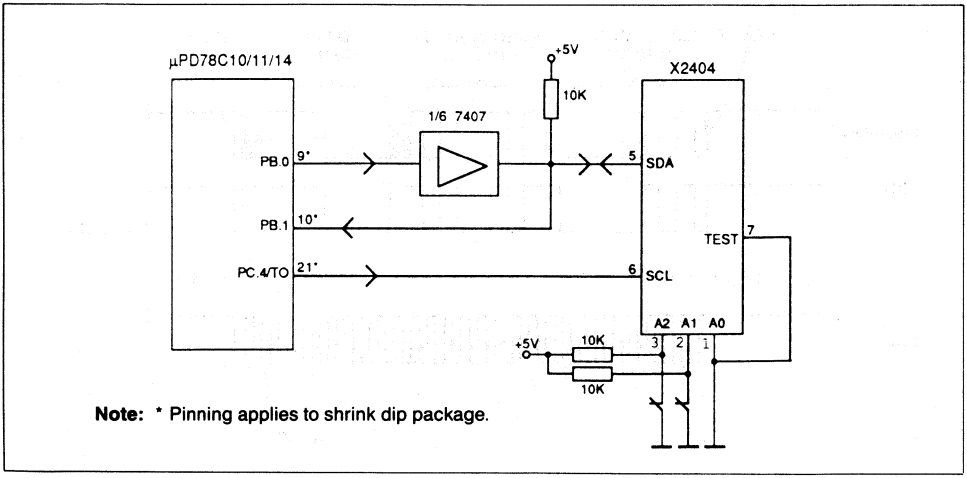


Figure 9. Serial Bus Interface for μPD78C10/C11(C14 (μCOM87) Microcomputers

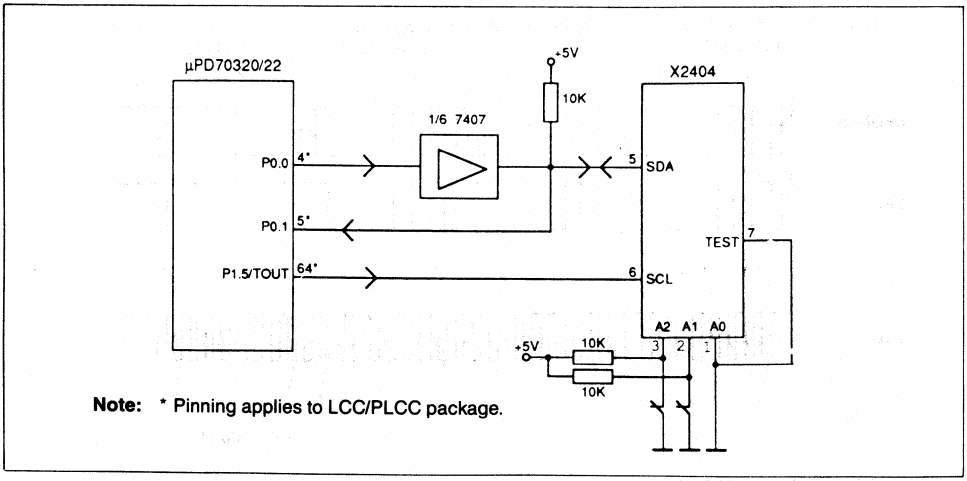


Figure 10. Serial Bus Interface for μPD70320/22 and μPD70330/32 (V25/V35) Microcomputers

FIP Display Interface to 8-Bit Single Chip Microcomputer μPD78310/312

Contents:

Authors: Thomas Weidemann, Heiner Tendyck
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD78310/12	Presentation
μPD78310/121	Product Description
IE 78310-R	User's Manual
FIP FM 20X1 AA-A	Specification

Related Products

μPD78310/312	8-Bit Microcomputers	CMOS

1. Hardware

1.1 EBIBM 78310 Evaluation Board

The μPD78310 is the Romless version of the μPD78312. The μPD78310 includes a 16-bit CPU, several Input/Output Ports, and an 8 priority level interrupt control. Apart from normal vectored interrupts, each maskable interrupt source has context switching capability and some of the vectored interrupts have a Macro Service capabillity.

The 78310 Evaluation board was designed to give as much flexibility for the design and evaluation of applications using a μPD78310/μPD78312. On-board there is 32K-byte EPROM containing monitor program and also a 32K-byte RAM for user programs. An RS 232 Interface is also provided on-board for communication to/from terminal or host system.

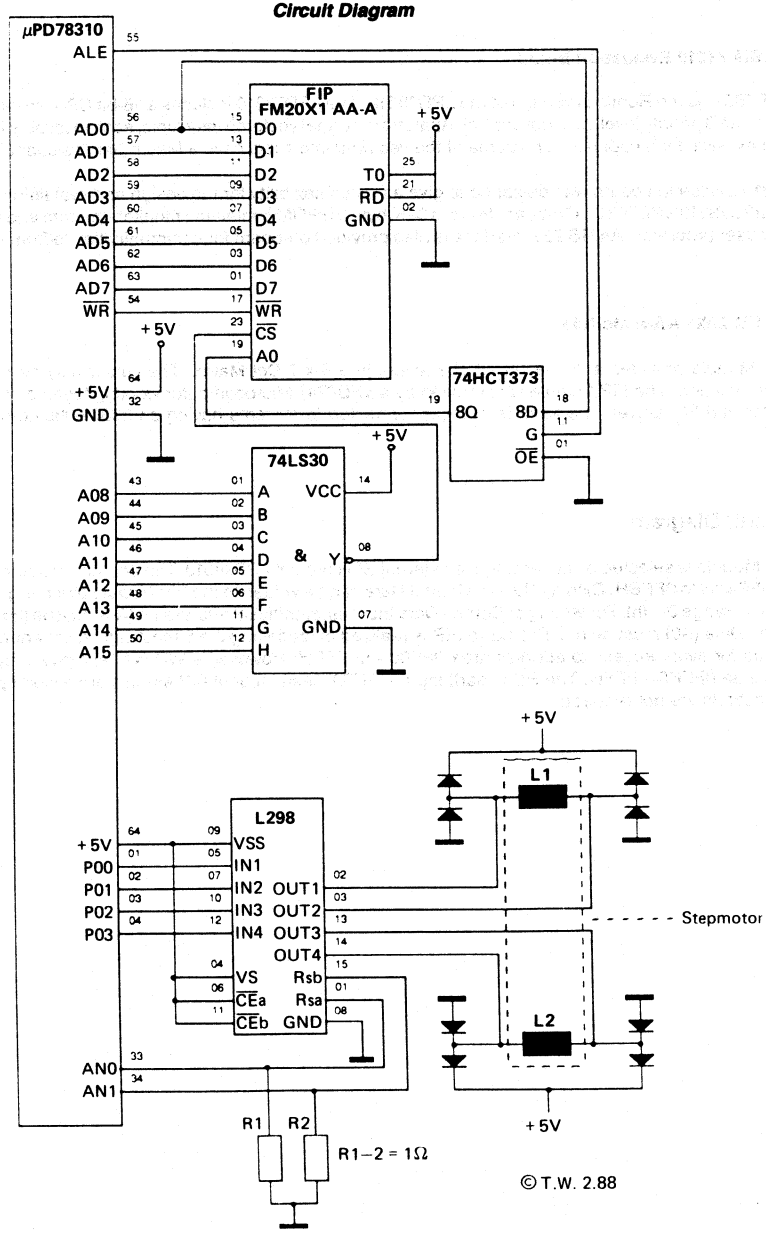
1.2 FIP FM 20X1 AA-A Module

The FIP Module provides a display of 20 characters in a 5 x 7 Dot Matrix. The cursor may be selected for any character position. The FIP Module is controlled by a μPD8741 Microcomputer working together with a Character Generator and FIP Driver. The Module may be connected to systems having an 8080/8085 compatible bus via D0—D7.

2. Circuit Diagram

The FIP Module is selected by accessing the external SFR area of μPD78310. The address area for external SFR's is from 0FFB0H to 0FFBH. Display Data or Control Data can be written to the FIP Module by writing data to address 0FFBx (x = range 0—fh). For writing of Control Data the least significant address bit (A0) must be (H), and for writing of Display Data (A0) must be (L). The CS to FIP is realized by decoding the 8 most significant address bits; thus a CS occurs for every access to address area 0FF00—0FFFFH. However, a WR (Write) signal only occurs within address area 0FFB0—FFFH. The RD (Read) input and TO (Test) input of FIP Module are pulled up to +5V, since these functions are not required.

Circuit Diagram

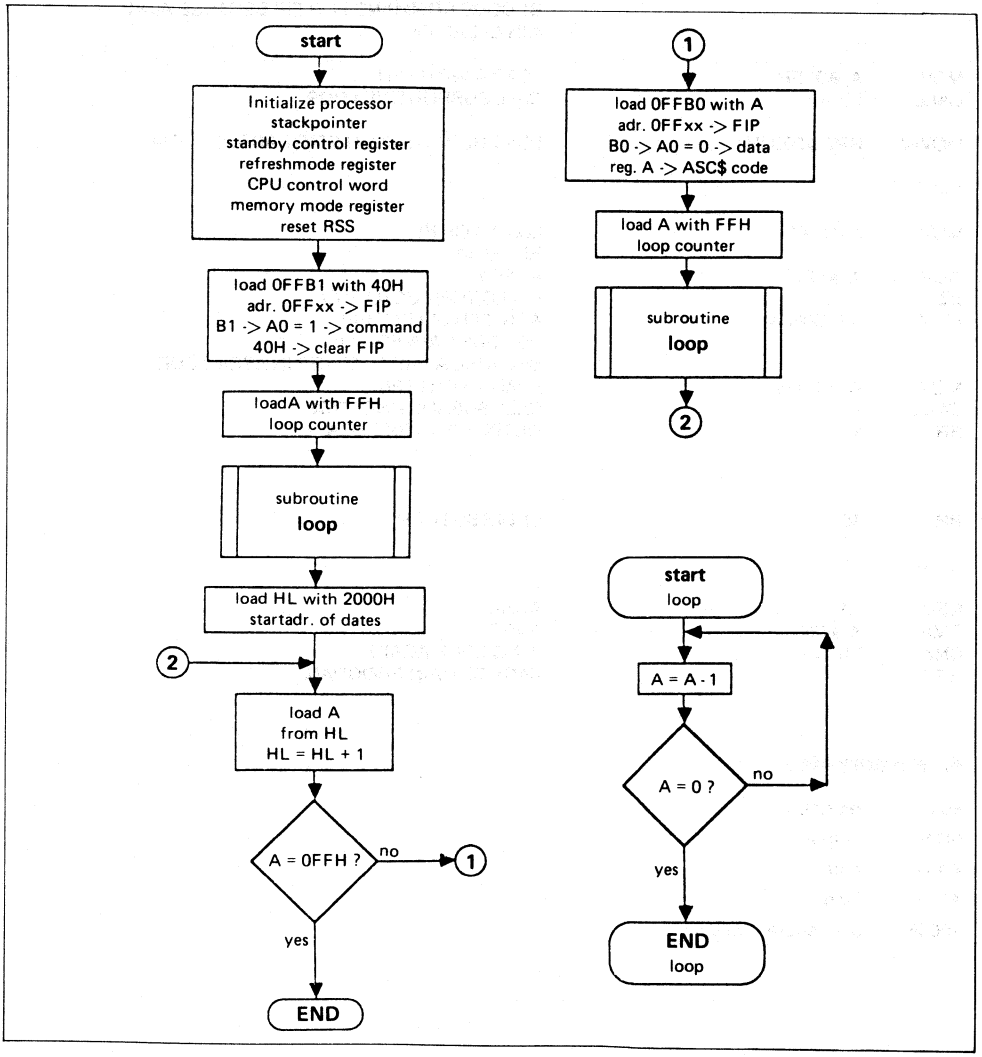


3. Software

The software routines for driving the FIP Module are very simple because only two addresses are used; 0FFB0h for writing Display Data to FIP, and 0FFB1H for writing Control Data to FIP. The FIP is controlled so that the cursor position shifts one position to the right after a display character has been written to the FIP Module.

N.B. The Software has a wait loop between writing Control Data and Display Data to FIP Module. This loop ensures that the FIP Module has enough time to process data.

3.1 Flow Chart



3.2 Programm

```
MOVW    SP,#0FE40 H     !
MOV     STBC,#8H        !
MOV     RFM,#0H         ! INITIALIZE PROCESSOR
MOV     CCW,#0H         !
MOV     MM,#7H          !
RSS     0               !

MOV     OFFB1,#40H      ADR. FF SELECT FIP
                        B1 CODE COMMAND FOR FIP BECAUSE A0=1;
                        40H CLEAR FIP

MOV     A,#0FFH         LOAD A WITH FFH
CALL    !LOOP           CALL SUBROUTINE LOOP

MOVW    RP7,#2000H      LOAD HL WITH 2000H START ADR. OF DATA

OUTPUT

MOV     A,[HL+]         LOAD A OF HL
                        HL=HL+1
CMP     A,#0FFH         A=FF ?
BZ      !END            IF YES THEN GOTO END
MOV     0FFB0H,A        ADR. FFH SELECT FIP
                        B0 CODE DATA FOR FIP,
                        BECAUSE A0=0; A LETTER IN ASCII CODE
MOV     A,#0FFH         LOAD A WITH FFH
CALL    !LOOP           CALL SUBROUTINE LOOP
BR      $OUTPUT         OUTPUT ROUTINE AGAIN

END

BR      $$              ENDLESS LOPP

LOOP

DEC     R1              A=A-1
CMP     A,#0H           A=0 ?
BNZ     !LOOP           IF NO LOOP AGAIN
RET                     BACK TO MAIN PROGRAM
```

4. Memory Map

80 H	program
9C H	output
AA H	end
AC H	loop
2000H	start address of data

5. Command List

A0 = 1

40H	clear display:	move cursur to left (1st Character Position)
41H	read cursor position	
42H	read data at cursor position	
43H	read data at cursor position	move cursor one character position to right

set writeposition

00H	01	05H	06	0AH	11	0FH	16
01H	02	06H	07	0BH	12	10H	17
02H	03	07H	08	0CH	13	11H	18
3H	04	08H	09	0DH	14	12H	19
04H	05	09H	10	OEH	15	13H	20

A0 = 0

Data ASCII-Code

20H	SP	30H	0	40H	@	50H	P
21H	!	31H	1	41H	A	51H	Q
22H	"	32H	2	42H	B	52H	R
23H	#	33H	3	43H	C	53H	S
24H	$	34H	4	44H	D	54H	T
25H	%	35H	5	45H	E	55H	U
26H	&	36H	6	46H	F	56H	V
27H	'	37H	7	47H	G	57H	W
28H	(	38H	8	48H	H	58H	X
29H	)	39H	9	49H	I	59H	Y
2AH	*	3AH	:	4AH	J	5AH	Z
2BH	+	3BH	;	4BH	K	5BH	[
2CH	,	3CH	<	4CH	L	5CH	¥
2DH	-	3DH	=	4DH	M	5DH	]
2EH	.	3EH	>	4EH	N	5EH	^
2FH	/	3FH	?	4FH	O	5FH	_

special commands

08H	backspace
09H	one step right
0AH	clear display
0DH	write position left
11H	automatic shift left for write position
12H	write position right
13H	scroll
14H	Cursor off
15H	Cursor on
16H	Cursor on and flashing

Stepper Motor Control using 8-Bit Single Chip Microcomputer μPD78310/312

Contents:

Authors: Thomas Weidemann, Heiner Tendyck
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD78310/12 CW/G Product Description
IE 78310-R Incircuit Emulator User's Manual
Specifications Driver L 298
FIP FM 20X1 AA-A Specification

Related Products

μPD78310/312 8-Bit Microcomputer	CMOS

1. Hardware

1.1 EBIBM 78310 Evaluation Board

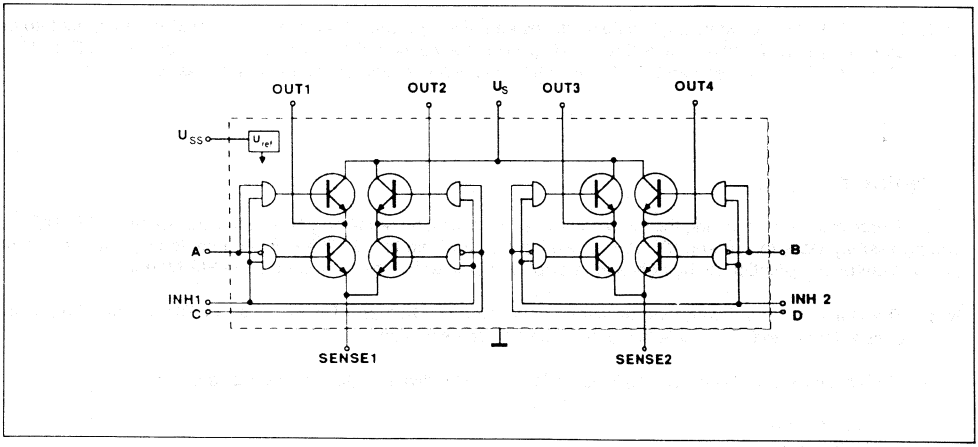
The μPD78310 is the Romless version of the μPD78312. The μPD78310 includes a 16-bit CPU, several Input/Output Ports, and an 8 priority level interrupt control. Apart from normal vectored interrupts, each maskable interrupt source has context switching capability and some of the vectored interrupts have a Macro Service capabillity. The 78310 Evaluation Board was designed to give flexibility for the design and evaluation of applications using a μPD78310/μPD78312. On-board there is 32K-byte EPROM containing Monitor Program and also a 32K-byte RAM for User Programs. An RS 232 interface is also provided on-board for communication to/from terminal or host system.

1.2 Stepper Motor

The Stepper Motor used in this application is a bipolar type with 48 steps/revolution. This corresponds to a step angle of 7,5 degrees. The Stepper Motor can be operated at frequencies between 600 Hz and 6 KHz. The rotor of the Stepper Motor is a cylindrical ferrite magnet with radial magnetization. The polarity of the stator poles, which depend upon the direction of the current in the phase winding, determines the direction of rotation.

1.3 Driver L 298

This device consists of dual bridge transistor circuits. In each bridge only two of the transistors will be operating simultaneously in order to provide opposite polarity outputs.



e.g. With INH1 (H), A (H) and C (L), "OUT1" will be "+ve", and "OUT2" will be "-ve". This will cause current to flow in a particular direction through winding L1 of Stepper Motor.

The other input signals; INH2, B and D control the other bridge circuit. The common emitters of each bridge circuit are SENSE1 and SENSE2. They are connected to 0V via a low value resistor and can be monitored in order to prevent over-current in Stepper Motor winding. The L 298 enables control of a two phase Stepper Motor with power consumption of 200W. This approximates to 46V, at 2,5A per bridge.

1.4 FIP FM 20X1 AA-A Module

The FIP Module provides a display of 20 characters in a 5 x 7 Dot Matrix. The cursor may be selected for any character position. The FIP Module is controlled by a μPD8741 Microcomputer working together with a character generator and FIP driver. The Module may be connected to systems via D0—D7, having an 8080/8085 compatible bus. For details refer to NEC Appl.-Note No. μCOM 39.

2. Circuit Diagram

The Stepper Motor phase windings L1 and L2 are driven by the 4 outputs of the L 298. For full power operation in either direction, the current in L1 and L2 flow in same direction:

e.g. OUT1 (H), OUT2 (L) and OUT4 (H), OUT3 (L)

The 4 inputs of the L 298 are driven by the lowest 4-bits (l.s. Nibble) of the μPD78310 real time output port. The SENSE1 and SENSE2 outputs of L 298 are monitored via AN0 and AN1 channels of A/D converter on μPD78310. The inhibit signals INH1 and INH2 are active (L). In this application the bridge circuits of L 298 are enabled continously since both inhibit lines are pulled up to (H). The FIP Module is selected by accessing of the external SFR area of the μPD78310. The address area for external SFR's is from 0FFB0H to 0FFBFH. Display Data or Control Data can be written to the FIP Module by writing data to address 0FFBx (x = range 0—fh). For writing of Control Data, the least significant address bit A0 must be (H), and for writing of Display Data A0, must be (L). The CS to FIP is realized by decoding the 8 most significant address bits; thus a CS occurs for every access to address area 0FF00—0FFFFH: However, a WR (Write) signal only occurs within address area 0FFB0—0FFBFH. The RD (Read) input and T0 (Test) input of FIP Module are pulled up to +5V, since these functions are not required.

N.B. If a μPD78312 is used in single-chip mode instead of a μPD78310, then the CS of FIP Module can be tied to 0v since the μPD78312 only activates WR (L) when accessing the external SFR address area 0FFB0H—0FFBFH. Thus the decoding via 74ls30 of 8 higher. Address bits becomes unnecessary.

3. Software

The software routines for driving the FIP Module are very simple because only two addresses are used; 0FFB0h for writing Display Data to FIP and 0FFB1H for writing Control Data to FIP. The FIP is controlled so that the cursor position shifts one position to the right after a display character has been written to the FIP Module.

N.B. The Software has a wait loop between writing Control Data and Display Data to FIP Module. This loop ensures that the FIP Module has enough time to process data.

The software routines for driving Stepper Motor can be divided into 4 operating sections:

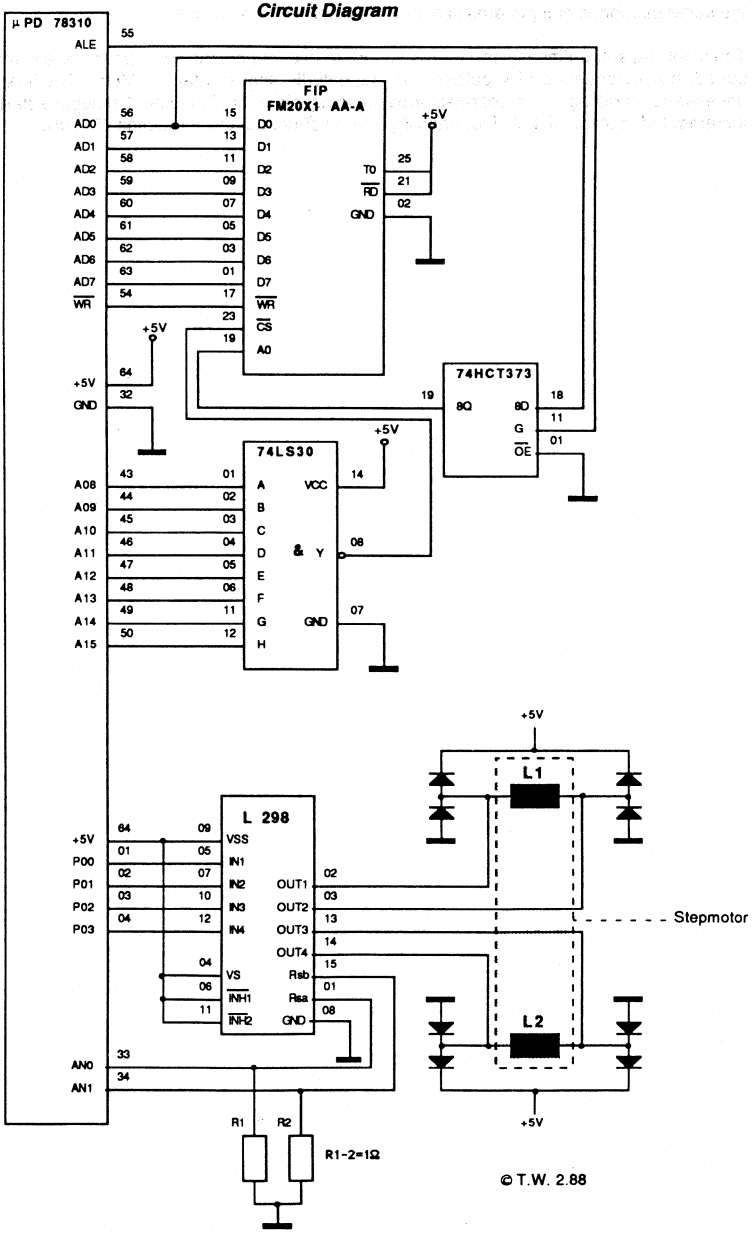
a. Acceleration
b. Half Speed
c. Full Speed
d. Deceleration (Braking)

The acceleration subroutine uses a fixed sequence of 8 patterns sent to Stepper Motor with a varying timebase provided by the timer 0. Timer 0 is initially set at 0F000H and at the completion of 8 patterns 400 H is subtracted from timer 0 timebase. The same 8 patterns are sent again and at completion another 400 H is subtracted from timer 0 timebase. This sequence is continued until timer 0 timebase has reached 1C00H. The acceleration cycle is now complete.

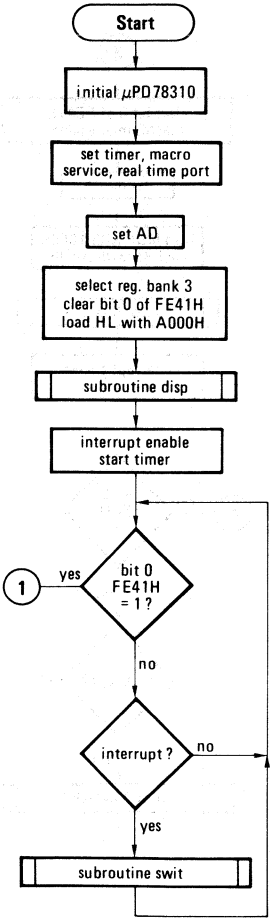
The next subroutine activated is the half speed subroutine. Here the timer 0 timebase is kept constant and the same sequence of 8 patterns is sent to Stepper Motor as above.

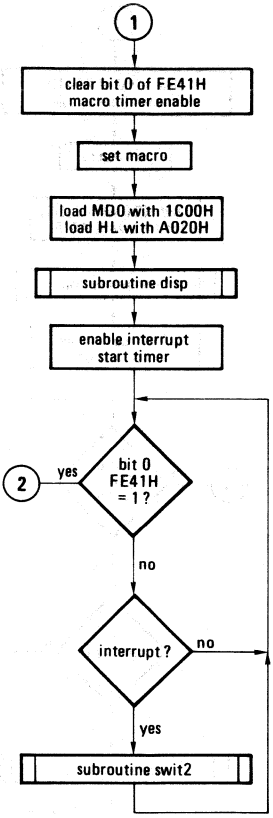
The following subroutine is used for full speed. In this routine, the timer 0 timebase is the same as for half speed, but a sequence of 4 patterns are repeatedly sent to Stepper Motor. The final subroutine is for decelaration (braking). This operates opposite to acceleration. The timer 0 timebase starts at 1C00H and is incremented in steps of 400 H for each 8 pattern sequence until it reaches 0F000H.

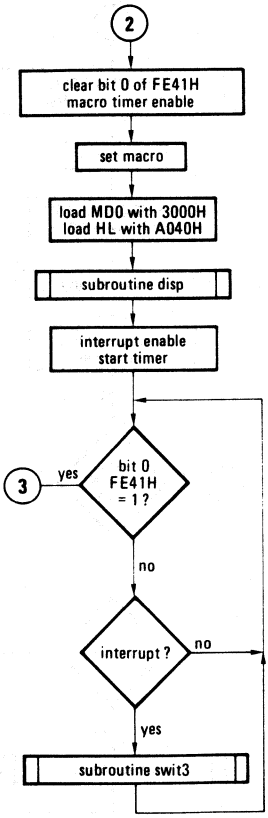
Circuit Diagram

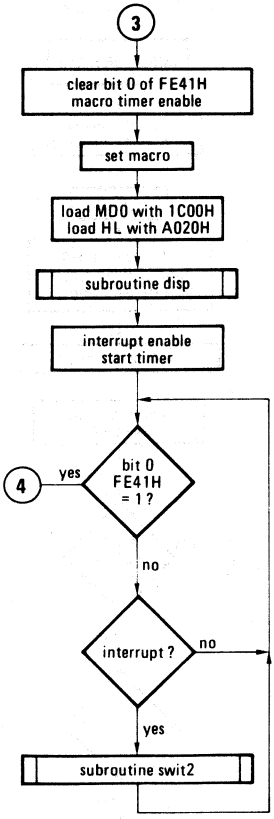


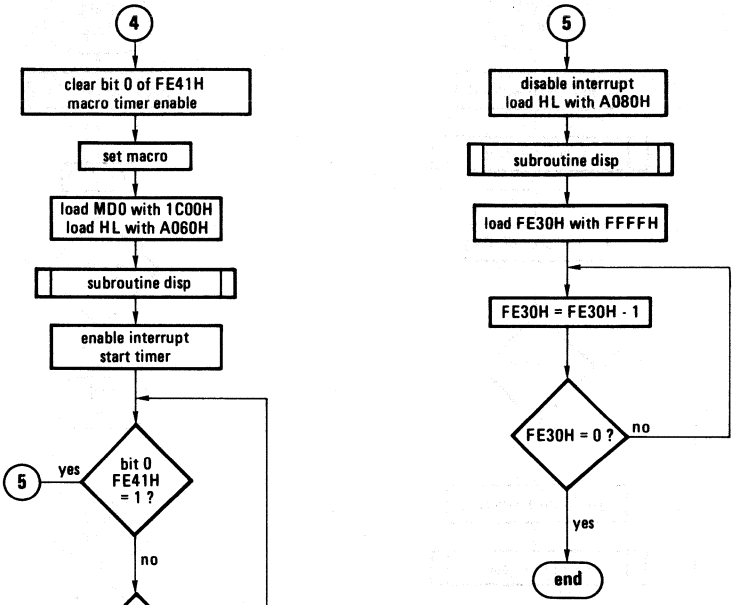
3.1 Flow Chart

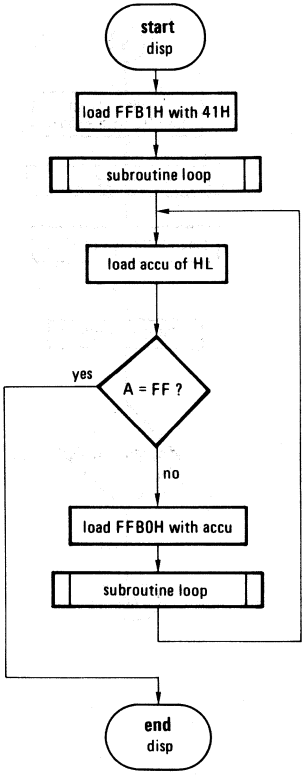


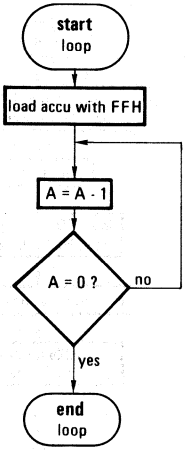


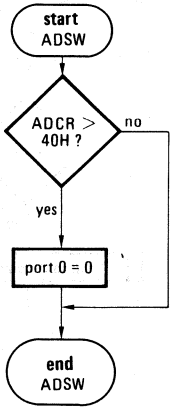

2
clear bit 0 of FE41H
macro timer enable
set macro
load MD0 with 3000H
load HL with A040H
subroutine disp
interrupt enable
start timer
3
yes
bit 0
FE41H
= 1 ?
no
interrupt ?
no
yes
subroutine swit3

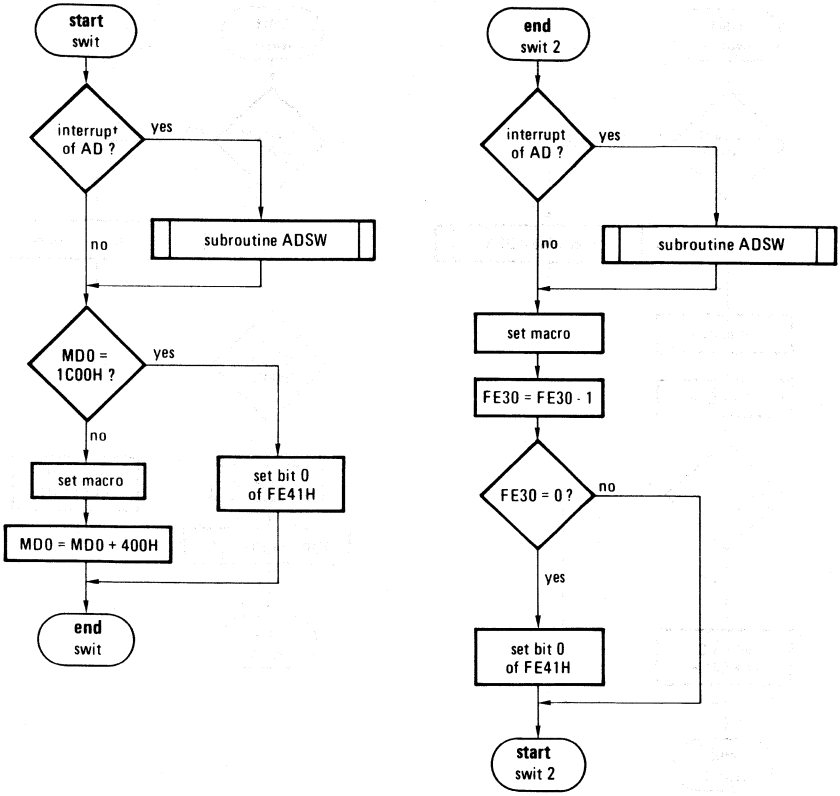

3
clear bit 0 of FE41H
macro timer enable
set macro
load MD0 with 1C00H
load HL with A020H
subroutine disp
interrupt enable
start timer
4
yes
bit 0
FE41H
= 1 ?
no
interrupt ?
no
yes
subroutine swit2

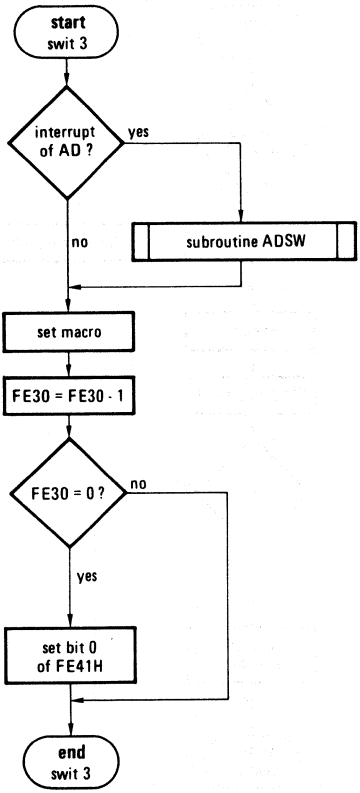

4
clear bit 0 of FE41H
macro timer enable
set macro
load MD0 with 1C00H
load HL with A060H
subroutine disp
enable interrupt
start timer
5
yes
bit 0
FE41H
= 1 ?
no
interrupt ?
no
yes
subroutine swit 4
5
disable interrupt
load HL with A080H
subroutine disp
load FE30H with FFFFH
FE30H = FE30H - 1
FE30H = 0 ?
no
yes
end

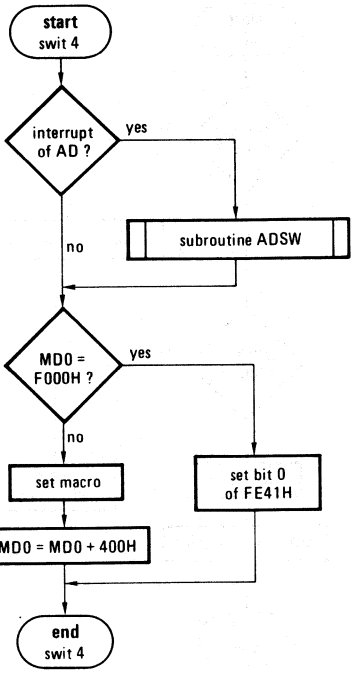

start
disp
load FFB1H with 41H
subroutine loop
load accu of HL
yes
A = FF ?
no
load FFB0H with accu
subroutine loop
end
disp


start
loop
load accu with FFH
A = A - 1
A = 0 ?
no
yes
end
loop


start
ADSW
ADCR >
40H ?
no
yes
port 0 = 0
end
ADSW


start
swit
interrupt
of AD ?
yes
no
subroutine ADSW
MD0 =
1C00H ?
yes
no
set macro
MD0 = MD0 + 400H
set bit 0
of FE41H
end
swit
end
swit 2
interrupt
of AD ?
yes
no
subroutine ADSW
set macro
FE30 = FE30 - 1
FE30 = 0 ?
no
yes
set bit 0
of FE41H
start
swit 2


start
swit 3
interrupt
of AD ?
yes
no
subroutine ADSW
set macro
FE30 = FE30 - 1
FE30 = 0 ?
no
yes
set bit 0
of FE41H
end
swit 3


start
swit 4
interrupt
of AD ?
yes
no
subroutine ADSW
MD0 =
F000H ?
yes
no
set macro
set bit 0
of FE41H
MD0 = MD0 + 400H
end
swit 4

3.2 Programm

1. programstep: speed up

```
        MOVW    SP,#0FE20H          !
        MOV     STBC,#8H            !
        MOV     RFM,#0H             ! initialize the
        MOV     CCW,#0H             ! μPD78310
        MOV     MM,#7H              !
        RSS     0                   !

        MOV     TMC0,#0H            ! set timer 0
        MOV     TMC1,#0H            ! set timer 1
        MOV     TMIC0,#00110010B    ! set timer 0 control reg.
        MOV     TMMS0,#04H          ! macro service priority 4
        MOV     RTPC,#3H            ! set real time output port
        MOV     PM0,#0H             ! set port 0 as output

        MOV     ADIC,#10H           ! set AD control reg.
        MOV     ADM,#2H             ! set AD
        MOVW    0FEF4H,#ADSW        ! context switching as ADSW

        MOVW    0FEE0H,#9000H       ! set macro service pointer
                                    ! at 9000H
                                    ! (pattern-data)
        MOV     0FEE2H,#08H         ! macro service counter at 8
        MOV     0FEE3H,#3AH         ! SFR is real time port
        MOVW    0FED4H,#SWIT        ! context switching at Swit

        SEL     RB3                 ! select reg. bank 3
        CLR1    0FE41H.0            ! set bit0 of FE41 at 0
        MOVW    RP7,#A000H          ! load HL with A000H
                                    ! (Data for FIP)
        CALL    ! DISP              ! call subroutine Disp
        EI                          ! enable interrupt
        SET1    TMC0.7              ! start timer
LB2:    BTCLR   0FE41H.0,$LB1       ! go LB1, if bit0 of FE41H is set
        BR      $LB2                ! go LB2
```

2. programstep: half step

```
LB1:    CLR1    0FE41H.0            ! clear bit 0 of 0FE41H
        SET1    TMIC0.5             ! enable macro service

        MOVW    0FEEE0H,#9000H      ! adr. pattern data
        MOV     0FEE2H,#08H         ! 8 data
        MOVW    0FEED4H,#SWIT2      ! context switching at SWIT2

        MOVW    MD0,#1C00H          ! set timer at 1C00H

        MOV     0FE30H,#20H         ! number of half steps

        MOVW    RP7,#A020H          ! FIP data at A020H
        CALL    !DISP               ! subroutine DISP

LB7:    BTCLR   0FE41H.0$LB8        ! go LB8, if bit0 of FE41H are set
        BR      $LB7                ! go LB7
```

3. programstep: full step

```
LB8:    CLR1    0FE41H.0          ! clear bit 0 of 0FE41H
        SET1    TMIC0.5           ! enable macro service

        MOVW    0FEE0H, #9010H    ! pattern-data at 9010H
        MOV     0FEE2H, #04H      ! 4 data
        MOVW    0FED4H, #SWIT3    ! context switching at SWIT3

        MOVW    MD0, #3000H       ! set timer at 3000H

        MOV     0FE30H, #0FFH     ! number of full steps

        MOVW    RP7, #3040H       ! FIP dta at 3040H
        CALL    !DISP             ! call subroutine DISP

LB10:   BTCLR   0FE41H.0,$LB11    ! go LB11, if bit0 of FE41H are set
        BR      $LB10             ! go LB10
```

4. program step: half step

```
LB11:   CLR1    0FE41H.0          ! clear bit 0 of 0FE41H
        SET1    TMIC0.5           ! enable macro service

        MOVW    0FEE0H, #9000H    ! pattern-data
        MOV     0FEE2H, #08H      ! 8 data
        MOVW    0FED4H, #SWIT2    ! context switching at SWIT2

        MOVW    MD0, #1C00H       ! set timer at 1C00H

        MOV     0FE30H, #20H      ! number ofhalf steps

        MOVW    RP7, #A020H       ! FIP data at A020H
        CALL    !DISP             ! call subroutine DISP

LB13:   BTCLR   0FE41H.0,$LB14    ! go LB14, if bit0 of FE41H are set
        BR      $LB13             ! go LB13
```

5. program step: brake

```
LB14:   CLR1    0FE41H.0          ! clear bit 0 of 0FE41H
        SET1    TMIC0.5           ! enablemacro service

        MOVW    0FEE0H, #9000H    ! pattern-data
        MOV     0FEE2H, #08H      ! 8 data
        MOVW    0FED4H, #SWIT4    ! context switching at SWIT4

        MOVW    MD0, #1C00H       ! set timer at 1C00H

        MOV     0FE30H, #20H      ! number of half steps

        MOVW    RP7, #A060H       ! FIP data at A060H
        CALL    ! DISP            ! call subroutine DISP

LB15:   BTCLR   0FE41H.0,$LB16    ! go LB16, if bit0 of FE41H are set
        RB      $LB15             ! go LB15
```

6. program step: last settings

```
LB16:   DI                              ! disable interrupt

        MOVW    RP7, #A080H             ! FIP-data at A080H
        CALL    ! DISP                  ! call subroutine DISP

        MOVW    0FE30H, #0FFFFH         ! load 0FE30H with 0FFFFH
LB18:   DECW    0FE30H                  ! decrement 0FE30H
        CMPW    0FE30H, #00H            ! Adr. 0FE30H=0 ?
        BNZ     $LB18                   ! go LB18, if not zero

        BR      $$                      ! endless loop
```

Data

pattern:

```
9000H   05 04 06 02 0A 08 09 01         (Half Step)
9010H   09 05 06 0A                     (Full Step)
```

FIP-dta:

```
A000H   53 50 45 45 44 20 55 50 14 FF
        S  P  E  E  D  _  U  P

A020H   48 41 4C 46 20 53 54 45 50 14 FF
        H  A  L  F  _  S  T  E  P

A040H   46 55 4C 4C 20 53 54 45 50 14 FF
        F  U  L  L  _  S  T  E  P

A060H   42 52 41 4B 45 14 FF
        B  R  A  K  E

A080H   2A 2A 2A 20 45 4E 44 20 2A 2A 2A 14 FF
        *  *  *  _  E  N  D  _  *  *  *
```

remark: 14 is cursor left; FF is data end

```
        MOV     0FFB1H, #40H            ! clear FIP
        CALL    !LOOP                   ! call subroutine LOOP

LB4:    MOV     A,[HL+]                 ! load A of HL; HL=HL+1
        CMP     A #0FFH                 ! A=0FFH ?
        BZ      $LB5                    ! go LB5, if zero

        MOV     0FFB0H,A                ! A as Data to FIP
        CALL    ! LOOP                  ! call subroutine LOOP

        BR      $LB4                    ! go LB4

LB5:    RET                             ! return from subroutine
```

```
LOOP:

        MOV     A, #0FFH            ! load A with 0FFH
LB4:    DEC     A                   ! A=A-1
        CMP     A, #00H             ! A=0 ?
        BNZ     $LB4                ! go LB4, if not zero

        RET                         ! return from subroutine

SWIT:

        DI                          ! disable interrupt
        MPW     MD0, #1C00H         ! MD=1C00H ?
        BZ      $LB3                ! go LB3, if zero
        MOVW    0FEE0H, #9000H      ! pattern-data
        MOV     0FEE2H, #08H        ! 0 data
        SUBW    MD0, #0400H         ! MD0=MD0-400H
        SET1    TMIC0.5             ! set macro service
        EI                          ! enable interrupt
        RETCS   ! SWIT              ! return from context switching

LB3:    SET1    0FE41H.0            ! set bit 0 of 0FE41H
        SET1    TMIC0.5             ! set macro service
        EI                          ! enable interrupt
        RETCS   ! SWIT              ! return from context switching

SWIT2:

        DI                          ! disable interrupt
        SET1    TMIC0.5             ! set macro service
        MOVW    0FEEE0H, #9000H     ! pattern data
        MOV     0FEE2H, #08H        ! 8 data

        DEC     0FE30H              ! 0FE30H=0FE30H-1
        CMP     0FE30H, #00H        ! 0FE30H=0 ?
        BNZ     $LB9                ! go LB9, if not zero

        SET1    0FE41H.0            ! set bit 0 of 0FE41H

LB9:    EI                          ! enable interrupt
        RETCS   ! SWIT2             ! return from Contest switching

SWIT3:

        DI                          ! disable interrupt
        SET1    TMIC0.5             ! set macro service
        MOVW    0FEE0H, #9010H      ! pattern-data
        MOV     0FEE2H, #04H        ! 8 data

        DEC     0FE30H              ! 0FE30H=0FE30H-1
        CMP     0FE30H, #00H        ! 0FE30H=0 ?
        BNZ     $LB12               ! go LB12, if not zero

        SET1    0FE41H.0            ! set bit 0 of oFE41H
```

```
LB12:   EI                          ! enable interrupt
        RETCS   ! SWIT3             ! return from context switching

SWIT4:

        DI                          ! disable interrupt
        CMPW    MD0, #F000H         ! MD=0F000H ?
        BZ      $LB17               ! go LB17, if zero
        MOVW    0FEE0H, #9000H      ! pattern-data
        MOV     0FEE2H, #08H        ! 8 data
        ADDW    MD0, #0400H         ! MD0=MD0+400H
        SET1    TMIC0.5             ! set macro serive
        EI                          ! enable interrupt
        RETCS   ! SWIT4             ! return from contect switching

LB17:   SET1    0FE41H.0            ! set bit 0 of 0FE41H
        SET1    TMIC0.5             ! set macro service
        EI                          ! enable interrupt
        RETCS   ! SWIT4             ! return from context switching

ADSW

        CMP     ADCR, #40H          ! changed worth = 40H ?
        BNL     $LB19               ! go LB19, if not lower
        MOV     RTPC, #8H           ! port 0 = normal port
        MOV     P0, #0H             ! 0 at port 0
        MOV     RTPC, #8H           ! port 0 = real time port

LB19:   RETCS   ! ADSW              ! return from context switching
```

4. Memory Map

8000H	program
9000H	data pattern half step
9010H	data pattern full step
A000H	data FIP

5. Command List

A0 = 1

40H	clear display:	Cursor left
41H	read position	
42H	read data of position	
43H	read data of position	Cursor one to right

set writeposition

00H	01	05H	06	0AH	11	0FH	16
01H	02	06H	07	0BH	12	10H	17
02H	03	07H	08	0CH	13	11H	18
3H	04	08H	09	0DH	14	12H	19
04H	05	09H	10	OEH	15	13H	20

A0 = 0

Data ASCII-Code

20H	SP	30H	0	40H	@	50H	P
21H	!	31H	1	41H	A	51H	Q
22H	"	32H	2	42H	B	52H	R
23H	#	33H	3	43H	C	53H	S
24H	$	34H	4	44H	D	54H	T
25H	%	35H	5	45H	E	55H	U
26H	&	36H	6	46H	F	56H	V
27H	'	37H	7	47H	G	57H	W
28H	(	38H	8	48H	H	58H	X
29H	)	39H	9	49H	I	59H	Y
2AH	*	3AH	:	4AH	J	5AH	Z
2BH	+	3BH	;	4BH	K	5BH	[
2CH	,	3CH	<	4CH	L	5CH	¥
2DH	-	3DH	=	4DH	M	5DH	]
2EH	.	3EH	>	4EH	N	5EH	^
2FH	/	3FH	?	4FH	O	5FH	_

special commands

08H	backspace
09H	one step right
0AH	clear display
0DH	write position left
11H	automatic shift left for write position
12H	write position right
13H	scroll
14H	Cursor off
15H	Cursor on
16H	Cursor on and flashing

Dynamic RAM Interface for 16-Bit Single Chip Microcomputer μPD70320 (V25)

Contents:

Author: Shyam Gupta
Application μCOM Department
NEC Electronics (Europe) GmbH

Related Documentation

Microprocessors and Peripherals	Data Book
User's Manual	
μPD70320/70322	
μPD70331/70332	
μPD79011	

Related Products

μPD70320/70320	16-bit Microcomputer	CMOS
μPD70330/70330	16-bit Microcomputer	CMOS
μPD79011	16-bit Microcomputer with Real Time Operating System	CMOS

1. Introduction

The μPD70322 (V25) is a 16-bit single chip microcomputer in CMOS technology from NEC. The μPD70320 is the romless version of the V25. The V25 supports a number of on chip peripheral functional blocks like two serial interface units, two DMA channels, interrupt controller threshold port comparator and I/O ports etc. It has 1M byte memory address range and 64K bytes I/O address space. The device supports programmable wait states for memory and I/O space. The use of dynamic RAM is supported by a built in dynamic RAM refresh control unit. The V25 refresh control unit can be programmed to generate refresh signals at various intervals (to suit different DRAM configurations) and it generates a 9 bit refresh address. Another feature of the refresh controller is that it also operates during the V25 standby mode. The same controller is also useful for supporting the pseudo static RAM. For details on this please refer to the μPD70320/70322 user's manual. The aim of this application note is to discuss the dynamic RAM interface issues to V25 microcomputer.

2. Dynamic RAM Interface Considerations

As already mentioned, the V25 microcomputer generates not only a refresh signal with programmable time interval but also a 9 bit refresh address (A0—A8). A typical dynamic RAM requires apart from the refresh signal and the refresh address also the dynamic RAM control signals like row and column address strobe (RAS, CAS) plus the row/column address multiplex signal (MUX). These signals can be generated through different methods viz:

a) usig a delay line,

b) using PALs,

c) using discrete logic.

Sofar as the control signal timing accuracy is concerned, the method using a delay line to generate dynamic RAM control signals is the most accurate one, however it is also the costliest method.

The method using the PALs requires the least possible number of devices to generate all the control signals needed, yet this method is still not very cost effective.

Taking the above mentioned facts into considerations this application note discusses realisation of the dynamic RAM control signal logic using off the shelf devices. This method is easy to use because as V25 generates the refresh signal on its own, there is no need to design an arbitration controller between normal memory access and the refresh cycle. Further, the state of the art dynamic RAM devices support a socalled CAS- before RAS- refresh mode in which the refresh address is generated by the dynamic RAM itself on chip. This feature frees the user from the limitation of 9-bit refresh address generated by the V25 and at the same time using the power saving refresh mode.

In the following, an interface for popular 256K x n bit and 1M x 1 bit type dynamic RAMs to V25 is described. The dynamic RAM interface is divided into three main functional sub blocks:

— Dynamic RAM chip select

— Dynamic RAM row/column address multiplexer

— Dynamic RAM RAS-/CAS- control signal generation

— Dynamic RAM CAS- before RAS- refresh cycle

These functional sub blocks are descussed in the dynamic RAM interface examples to V25 and shown for a 512K byte and 1M byte memory configuration each.

Note: Throughout the text of this application note, the active low signals are shown with a "—" sign following the signal name. In figures, the active low signals are shown with a bar on top of the signal name.

3. V25 — μPD424256-12 Interface

As is already known, the V25 supports a total of 1M byte memory address space. Let us consider a V25 memory system configuration with 512K bytes dynamic RAMs, and the rest of the 512K bytes for other types of memory devices (ROM, static RAM etc.). For the sake of simplicity let the dynamic RAM be located starting from the address 0. This would mean that the dynamic RAM can be selected by just decoding the address line A19.

A19 = 0; Dynamic RAM access

A19 = 1; Other memory access

A typical 512K bytes memory configuration using 256K x 4 bit devices would require four μPD424256-4 dynamic RAMs. Two each of these devices would than be accessed in parallel to realise a byte access as shown in figure 1.

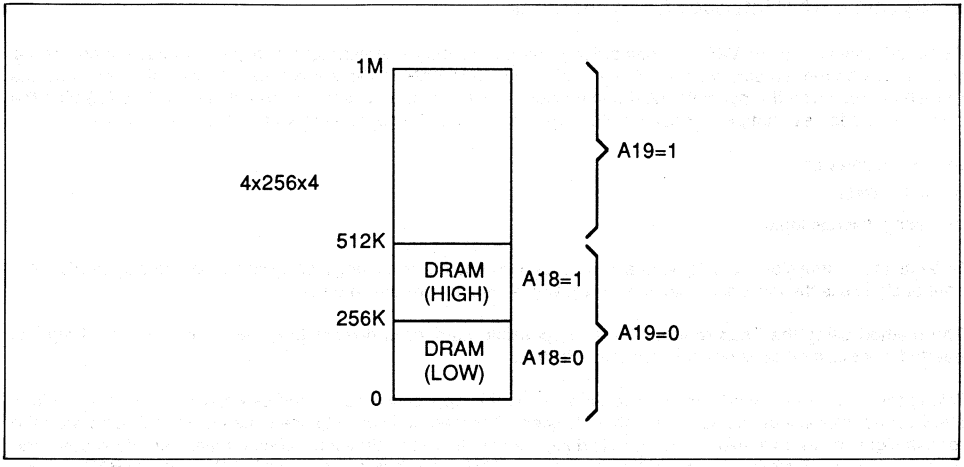


Figure 1. 512K Byte Dynamic RAM Address Map

The individual memory bits in a dynamic RAM are accessed by addressing row and the column of an array where it is located. For this, the row and the column addresses must be placed on the address lines and are than latched in the device through socalled row — and column address strobes (RAS- and CAS- signals).

In the example shown here the multiplexed row and column addresses for dynamic RAM are controlled using 74HCT257 multiplexer devices. Figure 2 shows the 9 bit row/column address multiplexing for μPD424256 (256K x 4 bits) dynamic RAMs.

The two RAM banks are controlled by RAS_L and RAS_H signals respectively. The CAS- signals is common for both memory banks. The μPD424256-X data I/O1—I/O4 pins are connected directly to the V25 D0—D7 lines and are controlled by R/W- (V25) signal connected to OE- and WE- pin of the memory devices. The multiplex signal MUX- switches the Row/Column signals.

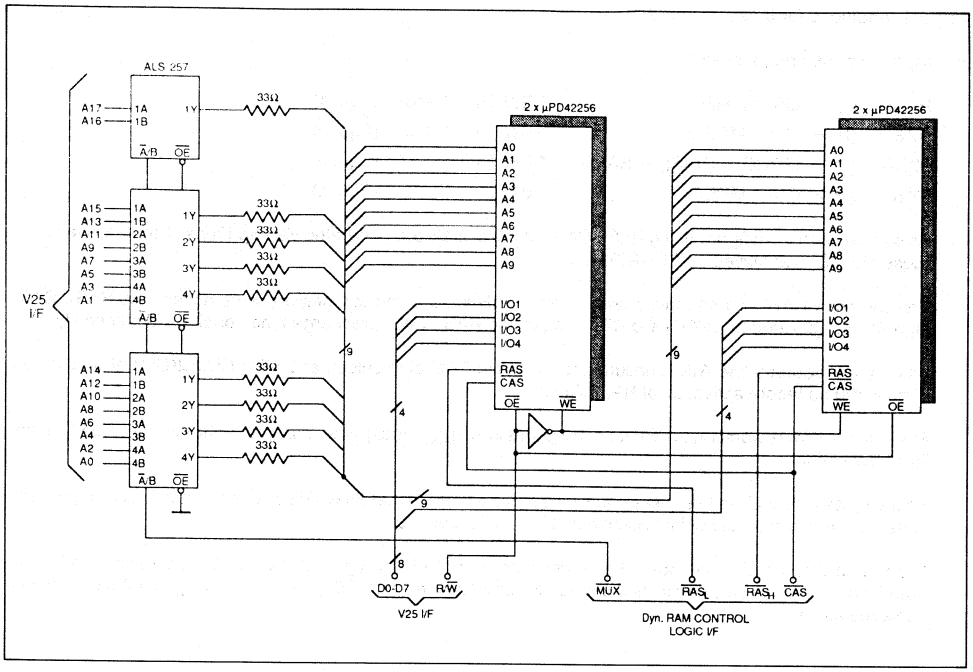


Figure 2. Row/Column Addresses Multiplexer for μPD424256-X

As already mentioned above, the RAS- and CAS- control signals for the dynamic RAM control are generated with the help of off the shelf devices driven by the V25 memory access and refresh control signals MREQ- and RERFQ-.

A standard bus access cycle for V25 is finished in two clock cycles. If one considers the min/max values of the most important signals needed for a memory access (MREQ-, MSTB-, REFRQ- etc.) from the clock (CLOCKOUT signal) edge, these are specified to be as 15/90 ns (min/max) respectively. Another important V25 timing parameter "Input data hold time (tHDK)" is specified as 60ns (min).

With the help of an additional clock (external) source which is at least twice the "CLOCKOUT" signal frequency plus some synchronisation logic for control signals, a zero wait state dynamic RAM interface may be realised for 5 MHz V25 systems.

With the declared aim to keep the total device and cost overhead of dynamic RAM interface to a minimum, one comes to the conclusion that the best compromise between access speed and cost would be reached if dynamic RAM interface is designed for V25 bus access with one wait state. The timing diagrams shown are valid for μPD424256-12 dynamic RAMs.

The 512K bytes are organised as two consecutive 256K byte RAM blocks. The chipselect (CS-) for the dynamic RAM is generated by decoding the V25 A19 address. The two dynamic RAM blocks are referred to here as RAM low (RAM_L) and RAM High (RAM_H). The selection of RAM_L and RAM_H memory access is done using the V25 A18 address.

In order to fullfill the worst case timing specification and yet be able to keep the device overhead to a minimum, the RAS-, MUX- and CAS- signals are triggered from clock (CLOCKOUT) edge.

Logical conditions for these signals are:

a) Memory read/write access

RAS_L- (↓) = CS- * MREQ- * BW (↑) * A18- (Equ. 1)

RAS_H (↓) = CS- * MREQ- * BW (↑) * A18 (Equ. 2)

MUX- (↓) = MREQ- * (RAS_L + RAS_H) * BW (↓) (Equ. 3)

CAS- (↓) = CS- * MREQ- * B2 (↑) (Equ. 4)

As can be seen from figure 5, the RAS-, CAS-, MUX- signals are generated via Flip-Flops. The deactivation of these signals is achieved by CS- (A19)line.

The dynamic RAM control logic shown utilises early write mode for memory write access. As RAS-, CAS- signals are activated by BW (↑) and B2 (↑) edges, the RAS-, CAS- precharge time condition is met easily.

As only the address line A19 is decoded for dynamic RAM chip select, at 5 MHz CLOCKOUT, the user has almost 100 ns before activation of RAS- signal.

Even under 8 MHz worst case condition, t_{DKA} (max) = t_{DKC} (max) = 90 ns the user has 35 ns setup time for RAS- activation.

As the system described here uses early write cycle, and the dynamic RAM cycle is maintained till chip select is deactivated, a memory write access has no critical parameter.

In a read cycle, the μPD424256-12 data access time form CAS- (t_{CAC} = 30 ns) condition is easily fulfilled for both 5 and 8 MHz V25 systems. As the read access is finished with CS- (↑) also, the T_{HKD} = 60 ns for V25 is achieved as well.

b) Refresh Cycle (CAS- before RAS-)

Dynamic RAMs require a periodic refresh of their memory contents. The refresh is achieved if all memory cells are accessed at least once every 2/4/8 ms (depending up on the size and the internal memory array configuration). Each row of dynamic RAM is refreshed by a read and/or write cycle. Since in a typical system it cannot be guaranteed that all rows will be accessed at least once (except in video RAM applications) within the specified interval, special refresh cycles must be generated to ensure this.

There are two ways to do dynamic RAM refresh, first, the RAS- only refresh and second the CAS- before RAS- refresh method.

In the RAS- only refresh method, the address of the row which is to be refreshed is placed on the dynamic RAM address lines and the RAS- is activated. The column address is not required and the CAS- strobe is suppressed (power saving).

In the CAS- before RAS- refresh method, the user only needs to activate the CAS- followed by the RAS- signal, the refresh address is taken care of by the dynamic RAM itself. This feature simplifies the dynamic RAM interface to a great extent.

As V25 not only generates a refresh signal but supplies a 9-bit refresh address as well in case of upto μPD42425 x n type dynamic RAMs it is no problem to use either of the two above mentioned methods.

Here, the CAS- before RAS- refresh method is used. The refresh cycle condition is given by the equation:

DRAMREF- = REFRQ- (Equ. 5)

For the dynamic RAM interface, during refresh cycle the address, the Chip select (CS-) and address multiplex (MUX-) signals fulfill the don't care condition.

CAS- (↓) = REFRQ- (Equ. 6)

RAS- (↓) = REFRQ- * CAS- * BW (↓) (Equ. 7)

Under the conditions given in equation Equ. 6 and Equ. 7, the t_{CSR} and t_{CHR}, t_{RAS} parameters are fulfilled in both 5 and 8 MHz V25 systems without problems.

Figure 4 shows V25 Vs dynamic RAM interface control signal logic for 5 and 8 MHz systems.

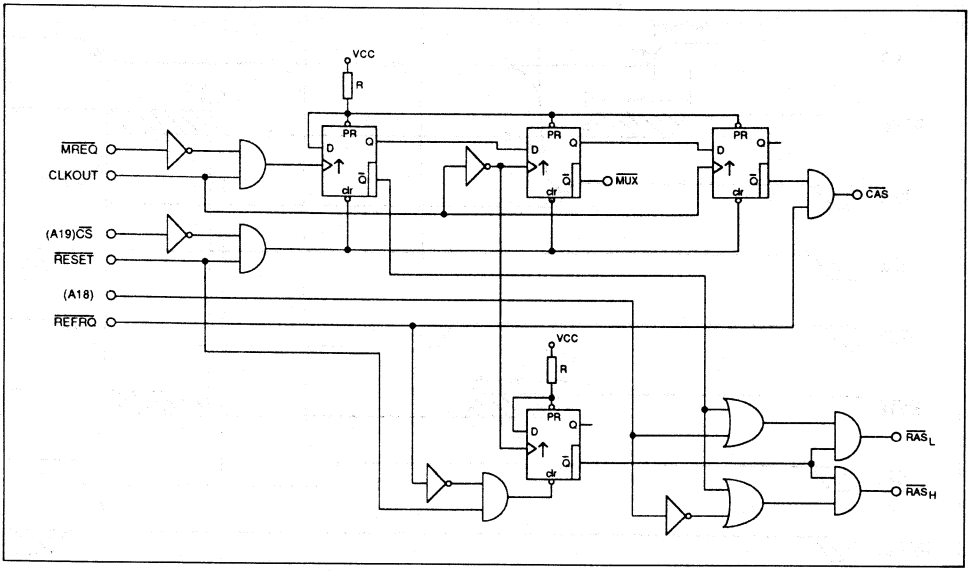


Figure 4. V25 Dynamic RAM Control Logic, 5/8 MHz / 1 Wait State 512K Bytes using four μPD424256-X

Figures 5, 6, 7, 8, 9 and 10 show μPD424256-12 Vs. V25 memory read, write and CAS- before RAS- refresh cycles for 5 and 8 MHz V25 systems respectively.

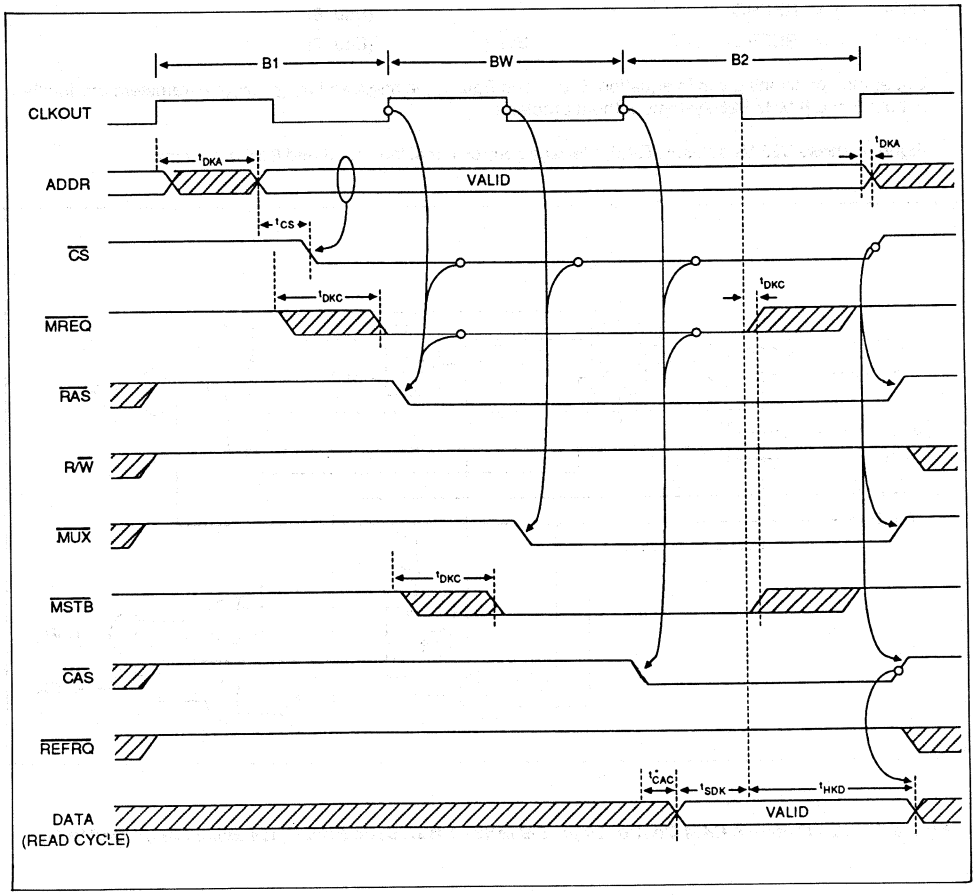


Figure 5. Dynamic RAM Read Cycle 5 MHz / 1 Wait State

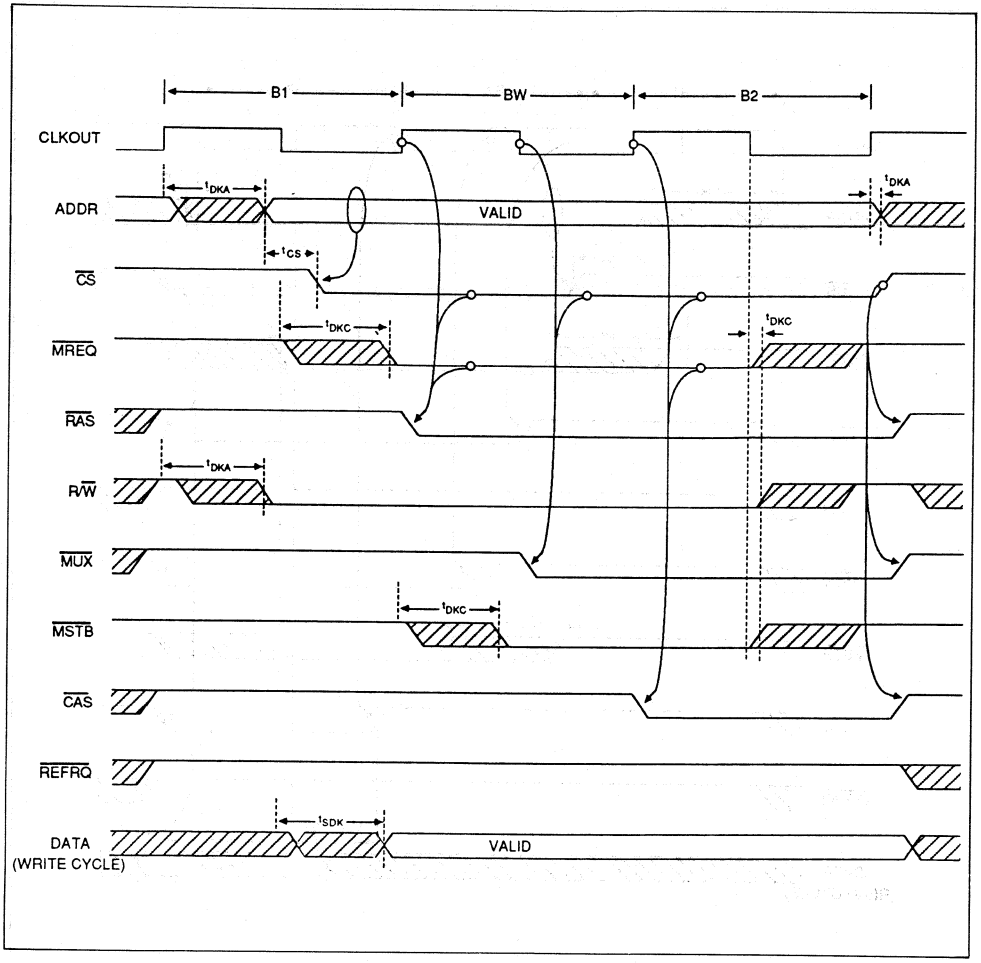


Figure 6. Dynamic RAM Write Cycle 5 MHz / 1 Wait State

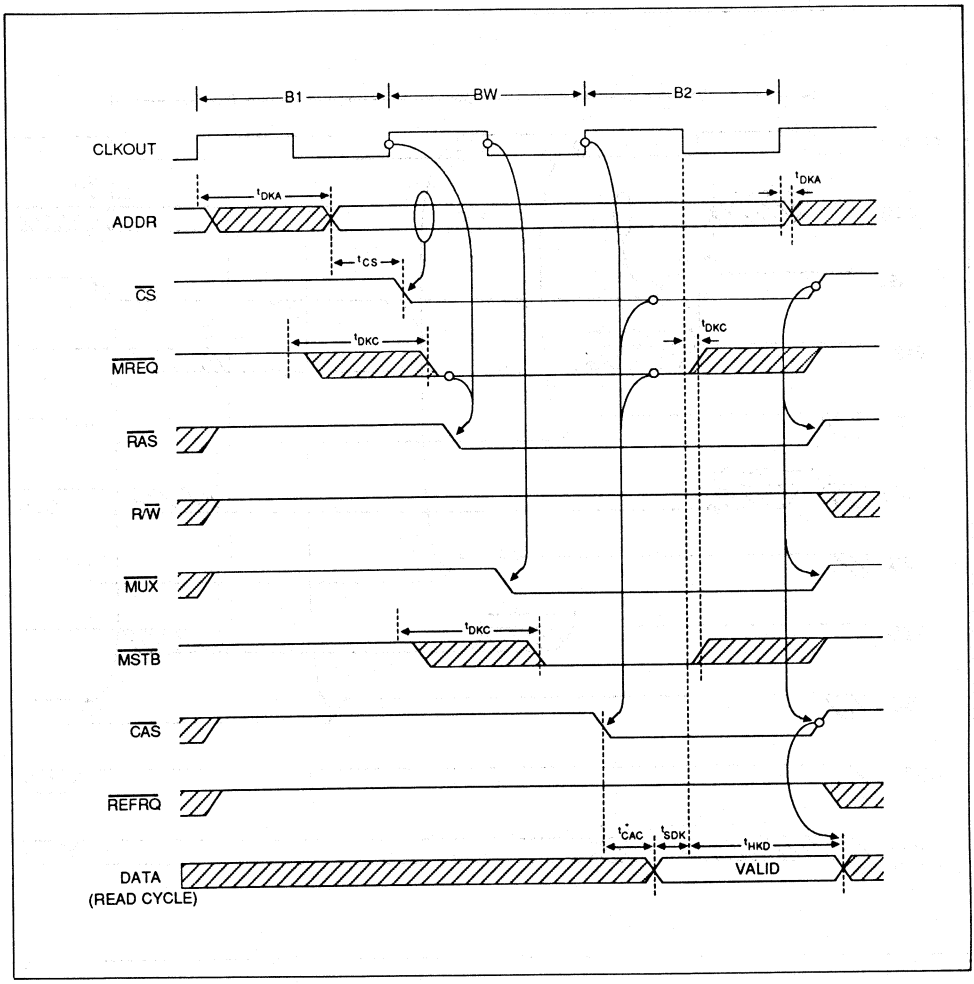


Figure 7. Dynamic RAM Read Cycle 8 MHz / 1 Wait State

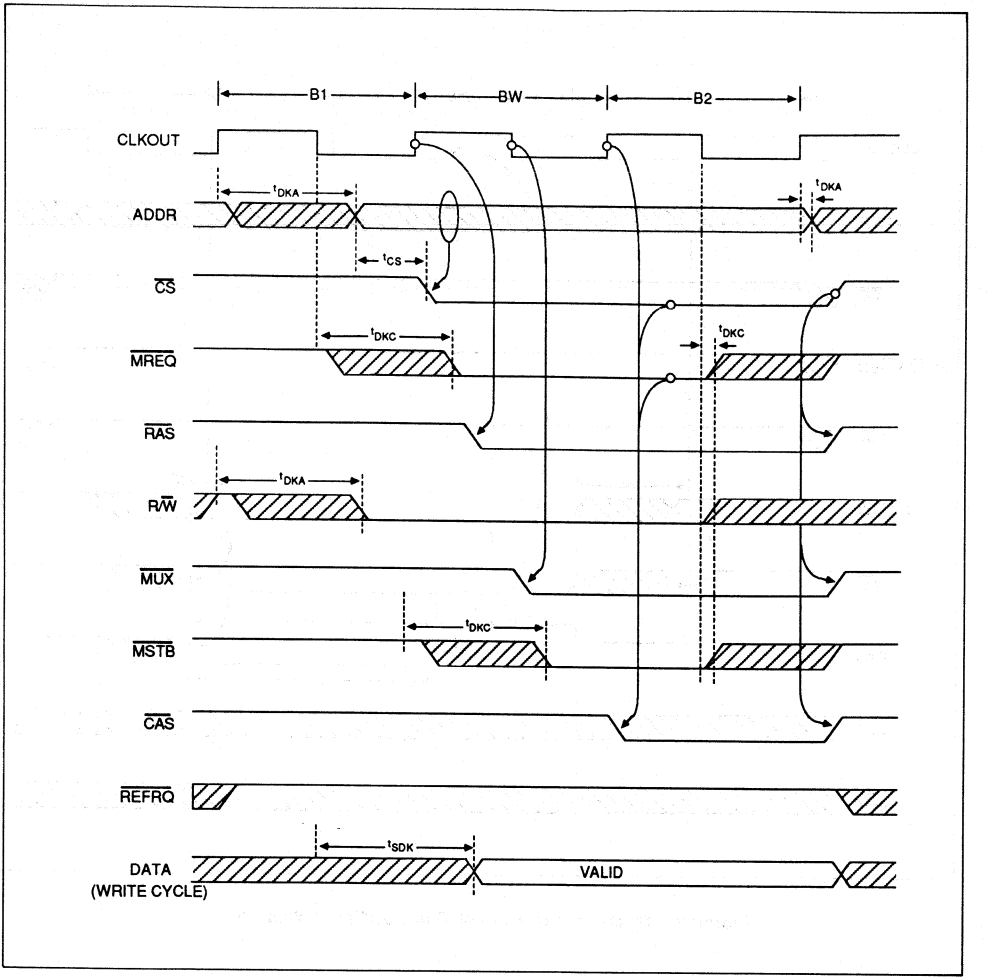


Figure 8. Dynamic RAM Write Cycle 8 MHz / 1 Wait State

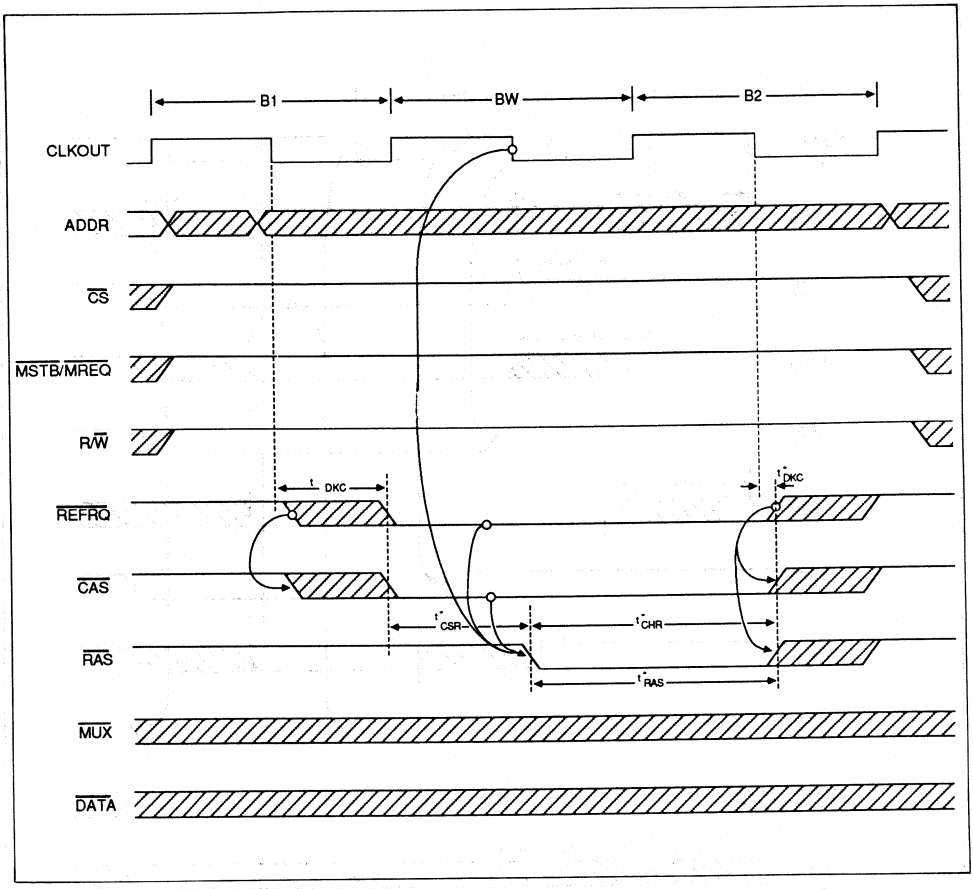


Figure 9. Dynamic RAM Refresh Cycle 5 MHz / 1 Wait State

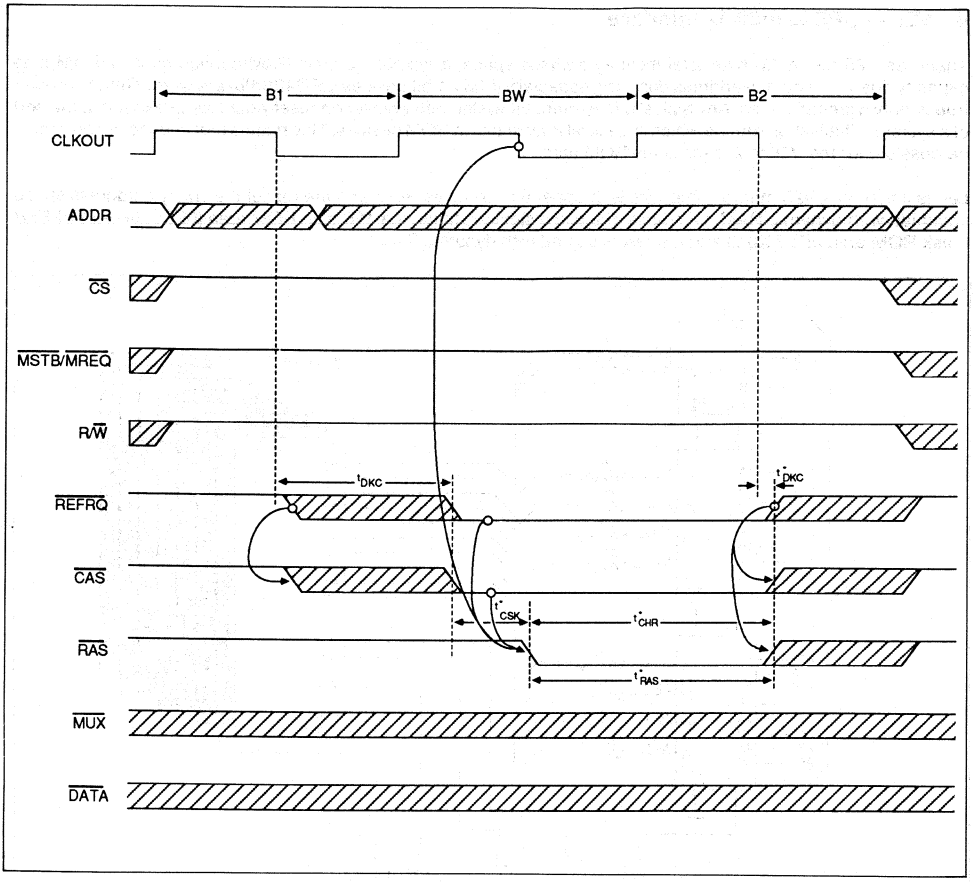


Figure 10. Dynamic RAM Refresh Cycle 8 MHz / 1 Wait State

4. V25 — μPD421000-12 Interface

Again, as V25 has a 1M byte total memory address space, a typical dynamic RAM configuration with 1M byte capacity doesn't need any address decoding especially if 1M x 1 bit devices (μPD421000) are used. This is however true in only a limited sense. Any typical microcomputer system after power on reset would require to run some sort of a system initialisation routine which in general is available in ROM devices. This means that after power on it must be possible for the V25 to access some ROM area.

Let us consider a V25 memory interface with upper 64K bytes as ROM and the rest of the 1M byte address space occupied with dynamic RAM. Of course one could similarly think of a 16K byte ROM area (0FC000H—0FFFFFH mask ROM area μPD70322) and the rest occupied with dynamic RAM.

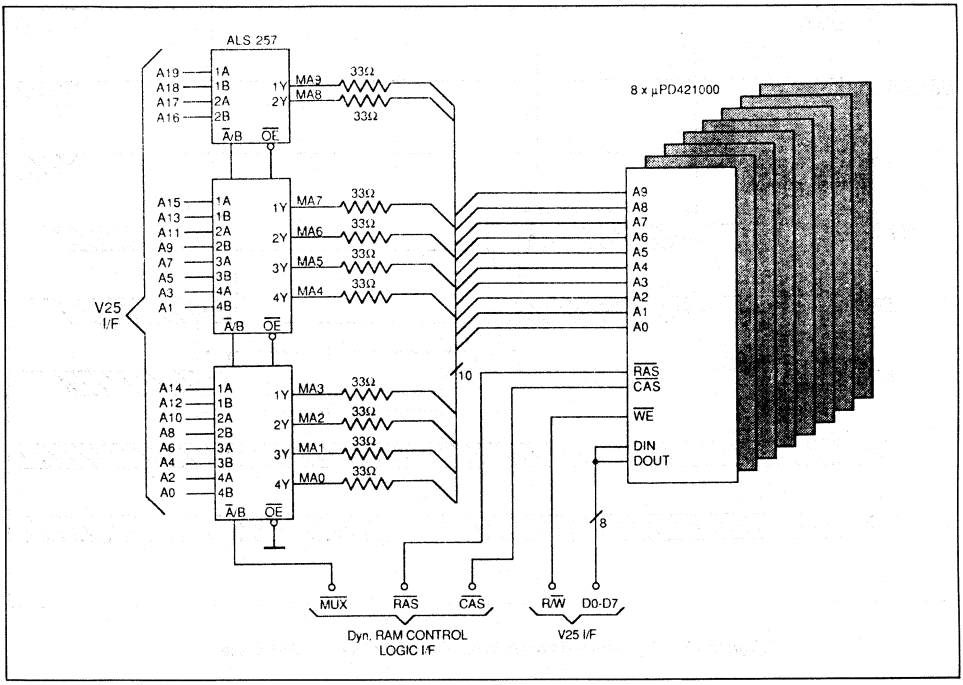


Figure 11.

Figure 11 shows the μPD421000-12 address multiplex logic using 74HCT257 multiplexer devices. As eight μPD421000-12 devices are connected in parallel, there is a common RAS- and CAS- control line for all of them. The R/W- signal of V25 controls the WE- pin of the dynamic RAM to differentiate between read and write cycle. As the "D-out" pin of μPD421000 is in high impedance state during a memory read cycle, the "D-in", "D-out" pins can be connected together and driven by D0—D7 lines of the V25.

The RAS-, MUX- and CAS- control signals are generated (CLOCKOUT synchronised) as shown in figure 12. The timings generated, allow easy interface for μPD421000-12 type dynamic RAMs.

The typical memory read, write cycle at 5 and 8 MHz V25 systems with 1 wait state runs identically as described in μPD424256-12 interface section. The timings diagram are shown figures 5, 6, 7 and 8 respectively.

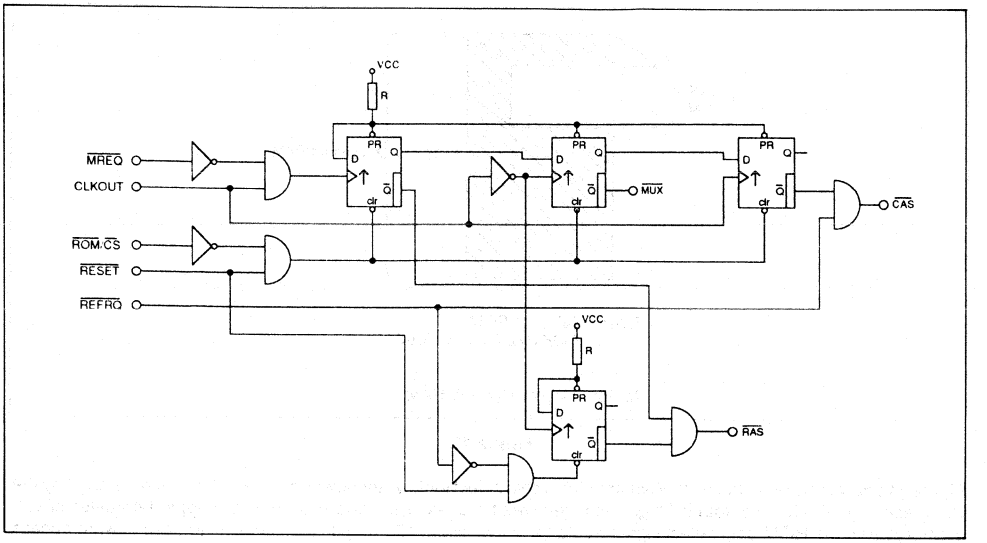


Figure 12. V25 Dynamic RAM Control Logic, 5/8 MHz / 1 Wait State 1M Bytes using eight μPD421000-X

As in case of μPD424245, the μPD421000 devices support also RAS- only and CAS- before RAS- refresh modes.

However as μPD421000 require a 10-bit refresh address, the 9-bit refresh address offered by V25 would not be sufficient for RAS- only refresh mode. If this refresh mode were to be selected, the user will have to generate the 10th refresh address bit externally. Thus need extra logic and is expansive.

The problem however eliminates itself is CAS- before RAS- refresh mode is used. The control logic shown in figure 12 is capable of generating the CAS- before RAS- refresh cycle every time the V25 activates its REFRQ- signal.

Figures 9 and 10 timing diagrams for such a refresh cycle are also valid for μPD421000.

In the μPD421000 dynamic RAM interface to V25 described in the foregone paragraphs the upper 64K bytes are mapped as ROM. Using some additional logic, a system configuration can be realised to map off the ROM (needed after power-on reset), once the initilisation is completed and all memory access is made to dynamic RAM in full 1M byte area.

One possible way to realisethis is reaslised as follows, immediately after power on reset, a flip-flop is set that generates the ROM enable signal. At this time the V25 sees no dynamic RAM in the upper 64K bytes memory address space for any read access. Memory write accesses however are made to the dynamic RAM even in the upper 64K bytes of the 1M byte memory address space. Once the system configuration is completed the flip-flop controlling the memory access to the upper 64K bytes of the memory space can be reset so that the ROM area is mapped off and all read/write access be then directed to the dynamic RAMs. Needless to say that the flop-flop controlling the memory map itself is located in the V25's 64K byte I/O address space. Figure 13 shows the 1M byte dynamic RAM mapping discused above.

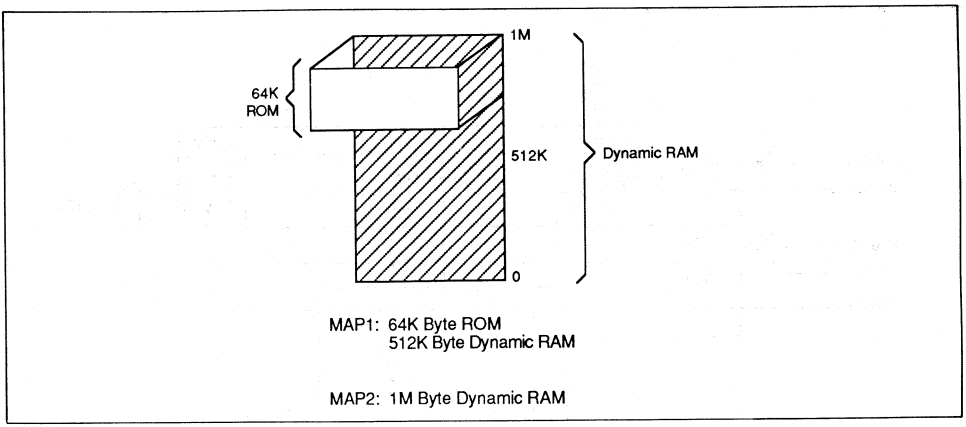


Figure 13.

Figure 14 shows one possible ROM/Dynamic RAM chip select logic. After power-on reset, the flip-flop is reset by the CPU reset signal. The Q output is "high". For every read access to any address within the upper 64K bytes of V25 memory area, the ROM chip select signal would be active (High) allowing a ROM access. In case of write access the ROMCS will be inactive.

Once the ROM access is no more required, the flip-flop can be set by an access in the V25 I/O area. The exact location of the flip-flop in the I/O map can be fixed by the I/O address decoder to suit individual applications.

Since the flip-flop is switched in toggel mode, the ROM can be switched "on" and "off" in the V25 memory map by every alternate access to the flip-flop location in V25 I/O map.

Other ROM configurations can be mapped by decoding the appropriate address select signal.

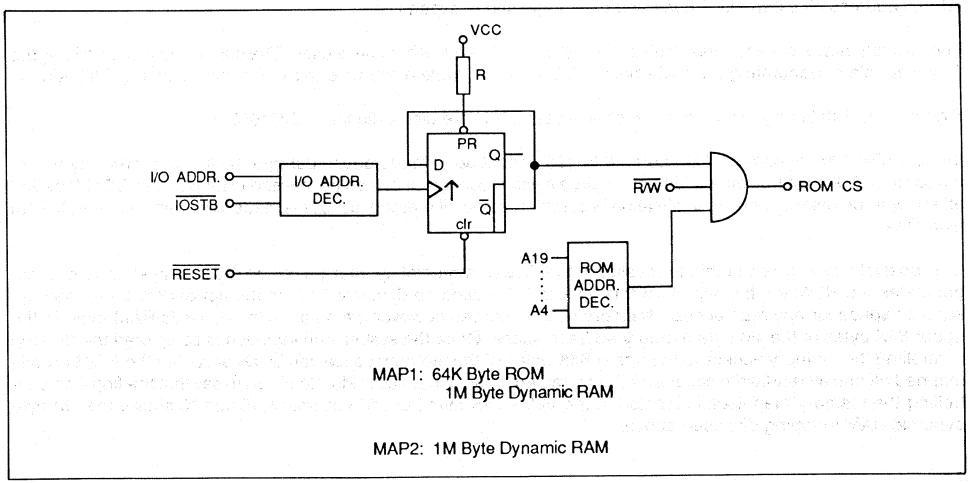


Figure 14.

Interfacing the μPD70320/70322 (V25) to four μPD7228S (Dot Matrix LCD, 2 Lines 40 Characters per Line)

Contents:

Author: Brian Gough
Application μCOM Department
NEC Electronics (Europe) GmbH

Related Documentation

User's Manual
μPD70320/70322
μPD70330/70332
μPD79011

Related Products

V25	16 Bit Microcomputer	CMOS
V35	16 Bit Microcomputer	CMOS
μPD79011	V25 + RTOS on chip	CMOS

1. Introduction

This application note interfaces the V25, NEC's state of the art 16 bit single chip microcomputer in CMOS technology, to 4 μPD7228's dot Matrix Driver Controllers. This idea could be used in many applications such as:

— ISDN telephone
— Single board computer
— Industrial control equipment
— Low power hand held applications

The V25 contains a lot of peripheral functions including serial interface (UART + I/O interface) hardware to make a relatively easy serial interface to the dot matrix controllers. In this application the serial channel 0 is utilized and is switched into NEC's I/O mode. Port 0 is used for handshaking and the timer unit produces the required LCD base frequency of 500 KHz.

2. Hardware Design

Figure 1 shows the circuit diagram of the V25 interface to the dot matrix driver controllers. The LCD display used has two lines and 40 characters per line when configured in the ASCII character generator mode. To control this display requires 4 driver controllers (μPD7228), working in synchronization with each other via the SYNC terminal. The driver controllers must be configured in the 16 multiplexing drive mode with chip 0 as the master chip providing 8 row drivers and chip 1 as a slave chip providing the other 8 row drivers, giving a 16 row back plane. The row drivers on chips 2 and 3 are left open. Also note that the master chip provides the output of the SYNC terminal.

To select each chip for sending a command or data the chips have a common chip select line called CS which will select all chips. To distinguish between which chip has been selected the serial data following the CS signal provides this, with respect to the CA1 and CA0 pin conditions on each driver controller. The biasing of each chip is provided by one biasing network as shown in circuit A.

Figure 2 shows a timing diagram of the first command byte sent to the driver controllers, please make a special note of the busy line condition at the end of this transmition. As you can see the busy line is not active for the first byte, but after this the busy line will go low for approximately 24 μs at the end of each byte sent to the driver controllers. Please see figure 3.

The V25 can easily be configured to communicate with the driver controllers using NEC's I/O mode which is available on serial channel 0, all the driver controllers have this serial configuration which transmits and receives data with a synchronized clock. Port 0 was used in this application because the bits of the port can be configured input or output independently of each other. There are two modes of operation of the driver controllers, command and data mode. The command mode is used to set up the driver controllers and the data mode is used for transmiting ASCII characters. The pin C/D is the command/data pin, when this is high, it means command mode and low means data mode.

The base frequency for the LCD driver controllers is 500 KHz and is provided by the V25s timer unit. This could easily be replaced by a frequency divider such as 4013 or 4017 utilizing the CPU clock output pin, to save using the timer unit. Pull up resistors are used, because of the possible high impedence condition which may occur on the communication lines.

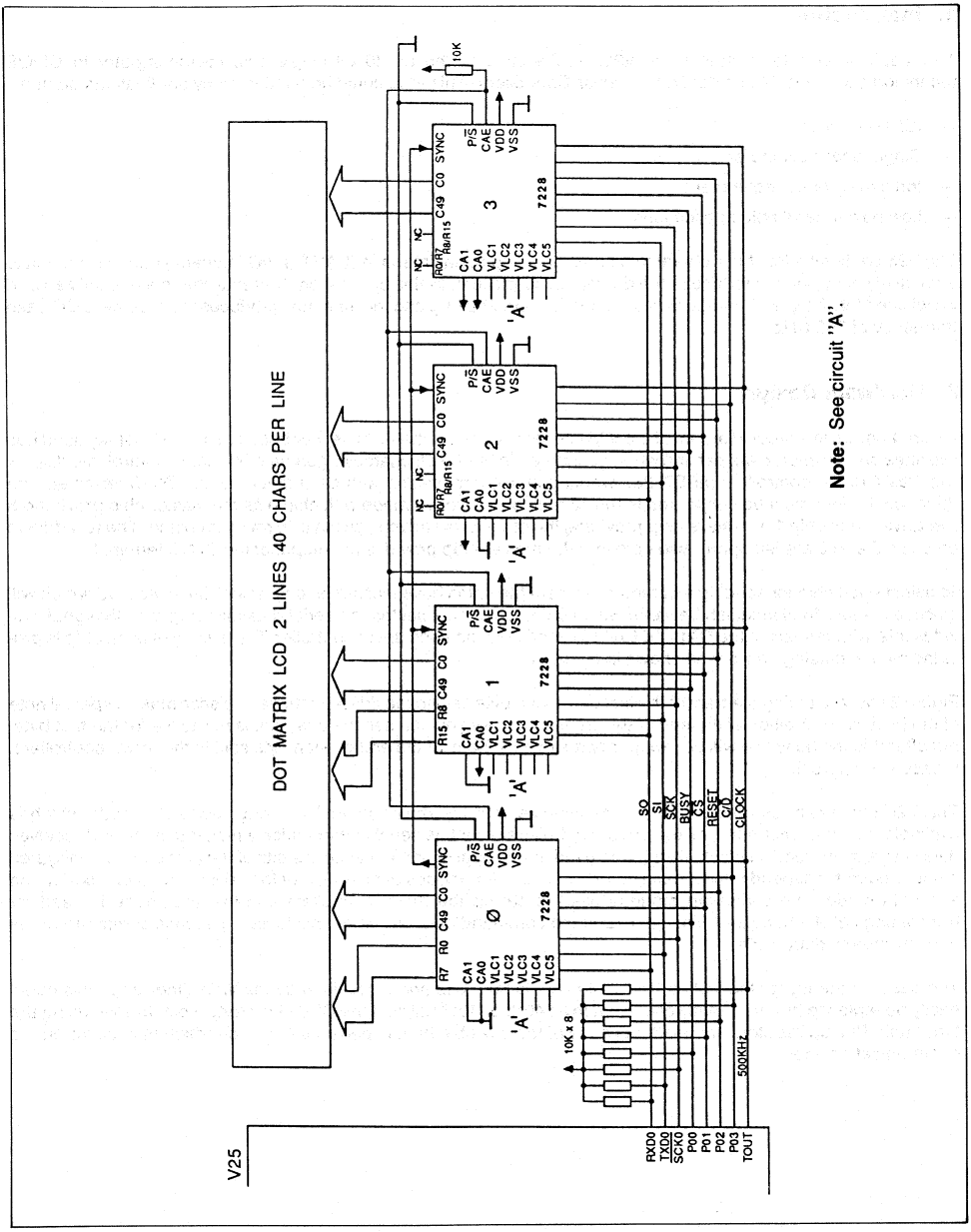
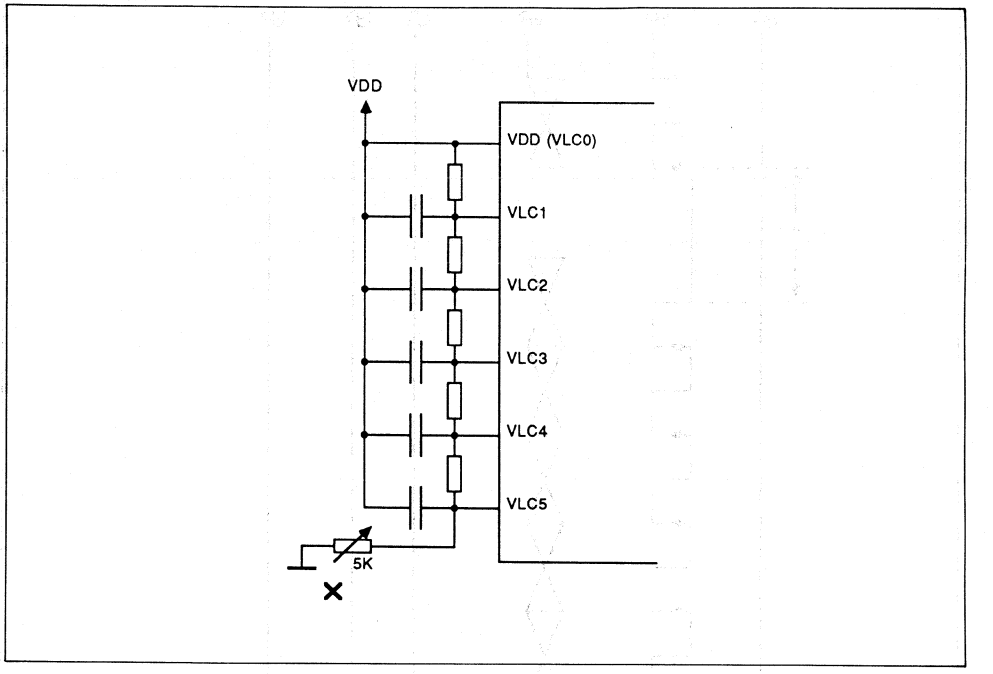


Figure 1. Circuit Diagram

Circuit "A"



Note: All μPD7228's are connected to the same BIAS network, The variable resistor R is optional, VLC5 can go directly to ground. (R gives contrast and temperature compensation.) Point X could be connected to the reset line, to switch off LCD during standby condition.

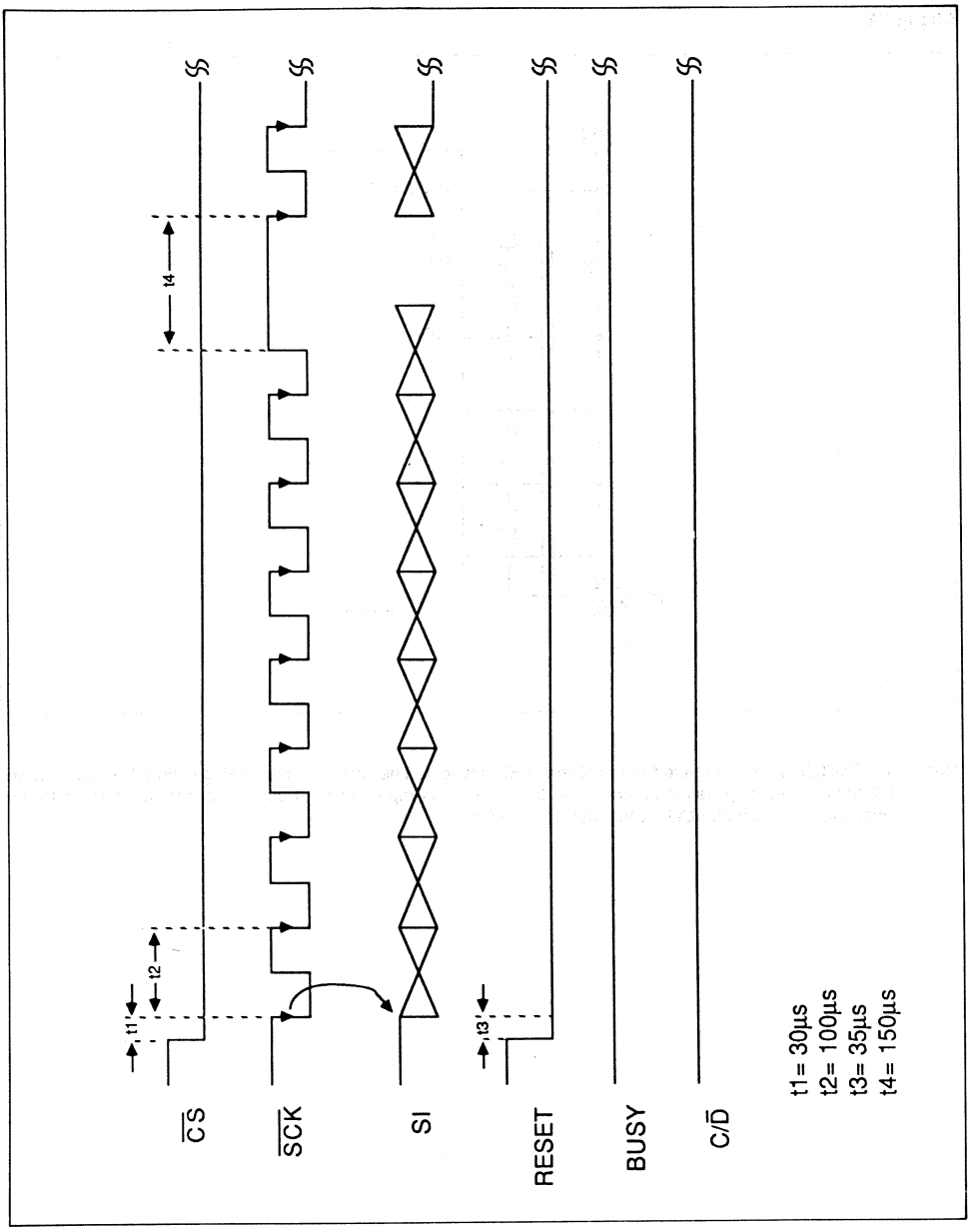


Figure 2. Timing Diagram of first Byte to the 7228

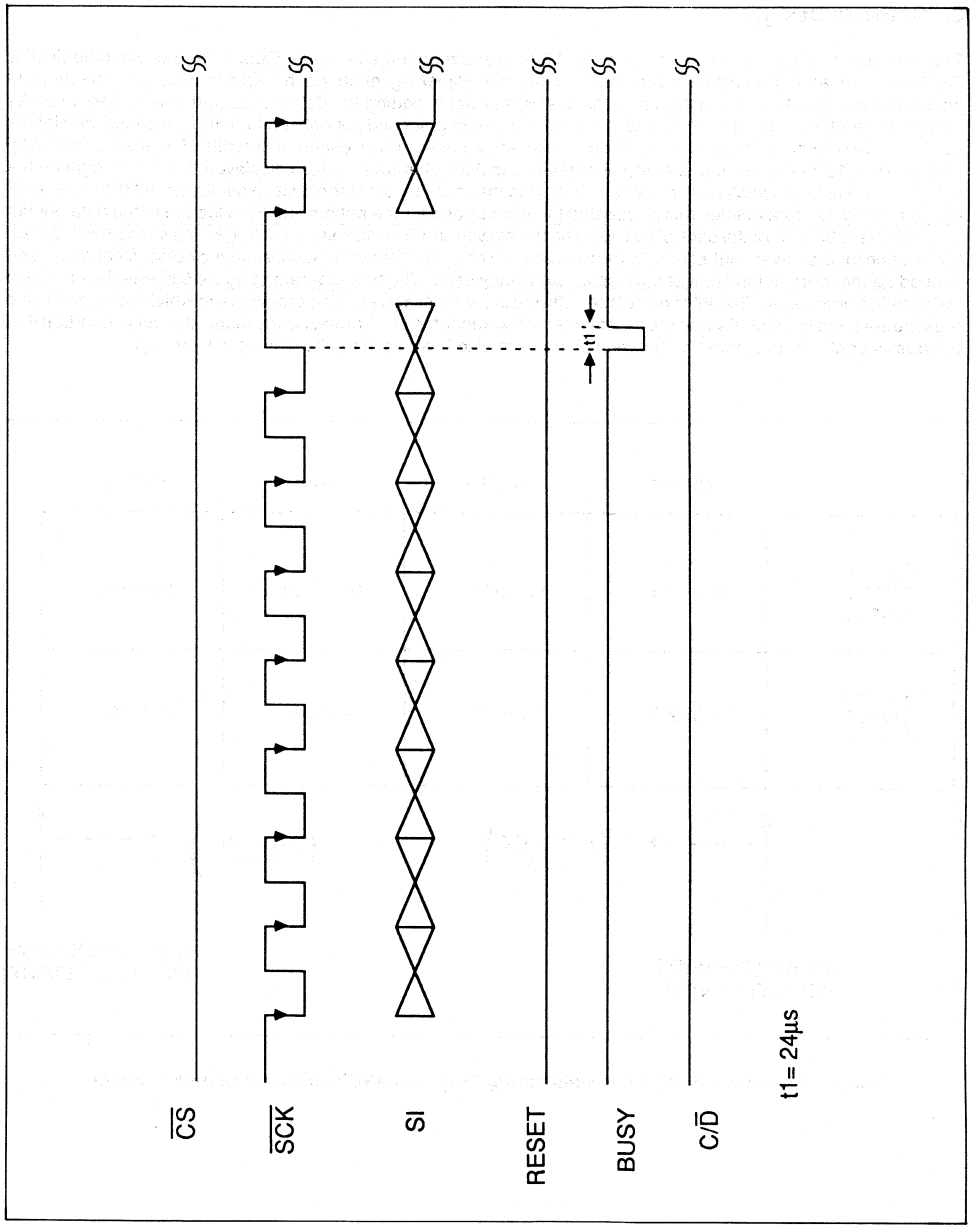


Figure 3. Timing Diagram of second Byte to the 7228

3. Software Design

The software is written for a display using the ASCII character generator mode. Figure 4 shows what the display RAM looks like after the chips are configured in the 16 multiplexing mode. Each chip can drive 20 characters, 10 characters per line. It is very important to make sure that when loading the data pointer with the highest value for bank 0 (B1) and bank 1 (F1) the character mode with left entry is set and not right entry. This is because the highest value of data pointer will be automatically decremented by 5 every time a character is written into the display RAM. After loading 10 characters into the display RAM of chip 0 the characters will be displayed from left to right on the top line of the display which is memory bank 0. To load the characters into the next chips display RAM the value B1 is again set to the data pointer, this is repeated for all chips and for the bottom line F1 is loaded to the data pointer which refers to bank 1. At the back of this application note you can find the software listing which shows the initialization routine for the driver controllers. As you will see each chip is initialized separatly and the cursor clear command is used so that both of the lines of the display are clearly seen. The flow diagram of figure 5 shows the complete initialization procedure. The other part of the software is the message sending procedure in which again each chip must be selected individually and the data pointer set correctly for left character entry mode. It is very important that the busy signal is high before transmission of data and also before setting the chip select line high.

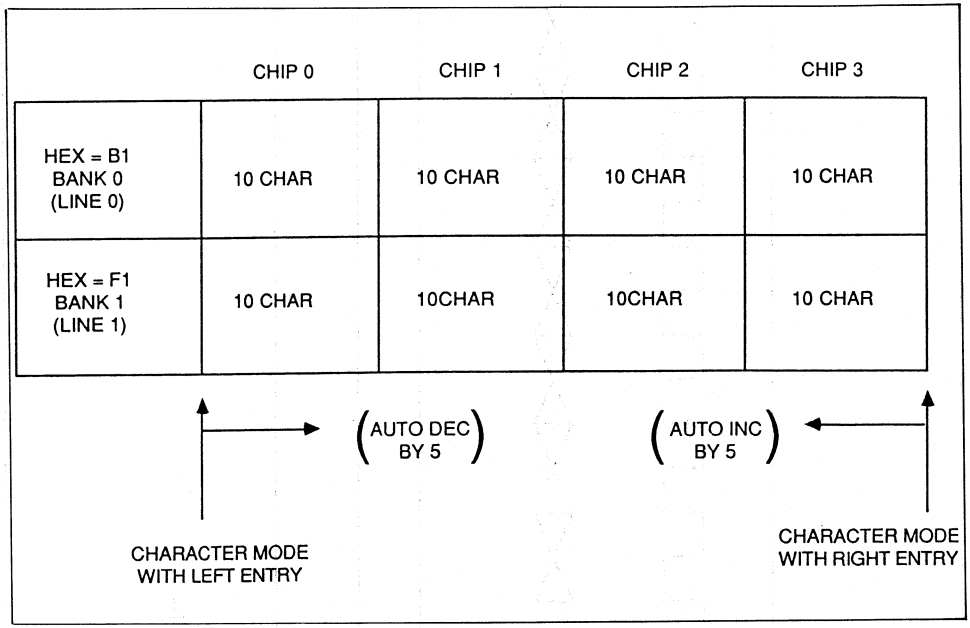


Figure 4. Display RAM, for 16 Multiplexing Mode and ASCII Character Generator Mode

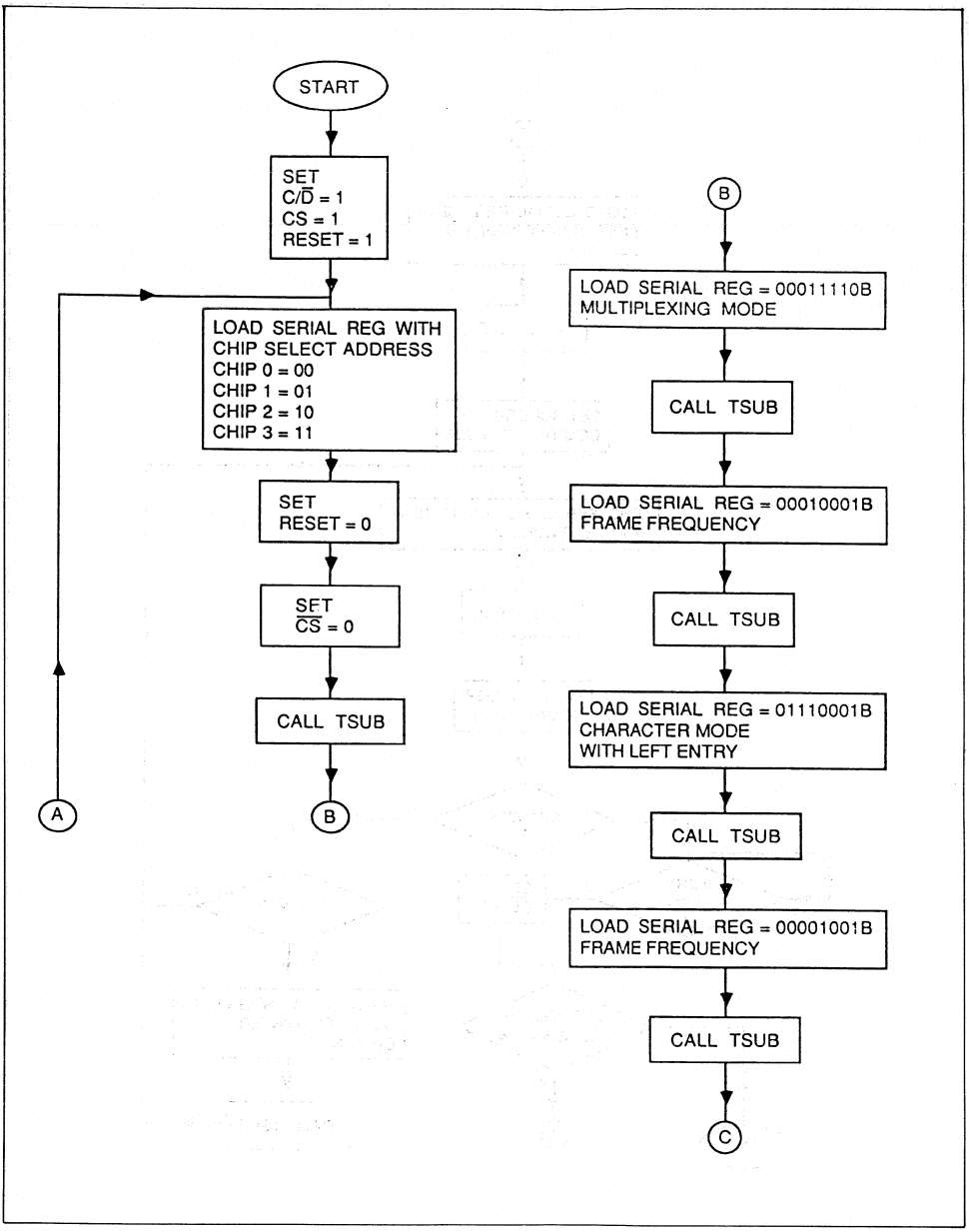
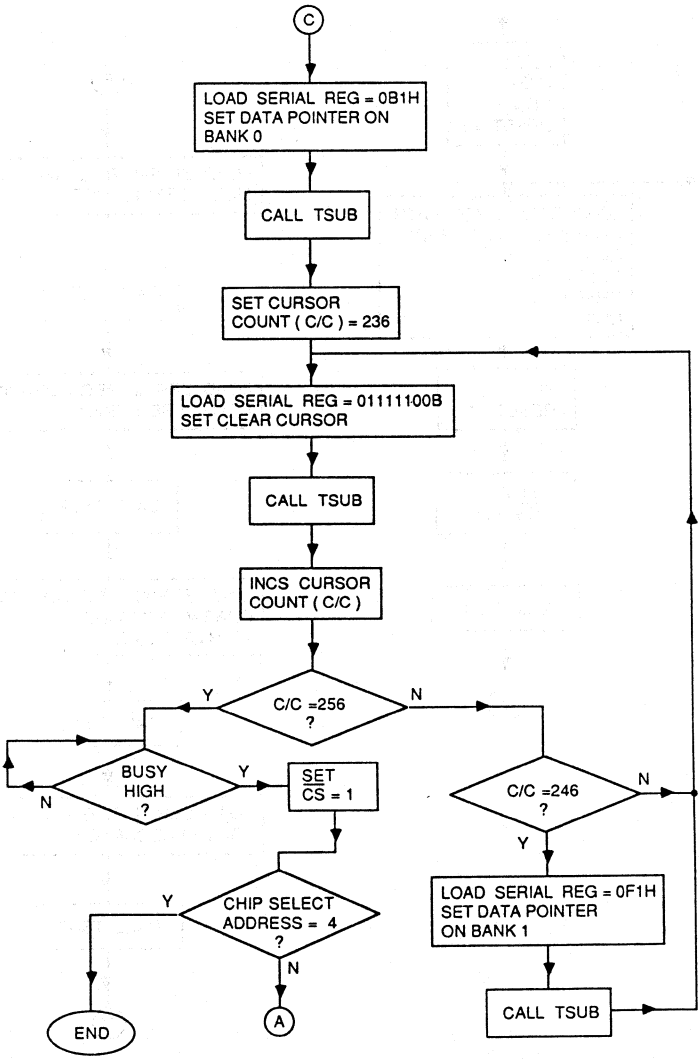
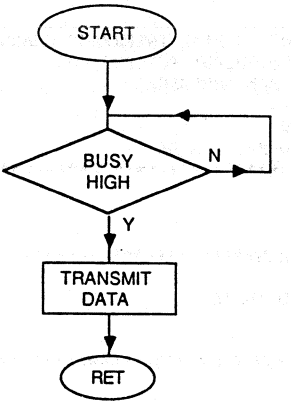


Figure 5. Display Driver Initialization


C
LOAD SERIAL REG = 0B1H
SET DATA POINTER ON
BANK 0
CALL TSUB
SET CURSOR
COUNT (C/C) = 236
LOAD SERIAL REG = 01111100B
SET CLEAR CURSOR
CALL TSUB
INCS CURSOR
COUNT (C/C)
C/C =256
?
Y
N
BUSY
HIGH
?
Y
N
SET
CS = 1
C/C =246
?
N
Y
CHIP SELECT
ADDRESS = 4
?
Y
N
LOAD SERIAL REG = 0F1H
SET DATA POINTER
ON BANK 1
CALL TSUB
END
A

TRANSMIT SUB
(TSUB)



4. Software Listing

```
        NAME    LCDDRS
;INTERFACING THE V25 TO FOUR UPD7228s(DOT MATRIX LCD,2 LINES 40 CHARS)
;WRITTEN FOR V25(UPD70320/22) BY B.GOUGH 7/88
;*********************************************************************
;NAME           :LCDGO
;DESC           :4*7228
;DEVICE                 :V25
;MEMORY BANK    :
;REGISTER BANK :7
;HARDWARE USED :TIMER UNIT,SERIAL INTERFACE CHANNEL 0 I/O MODE
;HARDWARE USED :INPUT P00,OUTPUT P01,P02,P03
;INPUT          :MESSAGE RAM 'MESSAGE'
;OUTPUT                 :-
;DESTROY        :AW,BW,CW
;CALLS          :PRO:SEND_MA,TRAN_DA
;*********************************************************************

        PUBLIC  START0
;*
;**********SET DATA SEGMENT*************************************************
;*
        DATA SEGMENT WORD 'DATA'
        DATA ENDS
;*
;**********SET SPECIAL FUNCTION REG*****************************************
;*
        ASGNSFR DATA
;*
;**********SET STACK SEGMENT************************************************
;*
        STACK SEGMENT WORD STACK
        STACKSIZE DW 40 DUP(?)
        STACK ENDS
;*
;**********SET CODE SEGMENT*************************************************
;*
        CODE SEGMENT WORD PUBLIC 'CODE'
;*
;**********SET ASSUME CONDITION*********************************************
;*
        ASSUME PS:CODE,DS1:DATA,SS:STACK
;*
;**********MAIN PROGRAM*****************************************************
;*
START0 LABEL    FAR
        MOV     AW,STACK                ;SET STACK CONDITION
        MOV     SS,AW
        MOV     SP,SIZE STACKSIZE
;
        SETIDB  50H                     ;SET IDB REG (INTERNAL DATA BASS)
                                        ;THIS REG LOCATES THE 8 REG BANKS
                                        ;SFR AREA DMA AND MACRO CHANNELS
;
        MOV     AW,5000H                ;SET DATA SEG1 TO INTERNAL RAM AREA
        MOV     DS1,AW                  ;ADDRESS 50000 (PLUS OFF SET)
;*
;********** SET UP V25 ***************************************************
;*
        MOV     PRC,40H                 ;SET PROCESS CONTROL REG
```

```
;
        MOV     PMC0,10000000B          ;SET PORT MODE CONTROL REG
        MOV     PM0,00000001B
        MOV     P0,0FFH                 ;SET CONTROL PORT HIGH
        MOV     PMC1,01100000B          ;SET PORT MODE CONTROL REG
;
        MOV     SCM0,10000100B          ;SET UP SERIAL INTERFACE REG(I/O MODE)
        MOV     BRG0,01111101B          ;SET CLOCK 9.6KHZ
        MOV     SCC0,00000001B          ;FCLK/2
;
        MOV     MD0,00000000B           ;TIMER INTERVAL =1 CLOCKS
        MOV     TMC0,10001000B          ;TM0 COUNT CLK=FCLK/6
                                        ;500KHZ LCD BASE FREQUENCY
;*
;********** INITIALIZE 7228*4 ****************************************
;*
        MOV     BH,00H                  ;CHIP SELECT COUNTER REG
        CLR1    P0,2                    ;SET RESET LINE LOW
        NOP                             ;DELAY FOR RESET
        NOP
OUT1:   CLR1    P0,1                    ;CHIP SELECT LINE LOW
        MOV     CH,BH                   ;CHIP SELECT 0,1,2,3
        CALL    TRAN_DA
;
        CMP     BH,00H                  ;MASTER OR SLAVE CHIP SELECT ?
        BZ      OUT0
        MOV     CH,00011100B            ;CHIP1 TO 3 SLAVE CHIP
        BR      OUT5
OUT0:   MOV     CH,00011110B            ;CHIP0=MASTER SET MULTIPLEXING MODE
OUT5:   CALL    TRAN_DA
;
        MOV     CH,00010001B            ;SET FRAME FREQUNECY
        CALL    TRAN_DA
;
        MOV     CH,01110001B            ;SET CHARACTER MODE WITH LEFT ENTRY
        CALL    TRAN_DA
;
        MOV     CH,00001000B            ;DISPLAY OFF (STOP DC ON DISPLAY)
        CALL    TRAN_DA
;
        MOV     CL,256-20               ;INIT BANKS COUNT (10*2)
        MOV     CH,0B1H                 ;SET DATA POINTER ON BANK 0
OUT4:   CALL    TRAN_DA
OUT3:   MOV     CH,01111100B            ;SET CLEAR CURSOR
        CALL    TRAN_DA
        INC     CL                      ;NEXT BANK BIT
        CMP     CL,00H                  ;END OF COUNT 10*2
        BZ      OUT2
        MOV     AL,256-10
        CMP     CL,AL                   ;END OF BANK 0 ?
        BNE     OUT3                    ;NO
        MOV     CH,0F1H                 ;SET DATA POINTER ON BANK 1
        BR      OUT4
;
OUT2:   BTCLR   P0,0,OUT7               ;BUSY HIGH ?
        BR      OUT2
OUT7:   SET1    P0,1                    ;SET CHIP SELECT LINE HIGH
        NOP                             ;DELAY FOR CHIP SELECT HIGH
        NOP
        NOP
```

```
        INC     BH                      ;NEXT CHIP INITILIZE
        CMP     BH,04H
        BE      OUT6                    ;OUT TO MAIN PROGRAM
        BR      OUT1                    ;NEXT CHIP
;*
;**********MAIN LCD ROUTINES*****************************************
;*

OUT6:   CALL    SEND_MA                 ;SEND TEST MESSAGE

;*
;**********SEND MESSAGE PROCEDURE************************************
;*

SEND_MA         PROC    NEAR
OUTS:   BTCLR   P0,0,OUTS8              ;BUSY HIGH ?
        BR      OUTS
OUTS8:  SET1    P0,1                    ;SET CHIP SELECT LINE HIGH
        SET1    P0,3                    ;SET CONTROL MODE
        MOV     BL,00H                  ;SET BANK 0 FLAG
        MOV     BH,00H                  ;SET CHIP SELECT CHIP 0
        MOV     IX,OFFSET MESSAGE       ;IX=MESSAGE POINTER
;

OUTS0:  MOV     CL,256-20               ;BANK 0,1 COUNT
        CLR1    P0,1                    ;CHIP SELECT LINE LOW
        MOV     CH,BH                   ;CHIP SELECT CHIP 0,1,2,3
        CALL    TRAN_DA
        MOV     CH,0F1H                 ;SET BANK 1 POINTER TOP OF RAM
        CMP     BL,0FFH                 ;BANK 0 OR 1
        BE      OUTS1
        MOV     CH,0B1H                 ;SET BANK 0 POINTER TOP OF RAM
OUTS1:  CALL    TRAN_DA
        CLR1    P0,3                    ;DATA MODE
;

OUTS2:  LDM     BYTE PTR PS:[IX]        ;GET ASCII CHAR IN AL,IX AUTO INC
        MOV     CH,AL                   ;LOAD SERIAL PARAMETER REG
        CMP     AL,00H                  ;END OF MESSAGES ?
        BZ      OUTS                    ;YES RESET TO START OF MESSAGES
        CALL    TRAN_DA
        INC     CL
        CMP     CL,256-10               ;END OF CHIP RAM ?
        BE      OUTS3                   ;YES
        BR      OUTS2                   ;NO
;

OUTS3:  BTCLR   P0,0,OUTS4              ;BUSY HIGH ?
        BR      OUTS3
OUTS4:  SET1    P0,1                    ;SET CHIP SELECT LINE HIGH
        SET1    P0,3                    ;CONTROL MODE
        INC     BH                      ;NEXT CHIP
        CMP     BH,04H                  ;END OF BANK 0,1 CHIPS ?
        BNE     OUTS0                   ;NO
;*
;***NEXT LINE 1
;*

        CMP     BL,0FFH                 ;BANK 1 FLAG SET ?
        BE      OUTS5
```

```
        MOV     BL,0FFH                 ;SET BANK 1 FLAG
        MOV     BH,00H                  ;RESET CHIP SELECT
        BR      OUTS0
;*
;***NEXT TEXT LINES 0,1
;*

OUTS5: CALL     DIS_ON                  ;DISPLAY ON
;

        MOV     BL,0F5H                 ;ON DELAY
        MOV     AW,0000H
OUTS6: INC      AW
        CMP     AW,0000H
        BNZ     OUTS6
        INC     BL
        CMP     BL,00H
        BNZ     OUTS6
;
        CALL    DIS_OFF                 ;DISPLAY OFF
;

        MOV     AW,0000H                ;OFF DELAY
OUTS7: INC      AW
        CMP     AW,0000H
        BNZ     OUTS7
;
        MOV     BL,00H                  ;RESET,BANK 0 FLAG/CHIP SELECT
        MOV     BH,00H
        BR      OUTS0
SEND_MA         ENDP

;*
;**********DISPLAY ON PROCEDURE****************************************
;*

DIS_ON PROC     NEAR
        MOV     BH,00H                  ;CHIP SELECT CHIP 0
OUTD00:         CLR1    P0,1                    ;CHIP SELECT LINE LOW
        MOV     CH,BH                   ;CHIP SELECT
        CALL    TRAN_DA
        MOV     CH,00001001B            ;DISPLAY ON
        CALL    TRAN_DA
OUTD01:         BTCLR   P0,0,OUTD02             ;BUSY HIGH ?
        BR      OUTD01
OUTD02:         SET1    P0,1                    ;CHIP SELECT HIGH
        NOP
        NOP
        NOP
        INC     BH                      ;NEXT CHIP INITILIZE
        CMP     BH,04H
        BNE     OUTD00
        RET
DIS_ON ENDP

;*
;**********DISPLAY OFF PROCEDURE***************************************
;*

DIS_OFF PROC    NEAR
```

```
        MOV     BH,00H                  ;CHIP SELECT CHIP 0
OUTDF0:         CLR1    P0,1                    ;CHIP SELECT LINE LOW
        MOV     CH,BH                   ;CHIP SELECT
        CALL    TRAN_DA
        MOV     CH,00001000B            ;DISPLAY OFF
        CALL    TRAN_DA
OUTDF1:         BTCLR   P0,0,OUTDF2             ;BUSY HIGH ?
        BR      OUTDF1
OUTDF2:         SET1    P0,1                    ;CHIP SELECT HIGH
        NOP
        NOP
        NOP
        INC     BH                      ;NEXT CHIP INITILIZE
        CMP     BH,04H
        BNE     OUTDF0
        RET
DIS_OFF         ENDP

;*
;**********TRANSMIT DATA PROCEDURE************************************
;*

TRAN_DA         PROC    NEAR
OUTT0: BTCLR    P0,0,OUTT1              ;BUSY LINE HIGH ?
        BR      OUTT0
OUTT1: MOV      TXB0,CH                 ;TRANSMIT DATA IN CH REG
;
        MOV     AL,7BH                  ;DELAY LOOP WAIT FOR TRANSMIT
OUTT2: INC      AL
        CMP     AL,00H
        BNZ     OUTT2
        RET
TRAN_DA         ENDP

;*
;**********ASCII CHAR STRINGS*****************************************
;*

MESSAGE         DB '   V25 NECs 16 BIT SINGLE CHIP MICRO    '   ;BANK 0
        DB '        UPD7228 LCD DEMONSTRATION       '   ;BANK 1
        DB '    5 AND 8 MHZ CPU SPEED VERSIONS      '   ;BANK 0
        DB ' MINIMUM INSTRUCTION CYCLE:250NS 8MHZ   '   ;BANK 1
        DB '    CPU WITH 6 BYTE PREFETCH QUEUE      '   ;BANK 0
        DB '                                        '   ;BANK 1
        DB ' BUILT IN ROM: 16,384 WORDS * 8 BITS    '   ;BANK 0
        DB '    BUILT IN RAM: 256 WORDS * 8 BITS    '   ;BANK 1
        DB '    1M BYTE OF MEMORY ADDRESS SPACE     '   ;BANK 0
        DB '       64K BYTE I/O ADDRESS SPACE       '   ;BANK 1
        DB '          4 * 16 BIT REGISTERS          '   ;BANK 0
        DB '              AW,BW,CW,DW               '   ;BANK 1
        DB '            8 REGISTER BANKS            '   ;BANK 0
        DB 'FAST INTERRUPT DRIVEN CONTEXT SWITCHING '   ;BANK 1
        DB '    MEMORY MAPPED ON CHIP PERIPHERAL    '   ;BANK 0
        DB '              HARDWARE                  '   ;BANK 1
        DB '    INPUT OUTPUT PORT WITH COMPARATOR   '   ;BANK 0
        DB '            8 BIT COMPARATOR            '   ;BANK 1
        DB '         INPUT PORTS: 4 BITS            '   ;BANK 0
        DB '      INPUT/OUTPUT PORTS: 20 BITS       '   ;BANK 1
        DB ' SERIAL INTERFACE WITH BAUD RATE GEN    '   ;BANK 0
        DB '     2 CHANNELS ASYNCH AND I/O MODE     '   ;BANK 1
```

```
        DB '          INTERRUPT CONTROLLER          '   ;BANK 0
         DB '  8 LEVELS,VECTOR AND MACRO FUNCTIONS   '    ;BANK 1
        DB '          DRAM AND PSEUDO SRAM          '   ;BANK 0
         DB '             REFRESH FUNCTION           '    ;BANK 1
        DB '              2 DMA CHANNELS            '   ;BANK 0
         DB '    MEMORY <-> MEMORY,I/O <-> MEMORY    '    ;BANK 1
        DB '        8 MACRO SERVICE CHANNELS        '   ;BANK 0
         DB '                                        '    ;BANK 1
        DB '         20 BIT TIME BASE COUNTER        '   ;BANK 0
         DB '                                        '    ;BANK 1
        DB '                TIMER UNIT              '   ;BANK 0
        DB '           THREE 16 BIT TIMERS          '   ;BANK 1
        DB '         BUILT IN CLOCK GENERATOR       '   ;BANK 0
         DB '                                        '    ;BANK 1
        DB '       PROGRAMMABLE WAIT FUNCTION       '   ;BANK 0
         DB '                                        '    ;BANK 1
        DB '             STANDBY FUNCTIONS          '   ;BANK 0
         DB '                HALT AND STOP           '    ;BANK 1
        DB '       LOW POWER CMOS TECHNOLOGY        '   ;BANK 0
         DB '                                        '    ;BANK 1
        DB '          SINGLE 5V POWER SUPPLY        '   ;BANK 0
        DB '                                        '   ;BANK 1
        DB '      OPERATION CURRENT AT 5V/8MHZ      '   ;BANK 0
        DB '             IDD1=120MA MAX             '   ;BANK 1
        DB '  STANDBY CURRENT HALT MODE AT 5V/8MHZ  '   ;BANK 0
        DB '             IDD2=50MA MAX              '   ;BANK 1
        DB '    STANDBY CURRENT STOP MODE AT 5V     '   ;BANK 0
        DB '             IDD3=30UA MAX              '   ;BANK 1
        DB 00
;*************************************************************************
;*

CODE    ENDS
        END
```

The Improved CMOS Graphics Display Controller μPD72020 Simplifies the Interface to Video RAMs

Contents:

Author: Uwe Schäfer
Application μCOM Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD72020 Data Sheet
μPD72020 Product Description
μPD7220A Application Note
μPD41264 Data Sheet

Related Products

μPD7220A	Graphics Display Controller (6 MHz) NMOS
μPD7220A-1	Graphics Display Controller (7 MHz) NMOS
μPD7220A-2	Graphics Display Controller (8 MHz) NMOS
μPD72020	Graphics Display Controller (8 MHz) CMOS

1. Outline

The μPD72020 is a graphic display controller — a CMOS device which is an enhanced version of the μPD7220A.

In addition to the functions of the μPD7220A, the device can perform the address space extended function, the image dual port RAM control function and the write mask function so that it can be applied to a wide range of areas from a simple display terminal to a high-resolution graphic display unit.

Features:

- Enhanced functions of the μPD7220A
  - — Video memory space 2MB max. (1M word x 16 bits)
  - — Image dual port and memory control
  - — Write mask of any selected bit for pixel mode
  - — Improved external synchronizing function
  - — CMOS

The improved interface for dual port memories simplifies the hardware design. Dual port RAM support was also possible with the μPD7220A with the help of external counters and BLANK signal mask circuits. As these parts are now implemented in the μPD72020 the chip-count is drastically reduced.

Note: Throughout the text of this application note, the active low signals are shown with a "—" sign following the signal name. In figures, the active low signals are shown with a bar on top of the signal name.

2. Hardware Description

2.1 The Memory Interface

The memory Interface in this application note is designed using a PAL. This is not the cheapest method but it saves boardspace and makes the design more simple.

The μPD72020 generates three different display memory access cycles:

Drawing cycles

Display cycles

Refresh cycles

All of these cycles start with the rising edge of ALE (also named RAS- in documentation). This signal is used by the two address latches (U6, U7) to latch the row and column address, as well as to start the state machine implemented in the PAL (U5).

The drawing cycles are always performed from the μPD72020 as Read-Modify-Write cycles. This access mode is available on all standard VRAMs.

The μPD72020 reads 16 bit of display data, modifies certain bits, and writes them back. This access always takes four 2xCCLK cycles.

Display cycles are generated successively during the active display time when no drawing is performed. They are also generated even if they have no meaning in Data Transfer Mode. As there is only an address output to the display memory bus and no read or write operation (DT-/OE-, WE- not active) is performed, this will have no effect to the board.

The only display cycles which are neccessary are Data Transfer cycles. They can be distinguished from normal display cycles by DT-signal active.

A DT cycle forces the VRAMs to load a complete row into its serial shift register beginning from the display address output. The DT cycle is identified from the VRAMs by a low level on the DT-/OE- pin at the falling edge of RAS.

Refresh cycles are output succesively only during the horizontal sync period.

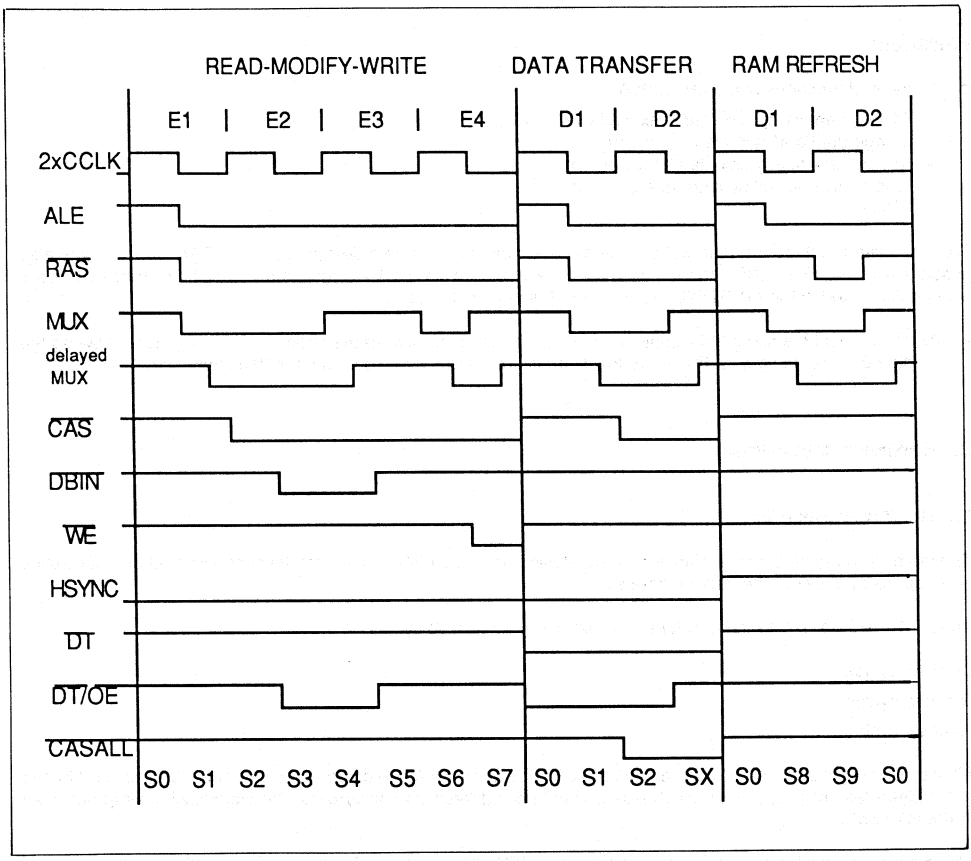


Figure 1. The three Different Memory Access Cycles generated by the PAL

The state machine determines with the signals HSYNC, DT- and DBIN- which kind of cycle is performed. If HSYNC is high, it is always a refresh cycle and state S0 changes to S8 in order to generate a RAM refresh. This cycle lasts always two 2xCCLK cycles.

If HSYNC is low, state S0 changes over S1 to S2.

After this the DT- signal is monitored: the low-state indicates a data transfer cycle which is then terminated by state Sx.

If DT- is high a transition to S3 occurs which monitors the DBIN- signal. If DBIN- is low, it is a drawing cycle (Read-Modify-Write) and states S4, S5, S6, S7, are entered successively.

A high signal on DBIN- indicates that it was a display cycle which is irrelevant in the dual port memory mode. This forces the state machine to enter state S0 and this cycle is terminated.

The PAL is clocked with 10 MHz to be synchronus to the μPD72020 running on 5 MHz. The MUX signal is externally delayed for a half clock cycle in order to switch the row- and column addresses between the falling edges of RAS- and CAS-.

2.2 The Video Parameters

It is important to note that for this board the calculation of the horizontal video parameters (HFP, HS, HB, AW; see figure 2) is different from normal GDC and standard dynamic memory systems. For the GDC-VRAM solution no fixed coupling of GDC clock and pixel frequency is necessary.

Normally the GDC 2xCCLK is the Pixel clock divides by 8. The reason for this is that every two 2xCCLK cycles the display address is incremented.

$$\text{Time to increment the display address} = 2x \frac{1}{2xCCLK}$$

$$\text{Time to display 16 pixel} = 16 \times \frac{1}{\text{Pixel clock}}$$

As these two times must be equal the following relation applies:

$$2xCCLK = \frac{\text{Pixel clock}}{8}$$

This is only required with normal DRAMs, however for VRAMs a display address is output during the Data Transfer cycle at the beginning of a scan line. This address is calculated as follows:

actual display address — AW + PITCH

The actual display address is the last value of the display address counter before the blanking period (last word + 1 of the previous line) or in other words: The start address of the previous line + AW.

Therefore the display address at the start of a scan line is only dependent on the value of PITCH and not on AW.

AW (Active Words) defines the number of words displayed during the active display time (non-blanking period).

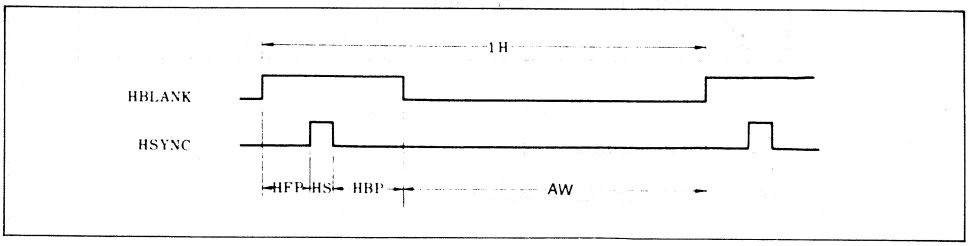


Figure 2. Generation of the Video Timing

The above timing and the pixel frequency depend on the utilized monitor. With a pixel clock of 20 MHz and the fixed relationship of GDC clock and pixel clock, the GDC would be forced to run only at 2.5 MHz which is very slow. But with the doubled value for AW and the other video parameters the GDC can be clocked on 5 MHz by generating the same video timing.

In this case the value of the display address counter does not match with the actually displayed word within the scan line. This is of no significance.

The only problem is that if the 8 lower bits of the display address counter are all zero the GDC will generate a new Data Transfer cycle.

This is important because normally the serial shift registers of μPD41264 VRAMs are empty in this case. With AW = 80 and PITCH = 64 in this design the GDC generates the following data transfer cycles:

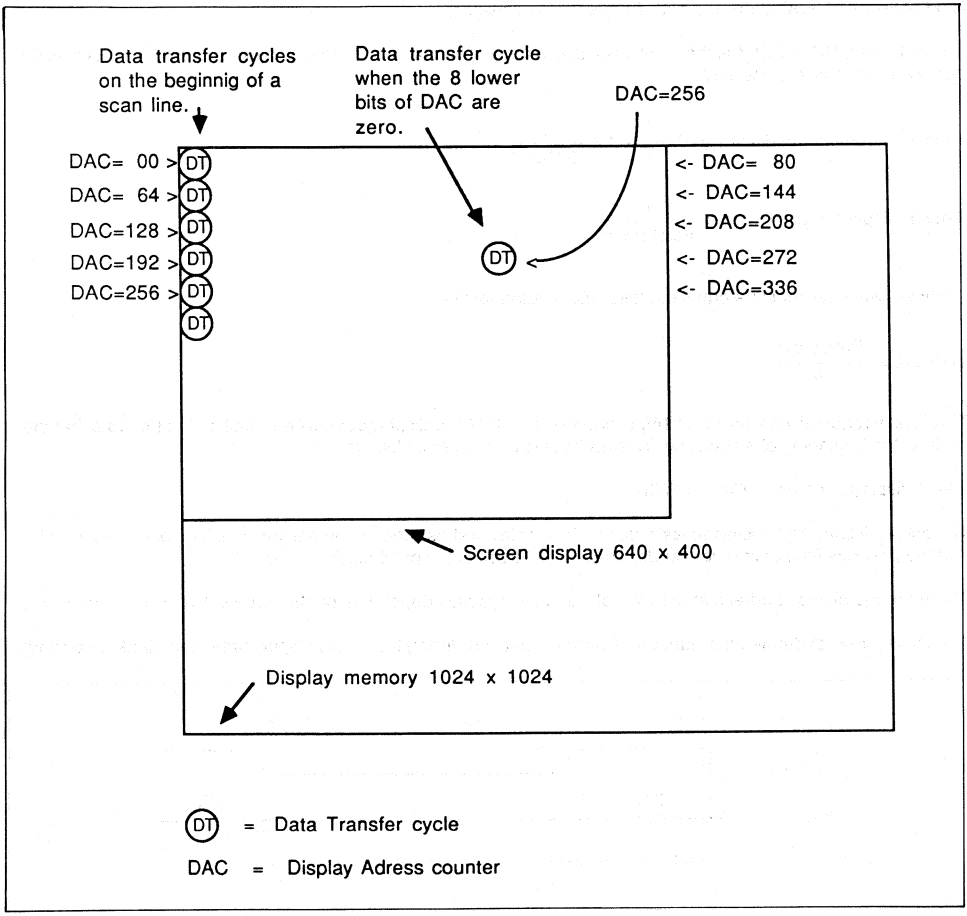


Figure 3. Generation of Data Transfer Cycles with AW = 80 and PITCH = 64

The Data Transfer cycle in the middle of the fourth scan line must be omitted in order to avoid a transfer with the wrong display address. (Otherwise the contents of the next scan line would be transferred.) The PAL takes care of this by monitoring the CLR- signal for the shift registers.

This allows DT cycles only at the beginning of a scan line. As described previously this DT cycle is unnecessary because the serial shift registers of the VRAMs are clocked with the half frequency and are therefore not empty.

2.3 Miscellaneous

The plane selection is done with the CAS- signals. The RAS-, WE-, OE-/DT- signals are the same for all memory planes. The memory refresh is done as RAS- only refresh and for all planes in parallel. For drawing operations the plane selection is done with the address lines A16 and A17. The CAS- signal enables the two bit binary decoder U9A which drives one of the three plane CAS- signals RCAS-, GCAS- or BCAS-. The CAS_ALL- signal is high in this case.

For a DT cycle all planes should be accessed in parallel and therefore CAS_ALL signal goes active together with CAS- in order to enable all the planes.

The 12 μPD41264 form a 1024 x 1024 x 3 planes display memory system. The three planes correspond to the three colour signals RGB of a TTL monitor. The display resolution is 640 x 400, but can be easily changed by software. 16 bits of display data are output in parallel with the rising edge of SC (Shift Clock) from each plane and then loaded byte-wise into the shift registers 74LS166.

This interleaved load operation is controlled by signals LOAD-, S0E1- and S0E2-. During the horizontal and vertical retrace period the shift registers are cleared by the delayed BLANK signal of the μPD72020.

This signal is delayed for five 2xCCLK cycle to allow the GDC to perform a DT cycle and to load the shift registers before the display starts.

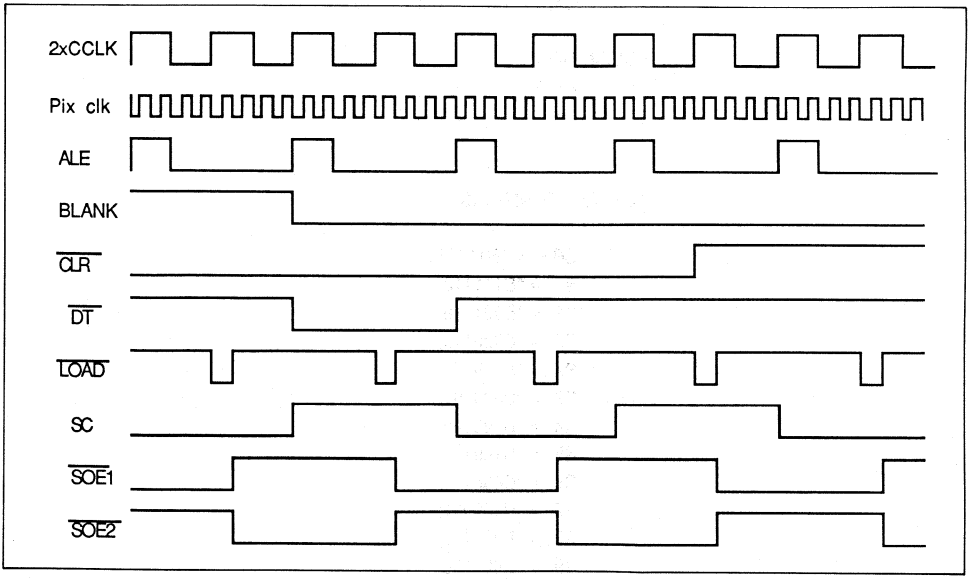


Figure 1. Interleaved Load Operation of the Shift Registers and DT-Cycle at the Beginning of a Scan Line.

3. Appendix A

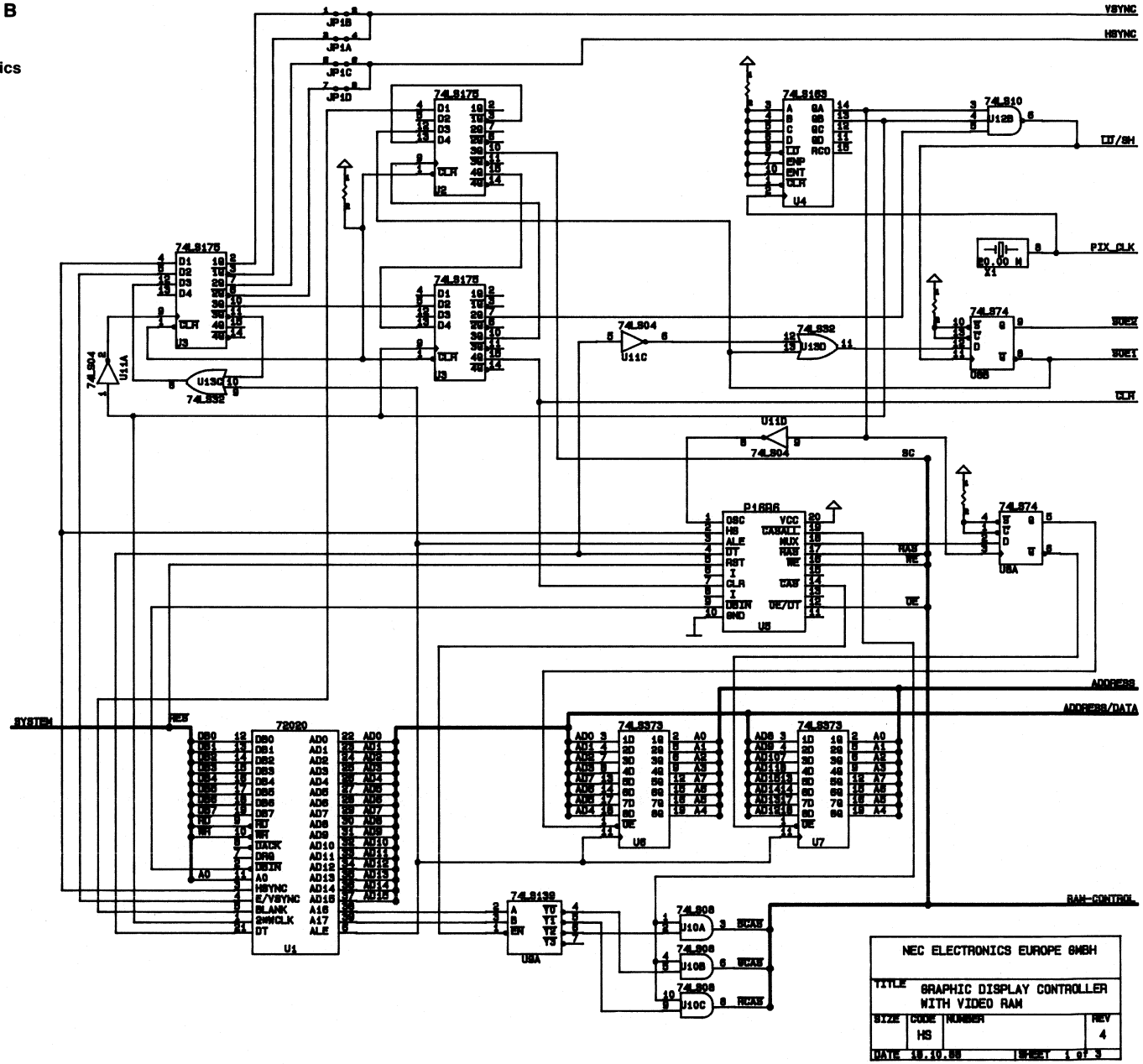
PAL Source Code

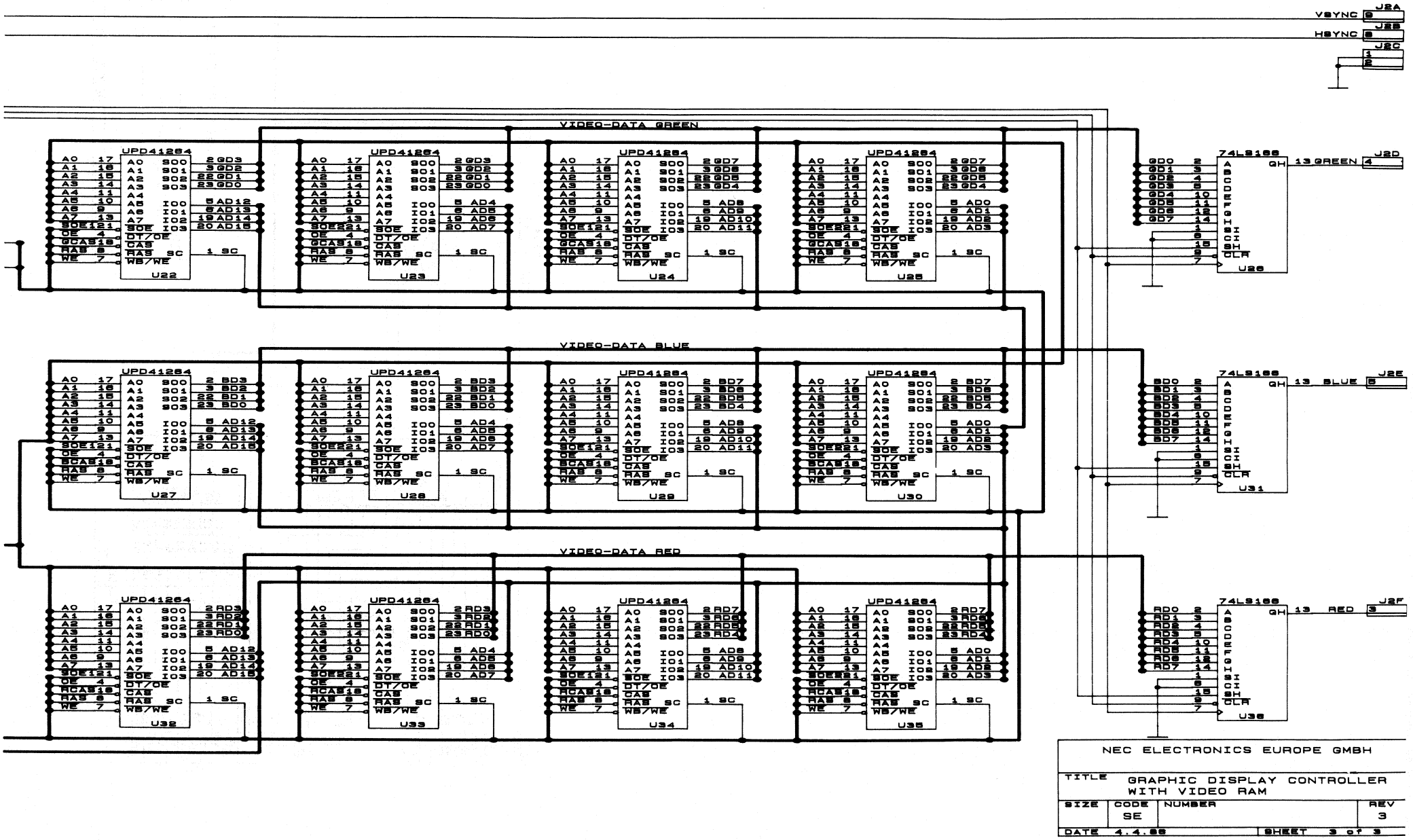
```
module vram
title 'Video RAM controller
EB-72020 Board
NEC electronics 02.08.1988'

        IC1     device  'P16R6';
"CLOCK
        OSC             pin     1;
"INPUT SIGNALS
        HS, ALE, DT     pin     2,3,4;
CLR pin 7;
        DBIN            pin     9;
        RST             pin     5;

"OUTPUT SIGNALS
        MUX,RAS,WE      pin     18,17,16;
        CAS,H1          pin     14,13;
        H2              pin     15;
        OE_DT           pin     12;
        CAS_ALL         pin     19;

"DEFINITIONS
        X = .X.;
        CK = .C.;


"STATE DEFINITIONS

        SRST = ^B111111;
        S0  = ^B111110;
        S1  = ^B001110;
        S2  = ^B000110;
        SX  = ^B100101;
        S3  = ^B000100;
        S4  = ^B100100;
        S5  = ^B100111;
        S6  = ^B000111;
        S7  = ^B100011;
        S8  = ^B011110;
        S9  = ^B001100;
```

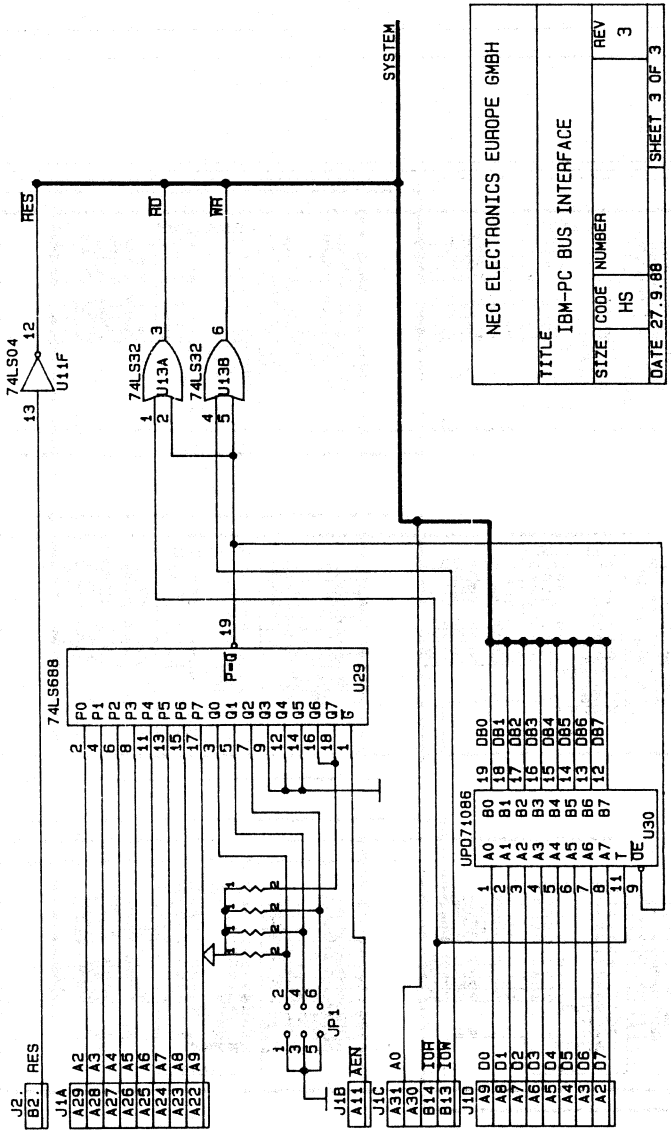
```
STATE_DIAGRAM [MUX,RAS,CAS,WE,H1,H2]

        STATE SRST:  IF (RST == 0) THEN SRST ELSE S0;

        STATE S0:    CASE  (ALE & !HS == 1)        : S1;
                           (RST == 0)          : SRST;
                           (ALE == 0)            : S0;
                           (ALE & HS == 1)                : S8;
                     ENDCASE;

        STATE S1:    IF (RST == 0) THEN SRST ELSE S2;

        STATE S2:    CASE  ( DT == 1)            : S3;
                           ( DT == 0)            : SX;
                           (RST == 0)            : SRST;
                     ENDCASE;

        STATE S3:    CASE  ( DBIN == 1)          : S0;
                           ( DBIN == 0)          : S4;
                           (RST == 0)            : SRST;
                     ENDCASE;

        STATE S4:    IF (RST == 0) THEN SRST ELSE S5;

        STATE S5:    IF (RST == 0) THEN SRST ELSE S6;

        STATE S6:    IF (RST == 0) THEN SRST ELSE S7;


        STATE S7:    IF (RST == 0) THEN SRST ELSE S0;

        STATE S8:    IF (RST == 0) THEN SRST ELSE S9;

        STATE S9:    IF (RST == 0) THEN SRST ELSE S0;

        STATE SX:    IF (RST == 0) THEN SRST ELSE S0;

EQUATIONS
        OE_DT    = ! ( !DT & !H2 & !CLR) & DBIN ;
        CAS_ALL =    DT # CAS;
end vram
```

4. Appendix B

Board Schematics




VSYNC
HSYNC
J2A
J2B
J2C
VIDEO-DATA GREEN
VIDEO-DATA BLUE
VIDEO-DATA RED
UPD41264
74LS166
J2D
J2E
J2F
U22
U23
U24
U25
U26
U27
U28
U29
U30
U31
U32
U33
U34
U35
U36
NEC ELECTRONICS EUROPE GMBH
TITLE
GRAPHIC DISPLAY CONTROLLER
WITH VIDEO RAM
SIZE
CODE
SE
NUMBER
REV
3
DATE
4.4.88
SHEET 3 of 3


NEC ELECTRONICS EUROPE GMBH
TITLE
IBM-PC BUS INTERFACE
SIZE
CODE
NUMBER
REV
3
HS
DATE 27.9.88
SHEET 3 OF 3
SYSTEM
RES
RD
WR
74LS04
U11F
74LS32
U13A
74LS32
U13B
74LS688
U29
P=Q
UPD71086
U30
OE
JP1
J2.
B2.
RES
J1A
A29
A28
A27
A26
A25
A24
A23
A22
A2
A3
A4
A5
A6
A7
A8
A9
J1B
A11
AEN
J1C
A31
A30
B14
B13
A0
IOR
IOW
J1D
A9
A8
A7
A6
A5
A4
A3
A2
D0
D1
D2
D3
D4
D5
D6
D7

V25 Vs. μPD72001 Interface

Contents:

Author: D. Cobbs
NEC Electronics USA

Related Documentation

User's Manual
μPD70320/70330/79011
Data Book
Microprocessors and Peripherals
Product Description
μPD7201/7201A/72001/7210/72105

Related Products

μPD70320	16 Bit Microcomputer	CMOS
μPD703305	16 Bit Microcomputer	CMOS
μPD72001	AMPSC Controller	CMOS

1. Introduction

The V25 has two independent DMA channels that provide fast service to external peripherals. The DMA controller has 20-bit source and destination memory addresses to allow access to the entire 1 megabyte address space.

Each DMA channel has an active high DMA request (DMARC) input, an active low DMA acknowledge (DMAAK*) output, and a terminal count (TC*) output. Each channel also has an associated interrupt that is generated when the specified number of transfers has been completed.

The V25's DMA controller is well suited for applications such as high speed serial communications. As an application example, the interface between the V25 and the μPD72001 advanced multiprotocol serial communications controller will be demonstrated in detail.

2. Memory-to-Memory DMA

The V25 has two memory-to-memory transfers modes: the burst and the single-step modes. In the burst mode, a rising-edge on the DMARQ line causes DMA transfers to begin and continue until the specified number of transfers is complete. In the single-step mode, a rising-edge on the DMARQ line causes execution of one instruction and one DMA transfers to be repeated alternately until the specified number of transfers is complete. Memory-to-memory DMA can also be initiated by the TDMA bit in the DMA mode registers (see DMA Control Registers).

Each memory-to-memory DMA transfer is performed in two consecutive cycles: a read from the source address, followed by a write to the destination address. The number of wait states for each access is determined by the programmable wait-state controller.

Aside from normal memory interfacing circuitry, little or no additional hardware is required when using memory-to-memory DMA.

3. Transfers between Memory and I/O

There are two modes for DMA between memory and I/O: the demand release and the single transfer modes. In the demand release mode, DMA transfers continue to execute as long as the DMARQ line is high. In the single transfer mode, each rising edge of DMARQ initiates one DMA transfer.

The V25's DMA controller utilizes the "Fly-by" method of transfer for DMA between memory and I/O. With this method each, DMA transfer requires only one access cycle. The controller provides all of the bus control signals for the memory address and DMA acknowledge (DMAAK*) pulse. The IOSTB* signal is not asserted. The I/O device must utilize the DMAAK* pulse to derive its chip select and any other necessary strobes to execute the data transfer. If necessary, the inverse of the R/W* line can be used by the I/O circuitry to determine the direction of the transfer. For example, normally when IOSTB* is asserted and R/W* is high, this condition corresponds to an I/O read cycle. However, if DMAAK* is asserted and R/W* is high, this condition corresponds to a memory to I/O DMA cycle, or equivalently, an I/O write. The circuitry required to interface I/O to the DMA controller is relatively simple and is the focus of the second part of this application note.

Note: An asterisk (*) indicates active low.

DMA Control Registers

DMA Service Channels

The DMA sercice channels are used to be specify the transfer source, destination and count used for DMA transfer and mapped into internal RAM. DMA service channel 0 is assigned to address XXE00—XXE07H, and channel 1 is assigned to XXE08—XXE0FH (XX is the value specified by the IDB register). Macro service channels 0, 1, and register bank 0 are als assigned to the same internal RAM area.

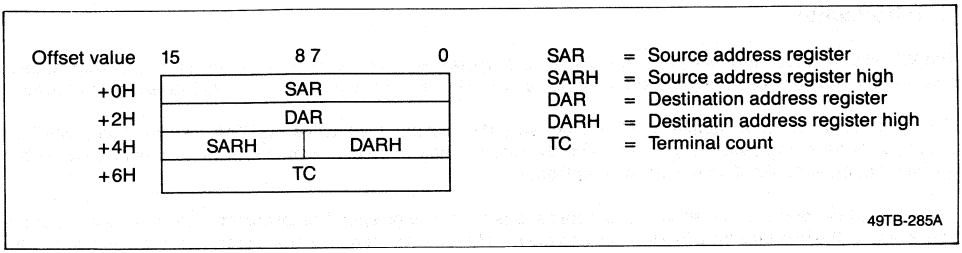


Figure 1. DMA Service Channel Format

The DMA controller generates a 20-bit physical source address by offsetting SARH 12 bits to the left and then adding it to SAR. This same procedure is also used to generate physical destination addresses using DARH and DAR.

DMA Mode Registers

The DMA mode registers select the mode of operation for a given DMA channel. The trigger DMA bit (TDMA) can be used to initiate memory-to-memory DMA via software. The enable DMA (EDMA) bit is cleared when the terminal count value reaches zero.

DMA Control Registers

These registers allow the user to specify how the source and destination memory address are updated.

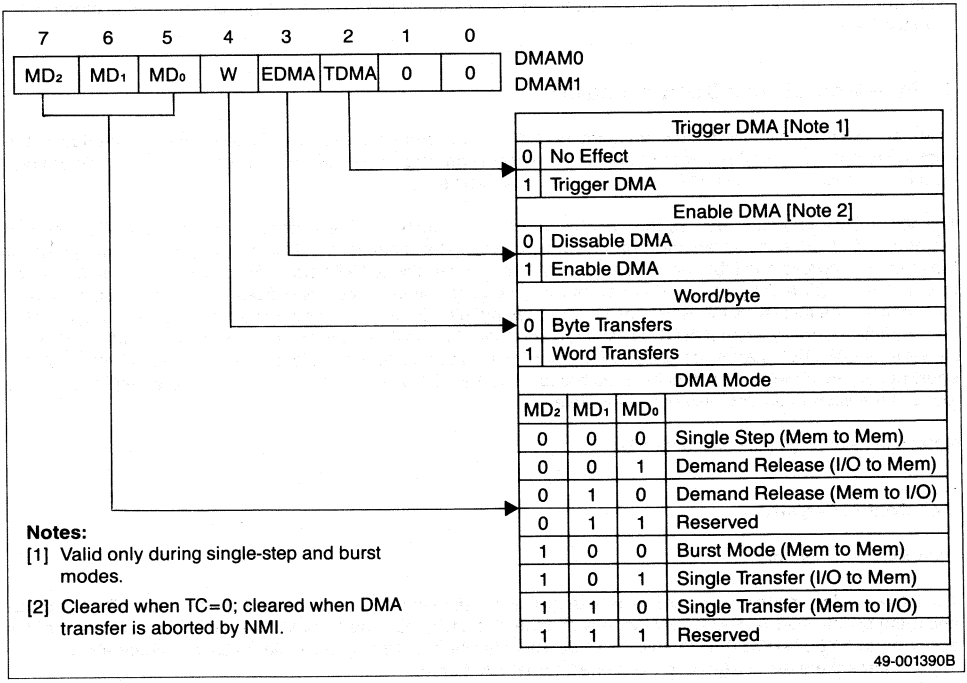


Figure 2. DMA Mode Registers (DMAM)

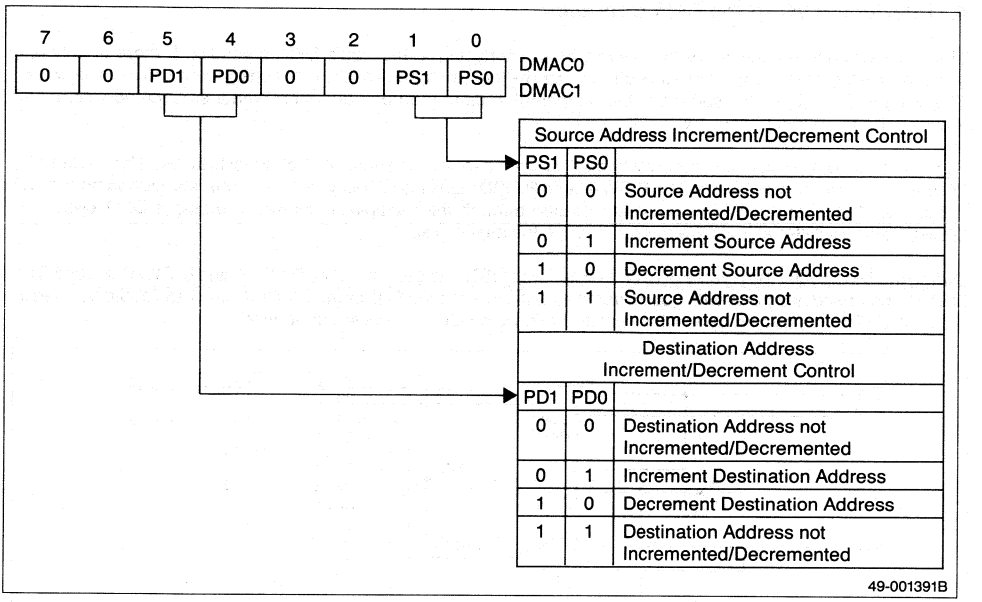


Figure 3. DMA Control Registers (DMAC)

DMA Latency

When a DMA request is issued, the V25 must first complete the instruction that is currently executing before the transfer can begin. Therefore, this time (N clocks) must be added to the overall latency. The latency for each mode of transfer is as follows:

Mode	**Clocks**
Burst	29 + N
Single-step	29 + N
Demand release	29 + N
Single transfer	31 + N

Transfer Rate

A certain number of CPU clocks is required for each DMA transfer. The times for each transfer mode are listed below:

Transfer Mode	**Byte Transfer**	**Word Transfer**
Burst	12 + WR + WW	16 + 2WR + 2 WW
Single-step	20 + WR + WW + N	24 + 2WR + 2 WW + N
Single transfer	34 + W + N	36 + 2W + N
Demand release	15 + W	17 + 2W

WR = number of wait states in read cycle
WW = number of wait states in write cycle
W = number of wait states in memory of I/O whichever is greater

4. Interfacing I/O to the DMA Controller

There are two alternatives to be considered when interfacing to the V25's DMA controller. If there is only one peripheral that requires DMA, one DMA channel can be used for reads and the other channel can be used for writes. This will provide full duplex service to the peripheral. If more than one peripheral requires DMA service, a half duplex scheme will be necessary.

Figure 4 is an example of a half duplex scheme. This circuit allows the same DMA channel to be used for memory-to-I/O and I/O-to-memory transfers. The PIOWR* and PIORD* Lines go directly to the peripherals. During normal I/O cycles the IORD* and IOWR* signals are passed through the multiplexer. However, during a DMA cycle, the DIORD* and DIOWR* signals are passed through the multiplexer.

During the DMA cycle, the inverse of R/W* is gated with MSTB* to generated the DIORD* and DIOWR* strobes. The MSTB* line provides a pulse with good symmetry, but either the MREQ* or the DMAAK* line itself could be used in place of MSTB* depending upon what specific timing is needed for a given peripheral.

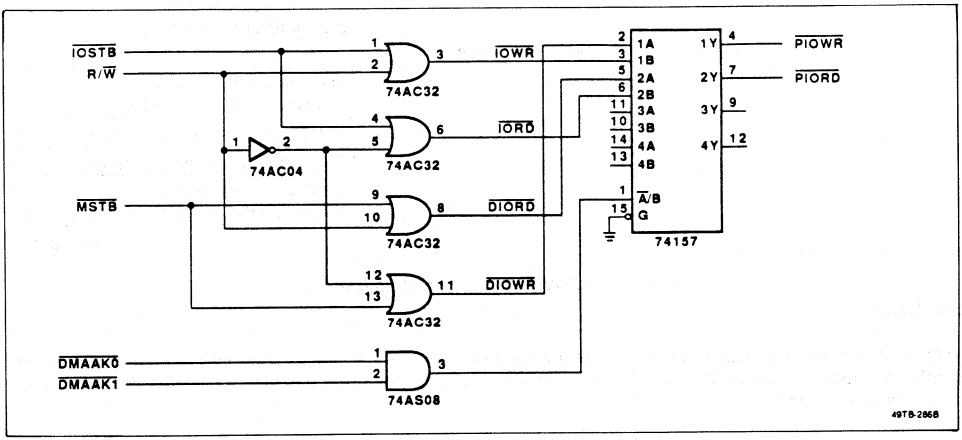


Figure 4. Half Duplex DMA Interface

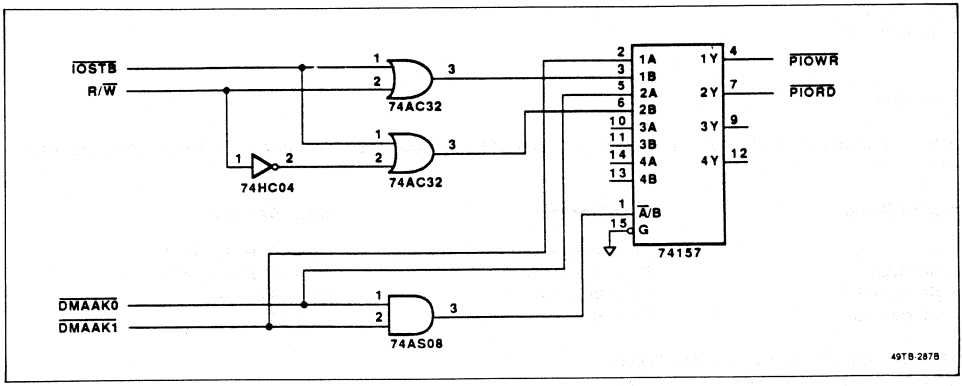


Figure 5. Full Duplex DMA Interface

In a typical system, the IORD* and IOWR* signals would go to all peripherals in the system while the PIORD* and PIOWR* lines would go only to the devices using DMA. The "P" signals could also be used as the system's I/O read/write strobes, but then the μPD74157 adds an unnecessary delay to the entire system. If chip select logic is required for the I/O device(s) then the appropriate DMAAK* line must be logically or'ed with the chip select signal so that the device is selected during DMA cycles.

If only one peripheral device requires DMA, a full duplex scheme can be used as shown in figure 5. The non-DMA I/O read and write are generated in the same way as the previous circuit. However, during a DMA cycle, the appropriate DMAAK* signal becomes the read or write strobe to the peripheral.

5. Interfacing the V25 to the μPD72001

The V25/μPD72001 interface shown in figure 6 is an example of a full duplex configuration. The μPD72001 does not have a chip select input, so the I/O read/write strobes must be qualified with a chip select input [see Static Memory I/O Interfacing Application Note for address decoding schemes]. A0 controls the C/D* (control data) line during regular I/O cycles. This line is driven to ground by the multiplexer during DMA cycles. The CS* signal is the μPD72001's select line which comes from the address decoding hardware.

The clock to the μPD72001 is provided directly from the V25 clockout (8 MHz in this example). Since the μPD72001 has its own internal baud rate generator, no additional clock circuitry is required.

The μPD72001 can be programmed to provide interrupts corresponding to numerous different internal status events. The INT* output from the μPD72001 can be tied directly to the V25's INTR line. Similarly, the V25's INTAK* can be tied directly to the μPD72001's INTAK* input. This setup will allow the μPD72001 to request interrupts and provide a vector number that depends on the source of the request.

DMA request lines are output from the μPD72001 for transfer buffer empty and receive buffer full; these lines can be input directly to the V25 DMA request input. Control registers CR1 and CR2 in the μPD72001 must be initialized properly to enable this function.

6. DMA Timing Analysis

Since V25 DMA cycles between memory and I/O occur within one bus cycle, when evaluating DMA timing, it is necessary to coordinate memory and I/O access timing. To better illustrate this situation, a detailed timing study of the V25/μPD72001 interface using μPD42356-15 SRAM will be discussed.

I/O to Memory Case

To perform I/O to memory DMA, I/O read and memory write cycles must be timed together. During these cycles, the V25 extends the DMAAK* pulse beyond the rising edge of MREQ and MSTB to provide data hold time for the memory. The following timing parameters are important to this interface.

tDW = memory device's data set-up time before the rising edge of write strobe
tWMRL = MREQ width low [1]
tIORD = delay from DMAAK* to IORD at μPD72001; delay caused by DMA interface circuitry
tDWR = delay from MREQ to write strobe low at μPD43256
tOEIO = data output access time of I/O device

A typical way that memory read/write strobes are generated is by using the same circuit that generates IORD* and IOWR* in figure 4, except that MREQ is used in place of IOSTB.

Once MREQ* goes low, the I/O device must put data on the bus before the tDW rising edge of MREQ*. The delays of the memory and DMA interface circuitry must also be accounted for. Given these parameters, the following equation can be derived:

Equation 1

tWMRL = tDW + tOEIO + tIORD

In most cases tIORD will be much longer than tDWR. Since MREO* and DMAAK* both go low at the same time, tIORD and tDWR overlap. Therefore, only the longer of these two specs is important, hence, only tIORD is used in the equation.

For this particular application, the following parameters are given:

tOEIO = 120 ns (for μPD72001)
tDW = 60 ns (μPD43256-15)
tWMRL = 220 ns (8 MHz, 1 wait state)

Therefore, to determine how fast the DMA interface must be (the combined delay of the 7408 and 74157),

tIORD = tWMRL — tDW — tOEIO
= 220 — 60 — 120 ns
= 40 ns

Note: 1. The value of tWMRL has been improved since the September 1987 data sheet.
tWMRL = (n+1)
T-30ns where "n" is the number of wait state and T is the CPU clock period.

The above timing equations assume that the memory device does not qualify addresses in its decoding logic, i.e. MREQ is not used to qualify address decoding and all decoding logic is performed within tDMAR [see figure 7]. If this were not the case, then the memory address decode time would have to be added to the right side of equation 1.

Memory to I/O Case

In the case of memory to I/O DMA timing, memory read cycles must be timed to coincide with I/O write cycles. The timing must be arranged so that the memory device has time to output data within the write set-up time constraints of the I/O device.

The following parameters are critical in this case:

tACC = memory access time + delay of address decode logic
tSDW = data set-up time to rising edge of I/O write strobe signal
tDAMR = MREQ delay from valid address = 0.5T — 30 nsecs
tWMRL = 220 ns [8 MHz, 1 wait state]

The following equation can be used to characterize this interface:

Equation 2

tWMRL [+ tDAMR] = tACC = tSDW

tDAMR is added to the left side of this equation when address decode logic is not qualified with MREQ. If the decode logic is qualified, the tDAMR term should be dropped from the equation. The delay of the DMA interface logic is not a factor in this equation because the memory access time will overlap this time and will usually be substantially longer.

The μPD72001's tSDW is specified as 120 ns. Since tACC and tSDW have been given in this example, the necessary duration of tWMRL will be calculated. Assuming infinitely fast non-qualified address decoding:

tWMRL = tACC + tSDW — tDAMR
= 150 + 120 — 33
= 240 ns

For tWMRL to be 240 ns or greater with the V25 operating at 8 MHz, 2 wait states must be inserted. This can easily be achieved by programming the V25's wait state controller. Unfortunately, it is not possible to program the controller for a different number of wait states on read cycles vs write cyles. Therefore, to implement the full duplex V25/μPD72001 interface, 2 wait states must be programmed for all DMA transfers.

7. Demonstration Software

Attached to this application note are two versions of a demonstration program for the V25/μPD72001 interface. This software was written to execute on the DDK-70320 (V25 evaluation board). The program in appendix 1 was assembled by the V20/V30 assembler (RA70116) and the program in appendix 2 was assembled by the V25 assembler (RA70320). The V25 assembler recognizes the names of special function registers and many other reserved memory locations. The V20 assembler has an appearance like that of other third party V-Series and 8086 assemblers currently available.

This program initializes the μPD72001 for 19.2K baud asynchronous operation. The V25's DMA controller is then initialized and memory to I/O DMA commences.

While the DMA is functioning, the μPD72001 outputs a message that can be observed on a terminal. A background program is executing on the V25 that performs PUSHR and POPR instructions. These instructions are the longest in the V25's instruction set in terms of execution time. Therefore, when these instructions are being executed, the worst case of DMA latency is exhibited.

The DMA interrupt is initialized for context switching. When the terminal count interrupt occurs, the CPU traps to register bank 6 and the interrupt service routine restarts the DMA process.

8. V25/μPD72001 Applications

The combination of the V25 and the μPD72001 creates a powerful communications system. The V25's processing power and the μPD72001's multiprotocol abilities make them an excellent choice for a wide range of communications applications.

One typical application is a communications controller or multiplexer. In this use, the serial channel of the V25 and one of the channels for the μPD72001 channel is used for asynchronous ports to terminals or PCs. The other μPD72001 channel is used for the synchronous connection to another multiplexer of the host computer. The DMA capabilities of the V25 allow the synchronous link to run at high speed for lower response time and high throughput.

Another application is in packet switched network equipment. The HLDC capabilities of the μPD72001 allow the V25 to run X.25 LAPB protocols. The V25/μPD72001 combination could be used as a packet assembler/disassembler (PAD), where one of the synchronous channels of the μPD72001 is used in HDLC mode to connect to the X.25 network. The other channel for the μPD72001 and the serial channel of the V25 are used for asynchronous interfaces to terminals of PC's. The frame handling capability of the μPD72001 and the processing power of the V25 give high packet throughput.

A packet switch is another X.25 application. The V25/μPD72001 combination moves data quickly. The true 16 bit power of the V25 allows even the most complicated routing algorithms to be completed quickly.

Integrated Services Digital Network (ISDN) is an emerging technology that is a perfect match for the V25 and the μPD72001. Because the 2B+D interface can be managed in a simple and cost effective way with this combination. In a typical ISDN terminal or Integrated Voice/Data Terminal (IVDT) the V25 would control the operation of the μPD72001, a CODEC for voice data, and the Digital Line Interface Controller (DLIC). The μPD72001 uses one channel for one of the 64 Kbit/second B channels and the other channel for the 16 Kbit/second D signaling channel. The additional B channel is used for the digitized voice data from the CODEC. The V25 coordinates the operation of the terminal and handles data transfer.

The V25 and μPD72001 combination is a very powerful one. There are few communications tasks that it cannot handle. The great processing power of the V25 and the comprehensive protocol support of the μPD72001 create a team that is difficult to match.

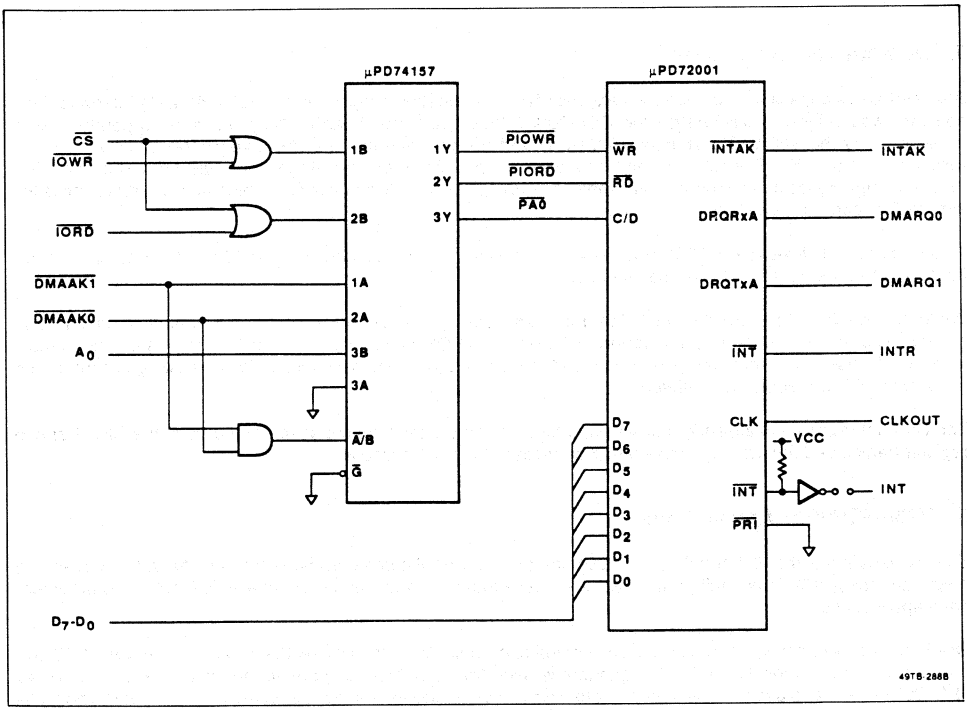


Figure 6. V25/μPD72001 Interface

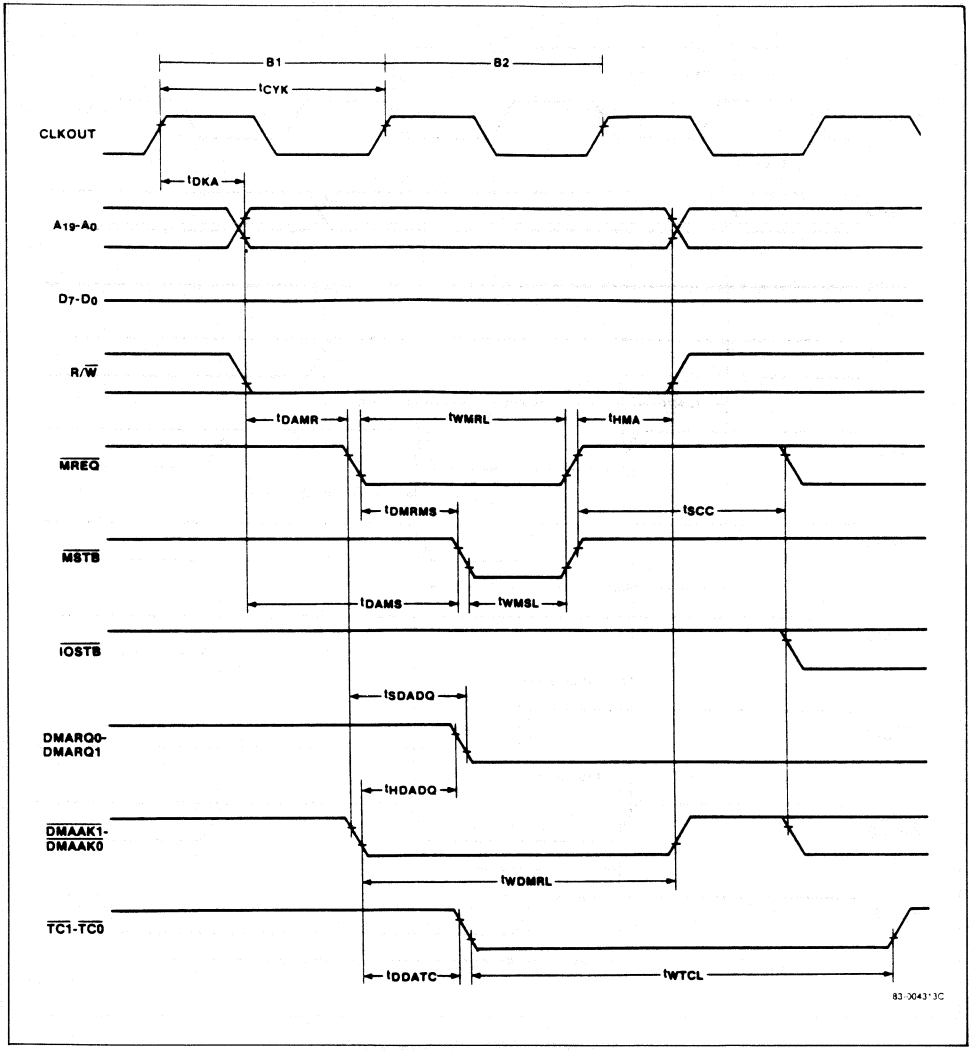


Figure 7. DMA, I/O to Memory

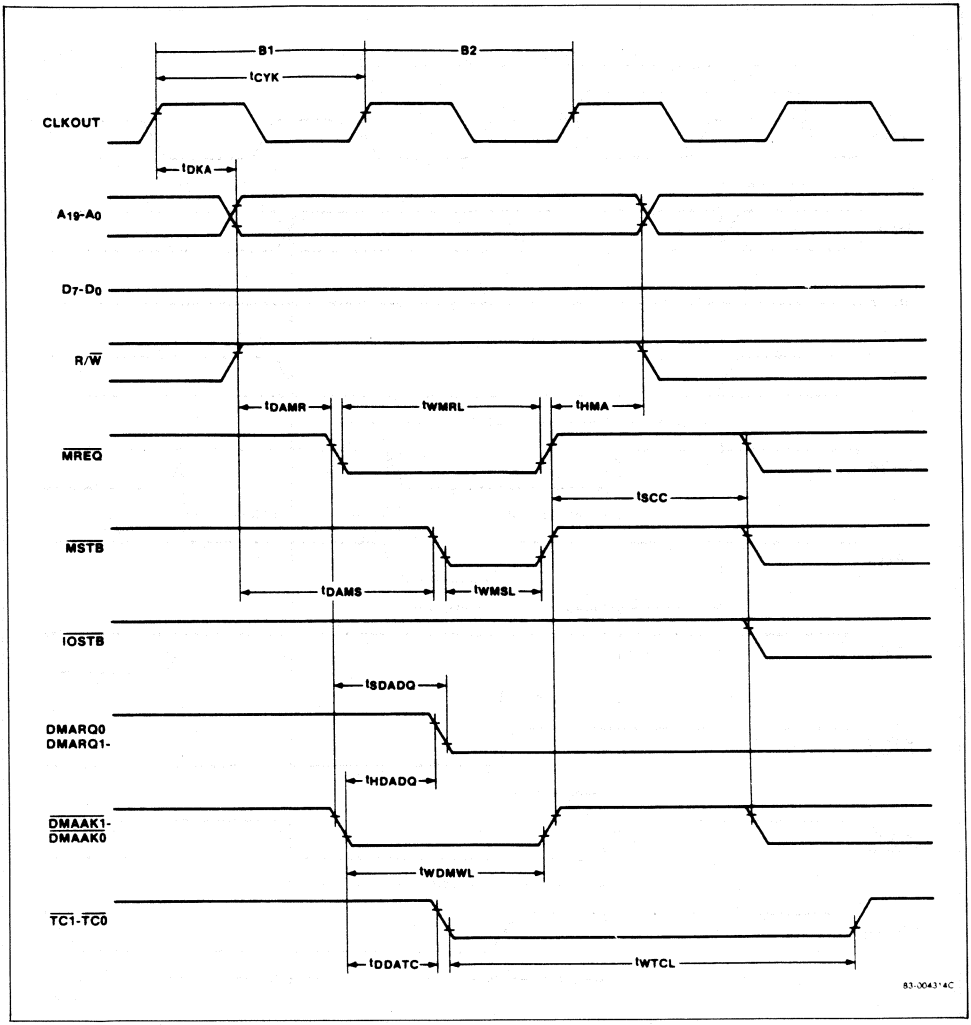


Figure 8. DMA, Memory to I/O

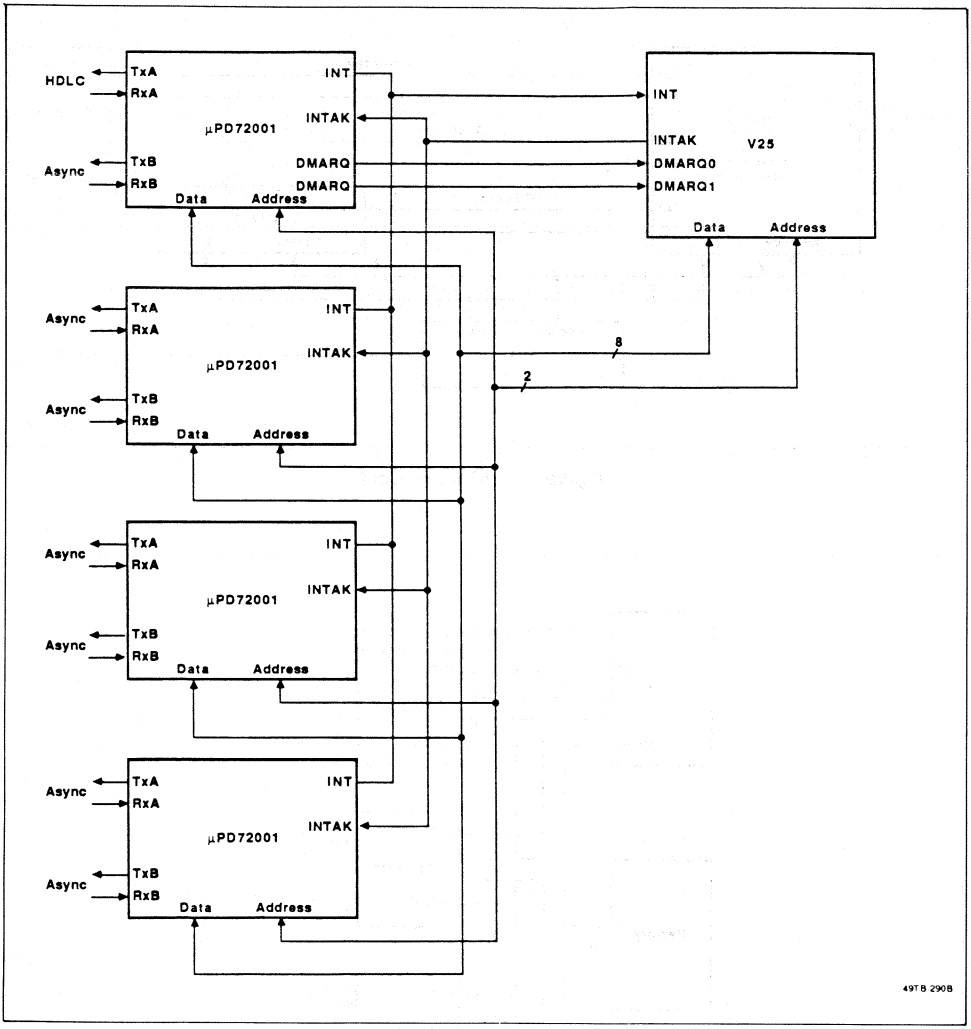


Figure 9. Multiplexer

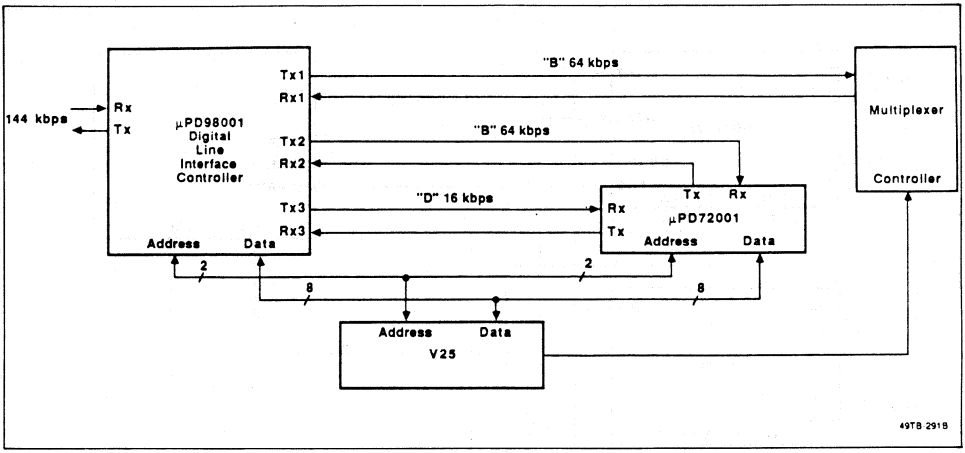


Figure 10. ISDN Line Card

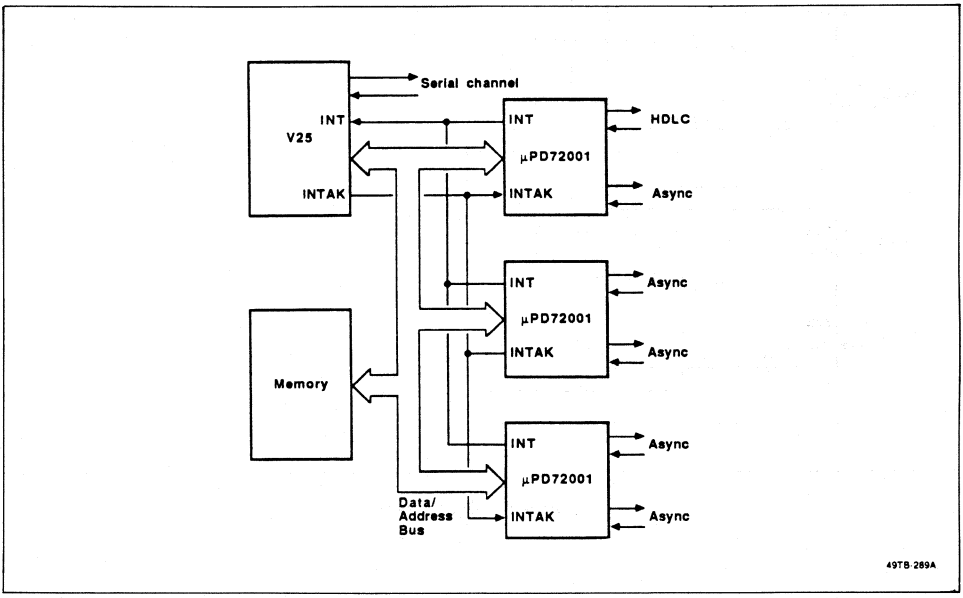


Figure 11. X.25 PAD

Appendix 1

V25/μPD72001 demonstration software assembled under RA7016.

```
UPD70108/70116 ASSEMBLER V1.0    DMA7

SOURCE FILE:  dma72x.asm
OBJECT FILE:  dma72x.rel (DMA7)
COMMAND    :  ra70116 dma72x.asm


LOC  OBJ                  LINE    SOURCE

                             1    ;*****************************************************************
                             2    ;
                             3    ;  This program tests the V25-72001 DMA interface.
                             4    ;
                             5    ;                       written by D. Cobbs
                             6    ;
                             7    ;*****************************************************************
                             8
                             9    NAME DMA7
                            10    ;
                            11    SFR_SEGMENT AT 0F700H
                            12    ;
                            13    DATA72  EQU     0FFFCH
                            14    CMD     EQU     0FFFDH
                            15    ;
                            16    ORG 0E00H
                            17    SAR1    DW      ?
                            18    ORG OFFSET ($ +3)
                            19    SARH    DB      ?
                            20    TC      DW      ?
                            21    ORG 0EC2H
                            22    VPC     DW      ?       ;VPC for rb6 @ XXEC2H
                            23    ORG 0ECBH
                            24    DSEG0   DW      ?       ;Data seg0 for rb6 @ XXEC8
                            25    ORG 0ECCH
                            26    PSEG    DW      ?       ;RB6 pseg XXECC
                            27    DSEG1   DW      ?       ;RB6 dseg1 XXECE
                            28    ORG 0EDAH
                            29    DM6     DW      ?       ;RB6 DW @XXEDA
                            30    ORG     0F12H
                            31    PMC2    DB      ?
                            32    ORG 0FA2H
                            33    DMAC1   DB      ?
                            34    DMAM1   DB      ?
                            35    DMS     DB      0ACH
                            36    DIC0    DB      ?
                            37    DIC1    DB      ?
                            38    SFR     ENDS
                            39    ;
                            40    ;
                            41    CODE_SEGMENT AT 00A0H
                            42    ASSUME FS:CODE,DS0:CODE, DS1:SFR
                            43    ;
                            44    ; All segment registers are initialised by the DDK monitor.
                            45    ; (Except for the ES register)
                            46    ;
                            47    START:  MOV     DW,CMD          IX,TABLE72
                            48            LDEA    IX,TABLE72
                            49            MOV     CW, TAB_COUNT
                            50

0000 BAFDFF
0003 8D368500
0007 B91A0090
```

```
UPD70108/70116 ASSEMBLER V1.0    DMA7

LOC  OBJ                    LINE     SOURCE

000B FC                       51             CLR1    DIR             ;DIR flag cleared for auto inc.
                              52     ;
000C AC                       53     INIT72: LDM     TABLE72 ;initialize 72001
000D EE                       54             OUT     DW,AL
000E E2FC                     55             DBNZ    INIT72
                              56     ;
0010 B800F7                   57             MOV     AW,0F700H
0013 8EC0                     58             MOV     DS1,AW          ; SET ES for SFR's
                              59     ; Initialize register bank 6 for interrupt service
                              60     ;
0015 26A3CE0E                 61             MOV     DSEG1,AW        ;INIT rb6's ES with ida addr.
0019 26C706C20E8700           62             MOV     VPC, OFFSET DMA_INT
0020 26C706CC0EA000           63             MOV     PSEG,00A0H
0027 26C706DA0EFDFF           64             MOV     DW6,CMD
                              65     ;
                              66     ; Initialize DMA channel and reg.'s
                              67     ;
002E 26C706080ECF0A           68             MOV     SAR1,(OFFSET MESSAGE + 0A00H)
0035 26C6060D0E00             69             MOV     SARH,0H
003B 26C7060E0E3F00           70             MOV     TC,LENGTH MESSAGE
0042 26C606A20F01             71             MOV     DMAC1,01
0048 26C606AC0F46             72             MOV     DIC0,46H  ;Set up cs int. for bank #6
004E 26C606AD0F16             73             MOV     DIC1,16H
0054 26C606A30FC8             74             MOV     DMAM1,0C8H
005A 26C606120F1F             75             MOV     PMC2,1FH
0060 BAFDFF                   76             MOV     DW,CMD
0063 B000                     77             MOV     AL,00
0065 EE                       78             OUT     DW,AL
0066 EE                       79             OUT     DW,AL           ;Get back to cr0
0067 B002                     80             MOV     AL,02           ;Point to cr02
0069 EE                       81             OUT     DW,AL
006A B066                     82             MOV     AL,66H
006C EE                       83             OUT     DW,AL   ;Set chA for DMA operation
006D B001                     84             MOV     AL,01
006F EE                       85             OUT     DW,AL   ;Set pointer to CR1
0070 B007                     86             MOV     AL,07   ;Enable DMA
0072 EE                       87             OUT     DW,AL
0073 B005                     88             MOV     AL,05H  ;Point to xmit enable reg
0075 EE                       89             OUT     DW,AL
0076 B068                     90             MOV     AL,68H
0078 EE                       91             OUT     DW,AL   ;Enable xmitter
0079 FB                       92             EI
007A EE                       93             OUT     DW,AL   ; Start xmit DMA
007B 60                       94     LAST:   PUSH    R
007C 60                       95             PUSH    R
007D 60                       96             PUSH    R
007E 60                       97             PUSH    R
007F 60                       98             PUSH    R
0080 61                       99             POP     R
0081 61                      100             POP     R
0082 61                      101             POP     R
0083 61                      102             POP     R
0084 61                      103             POP     R
0085 EBF4                    104             BR LAST
                             105     ;
                             106     ;    DMA Interrupt service routine
                             107     ;
0087 26C706080ECF0A          108     DMA_INT: MOV    SAR1, (OFFSET MESSAGE + 0A00H)
008E 26C6060D0E00            109              MOV    SARH,0H
0094 26C7060E0E3F00          110              MOV    TC,LENGTH MESSAGE
009B 32C0                    111              XOR    AL,AL
009D EE                      112              OUT    DW,AL
009E EE                      113              OUT    DW,AL     ;Reset pointer to 0
009F B005                    114              MOV    AL,05
00A1 EE                      115              OUT    DW,AL     ;Set 72001 addr. ptr to cr5
00A2 B060                    116              MOV    AL,60H    ;
00A4 EE                      117              OUT    DW,AL     ; Disable xmitter
```

```
UPD70108/70116 ASSEMBLER V1.0    DMA7

LOC  OBJ                        LINE     SOURCE

00A5 26C606A30FC8                118              MOV    DMAM1,0C8H ; Enable DMA
00AB B005                        119              MOV     AL,05
00AD EE                          120              OUT    DW,AL
00AE B068                        121              MOV    AL,68H
00B0 EE                          122              OUT    DW,AL     ;Enable xmitter
00B1 0F92                        123     FINT     DW     920FH
00B3 0F91                        124     RETRBI   DW     910FH
                                 125     ;
                                 126     ; 72001 Initialization data
                                 127     ;
00B5 00                          128     TABLE72          DB 00H,18H,02H,64H,04H,44H
00B6 18
00B7 02
00B8 64
00B9 04
00BA 44
00BB 0F                          129              DB 0FH,52H,03H,0C0H,05H,60H,0CH,02H,18H,00
00BC 52
00BD 03
00BE C0
00BF 05
00C0 60
00C1 0C
00C2 02
00C3 18
00C4 00
00C5 0C                          130              DB 0CH,01H,18H,00,0EH,07H,03H,0C1H,00,00
00C6 01
00C7 18
00C8 00
00C9 0E
00CA 07
00CB 03
00CC C1
00CD 00
00CE 00
00CF                             131     TABLE72_END      LABEL WORD
  001A                           132     TAB_COUNT EQU TABLE72_END - TABLE72
                                 133     ;
                                 134     ; Messages
                                 135     ;
00CF 0D                          136     MESSAGE          DB       0DH,'V25-72001 Demo/Test program',0DH,10,
00D0 5632352D373230303120
     44656D6F2F5465737420
     70726F6772616D

00EB 0D
00EC 0A
00ED 5472616E736D6974206F
     7065726174696F6E616C
     20776974682044D41
010A 0D
010B 0A
010C 0D
010D 0A
                                 137     & 'Transmit operational with DMA',0DH,10,13,10
                                 138     ;
----                             139     CODE ENDS
                                 140     END

 ASSEMBLY COMPLETE, 0000 ERROR(S) FOUND
```

Appendix 2

V25/μPD72001 demonstration software assembled under RA70320.

```
UPD70320 ASSEMBLER V1.0      DMA7
*

SOURCE FILE  : E:DMA72Z.ASM
OBJECT FILE  : E:DMA72Z.REL (DMA7)
COMMAND      : DMA72Z.ASM

LOC  OBJ                  LINE     SOURCE

                             1  ;*****************************************************************
                             2  ;
                             3  ; This program tests the V25-72001 DMA interface.
                             4  ;
                             5  ;
                             6  ;                       written by D. Cobbs
                             7  ;
                             8  ;
                             9  ;*****************************************************************
                            10  ;
                            11  NAME DMA7
                            12  ;
FFFC                        13  DATA72  EQU     0FFFCH
FFFD                        14  CMD     EQU     0FFFDH
                            15  ;
F700                        16  SFRLOC  EQU     0F700H  ; location of SFR area
                            17  ;
                            18  SFR SEGMENT AT SFRLOC
                            19  ORG 0E08H
0000                        20  SARL    DW      ?
0E08 ????                   21  ORG OFFSET ($ +3)
0E0A ??                     22  SAHM    DB      ?
0E0D ????                   23  TCR     DW      ?
0E0E ????                   24  SFR     ENDS
                            25  ;
                            26          ASGNSFR SFR
                            27  ;
                            28  ; Handy names for registers in Reg. Bank 6
00C2                        29  VPC     EQU     REGBANK.BK6.BVPC
00C8                        30  DSEG0   EQU     REGBANK.BK6.BDS0
00CC                        31  PSEG    EQU     REGBANK.BK6.BPS
00CE                        32  DSEG1   EQU     REGBANK.BK6.BDS1
00DA                        33  DW6     EQU     REGBANK.BK6.BDW
                            34  ;
                            35  CODE SEGMENT AT 00A0H
                            36  ASSUME PS:CODE,DS0:CODE, DS1:SFR
                            37  ;
                            38  ; All segment registers are initialized by the DDK monitor.
                            39  ; (Except for the DS1 register)
                            40  ;
0000 BAFDFF                 41  START:  MOV     DW,CMD
0003 8D36B500               42          LDEA    IX,TABLE72
0007 B91A0090               43          MOV     CW,TAB_COUNT    ;DIR flag cleared for auto inc.
000B FC                     44          CLR1    DIR
                            45  ;
000C AC                     46  INIT72: LDM     TABLE72 ;initialize 72001
000D EE                     47          OUT     DW,AL
000E E2FC                   48          DBNZ    INIT72
                            49  ;
0010 B800F7                 50          MOV     AW,0F700H
```

```
LOC  OBJ                      LINE      SOURCE

0013 8EC0                       51              MOV     DS1,AW          ; SET DS1 for SFR's
                                52      ; Initialize register bank 6 for interrupt service
                                53      ;
0015 26A3CE00           R       54              MOV     DSEG1,AW        ;INIT rb6's DS1 with IDA addr.
0019 26C706C2008700     R       55              MOV     VPC, OFFSET DMA_INT ; Init. RB6's VPC with int serv. routine addr.
0020 26C706CC00A000     R       56              MOV     PSEG,00A0H  ; INit. RB6's program segment register
0027 26C706DA00FDFF     R       57              MOV     DW6,CMD     ; Init RB6'S DW register to point to the 72001
                                58      ;
                                59      ; Initialize DMA channel and reg.'s
                                60      ;
002E 26C706080ECF0A             61              MOV     SAR1,(OFFSET MESSAGE + 0A00H)
0035 26C6060D0E00               62              MOV     SARH,0H
003B 26C7060E0E3F00             63              MOV     TC,MESSAGE_LEN
0042 26C606A20101       R       64              MOV     DMAC1,01
0048 26C606AC0146       R       65              MOV     DIC0,46H  ;Set up cs int. for bank #6
004E 26C606AD0116       R       66              MOV     DIC1,16H
0054 26C606A301C8       R       67              MOV     DMAM1,0C8H
005A 26C60612011F       R       68              MOV     PMC2,1FH ;Set proper port bits for control (DMA) mode
0060 BAFDFF                     69              MOV     DW,CMD
0063 B000                       70              MOV     AL,00
0065 EE                         71              OUT     DW,AL
0066 EE                         72              OUT     DW,AL           ;Get back to cr0
0067 B002                       73              MOV     AL,02           ;Point to cr02
0069 EE                         74              OUT     DW,AL
006A B066                       75              MOV     AL,66H
006C EE                         76              OUT     DW,AL   ;Set chA for DMA operation
006D B001                       77              MOV     AL,01
006F EE                         78              OUT     DW,AL   ;Set pointer to CR1
0070 B007                       79              MOV     AL,07   ;Enable DMA
0072 EE                         80              OUT     DW,AL
0073 B005                       81              MOV     AL,05H  ;Point to xmit enable reg
0075 EE                         82              OUT     DW,AL
0076 B068                       83              MOV     AL,68H
0078 EE                         84              OUT     DW,AL   ;Enable xmitter
0079 FB                         85              EI
007A EE                         86              OUT     DW,AL   ; Start xmit DMA
007B 60                         87      LAST:   PUSH    R
007C 60                         88              PUSH    R
007D 60                         89              PUSH    R
007E 60                         90              PUSH    R
007F 60                         91              PUSH    R
0080 61                         92              POP     R
0081 61                         93              POP     R
0082 61                         94              POP     R
0083 61                         95              POP     R
0084 61                         96              POP     R
0085 EBF4                       97              BR LAST
                                98      ;
                                99      ;     DMA Interrupt service routine
                               100      ;
0087 26C706080ECF0A            101      DMA_INT: MOV    SAR1, (OFFSET MESSAGE + 0A00H)
008E 26C6060D0E00              102              MOV    SARH,0H
0094 26C7060E0E3F00            103              MOV    TC,MESSAGE_LEN
009B 32C0                      104              XOR    AL,AL
009D EE                        105              OUT    DW,AL
009E EE                        106              OUT    DW,AL    ;Reset pointer to 0
009F B005                      107              MOV    AL,05
00A1 EE                        108              OUT    DW,AL    ;Set 72001 addr. ptr to cr5
00A2 B060                      109              MOV    AL,60H   ;
00A4 EE                        110              OUT    DW,AL    ; Disable xmitter
00A5 26C606A301C8       R      111              MOV    DMAM1,0C8H ; Enable DMA
00AB B005                      112              MOV     AL,05
00AD EE                        113              OUT    DW,AL
00AE B068                      114              MOV    AL,68H
00B0 EE                        115              OUT    DW,AL    ;Enable xmitter
00B1 0F92                      116              FINT            ;Finish internal interrupt
```

```
UPD70320 ASSEMBLER V1.0        DMA7

LOC  OBJ                       LINE      SOURCE

00B3 0F91                       117               RETRBI          ;Return from Reg Bank Int
                                118       ;
                                119       ; 72001 Initialization data
                                120       ;
00B5 00                         121       TABLE72 DB 00H,18H,02H,64H,04H,44H
     18
     02
     64
     04
     44
00BB 0F                         122               DB 0FH,52H,03H,0C0H,05H,60H,0CH,02H,18H,00
     52
     03
     C0
     05
     60
     0C
     02
     18
     00
00C5 0C                         123               DB 0CH,01H,18H,00,0EH,07H,03H,0C1H,00,00
     01
     18
     00
     0E
     07
     03
     C1
     00
     00
00CF                            124       TABLE72_END     LABEL WORD
   001A                         125       TAB_COUNT EQU TABLE72_END - TABLE72
                                126       ;
                                127       ; Messages
                                128       ;
00CF 0D                         129       MESSAGE DB      0DH,'V25-72001 Demo/Test program',0DH,10
     5632352D37323030
     312044656D6F2F54
     6573742070726F67
     72616D
     0D
     0A
00ED 5472616E736D6974           130               DB      'Transmit operational with DMA',0DH,10,13,10
     206F706572617469
     6F6E616C20776974
     6820444D41
     0D
     0A
     0D
     0A
010E                            131       MESSAGE_END     LABEL WORD
   003F                         132       MESSAGE_LEN     EQU     MESSAGE_END - MESSAGE
                                133       ;
010E 1E68FFFF1FC6060F R         134               SETIDB  HIGH SFRLOC     ; set IDB to top byte of SFR segment
     00F71F
                                135                                       ; required by RA70320 for SFR location
----                            136       CODE ENDS
                                137       END

ASSEMBLY COMPLETE,     0 ERROR(S) FOUND
```

Interfacing the μCOM75X Family to the μPD6252 (EEPROM)

Contents:

Author: Peter Diederichs
Application μCOM Department
NEC Electronics (Europe) GmbH

Related Documentation

μCOM75X	Family Product Description
μPD6252	Data Sheet

Related Products

μPD75008	4-Bit Microcomputer	CMOS
μPD75308	4-Bit Microcomputer	CMOS
μPD75328	4-Bit Microcomputer	CMOS
μPD75516	4-Bit Microcomputer	CMOS
μPD6252	2048 Bit EEPROM	

1. Introduction

This application note describes the interface between any member of the μCOM75X Family microcomputer with SBI interface (including 2-line I/O mode) with the EEPROM μPD6252.

The μPD6252 has 256 bytes of EEPROM and two possibilities of a serial bus connection, these are:

a) Two-wire serial bus interface mode (similar to the I²C* protocol).

b) The three-wire serial bus interface mode (common serial interface for all μCOM75 and 75X products) will be described in an other application note.

The 2-line I/O mode is a part of the μCOM75X SBI interface and offers the possibility to produce several types of protocols.

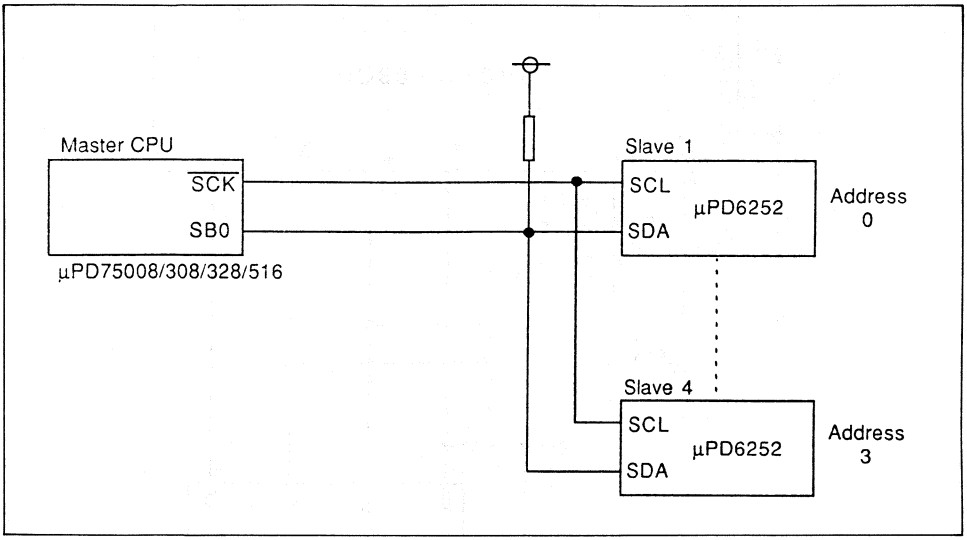


Figure 1. 2-Line System Configuration

The μPD6252 has two address lines, therefore it is possible to connect up to four EEPROMS on one microprocessor bus.

Note: * I²C is a patened protocol of Philips GmbH

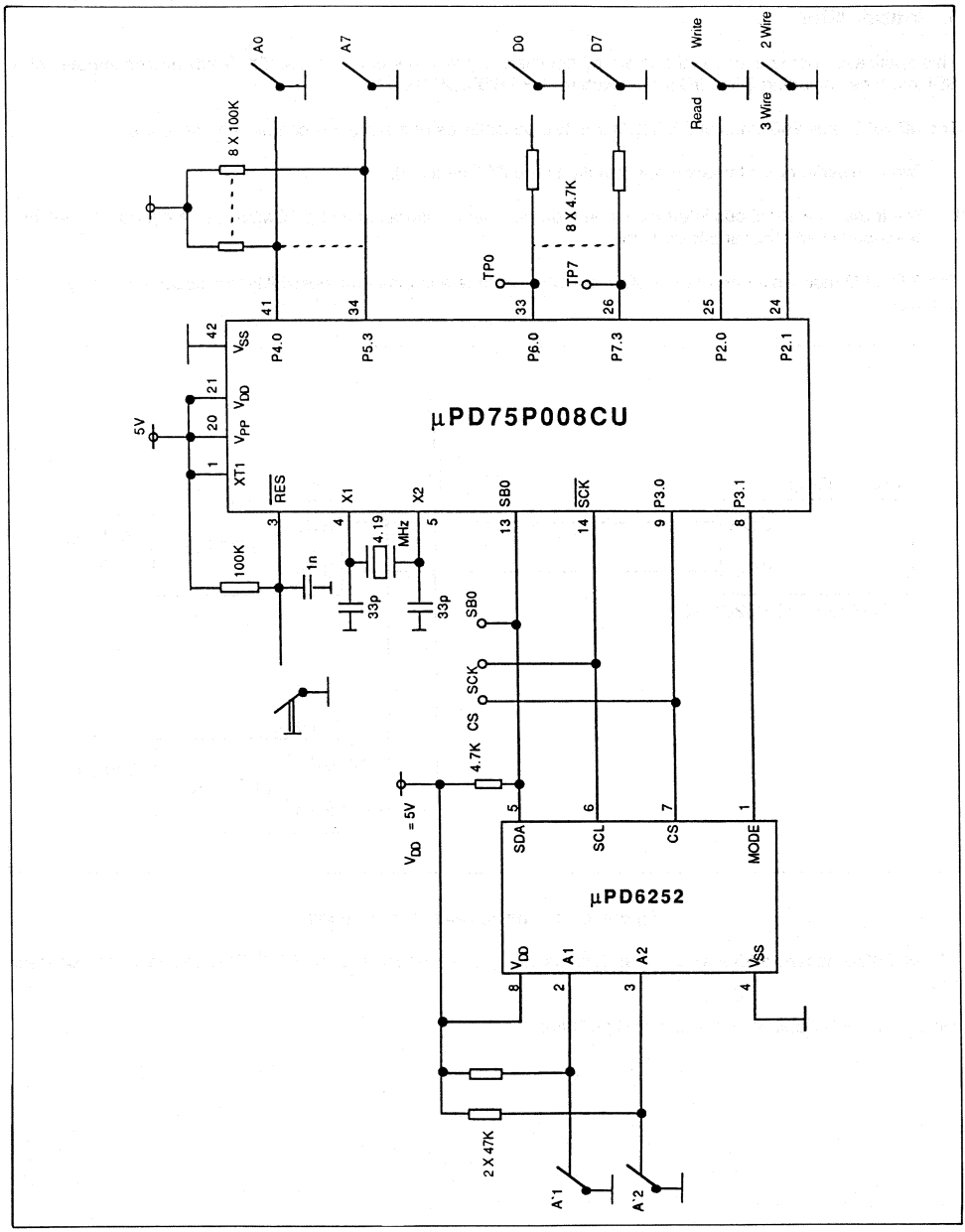


Figure 2. Circuit Diagram of Demonstration Board

2. Hardware

In this application note the used 2-line I/O mode is implemented with the following conditions:

The microcomputer is a master only and the EEPROM is a slave only. The 2-line I/O mode is a open drain bus system and it can be used to perform the I²C protocol. The μCOM75X devices with SBI interface have programmable pull-up resistors on the serial I/O (SB0) and on the serial clock line (SCK).

Figure 2 shows the hardware which is used to run the 2-line I/O mode between μPD75P008 and μPD6252. The μPD75P008 is supplied with V_{DD} = 5V and is running with a 4.19 MHz crystal in high speed mode, that means an instruction cycle time of 0.95 μs.

Ports 4 and 5 are used to select the word address for the μPD6252 (from 0H to FFH). Due to the fact, that P4, 5 are open drain Input/Output there must be 8 external pull-ups connected to V_{DD}. For all other ports the μPD75P008 contains internal pull-up resistors.

Ports 6 and 7 have two tasks:

First, they are used as inputs to generate 1 byte data which should be stored in the EEPROM during write mode.

The second task is to use them as an output to check the read out data on TP0 → TP7 with a scope or a logic analyzer.

Port 2.0 and Port 2.1 are used as inputs, they select between read or write and 3-wire or 2-wire serial I/O mode.

Port 3.0, 3.1 are defined as outputs to force chip select (CS) (choose between operation mode (high) or standby mode (low)) and MODE (choose between 3-wire- (high) or 2-wire mode (low)).

SCK and SB0 are the serial communication lines. SCK is the synchronized clock output with 65.5 KHz (using 4.19 MHz).

SB0 is the data Input/Output pin for 2-line I/O mode.

The address definition of the μPD6252 is done by using the A1', A2'dip switches.

3. Description of Communication Protocol

In 2-line I/O mode an interrupt occurs after the eighth clock, but the two wire serial bus of the μPD6252 requires an additional ninth clock to send an acknowledge. This ninth clock will be implemented by means of software.

The communication between master (μPD75P008) and the slave (μPD6252) starts always on the falling edge of the data line, when the clock line is high.

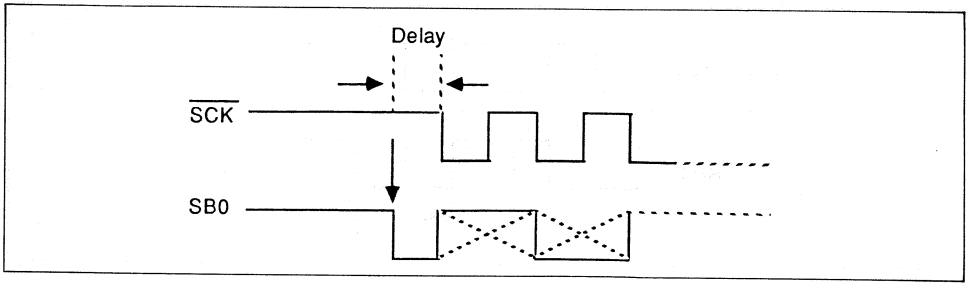


Figure 3. Start Condition

After transmitting an eight bis address of data the slave sends an acknowledge on the ninth clock. That means, if data transmission was ok, the data line is pulled low.

The master clocks out data on the falling edge of the clock pulse and the slave clocks in data on the rising edge.

First to be sent is the Device Address, then the Word Address (define the internal memory address for the EEPROM (0H → FFH)) and then the Data which will be stored.

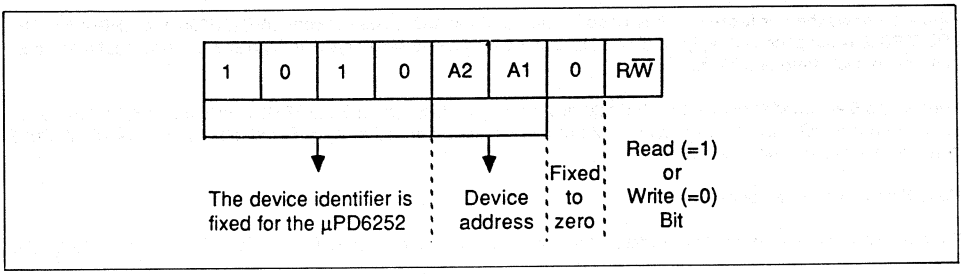


Figure 4. Slave Address Description

To chance the contents of a memory location the procedure is described in figure 5.

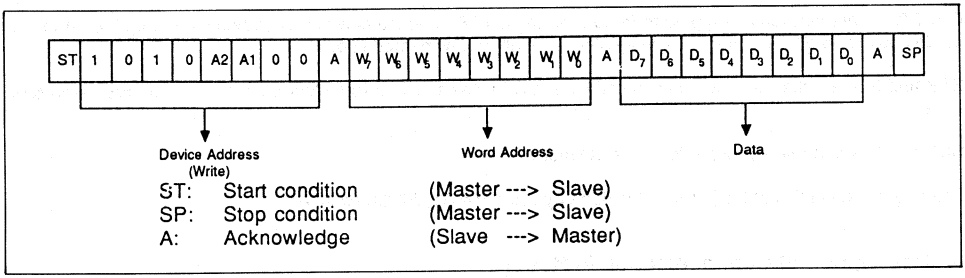


Figure 5. Timing to Write 1 Byte

Every command or data is followed by an acknowledge from the slave if the transmission was successfull (always on the ninth clock). Transmission is terminated by the master after sending the a stop condition. After the stop condition is set the internal write cycle of the EEPROM will begin.

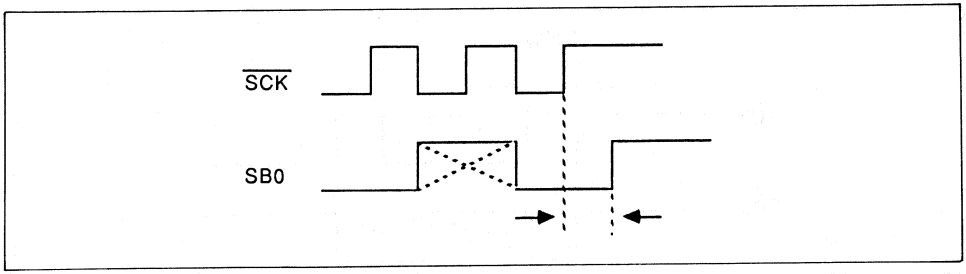


Figure 6. Stop Condition

The Start- and Stop Condition is given by setting two Trigger-bits, in software.

RELT (bus release trigger bit) = Set output latch high
CMDT (bus command trigger bit) = Set output latch low

To read an one byte data from a specified EEPROM Word Address is shown in figure 7.

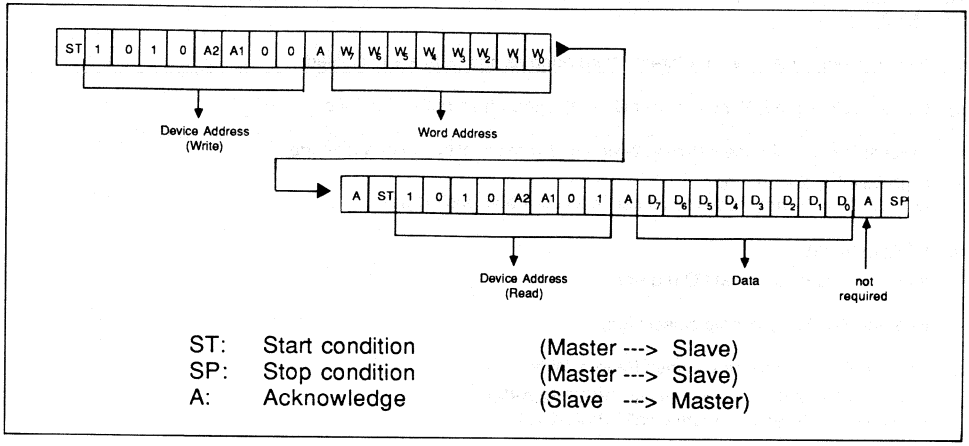


Figure 7. Timing to Read 1 Byte

4. Software

The software contains four modules with different tasks.

Module 1: POLE.ASM

This module represents the main routine for the communication between master (μPD75P008) and slave (μPD6252).

a) All parameters like Device Address, Word Address and Data are initialised.

b) Define: CPU speed; Ports (Input or Output) and programmable Pull-ups.

c) Level at Port 2.1 choose between 2-wire- or 3-wire serial bus interface mode.
High Level = 2 wire mode
Low Level = 3 wire mode (described in a later Application Note)

d) If 2-wire mode:
Initialilze and clear serial I/O register

e) Enable μPD6252 (set chip select high)

f) Level at Port 2.0 choose between Read or Write
High Level = Write (subroutine call "WRITE2.ASM)
Low Level = Read (subroutine call "READ2.ASM)

g) After return from any subroutine wait 31ms (required by the μPD6252 for internal write cycle), then set μPD6252 in Standby Mode and endless loop.

Module 2: READ2.ASM

Here the subroutine to Read Out 1 byte from a special Word Address and transfer the contents to Port 6, 7 is described.

a) Define Port 4, 5 as Inputs to read in Word Address; define Port 6, 7 as Outputs to show the contents of the specified address.

b) Set start condition (set CMDT-bit)

c) Transmit Device Address (write)

d) Transmit Word Address

e) Set next start condition

f) Transmit Device Address (Read)

g) Receive Data

h) Set stop condition (set RELT-bit)

k) Output Data at Port 6, 7

Note: After every command or Data Transmission there is a subroutine call "CLOCK 9", to send an additional ninth clock for the acknowledge.

Module 3: WRITE2.ASM

This module describes the Write procedure for 1 byte of Data.

a) Define Port 4, 5 as Inputs for the Word Address and Port 6,7 also as Inputs for the Data to be stored.

b) Set start condition

c) Transmit Device Address (Write)

d) Transmit Word Address

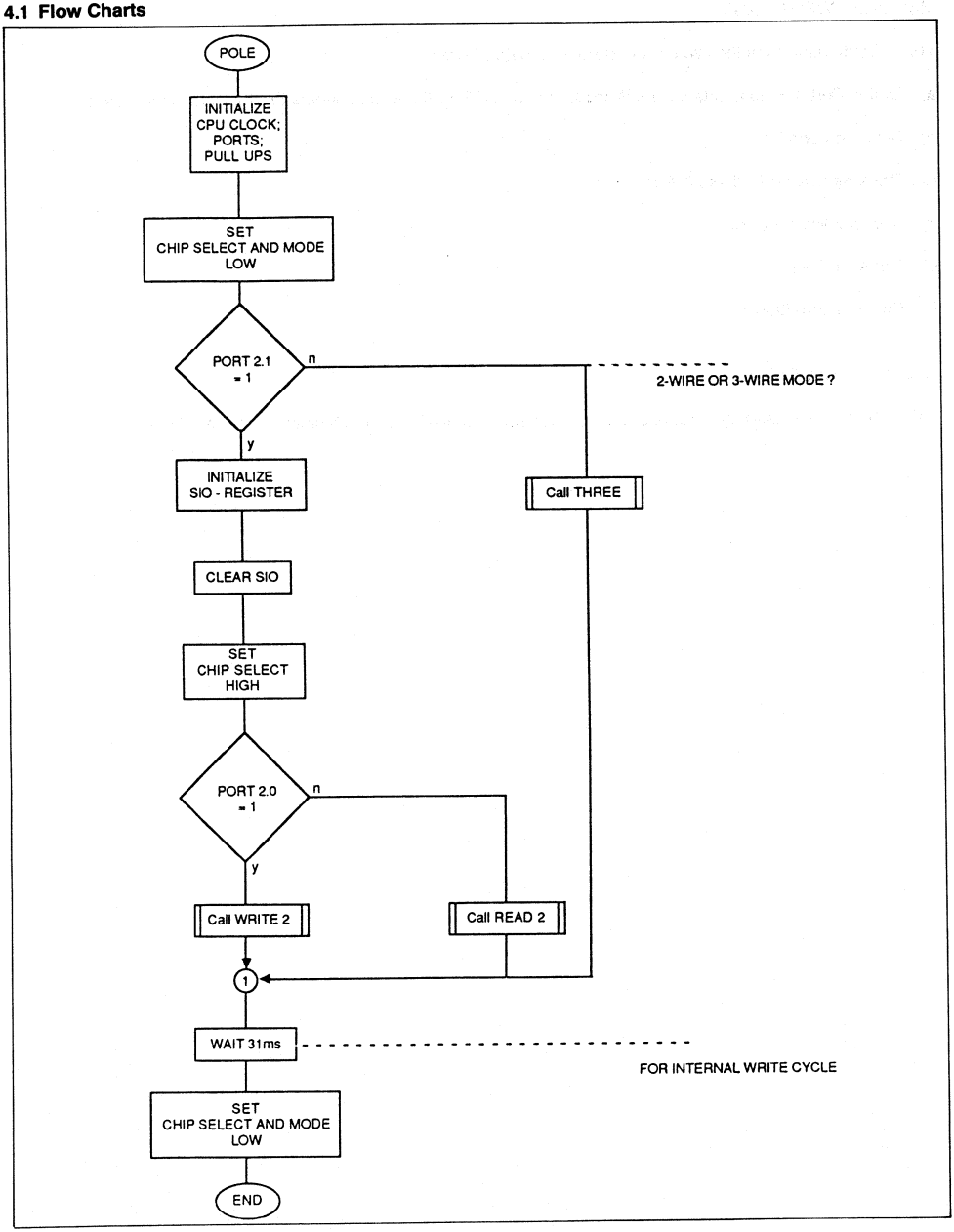
e) Transmit Data

f) Set stop condition

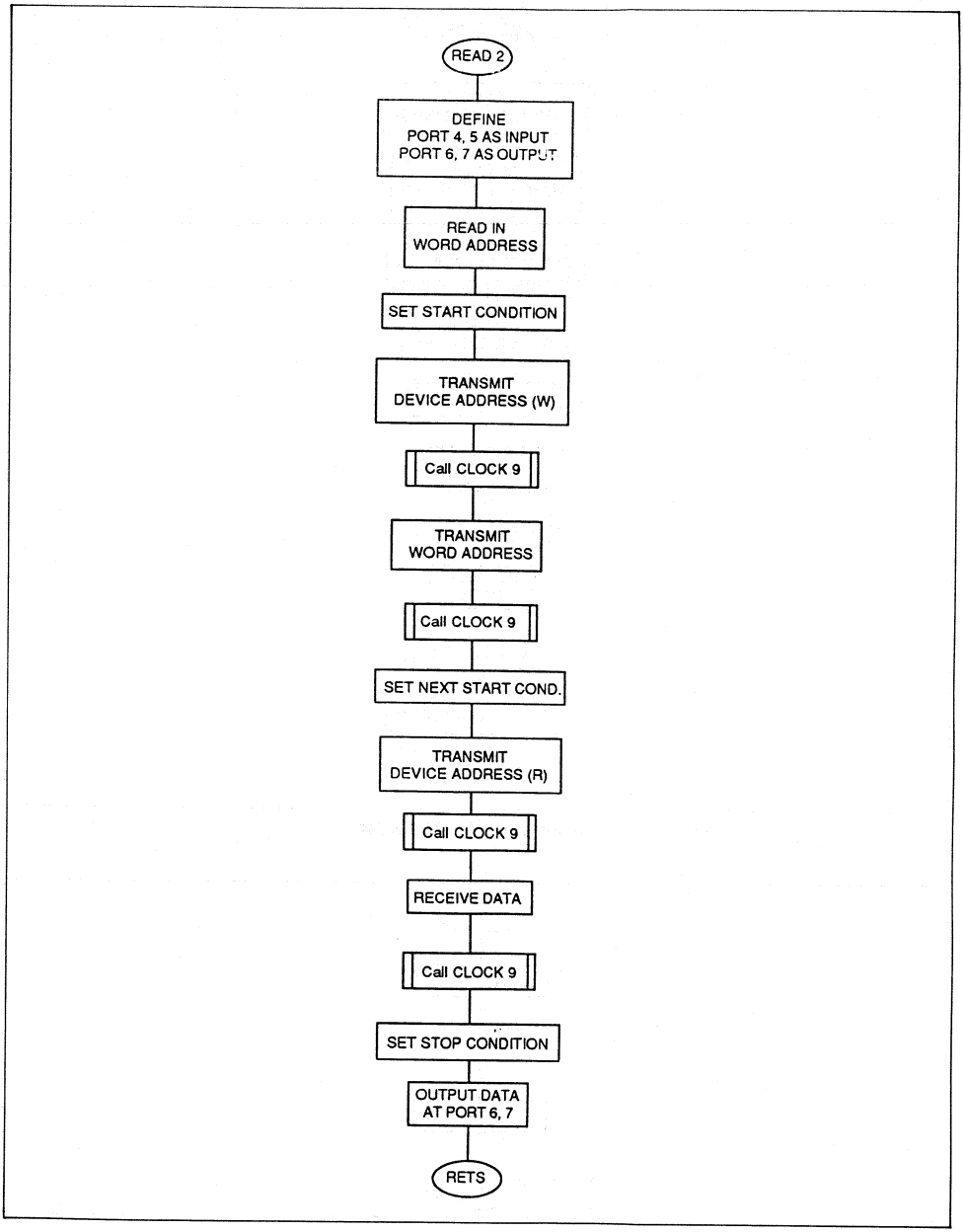
Module 4: CLOCK9.ASM

This subroutine is used by Module 2 and 3 to generate an additional ninth clock by means of software.

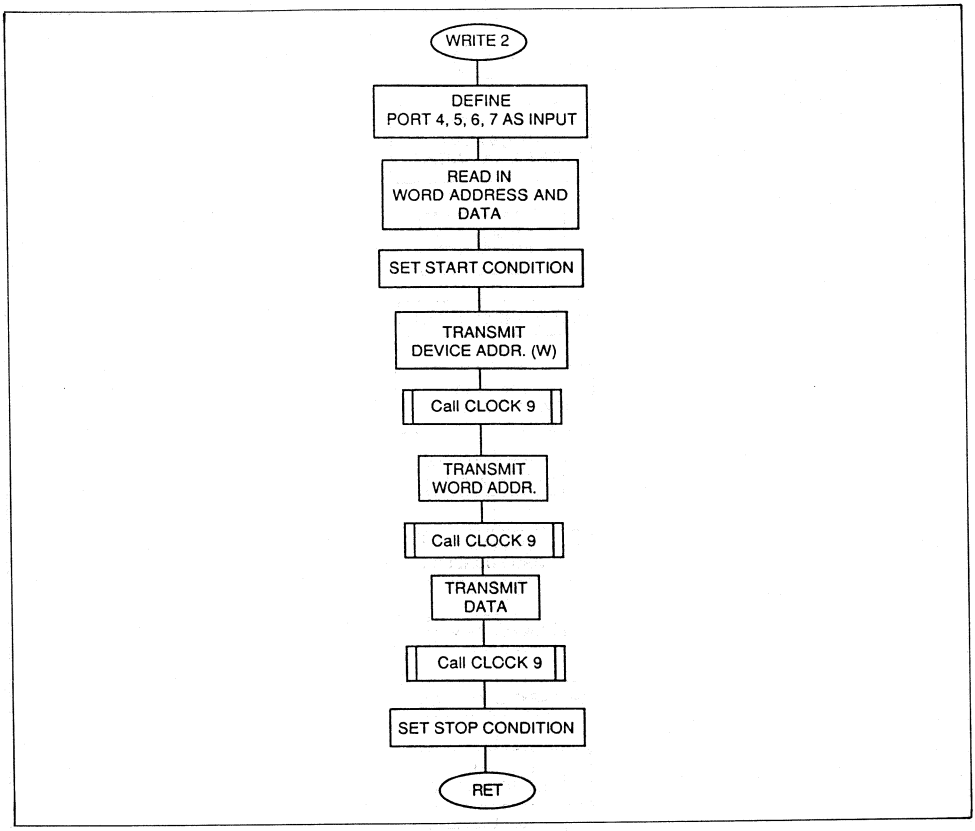
4.1 Flow Charts



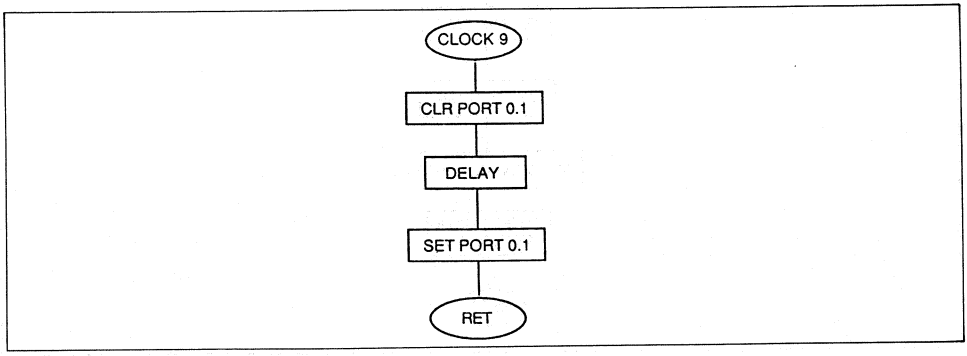
Module 1. POLE.ASM



Module 2. READ2.ASM



Module 3. WRITE2.ASM



Module 4. CLOCK9.ASM

4.2 Listings

```
        NAME    POLE
;
;*********************************************************************
;NAME            :POLE (28.2.89)
;DESC            :SET UP ROUTINE FOR UPD6252 DEMONSTRATION SOFTWARE
;                :SELECTION BETWEEN:
;                :TWO OR THREE WIRE MODE   AND   READ OR WRITE
;DEVICE          :UPD75X (WITH SBI INTERFACE, EXEPT UPD75402)
;MEMORY BANK     :0,15
;HARDWARE USED   :SOME EXT. PUPS AND SOME DIP SWITCHES
;INPUT           :PORTS 2,4,5,(6,7)
;OUTPUT          :PORTS 6,7,3, SB0,SCK
;DESTROY         :XA
;CALLS           :WRITE2,READ2,THREE
;*********************************************************************

        PUBLIC  SELM0,SELM1,SELM15
        PUBLIC  DEVADRW,DEVADRR,WRDADR,EEDATA
        EXTRN   CODE(WRITE2,READ2,THREE)

; ****************** VECTOR TABLE ***********************************

        STKLN   40H
        VENT0   MBE=1,RBE=0,GO  ;START VECTOR

; ****************** BUFFERS MEMORY BANK 0 **************************

D1SEG   DSEG    0       AT      10H
DEVADRW:DS      2               ;BUFFER FOR DEVICE ADDRESS (WRITE)
DEVADRR:DS      2               ;BUFFER FOR DEVICE ADDRESS (READ)
WRDADR: DS      2               ;BUFFER FOR WORD ADDRESS
EEDATA: DS      2               ;BUFFER FOR EEPROM DATA (1 BYTE)

; ****************** GETI TABLE *************************************

        GET1    CSEG    IENT
        SELM0:  SEL     MB0
        SELM1:  SEL     MB1
        SELM15: SEL     MB15

; ****************** PROCESSOR START CONDITION **********************

CSEG1   CSEG    SENT
GO:
        GETI    SELM15
        MOV     XA,#STACK       ;SET STACK
        MOV     SP,XA
        MOV     A,#0011B        ;SET HIGH SPEED CLOCK
        MOV     PCC,A

        MOV     XA,#0C7H
        MOV     POGA,XA         ;PUPS ON P0,P1,P2,P6,P7

        MOV     XA,#0FH
        MOV     PMGA,XA         ;PORT 3 AS OUTPUT, PORT 6 AS INPUT
        MOV     XA,#00H
        MOV     PMGB,XA         ;ALL OTHER PORTS AS INPUT
        MOV     A,#0H
        OUT     PORT3,A         ;SET CHIP SELECT (CS) AND MODE  LOW

; ****************** WHICH MODE ? ***********************************

        SKT     PORT2.1         ;TWO OR THREE WIRE MODE ?
        BR      SUB3WIRE        ;THREE WIRE MODE IS REQUESTED
```

```
; ****************** TWO WIRE MODE ***********************************

        MOV     XA,#10001111B
        MOV     CSIM,XA         ;BIT7 ENABLE SHIFT-REG.
                                ;P02/SBO = IN/OUTPUT,
                                ;SCK=65.5KHz (FIXED)
        MOV     XA,#0FFH
        MOV     SIO,XA          ;CLEAR SERIAL I/O LINE INTERFACE

        GETI    SELM0
        MOV     XA,#0A0H
        MOV     DEVADRW,XA      ;LOAD DEVEICE ADDR. WRITE INTO A BUFFER
        MOV     XA,#0A1H
        MOV     DEVADRR,XA      ;LOAD DEVEICE ADDR. READ INTO A BUFFER
        GETI    SELM15

CLEAR:
        SKTCLR  IRQCSI          ;WAIT UNTIL SERIAL I/O TRANSISTORS ARE
        BR      CLEAR           ;SWITCHED OFF

        SET1    PORT3.0         ;SET CS HIGH (ENABLE UPD6252)
        SKT     PORT2.0         ;READ OR WRITE ?
        CALLF   !READ2          ;READ MODE
        CALLF   !WRITE2         ;WRITE MODE
        BR      FINI

SUB3WIRE:
        CALLF   !THREE          ;SUBROUTINE 3 - WIRE MODE

FINI:
        MOV     A,#1011B
        MOV     BTM,A           ;WAIT 31MS FOR INTERNAL WRITE CYCLE
INTWA:                          ;REQUESTED FROM UPD6252 (DATA SHEET: MAX 40MS)
        SKTCLR  IRQBT
        BR      INTWA

        MOV     A,#0H
        OUT     PORT3,A         ;SET CS AND MODE LOW (UPD6252 IN STANDBY MODE)
FINISH:
        NOP
        BR      FINISH          ;ENDLESS LOOP

END
```

```
        NAME    READ2
;
;*********************************************************************
;NAME            :READ2  (30.1.89)
;DESC            :SUBROUTINE TO READ 1 BYTE FROM THE EEPROM UPD6252
;                :IN TWO WIRE I/O MODE
;DEVICE          :UPD75X (WITH SBI INTERFACE, EXEPT UPD75402)
;MEMORY BANK     :0,15
;HARDWARE USED   :SOME EXT. PUPS AND DIP SWITCHES
;INPUT           :PORT4,5,(SB0)
;OUTPUT          :SCK,SB0,PORT6,7
;DESTROY         :XA,DE
;CALLS           :CLOCK9
;*********************************************************************

        PUBLIC  READ2
        EXTRN   CODE(SELM0,SELM15,CLOCK9)
        EXTRN   DATA(DEVADRW,DEVADRR,WRDADR,EEDATA)

CSEG3   CSEG    INBLOCK
READ2:
        PUSH    BS
        PUSH    XA
        PUSH    HL
        PUSH    DE

        CLR1    MBE             ;SELECT MB0 AND MB15

        MOV     XA,#0FFH
        MOV     PMGA,XA         ;PORT 3 AND 6 AS OUTPUT
        MOV     XA,#80H
        MOV     PMGB,XA         ;PORT 7 AS OUTPUT, REST IS INPUT

        IN      XA,PORT4
        MOV     WRDADR,XA       ;READ WORDADDR. AND SAVE INTO BUFFER

; ***************** START CONDITION *********************************

        SET1    CMDT            ;SET SERIAL I/O LOW
        MOV     A,#0BH
DEL1R:
        INCS    A               ;DELAY BEFORE START TRANSMIT
        BR      DEL1R

; ***************** TRANSMIT DEVEICE ADDR. (WRITE) ********************

        MOV     XA,DEVADRW
        MOV     SIO,XA          ;SEND DEVEICE ADDR. (WRITE)
DEVWR:
        SKTCLR  IRQCSI          ;WAIT UNTIL TRANSFER IS FINISHED
        BR      DEVWR
        CALLF   !CLOCK9         ;NINTH CLOCK FOR THE ACKNOWLEDGE (ACK)

; ***************** TRANSMIT WORD ADDRESS ***************************

        MOV     XA,WRDADR
        MOV     SIO,XA          ;SEND WORD ADDR.
WRDR:
        SKTCLR  IRQCSI
        BR      WRDR
        CALLF   !CLOCK9         ;FOR THE ACK
```

```
; ***************** NEXT START CONDITION ****************************

        SET1    RELT            ;SET SERIAL I/O LINE HIGH
        CALLF   !CLOCK9
        NOP
        NOP
        NOP
        SET1    CMDT            ;SET SERIAL I/O LINE LOW (START COND.)
        MOV     A,#0BH
DEL2R:
        INCS    A               ;DELAY BEFORE START TRANSMIT
        BR      DEL2R

; ****************** TRANSMIT DEVEICE ADDR. (READ) ********************

        MOV     XA,DEVADRR
        MOV     SIO,XA          ;SEND DEVEICE ADDR. READ
DEVRR:
        SKTCLR  IRQCSI
        BR      DEVRR
        CALLF   !CLOCK9

; ****************** RECEIVE EEDATA *********************************

        MOV     XA,#0FFH        ;CLEAR SERIAL I/O TRANSISTOR
        XCH     XA,SIO          ;READ EEDATA
DATR:
        SKTCLR  IRQCSI
        BR      DATR
        MOV     XA,SIO          ;LOAD XA WITH EEDATA
        MOV     DE,XA           ;SAVE THE DATA IN DE
        CALLF   !CLOCK9
; ****************** STOP CONDITION *********************************
        SET1    CMDT            ;SET SERIAL I/O LINE LOW
        CALLF   !CLOCK9
        MOV     A,#0CH
DELR3:
        INCS    A               ;DELAY BEFORE STOP
        BR      DELR3
        SET1    RELT            ;NOW, SET SERIAL I/O HIGH,
                                ;DURING SCK IS HIGH

; ******************* OUTPUT THE DATA TO PORT6,7 ***********************

        MOV     XA,DE           ;GET THE DATA FROM DE
        OUT     PORT6,XA

; ******************* RETURN *************************************

        SET1    MBE
        POP     DE
        POP     HL
        POP     XA
        POP     BS
        RETS
END
```

```
        NAME    WRITE2
;*********************************************************************
;NAME            :WRITE2 (30.1.89)
;DESC            :SUBROUTINE TO WRITE 1 BYTE IN THE EEPROM UPD6252
;                :IN TWO WIRE I/O MODE
;DEVICE          :UPD75X (WITH SBI INTERFACE EXEPT UPD75402)
;MEMORY BANK     :0,15
;HARDWARE USED   :SOME EXT. PUPS AND SOME DIP SWITCHES
;INPUT           :PORT4,5,6,7
;OUTPUT          :SCK,SB0
;DESTROY         :XA
;CALLS           :CLOCK9
;*********************************************************************

        PUBLIC  WRITE2
        EXTRN   CODE(SELM0,SELM15,CLOCK9)
        EXTRN   DATA(DEVADRW,WRDADR,EEDATA)

CSEG2   CSEG    SENT
WRITE2:
        PUSH    BS
        PUSH    XA
        PUSH    HL
        CLR1    MBE             ;SELECTED MB0 AND MB15

; ***************** READ WORDADDR. AND DATA *************************

        IN      XA,PORT4
        MOV     WRDADR,XA       ;READ PORT4,5 AND SAVE AS WORDADDR.
        IN      XA,PORT6
        MOV     EEDATA,XA       ;READ PORT6,7 AND SAVE AS EEPROM DATA

; ***************** START CONDITION *********************************
        SET1    CMDT            ;SET SERIAL I/O LOW
        MOV     A,#0BH
DELW:
        INCS    A               ;DELAY BEFORE START TRANSMIT
        BR      DELW

; ***************** TRANSMIT DEVEICE ADDRESS ************************

        MOV     XA,DEVADRW
        MOV     SIO,XA          ;SEND DEVEICE ADDR. (WRITE)
DEVADW:
        SKTCLR  IRQCSI          ;WAIT UNTIL TRANSFER IS FINISHED
        BR      DEVADW
        CALLF   !CLOCK9         ;NINTH CLOCK FOR THE ACKNOWLEDGE (ACK)

; ***************** TRANSMIT WORD ADDRESS ***************************

        MOV     XA,WRDADR
        MOV     SIO,XA          ;SEND WORD ADDR.
WRDW:
        SKTCLR  IRQCSI
        BR      WRDW
        CALLF   !CLOCK9         ;FOR THE ACK

; ***************** TRANSMIT DATA ***********************************

        MOV     XA,EEDATA
        MOV     SIO,XA          ;SEND THE DATA
DATW:
        SKTCLR  IRQCSI
        BR      DATW
        CALLF   !CLOCK9
```

```
; ****************** STOP CONDITION ***********************************

        SET1    CMDT            ;SET SERIAL I/O LINE LOW
        CALLF   !CLOCK9
        MOV     A,#0CH
DELW1:
        INCS    A               ;DELAY >4.7US
        BR      DELW1
        SET1    RELT            ;SET SERIAL I/O LINE HIGH,
                                ;DURING SCK IS ON HIGH LEVEL
                                ;NOW BEGIN THE INTERNAL WRITE CYCLE (MAX.40MS)

; ****************** RETURN ***********************************

        SET1    MBE
        POP     HL
        POP     XA
        POP     BS
        RET
END
```

```
        NAME    CLOCK9
;*****************************************************************
;NAME            :CLOCK9 (30.1.89)
;DESC            :SUBROUTINE TO MAKE A NINTH CLOCK BY SOFTWARE, BECAUSE
;                :THE SLAVE MUST BE ENABLE TO SEND THE ACKNOWLEDGE
;DEVICE          :UPD75X (WITH SBI INTERFACE, EXEPT UPD75402)
;MEMORY BANK     :15
;HARDWARE USED   :EXTERNAL PULL UP ON SB0
;INPUT           :
;OUTPUT          :SCK
;DESTROY         :XA
;CALLS           :
;*****************************************************************

        PUBLIC  CLOCK9

CSEG4   CSEG    SENT
CLOCK9:
        PUSH    BS
        CLR1    0FF0H.1         ;SET PORT0.1 LOW
        MOV     A,#0EH
CLK1:
        INCS    A
        BR      CLK1
        SET1    0FF0H.1         ;SET PORT0.1 HIGH
        POP     BS
        RET
END
```

```
        NAME    THREE
;
;*********************************************************************
;NAME            :THREE
;DESC            :INITIALIZE ROUTINE FOR UPD6252 IN 3 - WIRE MODE
;DEVICE          :UPD75X (WITH SBI INTERFACE)
;MEMORY BANK     :0,15
;HARDWARE USED   :SOME EXT. PUPS AND SOME DIP SWITCHES
;INPUT           :PORTS 2,4,5, (6,7), SI
;OUTPUT          :PORTS 6,7,3,SO,SCK
;DESTROY         :XA
;CALLS           :READ3,WRITE3
;*********************************************************************
        PUBLIC  THREE
;       EXTRN   CODE(SELM15,WRITE3,READ3)

CSEG5   CSEG    SENT
THREE:
        PUSH    BS
        PUSH    XA
        PUSH    HL

        NOP                     ;LATER ON HERE WILL BE THE PROGRAM FOR THE
        NOP                     ; 3  WIRE MODE

        POP     HL
        POP     XA
        POP     BS
        RET
END
```

8-Bit A/D Conversion for the μCOM75X Familiy with the Dual Slope Method

Contents:

Author: Frank Jordan / Peter Diederichs
Application μCOM Department
NEC Electronics (Europe) GmbH

Related Documentation

μCOM75X	Family Product Description
μPD75308	Application Note (1)

Related Products

μPD750XX	4-Bit Microcomputer	CMOS
μPD751XX	4-Bit Microcomputer	CMOS
μPD752XX	4-Bit Microcomputer	CMOS
μPD753XX	4-Bit Microcomputer	CMOS
μPD755XX	4-Bit Microcomputer	CMOS

1. Introduction

This Application Note describes an A/D Conversion procedure using the μPD75X microcomputer family products. The A/D Conversion is done using the dual slope method. It is performed with a minimum of external hardware components, only two resistors and one capacitor are needed.

2. Description of A/D Conversion using the Dual Slope Method

The μCOM75X family has the advantage of bit setable I/O Ports (Port 3 and Port 6) and Schmitt Trigger inputs (e.g. INT0). In this Application Note the bit setable Port 3 is used as output respectively as input (high impedance state). The dual slope method is based on the comparison of two resistors (R_{ref} and R_{var}), which charge a capacitor.

At first Port 3 is cleared and set to output mode. In this case the capacitor is discharged very fast via Port 3.2 and prepared for the first measurement. Then Port 3.1 is set to 1 and output. The rest of Port 3 is set to input (high impedance). Also the timer is started at the same time. So the capacitor will be charged via the reference resistor. When the capacitor has reaches the threshold level of the Schmitt Trigger input INT0, an interrupt occurs and the interrupt request flag is set. Immediately after that the timer will be stopped and the timer value is read out. Now the capacitor will be discharged again. The same procedure starts once more with Port 3.0. The capacitor is charged via the unknown resistor R_{var}, and after INT0 occurence, the timer is read out again.

The unknown resistor R_{var} can be calculated by the two values above and the value of the reference resistor. (Please refer derivation).

Derivation

$$R_{ref} : Vcref = V_{DD}\left(1-e^{-\frac{tref}{Rref * C}}\right)$$

$$R_{var} : Vcvar = V_{DD}\left(1-e^{-\frac{tvar}{Rvar * C}}\right)$$

$$Vc = Vcvar = Vcref = const.$$

The threshold level of the Schmitt Trigger input does not have any influence on the accuracy of the measurement.

$$V_{DD}\left(1-e^{-\frac{tvar}{Rvar * C}}\right) = V_{DD}\left(1-3^{-\frac{tref}{Rref * C}}\right)$$

$$1-e^{-\frac{tvar}{Rvar * C}} = 1-e^{-\frac{tref}{Rref * C}}$$

$$e^{-\frac{tvar}{Rvar * C}} = e^{-\frac{tref}{Rref * C}}$$

$$\frac{tvar}{Rvar * C} = \frac{tref}{Rref * C}$$

$$\frac{tvar}{Rvar} = \frac{tref}{Rref}$$

$$Rvar = Rref * \frac{tref}{tvar}$$

C and V_{DD} do not have any influence on the accuracy of the measurement. Only the absolute value of the reference resistor R_{ref} has an influence. Using R_{ref}, t_{ref} and t_{var}, the resistor Rvar can be calculated.

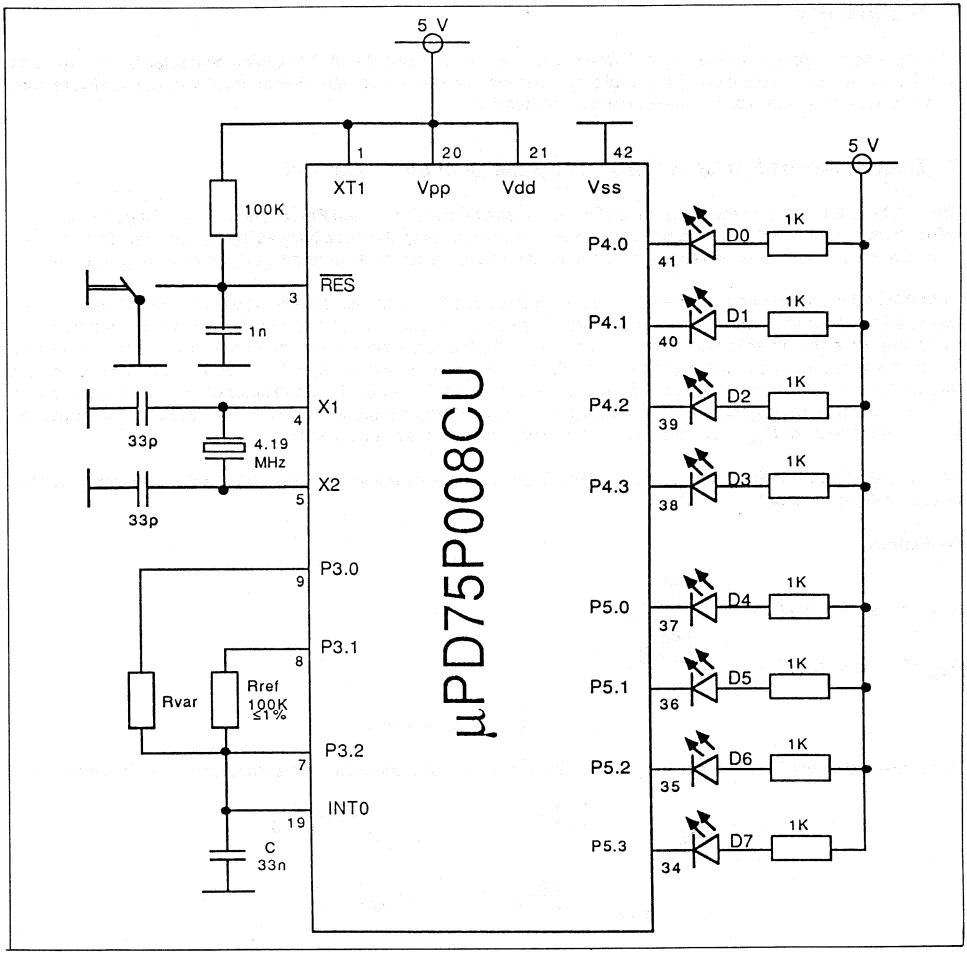


Figure 1. Circuit Diagram

3. Hardware

Port 3.1 is used to charge the capacitor via the reference resistor Ref. Port 3.0 is used to charge the capacitor via the unknown resistor Rvar. The time of charging the capacitor is measured in both cases. With the two times (REFTIM, VARTIM) and resistor Rref the unknown resistor Rvar can be calculated.

Port 3.2 is used to discharge the capacitor very quick. Port 1.0 is used as the interrupt input (INT0). This interrupt port has a Schmitt Trigger input.

Port 4 and 5 are used as output ports. They could drive immediate LEDs to get a visible result (percentage of R_{var} refer to R_{ref} in hex code). This is for demonstration purpose only.

4. Software

The software consists of two modules:

ADCONV.ASM: This module contains the system start condition:
Initialize stackpointer, CPU-clock, Ports 4 and 5 as output and all others as input. Pull up resistores at Ports 0, 2, 6, 7 and 8, and interrupt 0 active on rising edge.

This module also does the measurement of the charging times which are used to calculate the unknown resistor, and it outputs the result for demonstration purpose.

CALC.ASM: In this module the unknown resistor is calculated. At first the charging time of Rvar will be multiplied with 100 (for percentage). Then the resistor will be added to the result (it is only to round the result at the end). After that the result will be divided by the time of the reference resistor. This result is the percentage of the unknown resistor to the reference resistor and is stored to RESIST.

This module is mainly taken from the Application Note AN75308 . . . 128V10.

Charging Time for R_{ref}

$$Vc = V_{DD}\left(1-e^{-\frac{t}{R*C}}\right)$$

$$\frac{Vc}{V_{DD}} = 1-e^{-\frac{t}{R*C}}$$

$$1-\frac{Vc}{V_{DD}} = e^{-\frac{t}{R*C}}$$

$$-\frac{t}{R*C} = \ln\left(1-\frac{Vc}{V_{DD}}\right)$$

$$t = -R*C*\ln\left(1-\frac{Vc}{V_{DD}}\right)$$

For Example:

V_{DD} = 5V; $V_{Threshold} = V_C$ (0,4 . . . 0,7) V_{DD}
TYPICAL: $V_{Threshold} \approx 0{,}58\ V_{DD}$
R_{ref} = 100 KΩ; C = 33nF
$t = -R_{ref} * C * \ln(1 - V_C / V_{DD})$
$t_{TYP} = -100K\Omega * 32nF * \ln(1 - 0{,}58))$
t_{TYP} = 2,86 ms

TIMER VALUE FOR R_{ref}:

RESOLUTION: 15,25 μs MAX SETUP TIME: 3,906 ms

N_{TYP} = t_{TYP} / RESOLUTION

N_{TYP} = 2,86 ms / 15,26 μs

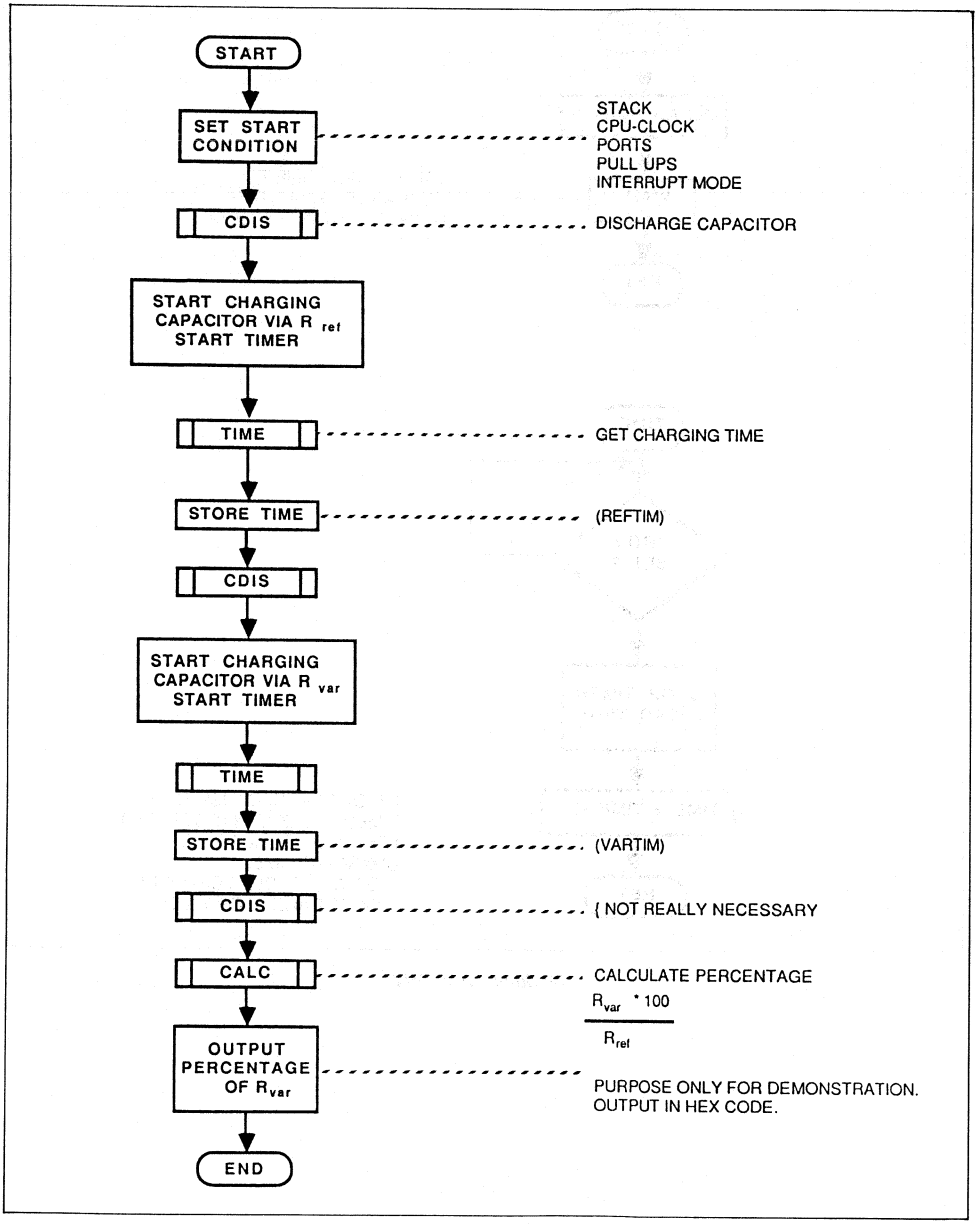
N_{TYP} = 187,4 DECIMAL

N_{TYP} = BB HEX

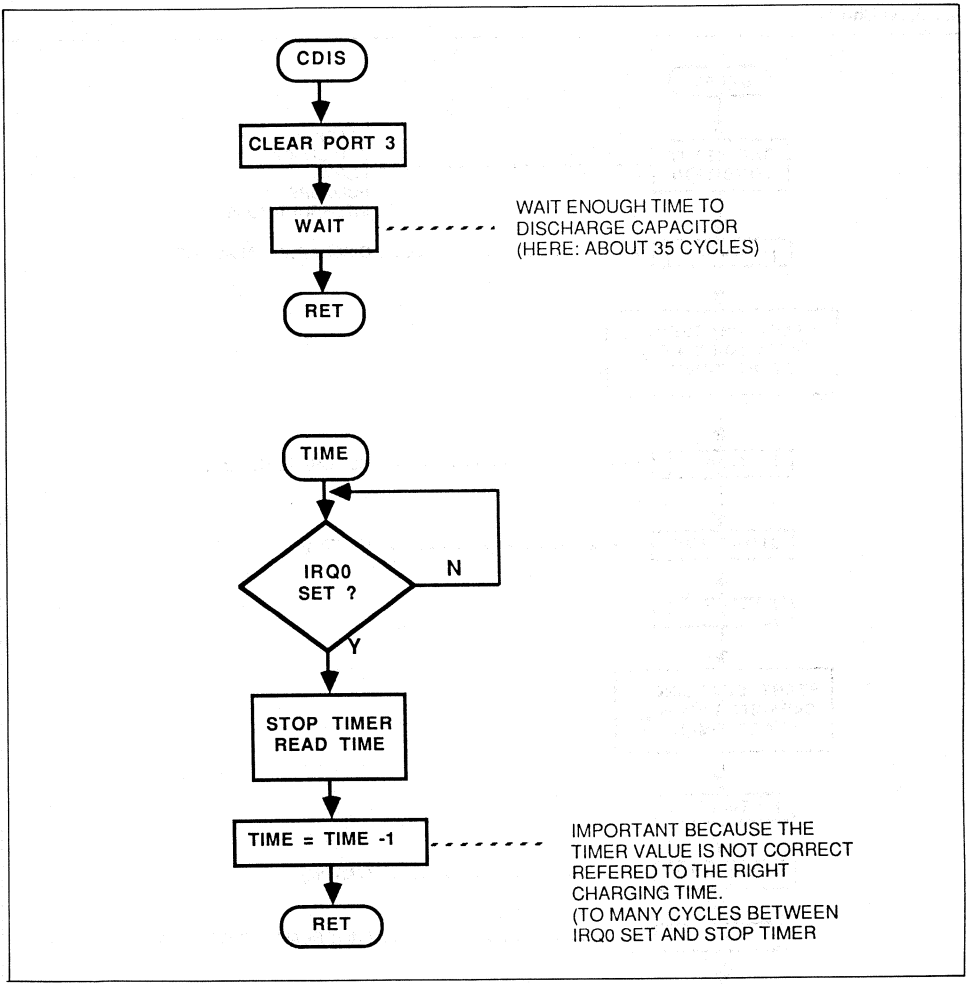
Note: It is recommended that the value of the measured resistors R_{var} and R_{ref} should not be higher than 100Ω to make sure that the error caused by leakage current is necligable. Also the timer may overflow,, if the resistor value is too high.

This procedure described in this Application Note takes less than 10 ms for a complete measurement and calculation.

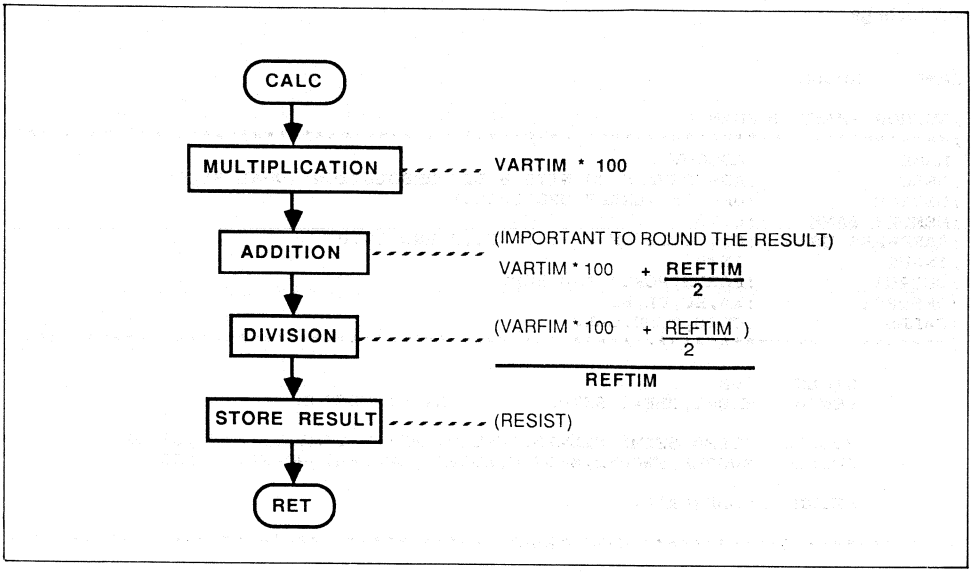
4.1 Flow Charts



Module: ADCONV



Module: ADCONV


CALC
MULTIPLICATION
VARTIM * 100
ADDITION
(IMPORTANT TO ROUND THE RESULT)
VARTIM * 100 + REFTIM / 2
DIVISION
(VARFIM * 100 + REFTIM / 2) / REFTIM
STORE RESULT
(RESIST)
RET

Module: CALC

4.2 Listings

```
NAME    ADCONV

;AUTHOR FRANK JORDAN
;*************************************************************************
;NAME            :ADCONV
;DESC            :AD-CONVERSION WITH 8 BIT RESOLUTION (DUAL SLOPE)
;DEVICE          :UPD75X (EXEPT UPD75P402)
;MEMORY BANK     :0,15
;HARDWARE USED   :EXT PUPS ON P1.1 TO P1.3 AND P0.0
;INPUT           :INT0
;OUTPUT          :PORT3,PORT4 AND PORT5
;DESTROY         :XA,BC,DE,HL
;CALLS           :CDIS,TIME,CALC
;*************************************************************************

        STKLN   20H
        VENT0   MBE=1,RBE=0,SETUP       ;START VECTOR

        PUBLIC  SELM0,SELM1,SELM15,REFTIM,VARTIM,TIME1,RESIST,SETUP
        PUBLIC  MWORKA,MWORKB,MWORKC,ADDRND,DWORKA,DWORKB,DWORKC

        EXTRN   CODE(CALC)

;************************ GETI TABLE ***************************************

GET1    CSEG    IENT
SELM0:  SEL     MB0
SELM1:  SEL     MB1
SELM15: SEL     MB15

;***************************** VARIABLES ***********************************

D1SEG   DSEG    0       AT      20H
REFTIM: DS      2
VARTIM: DS      2
TIME1:  DS      2
RESIST: DS      2

;************************ FOR MULTIPLIKATION *******************************

DMLT8   DSEG    0       AT      30H
MWORKC: DS      2H
MWORKB: DS      2H
MWORKA: DS      2H

;***************************** FOR ADDITION (ROUND) ************************

ADDRND: DS      4H
```

```
;*************************** FOR DIVISION ***********************************

DDIV16  DSEG    0       AT      40H
DWORKA: DS      4H
DWORKB: DS      4H
DWORKC: DS      4H

;*************************** INITIALIZE *************************************

C1SEG   CSEG    INBLOCK
SETUP:
        GETI    SELM15
        MOV     XA,#00H
        MOV     SP,XA                   ;SET STACK
        MOV     A,#0011B
        MOV     PCC,A                   ;SET HIGH SPEED CLOCK
        MOV     A,#0000B
        MOV     IM0,A                   ;SET INTERRUPT MODUS
        MOV     XA,#11000101B
        MOV     POGA,XA                 ;PULL UP RESISTORS PORT0,2,6,7
        MOV     POGB,XA                 ;PULL UP RESISTORS PORT8
        MOV     XA,#00110000B           ;SET PORT4,PORT5 AS OUTPUT
        MOV     PMGB,XA                 ;ALL OTHERS AS INPUT

;************************ REFERENCE RESISTOR TIME ***************************

        CALL    !CDIS                   ;DISCHARGE CAPACITOR
        SET1    PORT3.1
        MOV     XA,#00001010B           ;PORT3.1 AS OUTPUT, CHARGE CAPACITOR
        MOV     PMGA,XA                 ;VIA REFERENCE RESISTOR
        MOV     XA,#11101100B           ;RESOLUTION 15.3US, MAX TIME 3.9MS
        MOV     TM0,XA                  ;START TIMER
        CALL    !TIME                   ;LOAD TIME IN XA
        GETI    SELM0
        MOV     REFTIM,XA               ;SAVE REFERENCE TIME

;************************ VARIABLE RESISTOR TIME ****************************

        GETI    SELM15
        CALL    !CDIS                   ;DISCHARGE CAPACITOR
        SET1    PORT3.0
        MOV     XA,#00001001B           ;PORT3.0 AS OUTPUT, CHARGE CAPACITOR
        MOV     PMGA,XA                 ;VIA VARIABLE RESISTOR
        MOV     XA,#11101100B           ;RESOLUTION 15.3US, MAX TIME 3.9MS
        MOV     TM0,XA                  ;START TIMER
        CALL    !TIME                   ;LOAD TIME IN XA
        GETI    SELM0
        MOV     VARTIM,XA               ;SAVE VARIABLE TIME
        GETI    SELM15
        CALL    !CDIS                   ;MAY BE NOT NECESSARY
```

```
;****************** CALCULATE PERCENTAGE OF VARIABLE RESISTOR ****************

        GETI    SELM0
        CALL    !CALC                   ;CALCULATE RESISTOR

;******************* OUTPUT FOR DEMONSTRATION PURPOSE ONLY *******************

        MOV     XA,RESIST               ;RESISTOR IN XA

        XCH     XA,BC                   ;INVERT XA
        MOV     A,B
        NOT     A
        MOV     X,A
        MOV     A,C
        NOT     A

        GETI    SELM15
        MOV     PORT4,XA                ;OUTPUT RESISTOR
LOOP:
        NOP
        BR      LOOP                    ;ENDLESS LOOP

;******************* SUBROUTINE DISCHARGE CAPACITOR ***************************

CDIS:
        MOV     A,#00H
        MOV     PORT3,A                 ;CLEAR PORT3
        MOV     XA,#00001111B
        MOV     PMGA,XA                 ;PORT3 AS OUTPUT

        MOV     A,#0FH                  ;WAIT TO DISCHARGE CAPACITOR
WAIT:
        DECS    A
        BR      WAIT

        MOV     XA,#00000100B
        MOV     PMGA,XA                 ;PORT3.2 AS OUTPUT (LOW)
        CLR1    IRQ0
        RET
```

```
;**************** SUBROUTINE FOR CAPACITOR CHARGE TIME *********************

TIME:
        SKTCLR  IRQ0                    ;INTERRUPT FLAG SET?
        BR      TIME
        MOV     XA,#11100000B
        MOV     TM0,XA                  ;STOP TIMER
        MOV     XA,T0                   ;READ TIME

        GETI    SELM0                   ;SUB TIMER VALUE -1
        MOV     BC,XA                   ;REFERS TO CPU CLOCK
        MOV     XA,#01H
        MOV     TIME1,XA
        MOV     HL,#TIME1
        CLR1    CY
        MOV     A,C
        SUBC    A,@HL
        XCH     A,@HL
        INCS    L
        NOP
        MOV     A,B
        SUBC    A,@HL
        XCH     A,@HL
        MOV     XA,TIME1
        RET

END
```

```
NAME    CALC

;
;*************************************************************************
;NAME            :CALCULATE
;DESC            :CALCULATE AND ROUND VARTIM*100/REFTIM (PERCENTAGE)
;DEVICE          :UPD75X (EXEPT UPD75P402)
;MEMORY BANK     :0,15
;HARDWARE USED   :
;INPUT           :
;OUTPUT          :
;DESTROY         :XA,BC,DE,HL
;CALLS           :
;*************************************************************************

        PUBLIC  CALC

        EXTRN   DATA    (REFTIM,VARTIM,RESIST)
        EXTRN   DATA    (MWORKA,MWORKB,MWORKC,ADDRND,DWORKA,DWORKB,DWORKC)
        EXTRN   CODE    (SELM0,SELM1,SELM15)

;********************** MULTIPLIKATION: MESTIME X 100 **********************

C2SEG   CSEG    INBLOCK
CALC:
        MOV     XA,#64H
        MOV     MWORKA,XA
        MOV     XA,VARTIM
        MOV     MWORKC,XA

        MOV     XA,#00H
        MOV     MWORKB,XA
        MOV     C,#01H
MLOOP:
        MOV     B,#03H
        MOV     HL,#MWORKB
        MOV     A,#00H
SHLOOP:
        XCH     A,@HL
        INCS    L
        NOP
        DECS    B
        BR      SHLOOP

        XCH     A,B
FIGURE:
        DECS    B
        BR      MADD

        DECS    C
        BR      MLOOP
        BR      AR
```

```
MADD:
        PUSH    BC
        CLR1    CY
        MOV     B,#01H
        MOV     DE,#MWORKC
        MOV     HL,#MWORKB
ADLOOP:
        MOV     A,@DE
        ADDC    A,@HL
        XCH     A,@HL
        INCS    L
        NOP
        INCS    E
        NOP
        DECS    B
        BR      ADLOOP

        POP     BC
        MOV     HL,#MWORKA
        NOT1    CY
        SKT     CY
INCA:
        INCS    @HL
        BR      FIGURE

        INCS    L
        NOP
        BR      INCA
```

```
;******** ADDITION (VARTIM*100 + REFTIM/2 (TO ROUND THE RESULT) ***********

AR:
        MOV     XA,REFTIM
        MOV     HL,#(ADDRND + 1)
        MOV     B,A
        MOV     A,X
        CLR1    CY
        RORC    A
        XCH     A,@HL
        DECS    L
        NOP
        MOV     A,B
        RORC    A
        XCH     A,@HL
        MOV     XA,#00H
        MOV     (ADDRND + 2H),XA

        MOV     DE,#ADDRND
        MOV     HL,#MWORKB
        MOV     B,#03H
        CLR1    CY
ADDI:
        MOV     A,@DE
        ADDC    A,@HL
        XCH     A,@DE
        INCS    L
        NOP
        INCS    E
        NOP
        DECS    B
        BR      ADDI
```

```
;********************** DIVISION: TIMEMES100/TIMEREF ************************

        MOV     XA,REFTIM
        MOV     DWORKC,XA
        MOV     XA,#00H
        MOV     (DWORKC+2H),XA
        MOV     XA,ADDRND
        MOV     DWORKA,XA
        MOV     XA,(ADDRND + 2)
        MOV     (DWORKA+2),XA

        MOV     DE,#DWORKB
        MOV     C,#03H
CLRB:
        MOV     A,#00H
        XCH     A,@DE
        INCS    E
        NOP

        DECS    C
        BR      CLRB

        MOV     B,#0CH
DLOOP:
        MOV     DE,#DWORKA
        MOV     A,#00H
        MOV     C,#07H
SHFT:
        XCH     A,@DE
        INCS    E
        NOP
        DECS    C
        BR      SHFT
SUB:
        MOV     DE,#DWORKB
        MOV     HL,#DWORKC
        CLR1    CY
        MOV     C,#03H
SLOOP:
        MOV     A,@DE
        SUBC    A,@HL
        XCH     A,@DE
        INCS    L
        NOP
        INCS    E
        NOP
        DECS    C
        BR      SLOOP
```

```
        NOT1    CY
        SKT     CY
        BR      D0

        MOV     DE,#DWORKA
        XCH     A,@DE
        ADDS    A,#01H
        XCH     A,@DE
        BR      SUB
D0:
        MOV     DE,#DWORKB
        MOV     HL,#DWORKC

        MOV     C,#03H
ADD:
        MOV     A,@DE
        ADDC    A,@HL
        XCH     A,@DE
        INCS    L
        NOP
        INCS    E
        NOP
        DECS    C
        BR      ADD
        INCS    B
        BR      DLOOP

        MOV     DE,#DWORKA
        MOV     A,@DE
        MOV     X,A
        INCS    E
        NOP
        MOV     A,@DE
        XCH     A,X
        MOV     RESIST,XA
        RET

END
```

Image Compression and Expansion with the μPD72185
A Software Guide

Contents:

Author: Uwe Schaefer
Application μCOM Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD72185 User's Manual
μPD72185 Data Sheet

Related Products

μPD72185 Advanced Compression/ Expansion Engine
EB-72185 Evaluation Board for μPD72185

1. Outline

This application note explains how to program the μPD72185 for compression and expansion of binary images using MH, MR and MMR coding methods. It refers to the CDDC.EXE program running on the EB-72185. This program was written in "C" including some assembler modules.

2. Operation of CDDC.EXE

Syntax: CDDC / /mr /mr /mmr \ in-file out-file
(this option is necessary on coding only)

/mh:	generates MH	(modified Huffman) code
/mr:	generates MR	(modified relative element) code
/mmr:	generates MRR	(modified MR) code

Some status and statistical information about the coding/decoding process will be given during CDDC execution.

3. CDDC File Format

The image as well as the different code files start with a 26 byte wide header containing information about the image and code format. These 16 bytes have the following meaning:

Byte no.		
0:	Filetype	0 = Image, 1 = MH Code, 2 = MR Code, 3 = MMR Code
1:	low byte	
2:	high byte	number of pixels per horizontal line
3:	low byte	
4:	high byte	number of lines
5:	low byte	
6:	high byte	number of bytes per horizontal line
7:	0	(fixed)
8:	low byte	
9:	high byte	(optional) density (e.g. 200 DPI)
10:	low byte	
11:	high byte	(optional) minimum number of transmission bits per line
12:	0	
13:	0	
14:	0	
15:	0	(fixed)

The image itself must have the following format:

— white pixels represented by zeros

— black pixels represented by ones

— the leftmost pixel of a line is stored in ascending order from the LSB of a byte/word

Storage Example

Imageformat:

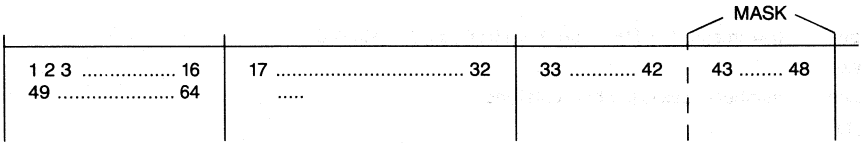
1 2 3 4 5 6 7 8 9 10 11 12 13 24

Fileformat:

	MSB							LSB	
Address	7	6	5	4	3	2	1	0	(bit no.)
0H	8	7	6	5	4	3	2	1	<— First data
1H	16	15	14	13	12	11	10	9	
2H	24	23	22	21	20	19	18	17	

— If the line length is not a multiple of 16 the remaining bits of the last word of a line are masked by the μPD72185 and the next line starts at bit 0 of the next word.

Example: line length = 42



write_head_recode ( )

Function to write output file header

initialize ( )

Function to assign system parameters. Parameters for each command are described below.

SYS command parameters

im8: Specifies image memory bus width.
Set to 1 for word mode access.

wait: Programmable wait states.
Set to 0 for wait states.

acs: Bus access priority.
Set to 3 for complete bus occupation.

sref: refresh enable
Set to 0 for refresh timing output.

imhl: Number of words of one horizontal line of image memory.
Set to the same value as prcl parameter.

Parameter of MOD command

cntcd: Not used in page mode operation.

erop: Processing after error detection.
Set to 1 to erase error lines and replace them by the next line.

rtc: Page end processing selection.
Set to 1 to add an RTC code to the end of the page.

cont: Continuation of processing.
Set to 1 to sweep out all code to the host.

fstln: First line processing.
Set to 1 to always start with the first line for page mode operation.

ii_hi: Processing mode specification.
Set to 0 for process execution between host bus and image memory bus.

lg_shr: Enlargement/ reduction of image.
Set to 0 for 1:1 operation.

kp: k counter
Used only in MR mode and time-division processing if process is halted somewhere in the k lines. Set to 1 in MR mode.

k: k parameters
Set to 4 in MR mode for coding one line one-dimensional and the following three lines two-dimensional.

SIMB command parameters

rlta: Start address of reference line.
Set to 0H but ignored in the used modes.

imba: Start address of image buffer.
Set to 0H.

imbs: Size of image buffer.
Set to the number of lines of the image to be precessed. The actual number of lines is taken from the header information of the processed image or code file.

SCDB command parameters

cdbba: Beginning bit location of code buffer.
Set to 0.

cdba: Start address of code buffer.
Set to 0 but not used due to code buffer location of host side.

cdbs: Size of code buffer.
Set to twice the value of the uncompressed image size. If the code size exceeds this value, coding will be aborted.

BLO command parameters

tcd: Top code data.
Not used due to code buffer beginning bit location at bit 0.

fil: Minimum number of transmission bits.
Set to the appropriate value specified in the image header. Used only for image compression.

f_skip: Initial error line skip processing.
Set to 0 for no skip processing.

f_err: Error setting.
Set to 0 for no error setting. Only used for time-division processing.

prcl: Number of words per line to be processed.
Calculated by line length parameter of the image or code file header.

rbit: Number of invalid bits on right end of the image.
If the line length is not a multiple of 16, rbit masks the remaining bits.

lbit: Number of invalid bits on left end of image.
Set to 0H for no invalid bits.

rmsk, lmsk: Number of words for white mask on right and left ends.
Both set to 0H for no white mask processing.

process ( )

Executes SYS command and coding or decoding function.

coding ( )

Main function for image compression.

First the image is loaded into on-board memory by the function load _image ( ). Then MOD, SIMB, SCDB and BLO commands are issued successively and the return parameters are checked for errors. If the image was coded correctly statistical information of the coding process like image and code size and compression ratio are printed on the screen.

decoding ( )

Main function for image expansion. MOD, SIMD, SCDB and SPRS command for resetting the statistical information table of the ACEE are issued successively. Function blod_cmd ( ) then issues BLO command and sends the compressed code to the ACEE. The SPRS command reads the statistical information table to check whether errors occured during image expansion. Based on this statistical information of the expansion process is printed on the screen.

sys_cmd ( ) Function to issue SYS command and parameters.

mod_cmd ( ) Function to issue MOD command and parameters.

simb_cmd ( ) Function to issue SIMB command and parameters.

scdb_cmd ( )	Function to issue SCDB command and parameters.
tro_cmd ( )	Function to issue TRO command and parameters for data transfer.
sprs_cmd ( )	Function to clear and read statistical information table of the ACEE.
bloc_cmd ( )	Function to issue BLO command and parameters for coding operation. Furthermore function save_code ( ) is called to store the generated code in a file.
blod_cmd ( )	Function to issue BLO command and parametersfor decoding. Also function load_code ( ) is called to read the code from a file and send it to the ACEE.
cmd_exe ( )	Function to start command execution of any command. After resetting the INTRST bit, CRQ bit is set and the interrupt request bit is polled. At the end, the response parameters are read into rsp_buf [ ].

load_image (long adr, long lines)

parameters:

adr:	Start address of image buffer.
lines:	Number of lines to be transfered.

Issues the appropriate commands for data transfer and transfers the image from a file into on-board memory. For the transfer operation assembler module out_bytes ( ) is used.

save_image (long adr, long lines)

parameters:

adr:	Start address of image buffer.
lines:	Number of lines to be transfered.

Function equal to load_image ( ) but with opposite transfer direction. The on-board memory is read and stored in a file. Transfer is done by assembler module inbytes ( ). Furthermore, the header with the image information is written to the image file. In case of errors in the code file, the expanded image will be shortened by the error lines and the reduced image size will be stored in the header.

load_code ( )

Function to send code from a file to the ACEE for image expansion. It is important to note that after transfer of the hole code, up to 6 additional words of dummy data have to be transfered until the INTR bit is set. This is due to the face that the ACEE reads up to 6 words of code in advance for internal pipeline operation. Also, if the code buffer is located in the image memory, its size must be defined six words larger to prevent code buffer empty errors.

save_code ( )

Function that receives code from the ACEE and stores it in a file.

4.2 Assembler Modules of CDDC

To perform fast data transfers between the host memory and the ACEE FiFo, the transfer parts of the program were written in assembler. These modules were assembled with the MASM using /mx option for upcase, lowcase sensitive translation.

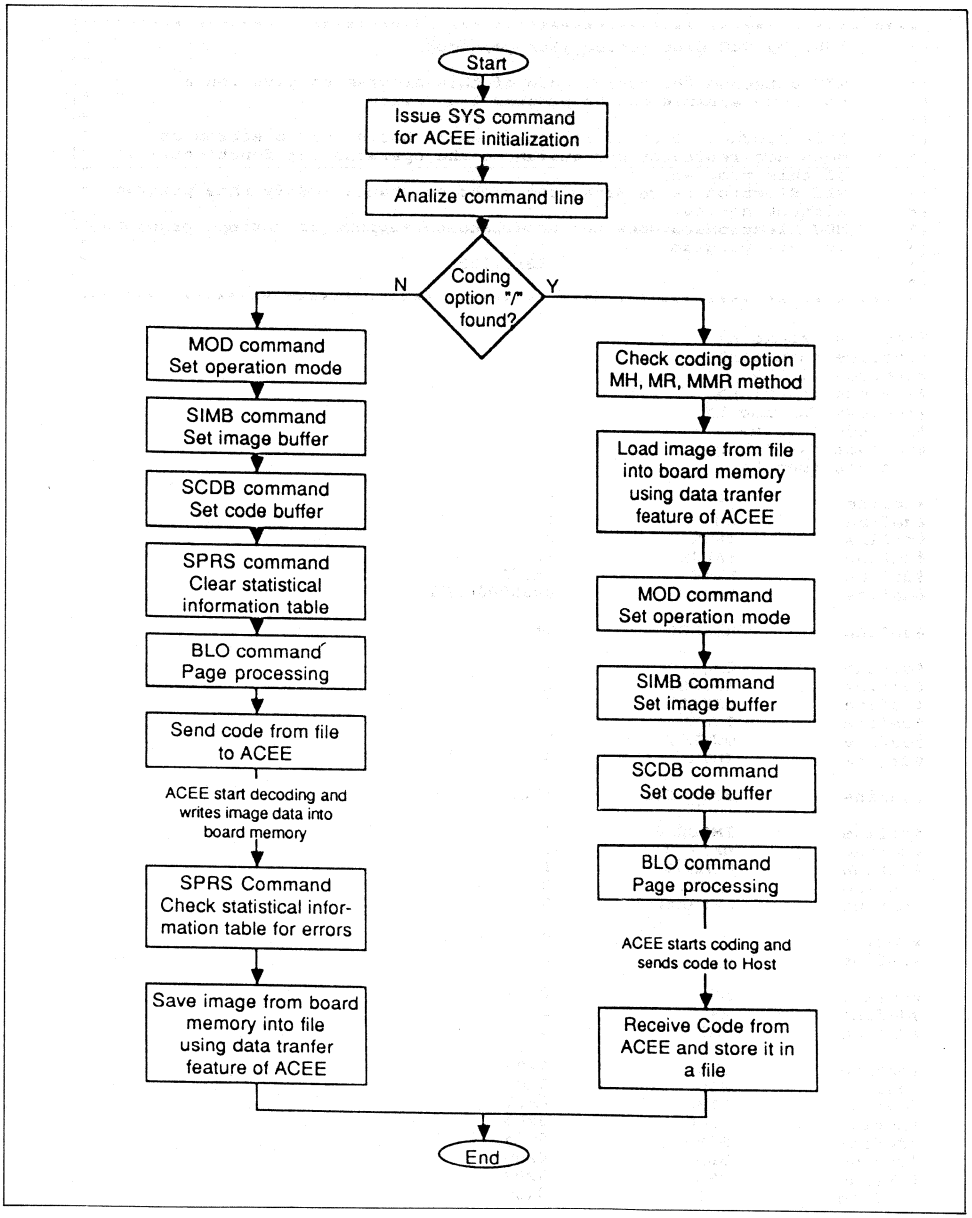
_inbytes ( )

Module to transfer data from ACEE FiFo into Host memory. The start address in the Host memory is determinded by the global variable "buffer" and the size in units of bytes by the variable "_trans". The transfer stops after transmission of all specified bytes or when the INTR bit is set. The function returns the number of bytes actually transfered.

_outbytes ( )

Module to transfer data from the Host memory into the ACEE FiFo. The start address, number of bytes and return value are the same as in function _inbytes ( ).

5. Software Flowcharts



6. Program Listings

```
/*************************************************************************}
{*      1989 by NEC Electronics (Europe) GmbH                            *}
{*                                                                       *}
{*      NEC attached the source code of this program to give you a       *}
{*      concrete example how to program the device.                      *}
{*                                                                       *}
{*      This product is delivered on an as-is basis. NEC Electronics     *}
{*      does not represent nor guarantee the operation or functionality  *}
{*      of this program.                                                 *}
{*      NEC Electronics reserves the right to change/modify this program *}
{*      without notice.                                                  *}
{*      NEC Electronics does not undertake to support or correct problems *}
{*      in this program.                                                 *}
{*                              EB-72185                                 *}
{*                                                                       *}
{*************************************************************************/

#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <string.h>
#include <dos.h>
#include <stdlib.h>
#include <mem.h>

#define         OFF             0
#define         ON              1
#define         TRUE            0
#define         FALSE           -1
#define         BYTE            0x00ff
#define         LBYTE           0x000000ffL

#define         HEADER          16

#define         CRQ             1
#define         INTRST          2
#define         SFTRST          4
#define         INRDY           1
#define         OUTRDY          2
#define         INTR            4

#define         BUF_SIZE        51200

#define         IMBUS16         1
#define         WAIT_0          0
#define         DEMAND          1
#define         OCCUPIE         3
#define         REFRESH         0

#define         CD              0
#define         DC              1

#define         MH              0
#define         MR              1
#define         MMR             2

#define         SYS             0x00
#define         MOD             0x01
#define         SIMB            0x02
#define         SCDB            0x03
#define         SPRS            0x04
#define         BLO             0x05
#define         TRO             0x07
#define         ABT             0x0c
```

```
#define         SYSOK           0x00
#define         MODOK           0x01
#define         IMBOK           0x02
#define         CDBOK           0x03
#define         PRSOK           0x04
#define         PRSTBL          0x12
#define         BCDOK           0x20
#define         BDCOK           0x21
#define         RDCOK           0x29
#define         BCDOKABT        0x30
#define         BDCOKABT        0x31
#define         BDCABT          0x41
#define         BDCABTABT       0x51
#define         CEMPT           0x8e
#define         CFULL           0x8f
#define         CFEERR          0xfd
#define         DBLCRQ          0xfe
#define         NORESP          0xff
#define         TRNOK           0x24

#define         LOW(x)           (x & BYTE)
#define         HIGH(x)         ((x >> 8) & BYTE)
#define         LLOW(x)          (x & LBYTE)
#define         LMIDL(x)        ((x >> 8L) & LBYTE)
#define         LHIGH(x)        ((x >> 16L) & LBYTE)

#define         ACEE(x)         outp(acee_io,x)

#define         IM8(x)           (x & 0x01)
#define         WAIT(x)         ((x & 0x03) << 1)
#define         ACS(x)          ((x & 0x03) << 3)
#define         SREF(x)         ((x & 0x01) << 5)

#define         CDDC(x)          (x & 0x01)
#define         CNTCD(x)        ((x & 0x01) << 1)
#define         EROP(x)         ((x & 0x01) << 2)
#define         RTC(x)          ((x & 0x01) << 3)
#define         CONT(x)         ((x & 0x01) << 4)
#define         FSTLN(x)        ((x & 0x01) << 5)
#define         PROC(x)          (x & 0x07)
#define         IIHI(x)         ((x & 0x01) << 3)
#define         LGSHR(x)        ((x & 0x07) << 4)

#define         TBLCLR(x)        (x & 0x01)
#define         ABEL(x)         ((x & 0x01) << 1)
#define         ABTEL(x)        ((x & 0x01) << 2)

#define         PRCL(x)         ((x % 16) == 0)?(x/16):((x/16)+1)

unsigned char   im8, wait, acs, sref, imhl ;

unsigned char   cd_dc ;
unsigned char   cntcd ;
unsigned char   erop ;
unsigned char   rtc ;
unsigned char   cont ;
unsigned char   fstln ;
unsigned char   proc ;
unsigned char   ii_hi ;
unsigned char   lg_shr ;
unsigned char   kp ;
unsigned char   k ;
unsigned char   tel ;
unsigned char   mxel ;
unsigned char   f_cd ;
```

```
unsigned long   rlta ;
unsigned long   imba ;
unsigned long   imbs ;
unsigned long   im_size;

unsigned char   cdbba ;
unsigned long   cdba ;
unsigned long   cdbs ;
unsigned long   cdb_total ;
unsigned int    total ;

unsigned int    tcd ;
unsigned int    fil ;
unsigned char   f_skip ;
unsigned char   f_err ;
unsigned int    prcl ;
unsigned char   lmsk ;
unsigned char   rmsk ;
unsigned char   lbit ;
unsigned char   rbit ;

unsigned char   type ;
unsigned int    length ;
unsigned int    line ;
unsigned int    wide ;
unsigned char   size ;
unsigned int    scan ;
unsigned char   filler[5] ;
unsigned char   color ;

unsigned int    nlcnt ;
unsigned int    out_bytes ;
unsigned int    in_bytes ;
unsigned int    bytes_read ,bufsize;


unsigned char   image_file[64] ;
unsigned char   code_file[64] ;

struct  { char  name[4] ; }     mode[3] =       {
                                                "MH",
                                                "MR",
                                                "MMR"
                                                } ;

struct  { char  p_mesg[9] ; } prcs[2] = {
                                        "CODING",
                                        "DECODING"
                                        } ;

unsigned char   buffer[BUF_SIZE] ;
unsigned char   rsp_buf[11] ;

FILE            *fin ;
FILE            *fout ;
unsigned int    infile, outfile, trans;
unsigned int    acee_io ;

extern int outbytes(void);  /* ASM routine to write data into ACEE fifo */
                        /* global integer "trans" defines the number of bytes *
                        /* to be transfered from array "buffer" */
extern int inbytes(void); /*ASM routine to read data from ACEE fifo */
                      /* global integer "trans" defines the number of bytes */
                      /* to be transfered into array "buffer" */
                      /* return value is the actual number of bytes transfered
```

```
/*------------------------------------------------------------------------------

                                M A I N

------------------------------------------------------------------------------*/

main(argc,argv)
        int     argc ;
        char *argv[] ;
{
    if (get_io() != TRUE) exit(1) ;
        if (command_analyze(argc,argv) != TRUE) exit(1) ;
        if (process() == FALSE){
                printf("\n??? program abort\n") ;
                exit(1) ;
        }
        printf("\n========== normal end ==========\n") ;
        exit(0) ;
}
/*------------------------------------------------------------------------------*/

get_io()
{
        if ((fin=fopen("ACEE.IO","rb")) == NULL){
                printf("File ""ACEE.IO"" not found !!!\n") ;
                printf("Please run IO.EXE .\n") ;
                return(FALSE) ;
        }
        if (fread(&acee_io,2,1,fin) != 1){
                printf("\nRead error in file ""ACEE.IO"" !!!");
                printf("\nPlease run IO.EXE .\n") ;
                return(FALSE);
        }
        fclose(fin);
        return(TRUE);
}
/*------------------------------------------------------------------------------

                C O M M A N D    L I N E    A N A L Y Z E

------------------------------------------------------------------------------*/

command_analyze(cnt,string)
        int     cnt ;
        char *string[] ;
{
        int     i ;
        int     j ;

        if (cnt < 2){
                printf("CDDC [/proc] input_file output_file\n") ;
                return(FALSE) ;
        }

        if (string[1][0] == 0x2f){
                if (proc_check(&string[1][1]) != TRUE) return(FALSE) ;
                f_cd=cd_dc=CD ;
                i=2 ;
        }
        else{
                f_cd=cd_dc=DC ;
                i=1 ;
        }
```

```
        if (cnt > 2){
                j=0 ;
                while(string[i][j] != 0x00){
                        image_file[j]=string[i][j] ;
                        j++ ;
                }
                image_file[j]=0x00 ;
                if (infile_check(image_file) != TRUE) return(FALSE) ;
                i++ ;
        }
        else{
                printf("no input/output file name\n") ;
                return(FALSE) ;
        }

        if (cnt > 2){
                j=0 ;
                while(string[i][j] != 0x00){
                        code_file[j]=string[i][j] ;
                        j++ ;
                }
                code_file[j]=0x00 ;
                if (outfile_check(code_file) != TRUE) return(FALSE) ;
                return(TRUE) ;
        }
        else{
                printf("no output file name\n") ;
                return(FALSE) ;
        }
}

/*-------------------------------------------------------------------------
                p r o c e s s    m o d e    c h e c k
-------------------------------------------------------------------------*

proc_check(string)
        char string[] ;
{
        int     i ;
        int     j ;
        char found ;
        char p_d ;

        for (i=0 ; i < 3 ; i++){
                found=ON ;
                for (j=0 ; mode[i].name[j] != 0x00 ; j++){
                        p_d=string[j] ;
                        if ((p_d >= 0x61) && (p_d <= 0x7b)) p_d=p_d & 0xdf ;
                        if (p_d != mode[i].name[j]) found=OFF ;
                }
                if (found){
                        proc=(char)i ;
                        break ;
                }
        }
        if (!(found)){
                printf("bad mode name\n") ;
                return(FALSE) ;
        }
        return(TRUE) ;
}

/*-------------------------------------------------------------------------
                        i n p u t    f i l e    c h e c k
-------------------------------------------------------------------------*
```

```
infile_check(string)
        char string[] ;
{
        if ((fin=fopen(string,"rb")) == NULL){
                printf("read open error - '%s'\n",string) ;
                return(FALSE) ;
        }

        if (read_head_recode()) return(FALSE) ;
        return(TRUE) ;
}

/*-------------------------------------------------------------------------------
                read   header   recode
-------------------------------------------------------------------------------*/

read_head_recode()
{
        int             ret ;
        unsigned int    rc ;

        infile = fileno(fin);
        rc=read(infile,&type,  1) ;
        rc+=read(infile,&length,2) ;
        rc+=read(infile,&line,  2) ;
        rc+=read(infile,&wide,  2) ;
        rc+=read(infile,&size,  1) ;
        rc+=read(infile,&scan,  2) ;
        rc+=read(infile,filler, 5) ;
        rc+=read(infile,&color, 1) ;
        im_size = (long)line * (long)wide;
        ret=TRUE ;

        if (rc != HEADER){
                if (feof(fin)){
                        printf("unexpected end of inputfile\n") ;
                }
                else{
                        printf("read error at header recode\n") ;
                }
                fclose(fin) ;
                ret=FALSE ;
        }
        return(ret) ;
}

/*-------------------------------------------------------------------------------
                output   file   check
-------------------------------------------------------------------------------*/

outfile_check(string)
        char    string[] ;
{
        if ((fout=fopen(string,"r")) != NULL){
              printf("\nFile   %s   already exists !!!  ",string);
              printf("Overwrite (Y/N) [N]? ");
              if ((toupper(getche())) != 'Y')  exit(1);
              else fclose(fout);
              puts("");
        }
        if ((fout=fopen(string,"wb")) == NULL){
              printf("write open error - '%s'\n",string) ;
              return(FALSE) ;
        }
        return(TRUE) ;
```

```
}
/*-------------------------------------------------------------------------------
                write   header   recode
-------------------------------------------------------------------------------*/

write_head_recode()
{
        int             ret ;
        unsigned int    rc ;
        unsigned char   o_type ;

        if (cd_dc == CD) o_type=proc+1 ;
        else o_type=0 ;

        outfile = fileno(fout);
        rc=write(outfile,&o_type,  1) ;
        rc+=write(outfile,&length,2) ;
        rc+=write(outfile,&nlcnt, 2) ;
        rc+=write(outfile,&wide,  2) ;
        rc+=write(outfile,&size,  1) ;
        rc+=write(outfile,&scan,  2) ;
        rc+=write(outfile,filler, 5) ;
        rc+=write(outfile,&color, 1) ;

        ret=TRUE ;

        if (rc != HEADER){
                printf("write error at header recode\n") ;
                fclose(fout) ;
                ret=FALSE ;
        }
        return(ret) ;
}
/*-------------------------------------------------------------------------------

                                INITIALIZE

-------------------------------------------------------------------------------*

initialize()
{
        total=0;
        prcl=length/16 ;
        if (rbit=length%16) prcl++ ;
        if (rbit) rbit=16-rbit;

                                /* sys parameter */
        im8=IMBUS16 ;
        wait=WAIT_0 ;
        acs=OCCUPIE ;
        sref=REFRESH ;
        imhl=prcl ;

                                /* mod parameter */
        cntcd=1 ;
        erop=1 ;
        rtc=1 ;
        cont=1 ;
        fstln=1 ;
        ii_hi=0 ;
        lg_shr=0 ;
        if (cd_dc == CD){
                if (proc == MR){
                        kp=1 ;
                        k=4 ;
                }
```

```
        }
        else{
                proc=type-1 ;
                kp=0 ;
                k=0 ;
        }
                                /* simb parameter */
        rlta = 0L;
        imba = 0L;
        imbs = (long)line;  /* hole image */

                                /* scdb parameter */
        cdbba= 0 ;
        cdba = 0L;              /* code buffer on host side */
        cdbs = im_size;         /* maximum codesize is 2x original image */
        cdb_total = 0L;

                                /* blo parameter */
        tcd=0 ;
        fil = filler[0]+filler[1]*0x100 ;
        f_skip=0 ;
        f_err=0 ;
        lmsk=0 ;
        rmsk=0 ;
        lbit=0 ;
}

/*-------------------------------------------------------------------------------

                                P R O C E S S

-------------------------------------------------------------------------------*

process()
{
        unsigned long   l ;
        int             ret ;

        initialize() ;
        ACEE(SFTRST) ;
        if (sys_cmd(im8,wait,acs,sref,imhl)) return(FALSE) ;

        if (f_cd == CD){
                printf("\n========= %s CODING =========\n",&mode[proc]);
                ret=coding() ;
        }
        else{
                printf("\n========= %s DECODING =========\n",&mode[proc]) ;
                ret=decoding() ;
        }
        return(ret) ;
}

/*-------------------------------------------------------------------------------
                        c o d i n g    p r o c e s s
-------------------------------------------------------------------------------*

coding()
{
        int     ret ;

        if (load_image(imba,imbs))  return(FALSE) ;
        if (mod_cmd(CD,cntcd,erop,rtc,cont,fstln,proc,ii_hi,lg_shr,kp,k))
                return(FALSE);
        if (simb_cmd(rlta,imba,imbs) != TRUE) return(FALSE) ;
        if (scdb_cmd(cdbba,cdba,cdbs)) return(FALSE) ;
```

```
        ret=bloc_cmd(tcd,fil,prcl,lmsk,rmsk,lbit,rbit) ;

        if  (ret == BCDOK){
                total= rsp_buf[1] + rsp_buf[2] *256;
        }
        else if(ret == CFULL){
                printf("\nCodesize exceeds uncompressed imagesize more ");
                printf("than 2 times !!!");
                printf("\nCoding aborted.");
                return(FALSE);
         }
         else{
                print_response(rsp_buf);
                printf("\nACEE error !!!");
                return(FALSE);
         }
         fclose(fin) ;
         fclose(fout) ;

         printf("\n\npixel per line  : %u"        , length) ;
         printf("\ntotal lines     : %u"        , total) ;
         printf("\ntotal imagesize : %ld bytes", im_size) ;
         printf("\ntotal codesize  : %ld bytes",cdb_total) ;
         printf("\ncompress factor : %3.2f\n",(float)im_size/(float)cdb_total);
         return(TRUE) ;
}

/*-------------------------------------------------------------------------------
                        d e c o d i n g    p r o c e s s
-------------------------------------------------------------------------------*/

decoding()
{
    unsigned int ret;
    rtc=0;

    if (mod_cmd(DC,cntcd,erop,rtc,cont,fstln,proc,ii_hi,lg_shr,kp,k))
        return(FALSE) ;
    if (simb_cmd(rlta,imba,imbs) != TRUE) return(FALSE) ;
    if (scdb_cmd(cdbba,cdba,cdbs)) return(FALSE) ;
    if (sprs_cmd(1,0,0,0,0,0,0,0,0) != TRUE) return(FALSE) ;

    ret=blod_cmd(f_skip,f_err,prcl,lbit,rbit) ;
    if (ret==BDCOK || ret==RDCOK){
        if (sprs_cmd(1,0,0,0,0,0,0,0,0) != TRUE) return(FALSE) ;
        nlcnt=rsp_buf[4] + rsp_buf[5]*256 ;
        mxel=rsp_buf[1];
        tel=rsp_buf[2];
        ret=save_image(0L,(long)nlcnt) ;
        im_size=(long)nlcnt*(long)imhl*2;
        if(tel){
                printf("\n\nError in Codefile !!!");
                printf("\n\npixel per line          : %u" , length) ;
                printf("\ntotal lines of org. image : %u" , line) ;
                printf("\nmax. of cont. error lines : %hu", mxel);
                printf("\ntotal error lines         : %hu", tel);
                printf("\nnormally processed lines  : %u ", nlcnt);
        }
        else{
                printf("\n\npixel per line            : %u" , length) ;
                printf("\ntotal lines               : %d" , nlcnt) ;
        }
        printf("\ntotal imagesize   : %ld bytes" , im_size) ;
        printf("\ntotal codesize    : %ld bytes",cdb_total) ;
        printf("\ncompress factor   : %3.2f\n",(float)im_size/(float)cdb_total
```

```
    }
    else{
        print_response(rsp_buf);
        printf("\nACEE error !!!");
        return(FALSE);
    }
    fclose(fin) ;
    fclose(fout) ;

    return(TRUE) ;
}

/*-------------------------------------------------------------------------------
                        s y s    c o m m a n d
-------------------------------------------------------------------------------*/

sys_cmd(p1,p2,p3,p4,p5)

        unsigned char   p1,     /* im8          */
                        p2,     /* wait         */
                        p3,     /* acs          */
                        p4 ;    /* sref         */
        unsigned int    p5 ;    /* imhl         */
{
        outp(acee_io+1,SYS) ;
        outp(acee_io+2,(SREF(p4)|ACS(p3)|WAIT(p2)|IM8(p1))) ;
        outp(acee_io+3,LOW(p5)) ;
        outp(acee_io+4,HIGH(p5)) ;

        if (cmd_exe(rsp_buf) != SYSOK) return(FALSE) ;

        return(TRUE) ;
}

/*-------------------------------------------------------------------------------
                        m o d    c o m m a n d
-------------------------------------------------------------------------------*/

mod_cmd(p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11)

        unsigned char   p1,     /* cd/dc        */
                                p2,     /* cntcd        */
                                p3,     /* erop         */
                                p4,     /* rtc          */
                                p5,     /* cont         */
                                p6,     /* lstln        */
                                p7,     /* proc         */
                                p8,     /* ii/hi        */
                                p9,     /* lg/shr       */
                                p10,    /* kp           */
                                p11 ;   /* k            */
{

       outp(acee_io+1,MOD) ;
       outp(acee_io+2,(FSTLN(p6)|CONT(p5)|RTC(p4)|EROP(p3)|CNTCD(p2)|CDDC(p1)))
       outp(acee_io+3,(LGSHR(p9)|IIHI(p8)|PROC(p7))) ;
       outp(acee_io+4,p10) ;
       outp(acee_io+5,p11) ;

       if (cmd_exe(rsp_buf) != MODOK) return(FALSE) ;

       return(TRUE) ;
}
```

```
/*-------------------------------------------------------------------------------
                        s i m b    c o m m a n d
-------------------------------------------------------------------------------*/

simb_cmd(p1,p2,p3)

        unsigned long   p1,     /* rlta         */
                        p2,     /* imba         */
                        p3 ;    /* imbs         */
{
        outp(acee_io+1,SIMB) ;
        outp(acee_io+2,LLOW(p1)) ;
        outp(acee_io+3,LMIDL(p1)) ;
        outp(acee_io+4,LHIGH(p1)) ;
        outp(acee_io+5,LLOW(p2)) ;
        outp(acee_io+6,LMIDL(p2)) ;
        outp(acee_io+7,LHIGH(p2)) ;
        outp(acee_io+8,LLOW(p3)) ;
        outp(acee_io+9,LMIDL(p3)) ;
        outp(acee_io+10,LHIGH(p3)) ;

        if (cmd_exe(rsp_buf) != IMBOK) return(FALSE) ;

        return(TRUE) ;
}

/*-------------------------------------------------------------------------------
                        s c d b    c o m m a n d
-------------------------------------------------------------------------------*/

scdb_cmd(p1,p2,p3)

        unsigned char   p1 ;    /* cdbba        */
        unsigned long   p2,     /* cdba         */
                        p3 ;    /* cdbs         */
{
        outp(acee_io+1,SCDB) ;
        outp(acee_io+2,(p1 & 0x0f)) ;
        outp(acee_io+3,LLOW(p2)) ;
        outp(acee_io+4,LMIDL(p2)) ;
        outp(acee_io+5,LHIGH(p2)) ;
        outp(acee_io+6,LLOW(p3)) ;
        outp(acee_io+7,LMIDL(p3)) ;
        outp(acee_io+8,LHIGH(p3)) ;

        if (cmd_exe(rsp_buf) != CDBOK) return(FALSE) ;

        return(TRUE) ;
}

/*-------------------------------------------------------------------------------
                        t r o    c o m m a n d
-------------------------------------------------------------------------------*/

tro_cmd(p1)
    unsigned int    p1 ;    /* prcl */
{
        outp(acee_io+1,TRO) ;
        outp(acee_io+7,LOW(p1)) ;
        outp(acee_io+8,HIGH(p1)) ;
        ACEE(INTRST);
        ACEE(CRQ);

        return(TRUE) ;
}
```

```
/*-------------------------------------------------------------------------
                        s p r s    c o m m a n d
-------------------------------------------------------------------------*/

sprs_cmd(p1,p2,p3,p4,p5,p6,p7,p8,p9)

        unsigned char   p1,             /* tblclr       */
                        p2,             /* abel         */
                        p3,             /* abtel        */
                        p4,             /* mxel         */
                        p5,             /* tel          */
                        p6 ;            /* cel          */
        unsigned int    p7 ;            /* wdl          */
        unsigned char   p8,             /* ell          */
                        p9 ;            /* tell         */
{
        outp(acee_io+1,SPRS) ;
        outp(acee_io+2,(ABTEL(p3)|ABEL(p2)|TBLCLR(p1))) ;
        outp(acee_io+3,p4) ;
        outp(acee_io+4,p5) ;
        outp(acee_io+5,p6) ;
        outp(acee_io+6,LOW(p7)) ;
        outp(acee_io+7,HIGH(p7)) ;
        outp(acee_io+8,p8) ;
        outp(acee_io+9,p9) ;

        if (cmd_exe(rsp_buf) != PRSTBL) return(FALSE) ;

        return(TRUE) ;
}

/*-------------------------------------------------------------------------
              b l o    c o m m a n d    ( c o d i n g )
-------------------------------------------------------------------------*

bloc_cmd(p1,p2,p3,p4,p5,p6,p7)

        unsigned int    p1,             /* top code data        */
                        p2,             /* fil                  */
                        p3 ;            /* prcl                 */
        unsigned char   p4,             /* lmsk                 */
                        p5,             /* rmsk                 */
                        p6,             /* lbit                 */
                        p7 ;            /* rbit                 */
{
        outp(acee_io+1,BLO) ;
        outp(acee_io+2,LOW(p1)) ;
        outp(acee_io+3,HIGH(p1)) ;
        outp(acee_io+4,LOW(p2)) ;
        outp(acee_io+5,HIGH(p2)) ;
        outp(acee_io+7,LOW(p3)) ;
        outp(acee_io+8,HIGH(p3)) ;
        outp(acee_io+9,p4) ;
        outp(acee_io+10,p5) ;
        outp(acee_io+11,((p6 << 4)|(p7 & 0x0f))) ;

 /* print_command() ;*/
        ACEE(INTRST) ;
        ACEE(CRQ) ;

        if (save_code(out_bytes) != TRUE) return(FALSE) ;
        read_response(rsp_buf);

        return(rsp_buf[0]) ;
}
```

```
/*-------------------------------------------------------------------------
            b l o   c o m m a n d   ( d e c o d i n g )
-------------------------------------------------------------------------*/

blod_cmd(p1,p2,p3,p4,p5)

        unsigned char   p1,             /* f_skip       */
                        p2 ;            /* f_err        */
        unsigned int    p3 ;            /* prcl         */
        unsigned char   p4,             /* lbit         */
                        p5 ;            /* rbit         */
{
        outp(acee_io+1,BLO) ;
        outp(acee_io+6,((p2 << 1)|(p1 & 0x0f))) ;
        outp(acee_io+7,LOW(p3)) ;
        outp(acee_io+8,HIGH(p3)) ;
        outp(acee_io+11,((p4 << 4)|(p5 & 0x0f))) ;

 /* print_command() ;*/

        ACEE(INTRST) ;
        ACEE(CRQ) ;

        if (load_code() != TRUE) return(FALSE);
        read_response(rsp_buf);

 /* print_response(rsp_buf) ;*/

        return(rsp_buf[0]) ;
}

/*-------------------------------------------------------------------------
        c o m m a n d   e x e c u t e   &   r e s p o n s e   r e a d
-------------------------------------------------------------------------*/

cmd_exe(rspbuf)

        unsigned char   rspbuf[] ;
{
        int     i ;
        int     j ;
        char    stat ;

 /* print_command() ;*/

        ACEE(INTRST) ;
        ACEE(CRQ) ;

        stat=0 ;
        while (!(stat & INTR)){
                stat=inp(acee_io) ;
        }

        if (stat & INTR){
                j=acee_io+1 ;
                for (i=0 ; i < 11 ; i++){
                        rspbuf[i]=inp(j) ;
                        j++ ;
                }
        }

 /* print_response(rsp_buf) ;*/

        return((int)rspbuf[0]) ;
}
```

```
/*-------------------------------------------------------------------------------
                        l o a d    i m a g e    d a t a
-------------------------------------------------------------------------------*/

load_image(adr,lines)
        unsigned long   adr ;
        unsigned long   lines ;
{
    unsigned int    rc ;
    long   words, bytes;

    words = (long)imhl * lines;
    bytes = words*2;

    if (mod_cmd(0,0,0,0,0,0,1,0,0,0,0)) return(FALSE) ;
    if (simb_cmd(rlta,adr,words) != TRUE) return(FALSE) ;
    if (tro_cmd(prcl) != TRUE) return(FALSE) ;
    printf("\nLoading image file ...");
    do{
          if (bytes > (long)BUF_SIZE) trans = BUF_SIZE;
          else                        trans = (int)bytes;
          rc = read(infile,buffer,trans);
          if (rc != trans){
                  printf("\nread error at imagefile !!!\n") ;
                  return(FALSE);
          }
          out_bytes = outbytes();
          bytes -= (long)out_bytes;
     }
     while(bytes);
     return(TRUE);
}

/*-------------------------------------------------------------------------------
                        s a v e    i m a g e    d a t a
-------------------------------------------------------------------------------*/

save_image(adr,lines)
        unsigned long   adr ;
        unsigned long   lines ;
{
    unsigned int    rc ;
    long            words, bytes;

    words = (long)imhl * lines;
    bytes = words*2;

    if (mod_cmd(0,0,0,0,0,0,0,0,0,0,0)) return(FALSE) ;
    if (simb_cmd(rlta,adr,words) != TRUE) return(FALSE) ;
    if (tro_cmd(prcl) != TRUE) return(FALSE) ;
    printf("\nStoring image file ...");
    if (write_head_recode()) return(FALSE) ;
    do{
        if (bytes > (long)BUF_SIZE) trans = BUF_SIZE;
        else                        trans = (int)bytes;
        inbytes();
        rc = write(outfile,buffer,trans);
        if (rc != trans){
                printf("\nwrite error at output imagefile!!!\n") ;
                return(FALSE);
        }
        bytes -= (long)trans;
     }
     while(bytes);
     return(TRUE);
```

```
}
/*-------------------------------------------------------------------------------
                        l o a d    c o d e    d a t a
-------------------------------------------------------------------------------*/

load_code()
{
        unsigned int rc,loop,i;

        trans=BUF_SIZE;
        cdb_total = 0L;

        do{
              printf("\nLoading code file ...");
              rc = read(infile,buffer,trans);
              trans = rc;
              printf("\nDecoding ...");
              out_bytes = outbytes();
              cdb_total += (long)trans ;
        }
        while (rc == BUF_SIZE);  /* transfer hole code */
        if (rc == -1){
                  printf("\nRead error at input codefile !!!\n") ;
                  return(FALSE);
        }

        trans=12;
        out_bytes=outbytes(); /* ACEE reads up to 6 words code in advance.
                                 Therefore up to 6 words of dummy data
                                 have to be transfered additionally
                                 until INT bit is set */

        return(TRUE) ;
}

/*-------------------------------------------------------------------------------
                        s a v e    c o d e    d a t a
-------------------------------------------------------------------------------*/

save_code()
{
    unsigned int    rc ;

    nlcnt=line;

    if (write_head_recode()) return(FALSE) ;
    trans=BUF_SIZE;
    printf("\nCoding ...");
    in_bytes=inbytes() ;

    while (in_bytes){
        printf("\nStoring Code ...");
        rc=write(outfile,buffer,in_bytes) ;
        if (rc != in_bytes){
                printf("\nwrite error at code data") ;
                return(FALSE) ;
        }

        cdb_total+=(long)in_bytes ;
        if (in_bytes < BUF_SIZE) break ;
        printf("\nCoding ...");
        in_bytes=inbytes() ;
    }

    return(TRUE) ;
}
```

```
/*-------------------------------------------------------------------------*/

print_command()
{
        unsigned int    i ;

        printf("\ncommand  :") ;
        for (i=acee_io+1 ; i < acee_io+0xc ; i++){
                printf(" %02x",inp(i)) ;
        }
}

print_response()
{
        unsigned int    i ;

        printf("\nresponse :") ;
        for (i=0 ; i < 11 ; i++){
                printf(" %02x",rsp_buf[i]) ;
        }
}


read_response(rspbuf)
        unsigned char   rspbuf[] ;
{
        int     i ;
        int     j ;

        j=acee_io+1 ;
        for (i=0 ; i < 11 ; i++){
                rspbuf[i]=inp(j) ;
                j++ ;
        }
}
```

```
        NAME    CRQ_FIFO
;                                               use MASM /mx option to assemble
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP   CONST,_BSS,_DATA
        PUBLIC  _outbytes,_inbytes
        EXTRN   _buffer:byte,_trans:word, _acee_io:word
        ASSUME  CS:_TEXT,DS:DGROUP,SS:DGROUP,ES:DGROUP

_TEXT   SEGMENT
;
INRDY           EQU     1
OUTRDY          EQU     2
INTR            EQU     4
;
_inbytes        PROC
        PUSH    DI
        MOV     DI,OFFSET _buffer
        MOV     CX,_trans
_OUTRDY_1:
        MOV     DX,_acee_io
        IN      AL,DX
        CMP     AL,OUTRDY
        JNZ     _OUTRDY_2
        ADD     DX,0CH
        IN      AX,DX
        STOSW
        DEC     CX
        DEC     CX
        JZ      _OUTRDY_3
        JMP     _OUTRDY_1
_OUTRDY_2:
        CMP     AL,INTR
        JNZ     _OUTRDY_1
_OUTRDY_3:
        MOV     AX,_trans
        SUB     AX,CX
        POP     DI
        RET

_inbytes        ENDP
;
```

```
_outbytes       PROC
        PUSH    SI
        MOV     SI,OFFSET _buffer
        mov     cx,_trans
        cld
_INRDY_1:
        MOV     DX,_acee_io
        IN      AL,DX
        CMP     AL,INRDY
        JNZ     _INRDY_3
        ADD     DX,0CH
        LODSW
        OUT     DX,AX
        DEC     CX
        DEC     CX
        JZ      _INRDY_4
        JMP     _INRDY_1
_INRDY_3:
        CMP     AL,INTR
        JNZ     _INRDY_1
_INRDY_4:
        MOV     AX,_trans
        SUB     AX,CX
        POP     SI
        RET
_outbytes       ENDP
_TEXT   ENDS
        END
```

Part B
High Complex Peripherals Application Notes

Table of Contents

Part B High Complex Peripherals Application Notes

Floppy Disk Controller μPD7265 provides compressed track Format for higher Storage Capacity

Contents:

Authors: Andreas Kohl, Peter Westerdorf
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD7265	Data Sheet
μPD765A	Data Sheet
μPD7265	Prod. Desc.
μPD765A	Appl. Note 8

Related Products

μPD7265AC	Floppy Disk Controller (IBM™ Format)	NMOS
μPD72065C	Floppy Disk Controller (IBM™ Format)	CMOS
μPD72066C	Floppy Disk Controller (ECMA Format)	CMOS
μPD7260D	Hard-/Floppy Disk Controller	NMOS
μPD7261AD	Hard Disk Controller (SMD, ST506)	NMOS
μPD9306C	Hard Disk Interface (ST506)	CMOS
μPD72067	FDC with on-chip digital/PLL	CMOS
μPD72068	FDC with DPLL and high current bus driver	CMOS

1. Introduction

The μPD7265 Floppy Disk Controller has been designed to control the ECMA/ISO compatible track format for diskettes. (ECMA = European Computer Manufacturers Association / ISO = International Organization for Standardization).

This format is also called Sony ECMA format and can greatly increase the data storage capacity of a diskette. The gain of storage depends on the data field length of a sector, the transmission data coding (FM/MFM) and the rotational speed tolerances of the applied floppy disk drive. (FM = Frequency Modulation / MFM = Modified Frequency Modulation).

The Sony ECMA track format is a subset of the older IBM™ format. Not essential parts of a track have been omitted and another has been shortent thus resulting in more storage capacity.

The continuous success of the IBM™ compatible controller μPD765A depends only on the face that it is longer on the market and that IBM™ utilizes it though the μPD7265 controller is much more advanced.

2. The Floppy Disk Controller μPD7265

2.1 ECMA/ISO Versus IBM™ Track Format

As figure 1 demonstrates, an IBM™ compatible track format starts with GAP4a, a SYNC field and the index address mark (IAM). These recorded bytes are not used by both floppy disk controllers, μPD7265A and μPD765A, when reading or writing data. Even the IBM™ compatible controller μPD765A skips part of this area on IBM™ formatted diskettes. The rest of the track (except GAP1 which length has been shortened) is the same for IBM™ and ECMA/ISO format. Though the μPD7265 doesn't skip this IBM™ formatted diskettes by this device.

The other direction is not practicable because the μPD765A skips a certain area of the ECMA/ISO formatted track and thereby jumps into the ID field of the first sector on the track. So it is not possible to read the first sector of a track but the following sectors are accessable.

For short:

— The μPD7265 ECMA/ISO compatible Floppy Disk Controller can fully read/write IBM™ compatible formatted disketes but cannot format these.

— The μPD765A IBM™ compatible Floppy Disk Controller cannot fully read/write ECMA/ISO formatted diskettes and also not format them.

2.2 Gain for Certain Data Field Length

As can be seen at figure 1 the storage capacity gain is 47 + 10 bytes in FM mode or 96 + 18 bytes in MFM mode. Figure 2 points out that the μPD7265 ECMA/ISO controller can store one additional sector for both operation modes (FM and MFM, except for the data field length of 512 bytes per sector) without modifying the length of GAP3 and thereby GAP4.

μPD765A [FM Mode]

GAP 4a	SYNC	IAM	GAP 1	SYNC	IDAM	C		S	N	C	GAP2	SYNC	DATA AM	DATA	C	GAP3	GAP
40x	6x		26x	6x		Y		E	O	R	11x	6x			R		4b
FF	00	FC	FF	00	FE	L		C		C	FF	00	FB or F8		C		

Index — Repeat N Times

μPD765A [FM Mode]

GAP 1	SYNC	IDAM	C	H	S	N	C	GAP2	SYNC	DATA AM	DATA	C	GAP3	GAP4
16x	6x		Y	D	E	O	R	11x	6x			R		
FF	00	FE	L		C		C	FF	00	FB or F8		C		

Index — Repeat N Times

μPD765A [MFM Mode]

GAP 4a	SYNC	IAM		GAP 1	SYNC	IDAM		C	H	S	N	C	GAP2	SYNC	DATA AM		DATA	C	GAP3	GAP
80x	12x	3x	FC	50x	12x	3x	FE	Y	D	E	O	R	22x	12x	3x	FB		R		4b
4E	00	C2		4E	00	A1		L		C		C	4E	00	A1	F8		C		

Index — Repeat N Times

μPD765 [MFM Mode]

GAP 1	SYNC	IDAM		C	H	S	N	C	GAP2	SYNC	DATA AM		DATA	C	GAP3	GAP4
32x	12x	3x	FE	Y	D	E	O	R	22x	12x	3x	FB		R		
4E	00	A1		L		C		C	4E	00	A1	F8		C		

Index — Repeat N Times

Figure 1. ECMA/ISO and IBM™ Track Formats for FM and MFM Mode

If it is desired to record more sectors than given in the tables of figure 2, the appropriate GAP length has to be calculated (see chapter 2.4).

		PREAMBLE	SECTOR		GAP3		POSTAMBLE	
SINGLE DENSITY (10)	128	73	188		27		232	15
	256	73	331		42		73	9
	512	73	603		58		640	4
DOUBLE DENSITY (10)	256	146	372		54		524	15
	512	146	658		84		182	9
	1024	146	1202		116		1296	4
	DATA				GAP3		GAP4	SECTOR/ TRACK

μPD765 IBM Type Standard Format

		PREAMBLE	SECTOR		GAP3		POSTAMBLE	
SINGLE DENSITY (10)	128	16	188		27		101	16
	256	16	331		42		130	9
	512	16	603		58		94	5
DOUBLE DENSITY (10)	256	32	372		54		266	16
	512	32	658		84		296	9
	1024	32	1202		116		208	5
	DATA				GAP3		GAP4	SECTOR/ TRACK

μPD7265 SONY Recommending Format

Figure 2. Recommended Values for Certain Data Field Length

2.3 ECMA/ISO Compatible Controller μPD7265 Versus IBM™ Compatible Controller μPD765A

The two Floppy Disk Controller μPD7265 and μPD765A are exactly identical except two items.

Firstly, the μPD765A controller can only recalibrate 77 tracks as the IBM™ standard states. Modern floppy disk drives (5.25'' and 3.5''), however, do have 80 tracks which habe to be recalibrated by the Floppy Disk Controller. Using such drive the μPD765A IBM™ controller will report a seek error if it does not recieve the track zero signal from the drive after 77 step pulses. This problem can be solved for the μPD765A by sending the RECALIBRATE command twice and disregarding a possible seek error after the first recalibaration command. Note that the μPD765A can perform the SEEK command over 255 tracks without problem.

The μPD7265 is able to recalibrate up to 255 tracks directly and is thereby prepared for future floppy disk drives which may have more than 80 tracks.

The second difference between these controllers is the so called 'jump over time' after detection of an index pulse. After detecting the index pulse, the μPD765A stops reading disk data for 1.2 milliseconds whereas the μPD7265 stops for 0.2 milliseconds only (both values re valid for 4 MHz Floppy Disk Controller input clock). This is done to suppress the splice point generated in GAP4a (IBM™) or GAP1 (ECMA/ISO) when formatting the diskette. (The format writing of these gaps is stopped abruptly after detecting the rising edge of the index pulse and thereby glitches will be produced.)

If these glitches would not be absorbed, the PLL (Phase Lock Loop), necessary for data recovery, would run out of range. Data reading of the first sector following the index pulse would then be impossible.

The attached calculation example shows this in detail.

Assumption: 3.5'' micro floppy disk drive
600 RPM rotational speed
6250 bytes/track (unformatted)
500 Kbit/s = > Floppy Disk Controller clock = 8 MHz

μPD765A: Jump over time at 8 MHz = 0.6 ms
μPD7265: Jump over time at 8 MHz = 0.1 ms

$$\text{Amount of bytes to jump over (}\mu\text{PD765A)} = \frac{0.6 * 6250 \text{ bytes}}{100 \text{ ms}} = 37.5 \text{ bytes}$$

$$\text{Amount of bytes to jump over (}\mu\text{PD7265)} = \frac{0.1 * 6250 \text{ bytes}}{100 \text{ ms}} = 6.25 \text{ bytes}$$

The above calculation shows transparently why the μPD765A cannot read diskettes which have been ECMA/ISO formatted. The μPD765A would start reading the first sector within GAP2 that follows the SYNC and ID field. So it is impossible to read this sector. All other sectors are accessable.

On the other side the very short jump over time of the μPD7265 allows the access of all sectors though the diskette has been formatted IBM™ compatible.

2.4 Track Format Calculation for the μPD7265

If it is desired to record more sectors on one track it will be necessary to calculate the appropriate GAP3 length to be given to the Floppy Disk Controller for the WRITE ID command.

To format a track four parameter are requested by the WRITE ID command:

— N (a code for the number of bytes per sector)
— SC (the number of sectors per track)
— GPL (length of GAP3)
— D (filler byte)

Parameter D specifies the pattern which is written during formatting into all sectors.

Parameters N and SC are dependend on the application.

Parameter GPL is unknown, if it is not listed in the data sheet table which gives the relationship of GPL versus N/SC. In the following the procedure to calculate this parameter will be given.

The main purpose of GAP3 (which length is specified by GPL) is to compensate variations of the rotational speed of the drive. GAP4 makes up the rest of the track, that remains after formatting of all sectors. As can be seen from figure 3, the smaller GAP3 and GAP4 are, the more sectors fit on the disk.

How to calculate GAP3 for μPD7265?

Example: 3.5 inch micro disk drive
17 sectors/track
256 bytes/sector
double density (500 kBit/s)
disk rotation = 600 rpm +/— 2 %

Total track capacity = 6250 Bytes +/— 2 %
= 6250 Bytes +/— 125 Bytes = >

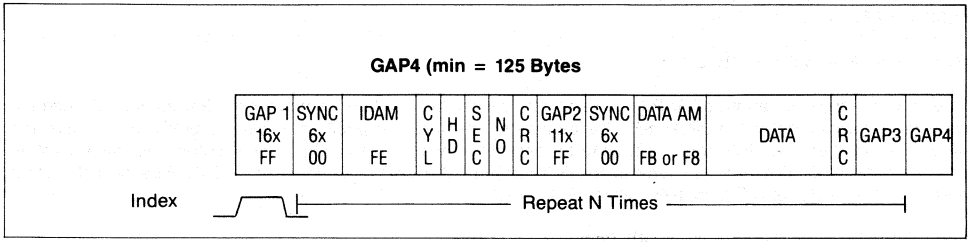


Figure 3. Detailed Track Format Layout of the μPD7265

Track equation:

6250 = GAP1 + SC * (SYNC + ID + GAP2 + SYNC + DAM + DATA + CRC + GAP3) + GAP4
6250 = 32 + 17 * (12 + 10 + 22 + 12 + 4 + 256 + 2 + GAP3) + 125
6250 = 32 + 17 * (318 + GAP3) + 125
6250 = 157 + 5406 + 17 * GAP3
6250 = 5406 + 17 * GAP3

= > GAP3 = (6250 — 5406) / 17

GAP3 = 49.7 bytes = > GAP3 (max) = 50 bytes

The calculated values are maximum (GAP3) and minimum (GAP4) values. In the example, if GAP4 is set to 125 Bytes then GAP3 may not exceed 50 Bytes. In order to get more margin it is desirable to increase the length of GAP4. This can be achieved by decreasing the length of GAP3. To determine how far GAP3 may be shortened, the rotational speed tolerance that can occur during reading/writing of one sector, will be calculated (because the sense of GAP3 is to compensate such tolerances).

Overall sector length = (SYNC + ID Field + GAP2 + SYNC + Data AM + Data + CRC)
= (12 + 10 + 22 + 12 + 4 + 256 + 2)
= 318

Sector tolerance (2 %) = (2 * 318) / 100 Bytes = 64 Bytes

= > GAP3 (min) = 7 Bytes

The optimum value of GAP3 is the mean value of GAP3 (min) and GAP3 (max).

GAP3 (opt) = (GAP3 (min) + GAP3 (max)) / 2

and in our example:

GAP3 (opt) = (7 + 50) / 2 Bytes = 28.5 Bytes

=> **GAP3 (opt) = 29 Bytes**

Using GAP3 (opt) to calculate the actual value of GAP4:

6250 = GAP1 + SC * (SYNC + ID + GAP2 + SYNC + DAM + DATA + CRC + GAP3) + GAP4
6250 = 32 + 17 * (12 + 10 + 22 + 12 + 4 + 256 + 2 + 29) + GAP4
6250 = 32 + 17 * (318 + 29) + GAP4
6250 = 5931 + GAP4

GAP4 = 6250 — 5931 = 319 bytes

Now that the physical length of GAP3 is determined, we still have to find a value for the Parameter GPL used in read/write commands of the μPD7265. This parameter specifies the position within the GAP3 field where the μPD7265 switches the R/W head from write to read mode when a sector has been written. As this operation produces splice points it must be asured to do this switching in the GAP3 area only. For the GPL Parameter therefore a value in the center of GAP3 is used. (Note: Gp13 = GPL)

GPL (Format) = (GPL (max) + GPL (min)) / 2
= (50 + 7) / 2
= 28.5 bytes

GPL (Read/Write) = GPL (format) / 2
= 29 / 2
= 14.6 bytes

=> **GPL (Format) = 29 Bytes**
=> **GPL (Read/Write) = 15 Bytes**

3. Application Board Hardware Description

The following schematic shows the absolute minimum IC consuming solution for an ECMA/ISO compatible floppy disk drive interface board.

As the utilized Floppy Disk Interface device μPB9201c is able to directly drive head load, write data and read data of floppy disk drives no additional open collector drives are necessary.

Moreover all important hardware for data reading and writing, clock generation and switching and some other convenient features are implemented into this device. So this is much more sophisticated as discrete, device consuming solutions and will also be less expensive, smaller and much more reliable.

A DMA controller has also been omitted so the Floppy Disk Controller has to be programmed for interrupt operation mode. (It is also practicable to poll the status register and check the RQM bit (RQM = ReQuest for Master) if regarding the specified access timing.)

Additionally, a motor on control must be done. It is also possible to set the device's motor on signal permanently valid decreasing the hardware additionally.

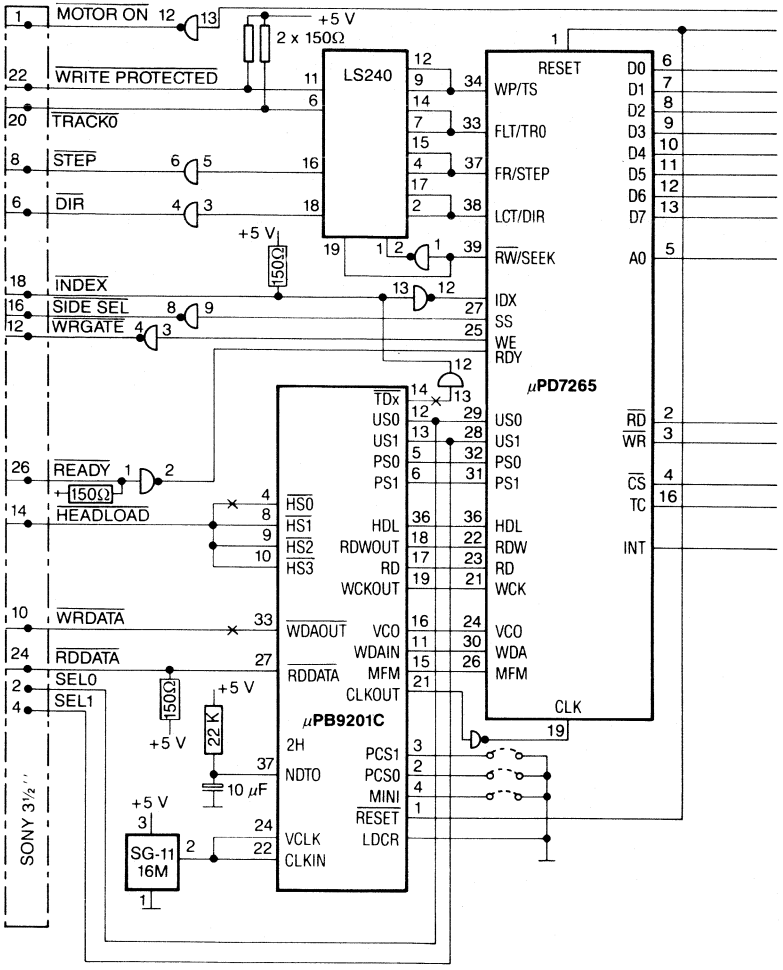


Figure 4. Floppy Controller Board for EXMA/ISO Track Format Support

Hard Disk Interface μPD9306 and Controller μPD7261A build the perfect link between Hard Disk and Computer

Contents:

Authors: Andreas Kohl, Peter Westerdorf
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD7261A	Data Sheet
μPD7261A	Prod. Desc.
μPD9306	Prod./Data Sheet

Related Products

μPD765AC	Floppy Disk Controller (IBM™ Format)	NMOS
μPD7265AC	Floppy Disk Controller (ECMA Format)	NMOS
μPD72065C	Floppy Disk Controller (IBM™ Format)	CMOS
μPD72066C	Floppy Disk Controller (ECMA Format)	CMOS
μPD7260D	Hard-/Floppy Disk Controller	NMOS

1. Introduction

The μPD7261A has been proven to be a flexible and high performance hard disk controller. It not only features confortable high level macro commands for ease of programming, it also makes very compact and low cost hardware designs possible.

A further simplification of the hardware by drastically reducing the part count against even most advanced conventional solutions, is offered by the hard disk interface μPD9306. It integrates lot of hardware needed to connect the μPD7261A to the popular ST506/ST412 drive interfaces. The μPD9306 integrates a fully digital data separation circuitry (which in the past turned out to be a complecated part of the design) the precompensation logic as well as the clock generator. It is fabricated in CMOS technology and is available in a standard 28pin package.

2. Hard Disk Interface μPD9306 Versus Conventional Solutions

In the following a short comparison of the μPD9306 against a conventional analog application will be given. Although numerous solutions have been realised already, only one state of the art approach with an analog single chip PLL (phase locked loop) will be discussed. From the performance point of view both analog and DPLL solutions show comparable results. But by looking at the costs, simplicity of design and part count the advantages of the μPD9306 solution become apparent.

2.1 Clock Generation

The μPD7261A has two clock inputs: CLK (pin 37) and R/WCLK (pin 3). The first is used for the on-chip processor, whereas the second serves as clock for the internal format controller. The format controller handles high speed operations like ECC/CRC calculation, SYNC field detection, ID field check . . . . Both clocks are generated by the μPD9306, which also performs the clock switching from read clock (RCLK) to write clock (WCLK) and vice versa.

2.2 Precompensation

In order to compensate the functional peak shift that takes place during reading data form the disk, the expected shift has to be "precompensated" when writing data to the disk. For this purpose logic has been integrated into the μPD7261A that controls whether data must be delayed or hastened. The actual delay of the data itself has to be done outside the HDC. The μPD7261A supplies the two signals PCE and PCL (precompensation early and late) to instruct external circuitry whether to delay or hasten the data. See figure 1 for the required timing. A circuitry that performs this operation is shown in figure 2.

The μPD9306 has all precompensation logic on-chip (figure 3).

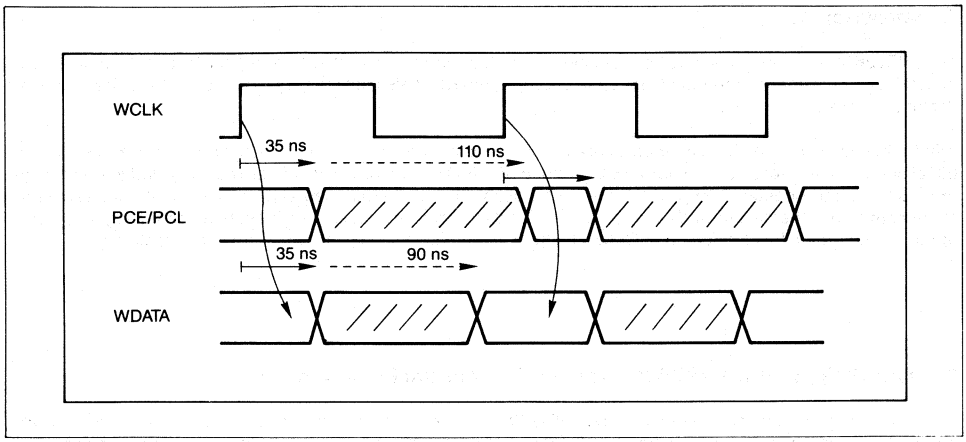


Figure 1. Precompensation Timing of μPD7261A

2.3 Data Separation

The main purpose of the phase locked loop is to derive a clock signal from the data read from the disk. This RCLK has to be phase and frequency locked with the read data stream. As this clock is used to refresh internal registers for the μPD7261A it is essential to exactly meet all specs concerning this clock. Following items are most important:

— Maximum clock frequency of RCLK

It must be assured that the frequency never exceeds the maximum allowed frequency of approximately 12 MHz. The PLL has to be designed to never generate such high frequency even or especially not during locking-in to the SYNC field.

— RCLK has to be glitchfree

Usually the output of a PLL is glitchfree. (Here a glitch is defined as a pulse which is shorter than 30ns; see data sheet for R/WCLK specs.). Special care must be taken when switching from RCLK to WCLK and vice versa using the RGATE signal.

— Data setup and hold times from/to R/WCLK

Data setup and hold times must be kept at any time. Even during locking-in of the PLL to a SYNC field the specs must be met.

The μPD9306 produces all clock and data signals according to the requirements of the μPD7261A. The performance of a PLL can be described in terms of

— Acquisition Time
— Frequency Range of Operation
— Peak Shift (Bit Jitter) Tolerance

— Acquisition Time

This parameter specifies the time needed by the PLL to lock-in into the data stream from the disk. This criterion is of less importance. Today PLL's are capable of locking-in in app. 2—4 bytes of SYNC data. The μPD7261A itself needs 0.5 bytes for synchronization. The μPD9306 does it in 4 bits as well.

— Frequency Range

This value is of medium importance only. The frequency variation of the RDATA stream itself is only around 1 %, due to small tolerances of rotational speed of hard disk drives. In order to compensate discrete component tolerances, analog PLL's must have a wide range of operation (app. 10 %). The μPD9306 integrates a totally digital PLL. It features a tracking range of 3 % which is more than enough to cope with the rotational tolerances of hard disk drives.

— Peak Shift

This is an important criterion of a PLL. Digital and good analog PLLs show good results here. The μPD9306 can handle bit jitter up to 40ns.

3. Application Board Hardware Description

In the following, two application boards will be described. One maskes use of an analog single chip PLL and the other works with the μPD9306. For the sake of simplicity, both applications support one hard disk drive only. In order to control up to four drives it is a simple matter to add one 75118 per additional drive and to enable these buffers with the appropriate Drive Select signals.

Both boards have identical interfaces to the host system.

3.1 Drive Interface

Figure 2 shows the solution with the analog single chip PLL DP8460 from National Semiconductor. As can be seen from the schematic, a couple of discrete components are needed. The Flip Flop between the PLL and the HDC is used to adapt the PLL output to the clock requirements of the μPD7261A. The clock generator and the precompensation circuitry are realised in TTL and Schottky TTL logic.

In figure 3 the solution using μPD9306 is given. As can be seen on the first view, this is a very compact (and low cost) solution. Only a crystal and two delay lines have to be attached to the HDI. No discrete components are needed, avoiding all problems of analog designs like component tolerances, adjustments, board layout, etc. The μPD9306 generates all signals according to the requirements of the μPD7261A. Therefore, no additional synchronization logic is necessary between both devices. In case that a 10 MHz clock is already available in the system, this signal can directly be connected to the XTAL1 (pin 13) input of μPD9306. No connection to input XTAL2 (pin 12) is needed.

3.2 Host System Interface

Figure 4 shows the schematic of the host interface. The host CPU used is the μPD780 (Z80™ compatible), the bus is the ECB bus. A DMA controller μPD8273A-5 is used for fast data transfer. No local memory is needed as the DAM controller has highest priority in the system. Part of address decoding and control signal generation is done in the PAL16L8.

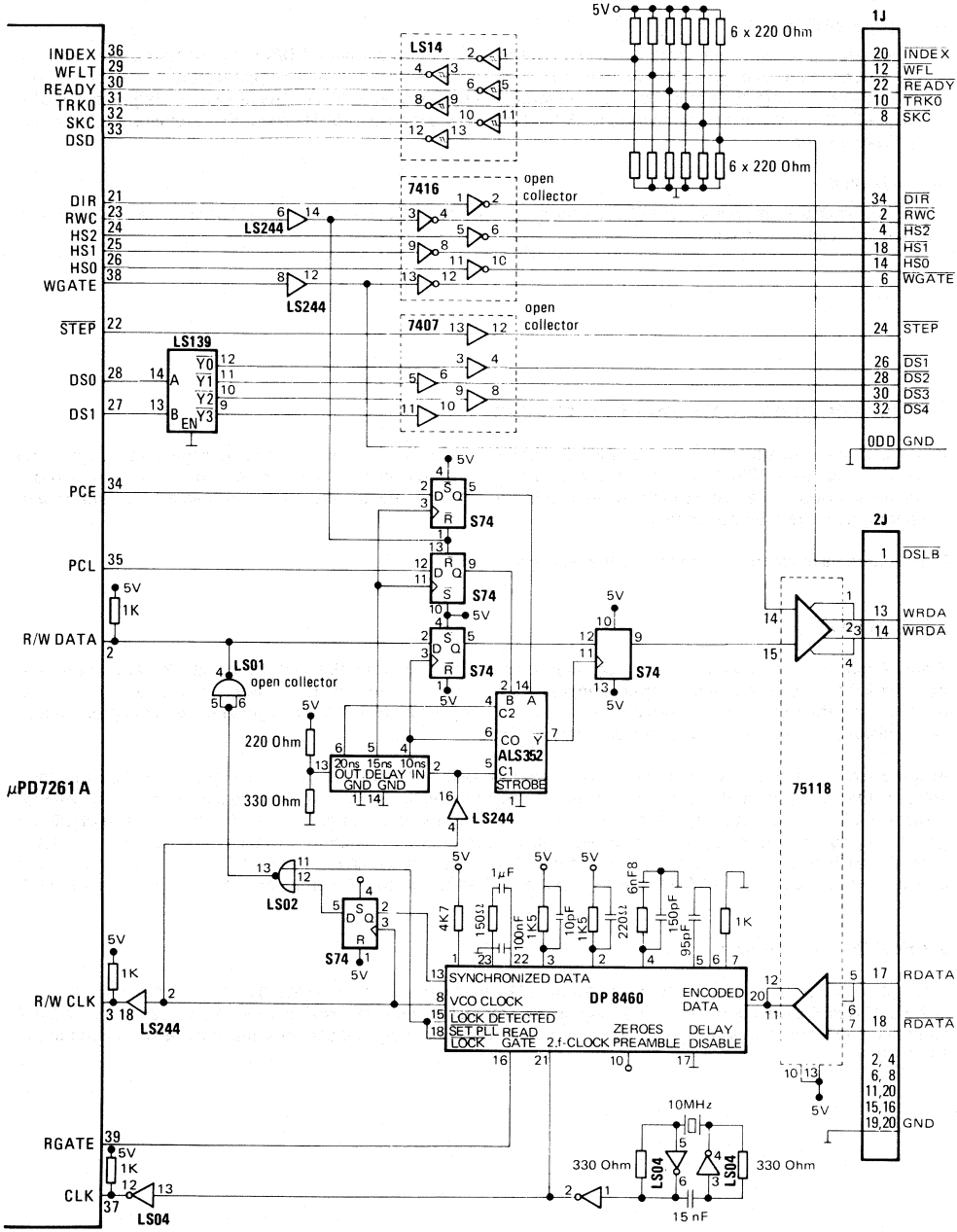


Figure 2. Analog Single Chip PLL Solution

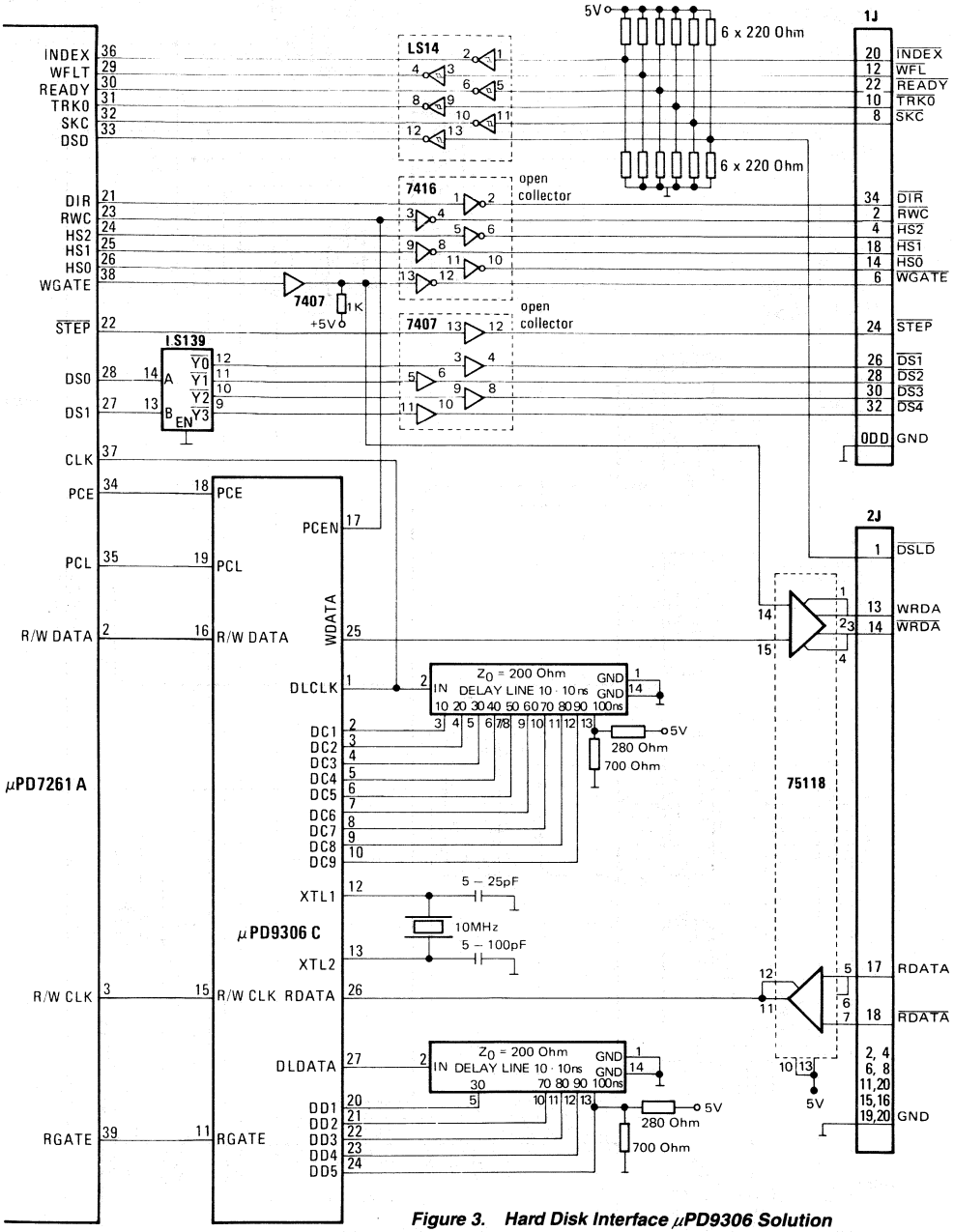


Figure 3. Hard Disk Interface μPD9306 Solution

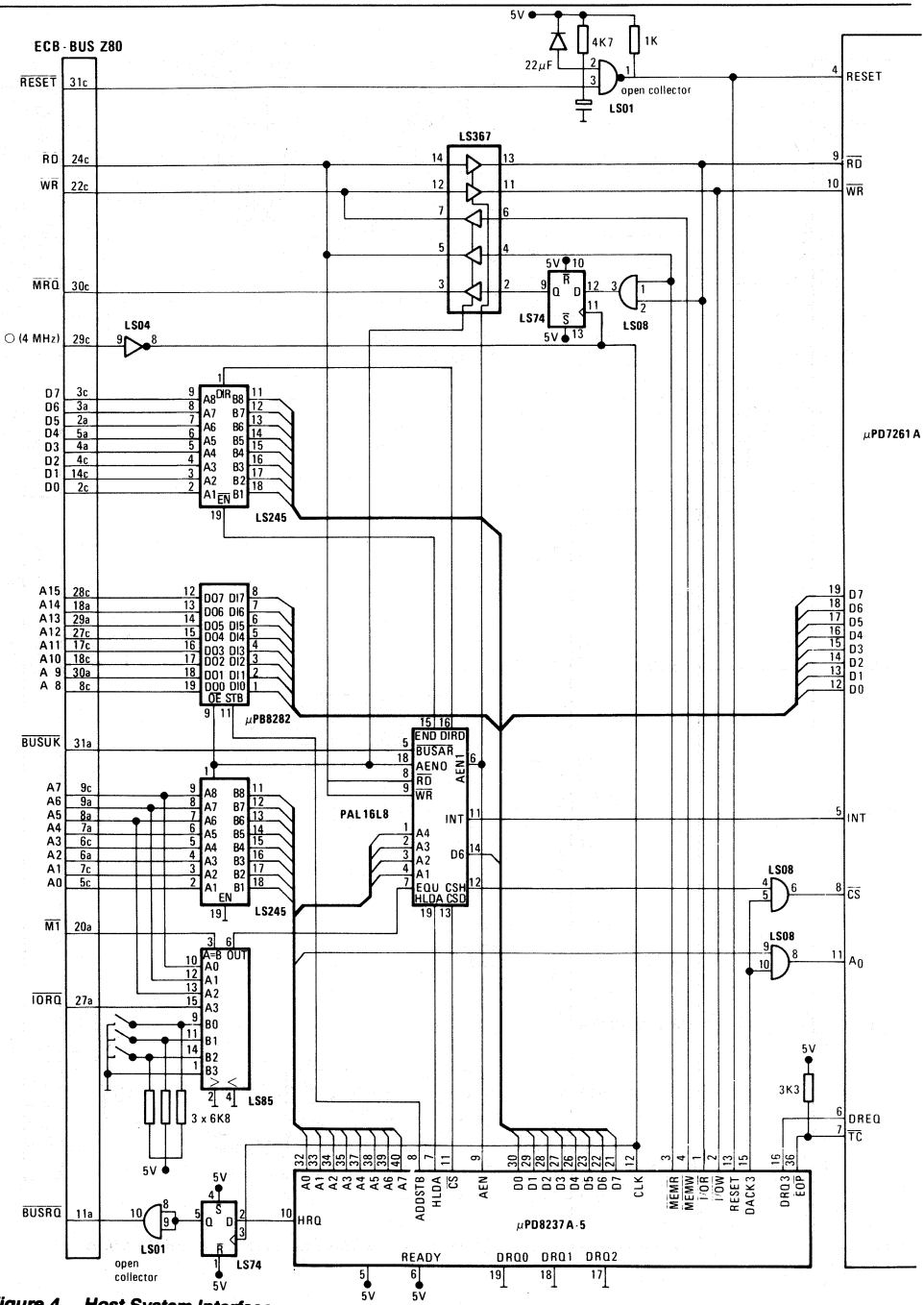


Figure 4. Host System Interface with DMA Controller µPD8273A-5

Video RAM μPD41264 Plus Graphics Display Controller μPD7220A Form a High Speed/High Resolution Graphics System

Contents:

Author: Peter Westerdorf
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD7220A	Data Sheet
μPD7220A	Product Description
μPD7220A	Application Note
μPD41264	Data Sheet

Related Products

μPD7220A	Graphic Display Controller (6 MHz)	NMOS
μPD7220A-1	Graphic Display Controller (7 MHz)	NMOS
μPD7220A-2	Graphic Display Controller (8 MHz)	NMOS
μPD72020	Graphic Display Controller (8 MHz)	CMOS

1. Introduction

The demand for high resolution color graphics systems is increasing rapidly. Along with higher resolutions, more and more video memory capacity is required. The higher the amount of video memory, the longer the time needed to draw vectors, circles etc. To have a comparable high drawing speed, with high resolution graphics systems, it is necessary to also increase this speed. Especially for this purpose NEC has developed the μPD41264 VIDEO RAM. With the help of this memory device it is possible to increase the drawing speed by four of five times without modifying the drawing logic to operate at higher speed.

2.1 Solution to Increase the Drawing Speed

An image memory must be served in two different manners. At first this RAM must be accessed by the display logic to transfer the data to the display monitor. This data read operation has to be done continuously to have a not disturbed data stream to the monitor, and therefore a not disturbed display.

For short: During the active display time (= time the beam scans the visible part of a screen) of a monitor a continuous data stream must be generated to have a not disturbed display. This kind of video memory access has highest priority so the modification of the memory data can just be done during blanking time.

The second kind of video memory service is to read the content, modify this data and write it back again. This operation mode is called read-modify-write cycle and is used to alter the image memory.

The third kind of video memory access is the refresh cycle. This is necessary in case of having a large dynamic image memory where the row access succession of the normal display operation is too slow.

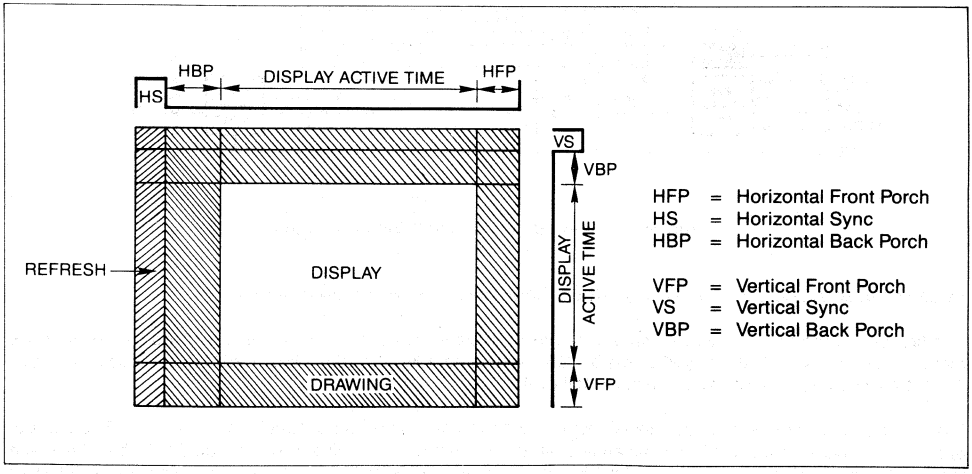


Figure 1 Timing Chart of a Display Monitor

Figure 1 emphazises that most of the time is used for displaying the video memory content and a relatively small part can be used for read-modify-write operations (= drawing). So only during the blanking period the video RAM is free to be accessed randomly. Depending on the relation of active display time to blanking time the drawing can only be done at a fourth or a fifth of the theoretical maximum speed.

To overcome this waste of possible drawing time without the disadvantage of a disturbed display or additional hardware, the NEC μPD41264 VIDEO RAM (also called DUAL PORTED RAM) can be used. So an approximately "100 % of the time" image memory access for read-modify-write operations is available resulting in the four to five time higher drawing speed by simultaneously providing display data for the monitor.

2.2 The μPD41264 VIDEO RAM

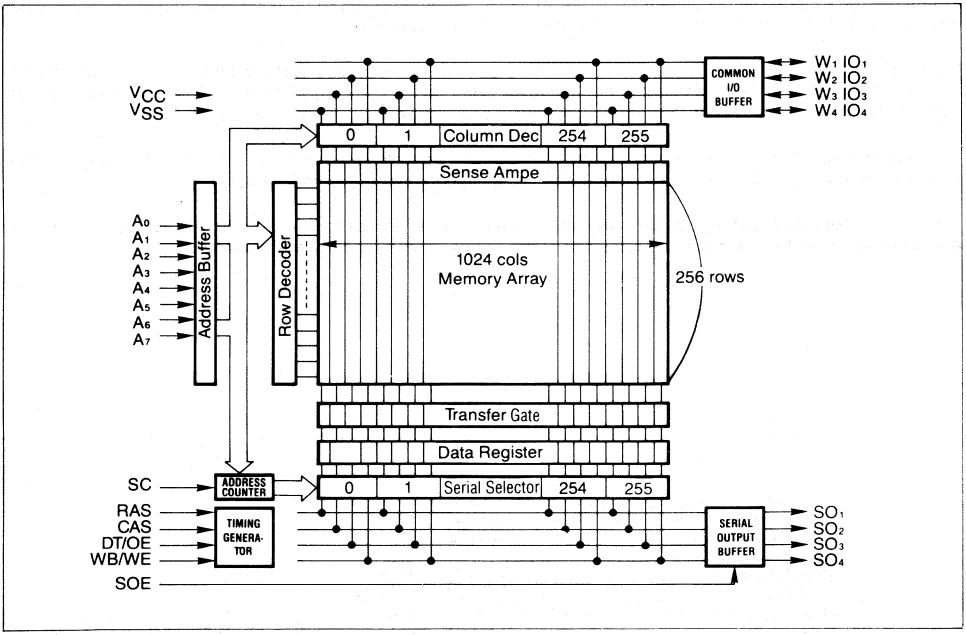


Figure 2 μPD41264 VIDEO RAM Block Diagram

As can be seen at figure 2 the μPD41264 VIDEO RAM consists of two different blocks.

The first one is the memory part which is very similar to standard dynamic RAMs. It is realized by four 64 Kbits memories. (Four bits in/output by 256 rows by 256 columns per row : 4 * 256 * 256 = 262144 bits.) So this VIDEO RAM is a 256 Kbit memory organized in 64 Kbit by 4.

The second block can be seen as a serial memory output circuit. This output circuit possesses a 1024 bit output buffer which is also called line buffer because it can store an entire display line of less or equal 1024 bits. Also here the serial output buffer is organized by 4 bit (256 * 4). The line buffer is loaded with the content of one of the 256 lines (= rows) of the 256 Kbit memory array.

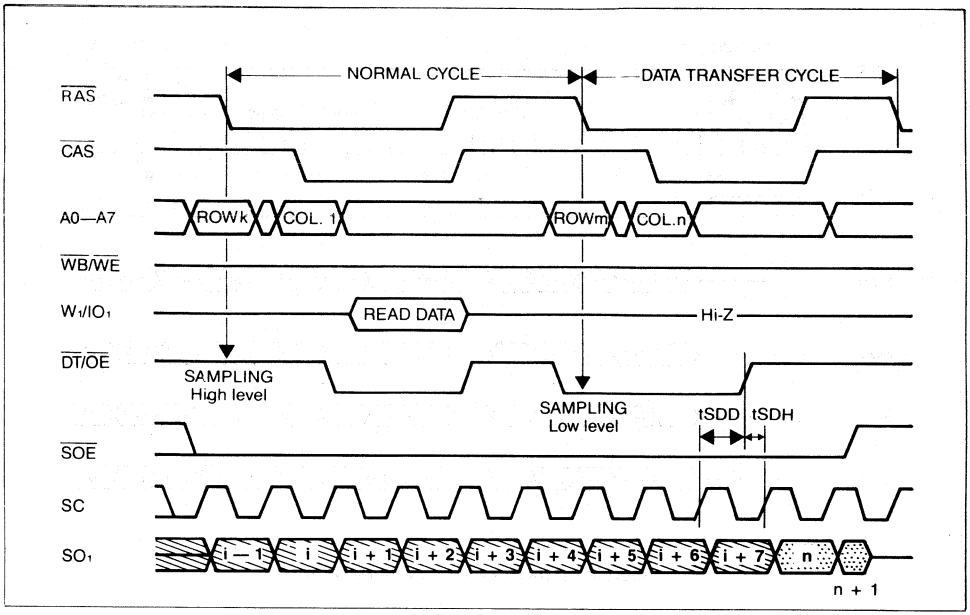


Figure 3 The Data Transfer Cycle

To load a certain row into the line buffer a special video memory timing called data transfer cycle is required. This timing differs from a standard read or write cycle in one thing only that is the level of the output enable input that is also called data transfer input. If this input line (DT(OE) is high level when RAS (= Row Address Strobe) falls it is a normal read or write cycle, if it is low level it is a data transfer cycle. The row to be transferred into the line buffer is determined by the row address that is read by the falling edge of RAS when DT/OE is low. The read column address in a data transfer cycle is used for the serial output logic to define the column start position whithin the four times 256 serial data buffer. Starting at this position the four times 256 bit line buffer register can be read out serially by the help of a read clock applied at the SC pin (SC = Serial Clock) of the μPD41264.

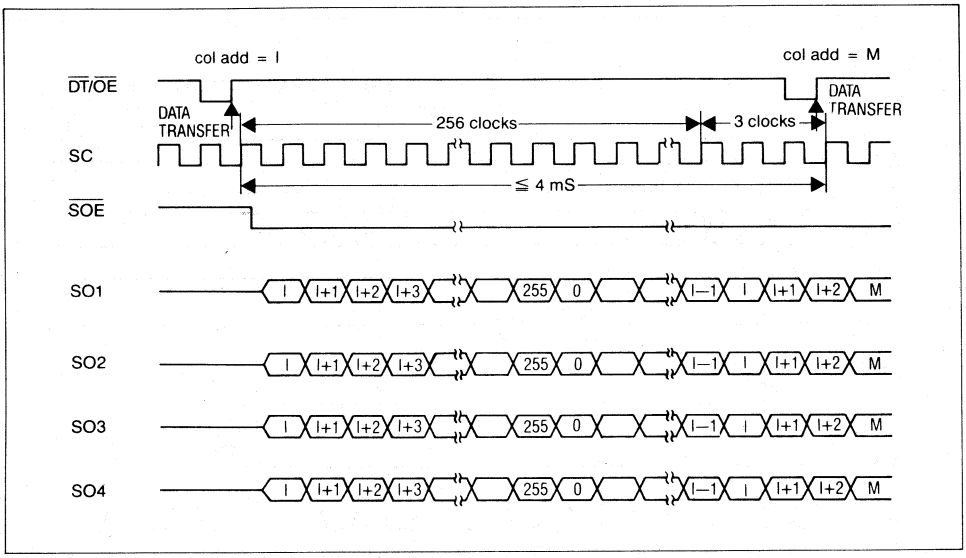


Figure 4 Data Stream of the Serial Port

Each clock pulse at the SC pin yields four bits dta at the serial output pins SO0—SO3. Also an internal 256 bit serial address counter is incremented to point to the next four serial data bits. If this address counter reaches position 255 then the next position is 0. The serial data port provides four times 256 bits but also a continuous data stream of one time 1024 bit can easily be achieved. For this purpose an external shift register must be utilized.

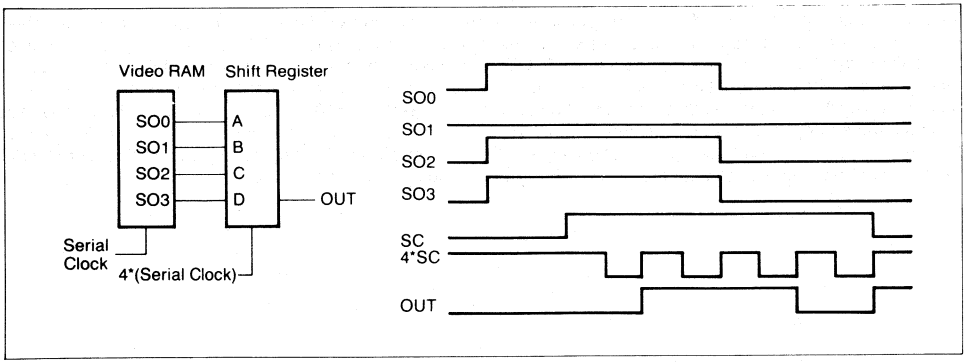


Figure 5 Transforming a 4 * 256 Bit Data Stream into a 1 * 1024 Bit Data Stream

At figure 5 shows the clock to generate the serial output data stream of the external shift register is four times faster than the read clock applied to the VIDEO RAM. The maximum serial clock input frequency to the VIDEO RAM is 25 MHz but the four bit shift register can generate a data stream that is equivalent to a 100 MHz VIDEO RAM.

It should be mentioned that the real time data transfer cycle to load a new line into the serial line buffer may occur any time even within the active display time interval. This is very useful for windowing that is to overlay one image with a part of another one. The most important fact of the new VIDEO RAM is that both ports, the parallel one to access the memory array and the serial one to read out bits of a line, can operate fully independent from each other except around 40 ns within data transfer cycles when serial and parallel port must be synchronously. If utilizing this memory as image memory an all the time drawing access and a not disturbed display can be easily achieved. The line buffer frees the normally prohibited active display time interval for read-modify-write operations.

2.3 Example how to Interface the μPD7220A Graphics Display Controller to the μPD41264 VIDEO RAM

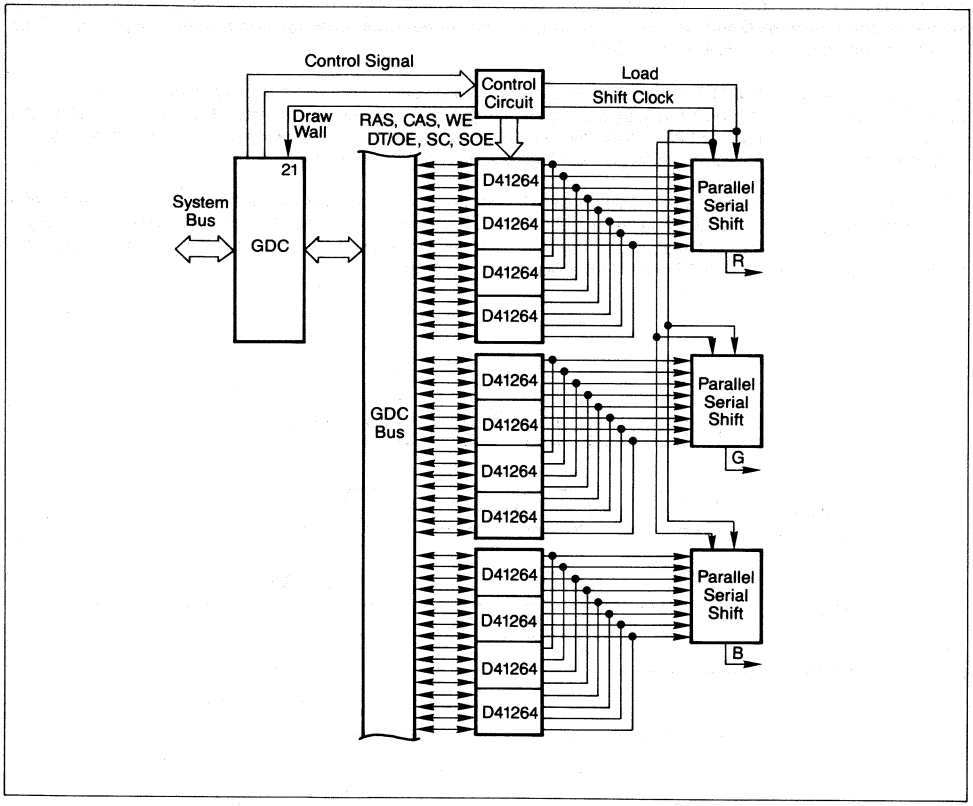


Figure 6 Block Diagram of a GDC-VIDEO RAM Graphics System

Using a graphics display controller offers many features which are not available or hard to realize by a microprocessor. First of all, the drawing speed is much higher with such a controller as the appropriate drawing calculations are done by on chip hardware and not by the CPU.

The configuration of the μPD7220A Graphics Display Controller connected to the μPD41264 VIDEO RAM will be explained. Figure 6 shows a block diagram of a high resolution graphics system having 1024 dots per line by 1024 lines in 8 colors. This has been achieved by three color planes (red, green, blue) of 128 KBytes each. The GDC is programmed for full graphics mode so it can access four times 128 KBytes memory that is a total of 512 KBytes maximum. The controller operates in word access (16 bits = one word) so 16 address lines are sufficient to address the entire memory directly. The selection of the color planes can be done by utilizing the address lines A16 and A17. (Green plane: 00000—0FFFF / red plane: 10000—1FFFF / blue plane: 200000—2FFFF)

As already mentioned, the GDC provides two basic image memory access modes. One is called the DISPLAY CYCLE, which is used to read the video memory successively to generate an image on a monitor. The other one is the READ-MODIFY-WRITE CYCLE which is utilized when drawing into the image memory. Only one of these cycles can operate at a time so in normal dynamic RAM graphics systems the GDC generates display cycles during the active display time interval and the read-modify-write cycles during blanking periodes if drawing is in progress. This kind of operation is called FLASHLESS DRAWING.

By the help of the new VIDEO RAMs μPD41264 the Graphics Display Controller is allowed to do read-modify-write cycles during this former prohibited time interval, resulting in a four to five times faster drawing speed. Here the GDC operates in the so called FLASH DRAWING mode, which means that if a drawing operation has been started, only read-modify-write cycles are performed until the last pixel has been written.

The internal display cycle logic still counts on also during read-modify-write operations so that the beginning of a new display line (= Horizontal Blank falling edge) the appropriate row and column address for the image memory is available. The new μPD7220A Graphics Display Controller can be programmed to output this address needed for the data transfer cycle of the VIDEO RAM inbetween drawing operations.

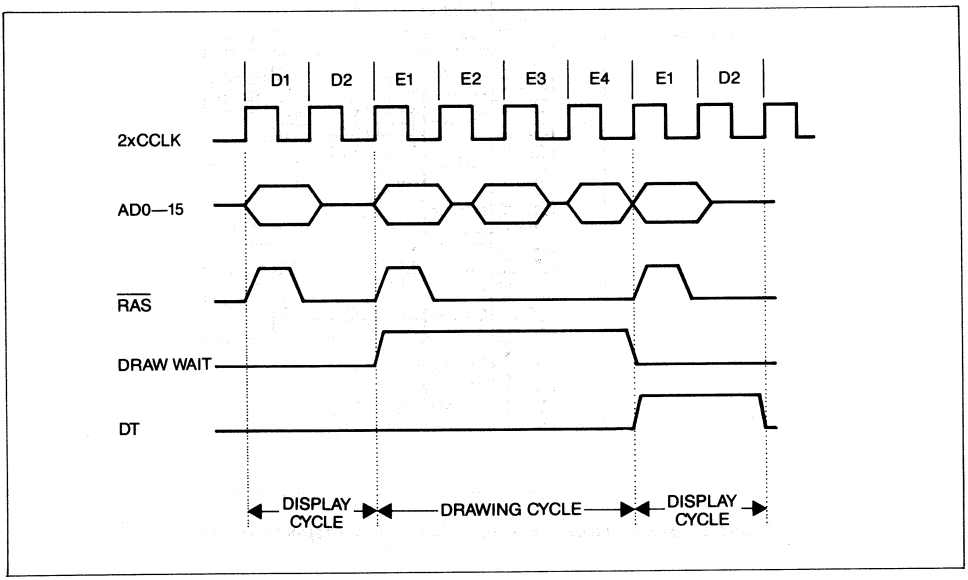


Figure 7 Draw Wait Function of the μPD7220A

After enabling the so called draw wait function of the GDC the light pen input operates an input to interrupt a currently active read-modify-write operation. A high level signal must be applied for the duration of four GDC clock cycles (= 4 * 2xWCLK) to ensure that a just started read-modify-write cycle is terminated when entering a display cycle. This following display cycle provides the desired row and column address for the VIDEO RAM and after this display cycle the GDC reenters the read-modify-write operation to continue the interrupted drawing process.

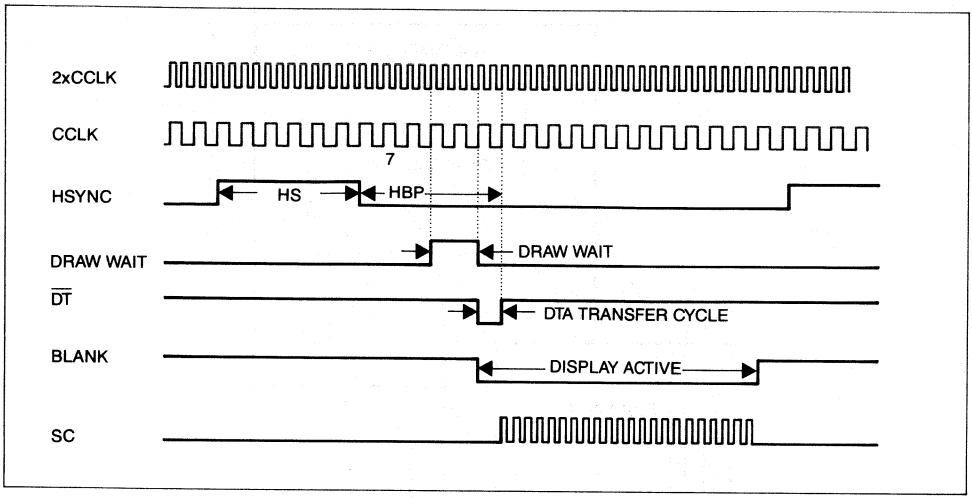


Figure 8 Timing Chart of Data Transfer Control

Figure 8 points out more detailed that the falling edge of the GDC Blank signal is the termination of a read-modify-write cycle and the start point of a display cycle which supplies a start address for the VIDEO RAM real time data transfer. At the end of the succeeding data transfer cycle (DT) the serial read clock (SC) is applied to the μPD41264 to get the display data out of the line buffer.

3. Application Board Hardware Description

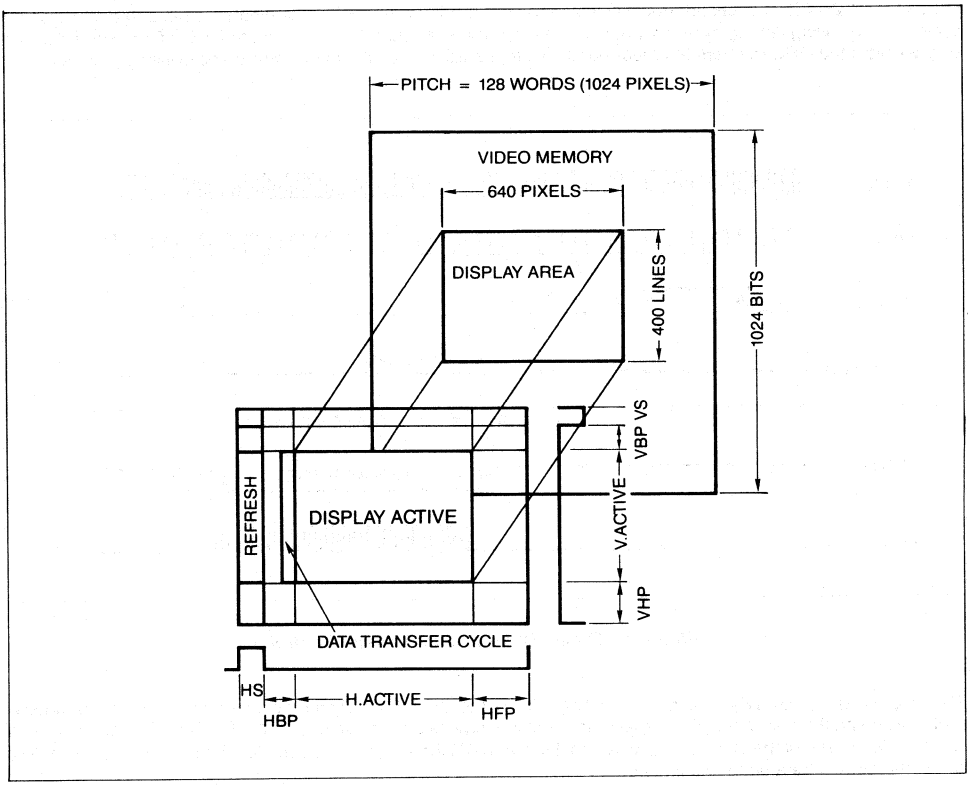


Figure 9 Organisation of a one Plane System

The following description is related to a high resolution color graphics system. The image memory can store 1024 dots per line by 1024 lines in 3 color planes which form 8 colors. The resolution for the active display is 640 dots per line by 400 lines. Here the GDC is used to provide a 16 bit address and data bus for the image memory. As one VIDEO RAM has got four data in/output lines four devices have to be used to meet a 16 bit bus. The address bits A8 through A15 are used as row address and A0 through A7 as column address. (Note: Each Colum stores four bits.)

The GDC provides the refresh addresses at the address lines A0 through A7. For refresh cycles the low addresses have to be switched to operate as row addresses. This can easily be achieved by the help of the HSYNC (Horizontal Sync) signal. If HSYNC is actie refresh cycles are generated so this signal can be used to activate the column address latch output when RAS falls. In this application there is no problem as two 8 bit address latches have to be used meet the GDC requirements.

Figure 10 points out the difference in addressing between normal dynamic RAMs and VIDEO RAMs. With standard RAMs the address bits A8 through A15 function as column address, whereas the VIDEO RAM needs these as row address. Therefore, the low address bits A0 through A7 are used as row address bits in dynamic RAM applications and in VIDEO RAM systems as column address.

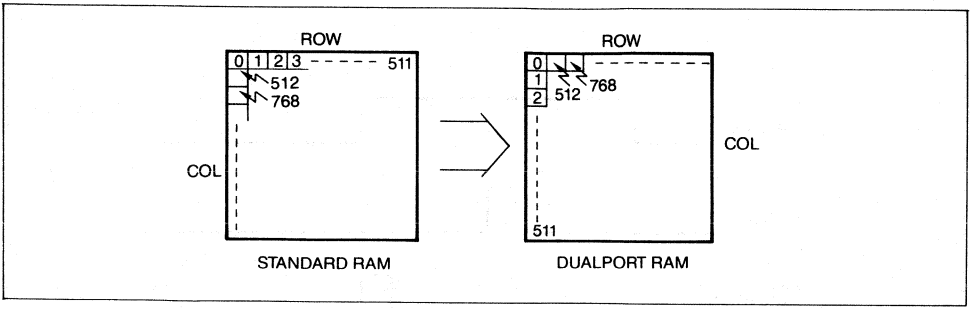


Figure 10 Standard Dynamic RAM Address Distribution Versus VIDEO RAM

The three color planes are accessable in two ways. For read-modify-write cycles each color plane is accessed separately, but for data transfer cycles all planes are accessed simultaneously. The GDC's address lines A16 and A17 are used to generate CAS (Column Address Strobe) signals for each color plane in coordination with a data transfer, horizontal sync and a delayed RAS signal.

If there is a read-modify-write cycle HSYNC and the data transfer signal are invalid so the utilized LS139, the three AND gates LS08 and the OR gate LS32 generate a CAS signal for only one color plane.

A data transfer cycle is in progress when HSYNC is invalid but the data transfer signal is valid. So the CAS generator logic produces three CAS signals simultaneously. The data to be displayed on a color monitor can then be read out of the three color planes at the same time.

Refresh cycles occur only during HSYNC active intervals when the data transfer signal is invalid. Here no CAS signal following the RAS signal is generated but this is exactly what has to be done to perform the RAS ONLY refresh mode for the VIDEO RAMs.

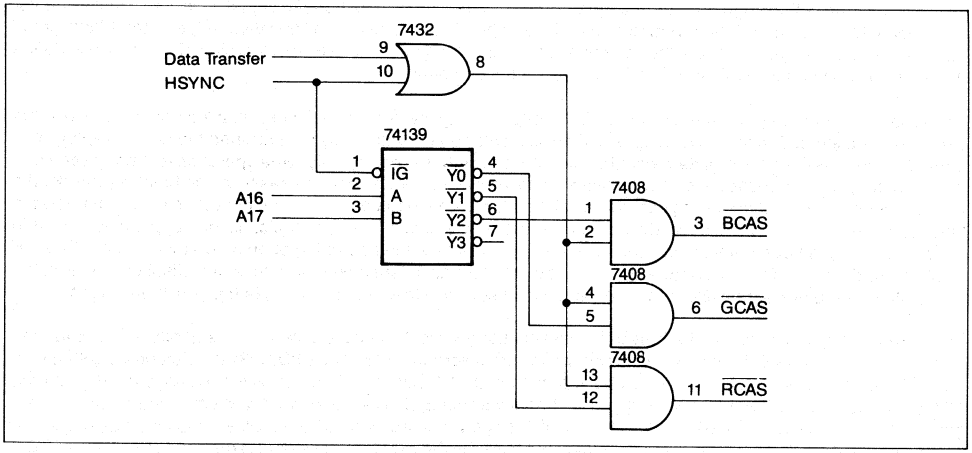


Figure 11 CAS Generation Logic

The data transfer logic, which is necessary for the proper loading of the VIDEO RAM line buffers, has to generate the draw wait and the data transfer signal synchronized by the HSYNC signal.

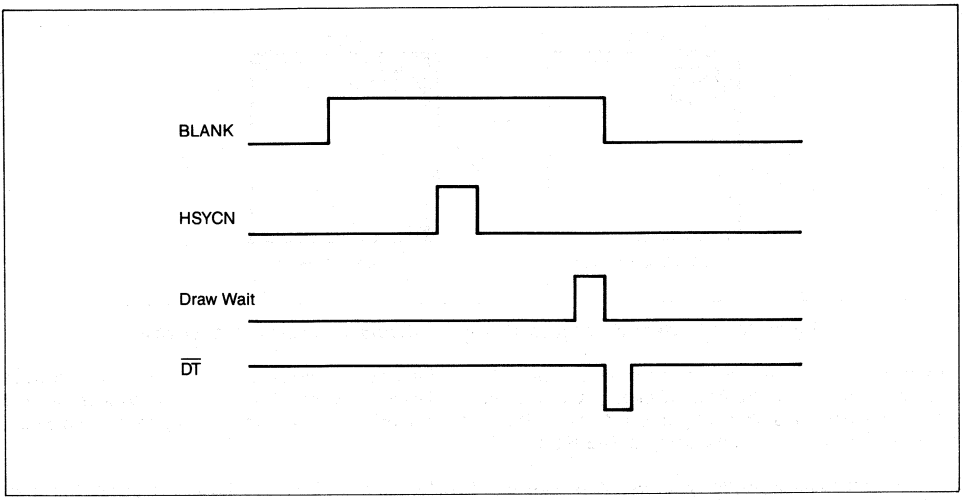


Figure 12 Data Transfer Timing Chart

The first falling edge of the GDC BLANK signal after HSYNC was active defines the start position to send the row and column address to the VIDEO RAMs. Because the μPD7220A always generates a blank signal even during read-modify-write operations when accessing a word to be modified, the falling edge of this signal cannot be used directly. A second point is that a draw wait signal has to be applied to the controller for 4 * 2xWCLK cycles before the falling edge of blank and thereby the activation of the data transfer signal. Due to this fact a counter has been used to count the time between the falling edge of the HSYNC and the falling edge of the blank signal minus 4 GDC clock cycles. This is the horizontal back porch time minus 4 GDC clock cycles. So it is easy to define the start position of draw wait and thereby the data transfer signal.

Because of the blank signal behaviour it is also necessary to generate a special new blank signal for the external serial shift registers. The task is to suppress all GDC blank signals which may occur during the active display time, otherwise the external shift registers would be cleared randomly if a read-modify-write operation is in progress. Here it is a similar problem because the active display time interval cannot be defined easily by the blank signal like the time between falling and rising edge of this signal. So also it is necessary to count out the active display time by the help of a counter. For the active display time the blank signal of the controller is not applied to the shift registers. This has been done by latching the blank signal state at the beginning of an active display time (BLANK = 0) by the data transfer signal into a flip flop. If the counter has finished counting the active display time then the new state of the blank signal (BLANK = 1) can be latched. The state of this flip flop is the new blank signal.

In this application one 8 bit counter is used for both counting tasks. It consists of two 4 bit parallelload counters with ripple carry output (LS163). The counter preset values are determinated by two μPD71082 8 bit latches together with the HSYNC signal. At first the horizontal back porch minus 4 * 2xWCLK time equivalent value is loaded into the counter. If the counter has finished counting (= ripple carry output is set) draw wait and data transfer signals are generated and the counter loads itself with the active display time equivalent value. When this time has elapsed a second draw wait/data transfer cycle is generated but this time the data transfer signal is only used to allow the GDC blank signal to pass to the external shift registers.

So the two data transfer signals form the new blank signal for the external shift registers and monitor. The counter is clocked with 1/2 of 2xWCLK which is the display cycle time therefore it is no problem to exactly define the appropriate values for the counter.

Calculation example for a 640 by 400 resolution display

Pixelclock= 20 MHz
2xWCLK= 5 MHz
Counter clock = 2.5 MHz => Tcounter = 400 ns
Horizontal Front Porch = 2.4 μs
Horizontal Sync = 2.4 μs
Horizontal Back Porch = 3.2 μs
Monitor Active Display Time = 32 μs

Resolution = act. disp. time * pixelclock
= 32 μs * 20 MHz
= 640 pixel/line

Active words/line = (1/2 * 2xWCLK) * 32 μs
= 80 words/line

Horizontal Front Porch = 2.4 μs / 400 ns = 6
Horizental Sync = 2.4 μs / 400ns = 6
Horizontal Back Porch = 3.2 μs / 400ns = 8

1. counter value: horizontal back porch — (4 * (1/2xWCLK)))
3.2 μs — 4 * (1/5 MHz) = 2.4 μs
2.4 μs / Tcounter = 2.4 μs / 400 ns = 6
1. Value = 256 — 6 = 250
249 has to be loaded as first value

2. counter value: active display time
32 μs / Tcounter = 32 μs / 400 ns = 80
2. value = 256 — 80 = 176
175 has to be loaded as second value

Normally three 16 bit external shift registers could be used to generate the red, green, blue data stream to the color monitor. The clock frequency for shifting this registers is 16 times the serialclock of the VIDEO RAM. Here the dot clock is 20 MHz and therefore the serial clock is 20 MHz / 16 = 1.25 MHz.

A better solution is to utilize 8 bit shift registers because of less pins and smaller packages. Now the 8 bit shift registers are cloaded twice.

At first the low data are transfered into the register. After all 8 bits have been shifted the high 8 data bits are loaded and shifted out. Each load cycle therefore affects two VIDEO RAMs of a color plane. Figure 13 describes this more detailed.

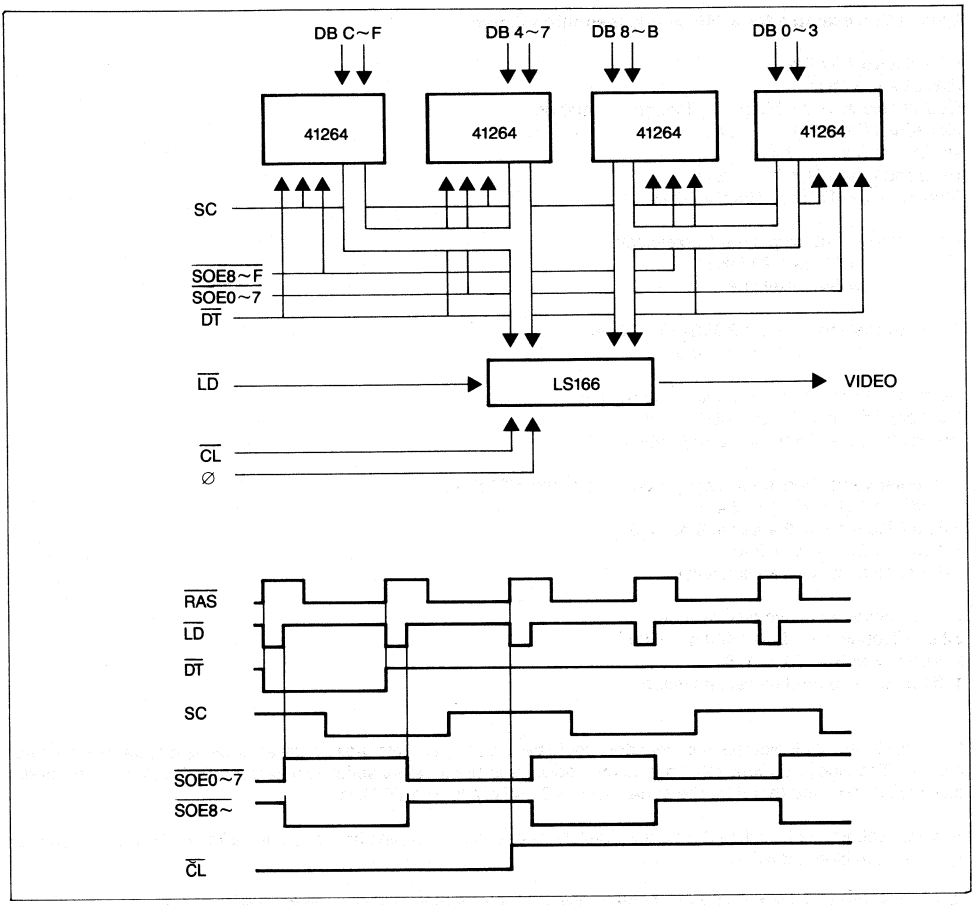
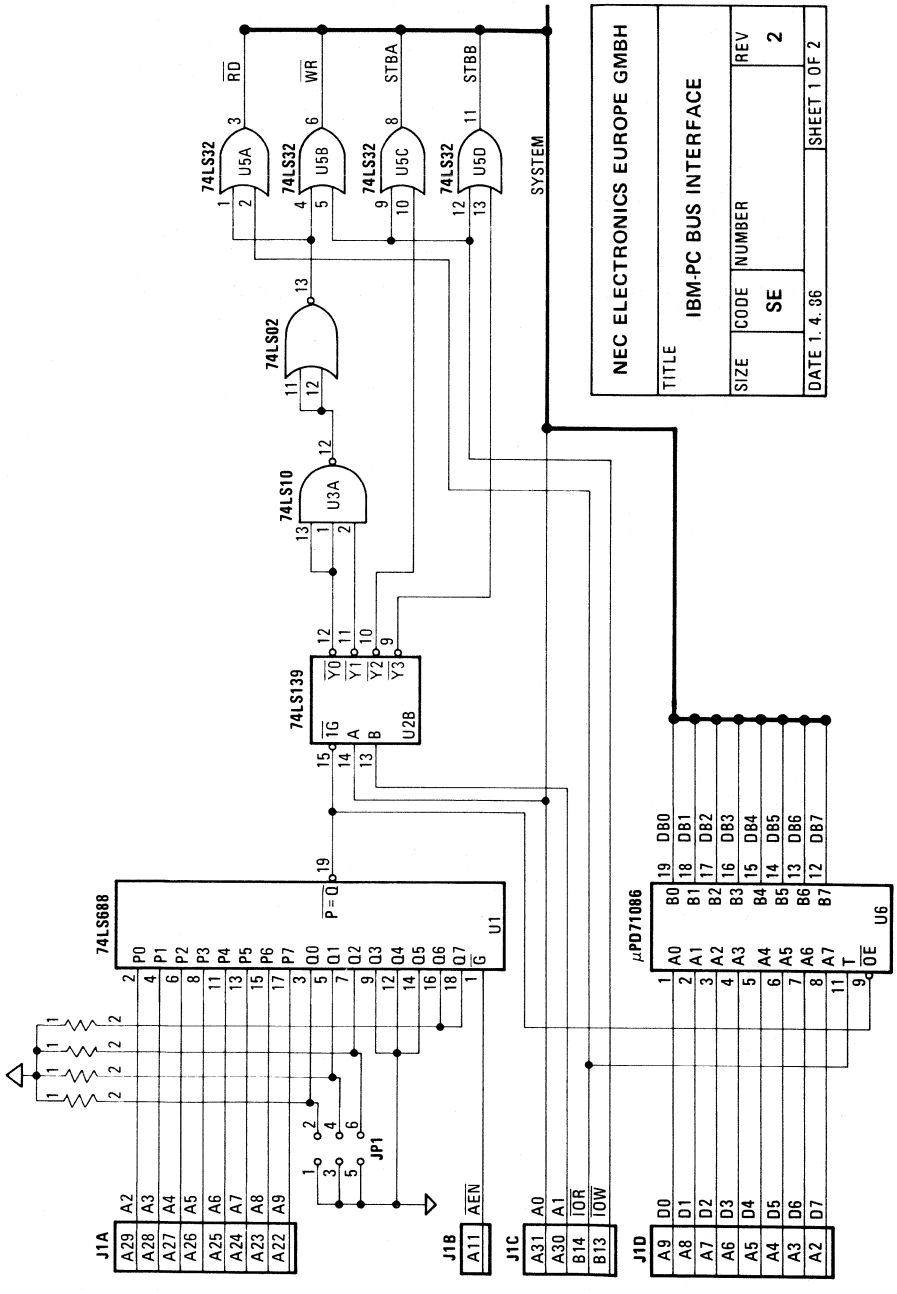


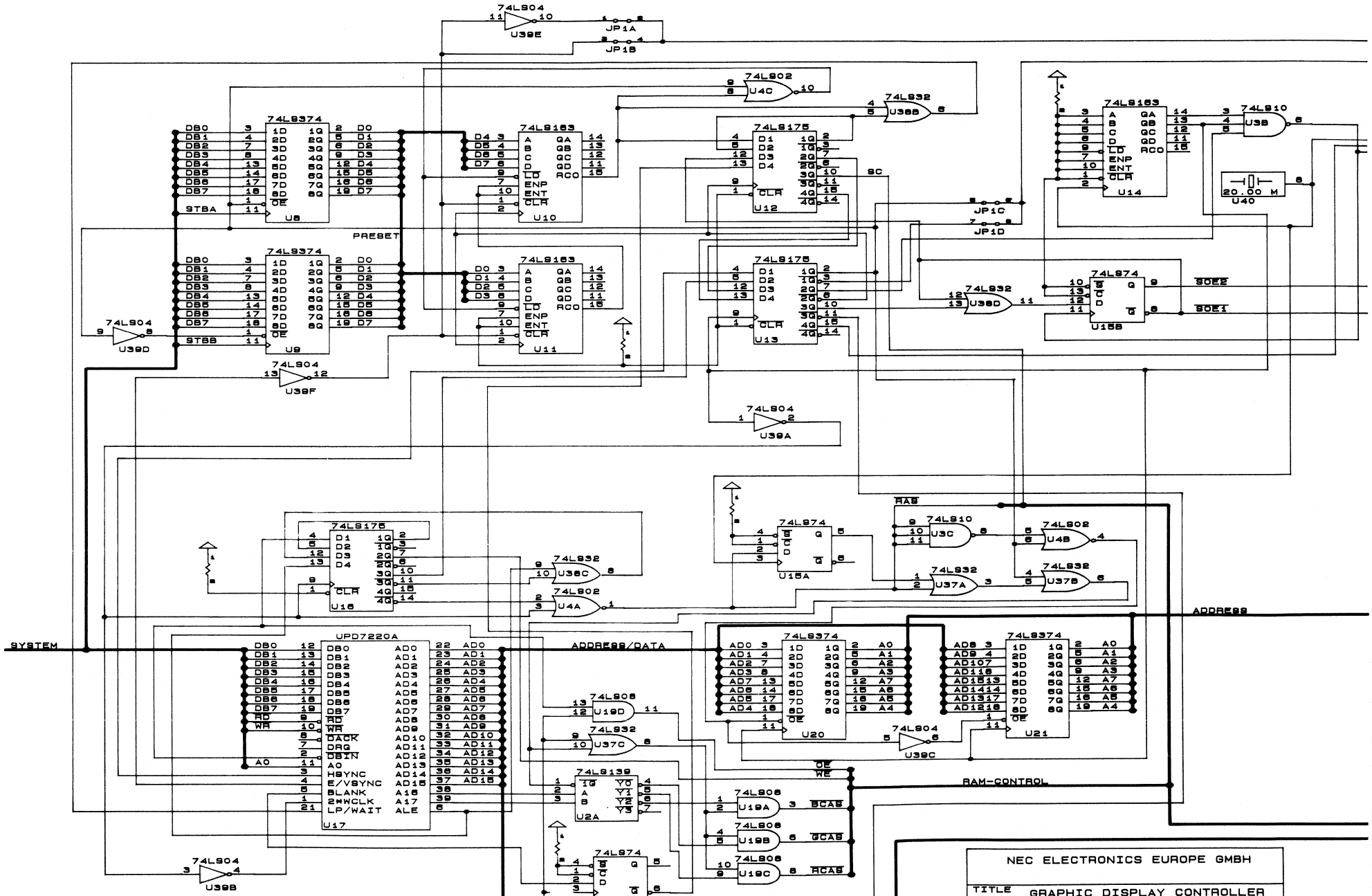
Figure 13 External Shift Registers Control

4. Conclusion

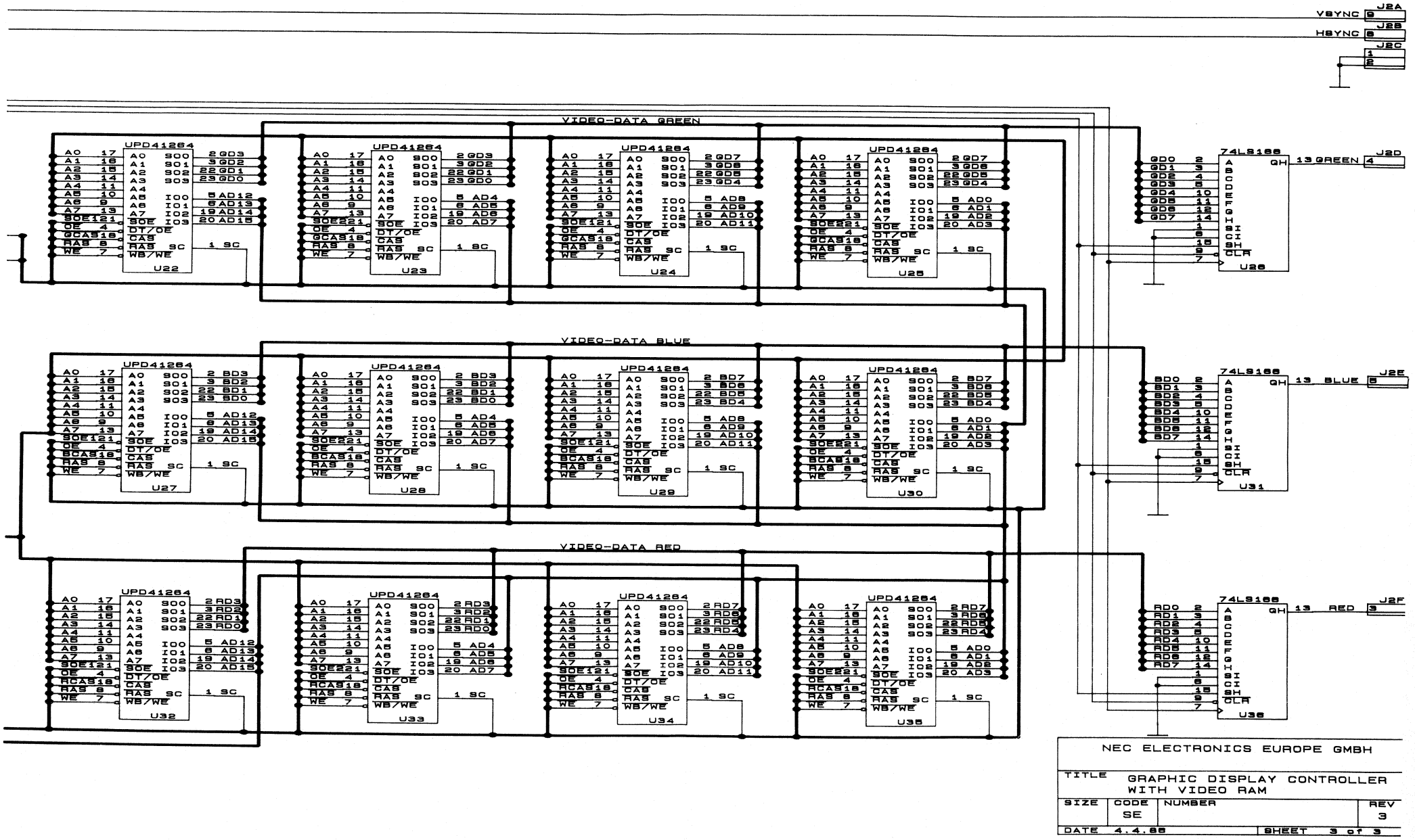
The usage of the VIDEO RAMs μPD41264 together with the μPD7220A Graphics Display Controller speeds up the drawing by factor four or five. Therefore, the new bottleneck is the microprocessor that has to generate the drawing preset parameters for the controller.

The advantage of higher drawing speed will become more visible the larger the image memory is and the more the controller has to work. (Note! An 8 MHz GDC with VIDEO RAMs drawing in Flashmode will have the same drawing speed as 32 MHz or 40 MHz GDC would have if operating in Flashless mode.) These VIDEO RAMs can significantly increase the drawing specially when 'drawing' proportional characters onto a screen that is only possible when operating in full graphics mode. Important is that the pixel clock and the clock to the controller can be selected independently from each other. Now it is possible to build up a high resolution graphics board in a small size only using some additional TTL devices.


NEC ELECTRONICS EUROPE GMBH
TITLE
IBM-PC BUS INTERFACE
SIZE
CODE
SE
NUMBER
REV
2
DATE 1. 4. 86
SHEET 1 OF 2
SYSTEM
74LS32
U5A
U5B
U5C
U5D
RD
WR
STBA
STBB
74LS02
74LS10
U3A
74LS139
U2B
74LS688
U1
P=Q
JP1
AEN
IOR
IOW
μPD71086
U6
J1A
J1B
J1C
J1D



NEC ELECTRONICS EUROPE GMBH			
TITLE	GRAPHIC DISPLAY CONTROLLER WITH VIDEO RAM		
SIZE	CODE SE	NUMBER	REV 3
DATE 4.4.86		SHEET 2 of 3	


VSYNC
HSYNC
J2A
J2B
J2C
VIDEO-DATA GREEN
VIDEO-DATA BLUE
VIDEO-DATA RED
UPD41264
74LS166
U22
U23
U24
U25
U26
U27
U28
U29
U30
U31
U32
U33
U34
U35
U36
J2D
J2E
J2F
13 GREEN
13 BLUE
13 RED
NEC ELECTRONICS EUROPE GMBH
TITLE GRAPHIC DISPLAY CONTROLLER WITH VIDEO RAM
SIZE
CODE SE
NUMBER
REV 3
DATE 4.4.88
SHEET 3 of 3

External Components for the Subscriber Line Interface Circuits (SLIC) μPC7062K and μPC7069K

Author: Berthold Heck
Application Department
NEC Electronics (Europe) GmbH

1. Introduction

The μPC7062K and the μPC7069K are subscriber line interface ICs for PABX. They serve as an interface between a two-wire, analog subscriber line and a four-wire, time division multiplex switch. This requires the following functions, collectively referred to by the acronym BORSCHT.

B: Battery feed to supply DC current to terminal devices, e.g. telephne sets

O: Overvoltage protection to protect the exchange from overvoltages or surges induced by lighting or power lines

R: Ringing to transmit ringing signals

S: Supervision to supervise the states if the subscriber line such as on-hook and off-hook

C: Codec to encode/decode analog signals to/from a PCM code

H: Hybrid to convert a two-wire signal transmitted from the subscriber line into a four-wire signal to be transmitted to the speech path switch or vice versa

T: Testing of the subscriber line

The μPC7062K and the μPC7069K are provided with functions B, S and H.

This application note describes the external components which are required for the DC power feeding and the realization of desired AC input impedance and transmission characteristics. Figure 1 shows an application circuit. Overvoltage protection is not included in this circuit. Additionally, noise bypass capacitors should be connected between the power supply pins and ground.

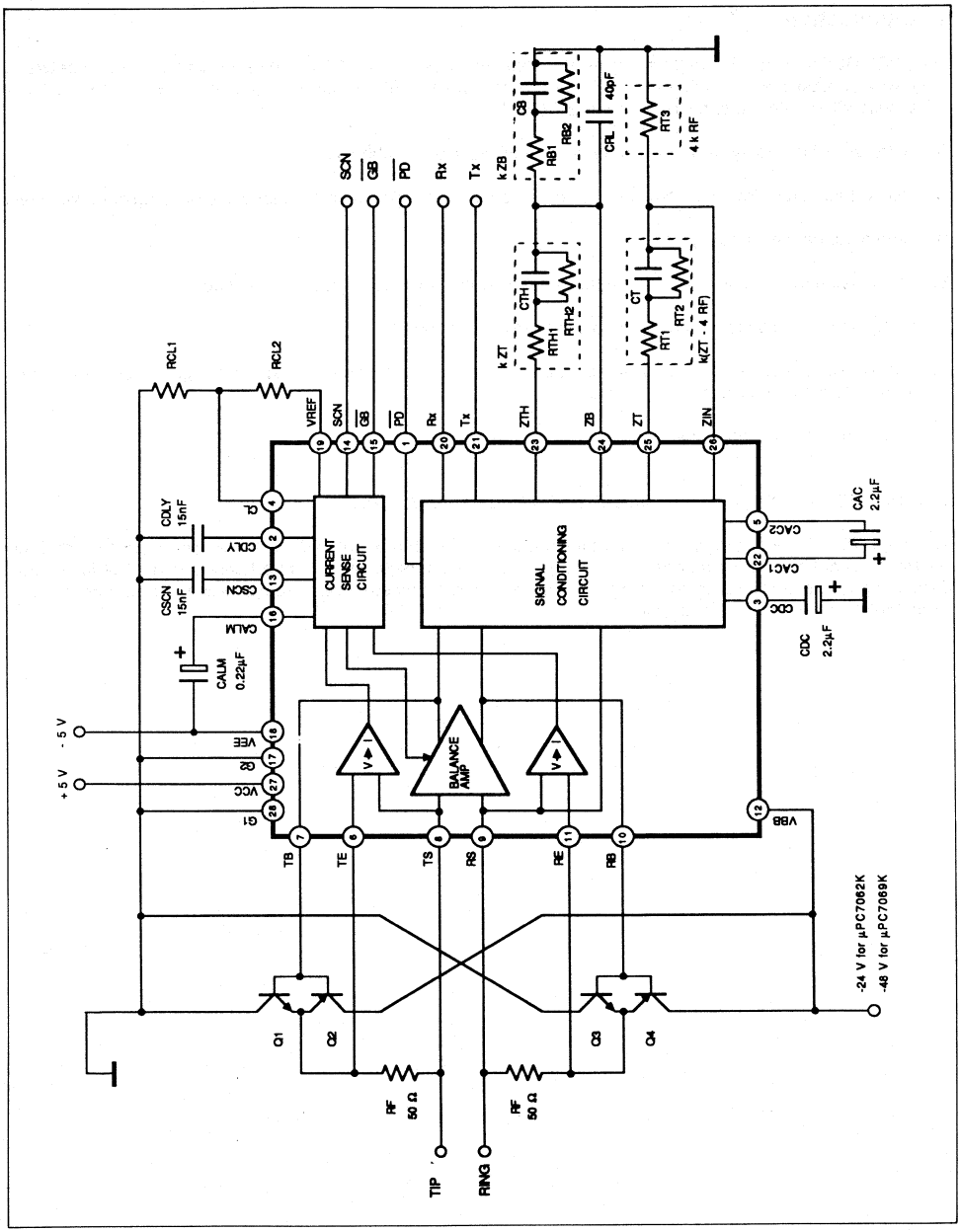


Figure 1. Application Example

2. DC Power Feeding

The μPC7062K and the μPC7069K incorporate a resistance-synthesis circuit. The external resistors R_F at TIP and RING terminals determine the feed resistance with a scale factor of four, thus providing a 2-wire DC feed resistance of 2 x 4 R_F. With R_F = 50 Ω, the 2-wire DC feed resistance is 2 x 200 Ω.

The μPC7062K and the μPC7069K represent a so-called semi-constant-current power supply system. This power supply system supplies power to the subscriber load at a fixed DC feeding resistance as long as the subscriber line has a specific resistance or more. If the resistance of the subscriber line is lower than the specific value however, this system supplies the power at a constant current I_{CNS}. Figure 2 shows a model for the power supply system and the relation between resistance and current of the subscriber line.

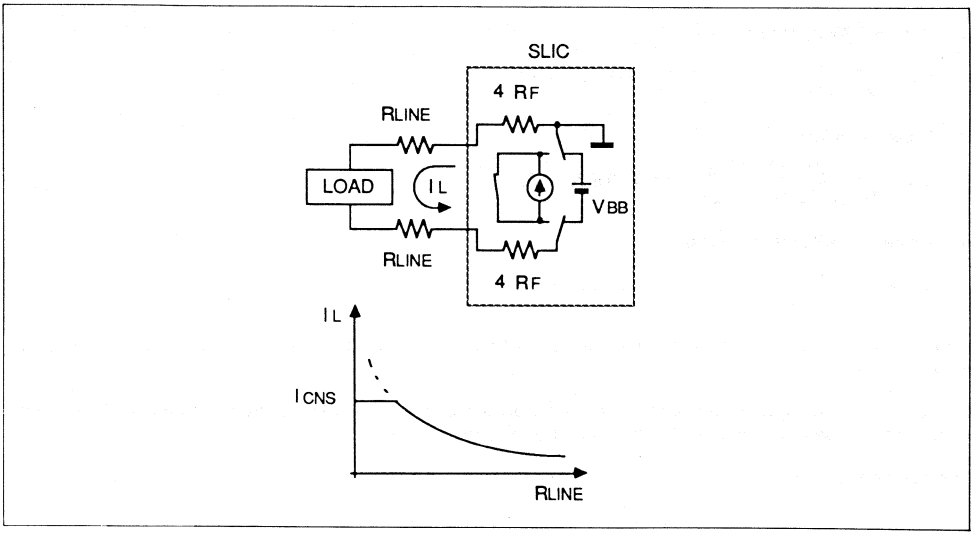


Figure 2. Semi-Constant-Current Power Supply System

The constant loop current I_{CNS} can be set within the range 20 mA ≦ I_{CNS} ≦ 50 mA by attaching resistors R_{CL1} aND R_{CL2} to the SLIC. The value is determinated by the following equation:

$$I_{CNS} = \frac{7.9\ V}{2R_F} \cdot \frac{R_{CL}}{R_{CL1} + R_{CL2}}$$

$R_{CL1} + R_{CL2}$ should be in the range 50 kΩ ≦ $R_{CL1} + R_{CL2}$ ≦ 100 kΩ. With R_F = 50 Ω, and setting resistances R_{CL1} and R_{CL2} to $R_{CL1} + R_{CL2}$ = 79 kΩ we get

$$\frac{I_{CNS}}{mA} = \frac{R_{CL1}}{k\Omega}$$

The 2-wire-line drive circuit requires to NPN transistors Q_1, Q_3 and two PNP transistors Q_2, Q_4. They are driven by the internal operational amplifier. Loop current is supplied from these externally attached power transistors. No special heat precautions for the SLIC are required. The use of two NPF and PNP transistors enables bidirectional current flow of AC induction current exceeding the loop current.

For the power transistors any transistor is acceptable, if its V_{CEO} is over 100 V,
I_C max = 2A, $f_T > 50$ MHz and $h_{fe} > 60$, e.g. NPN Transistor 2SC2331 and PNP Transistor 21Sa 1008.

3. 2-Wire AC Input Impedance and Transmission Characteristics

The transmission characteristics are as follows:

- 2-wire return loss

$$h_{22} = \frac{Z_L - Z_T}{Z_L + Z_T} \qquad h_{22} = 0 \text{ if } Z_L = Z_T$$

- 4-wire — 2-wire insertion loss

$$h_{42} = \frac{2 \quad Z_L}{Z_L + Z_T} \qquad h_{42} = 1 \text{ if } Z_L = Z_T$$

- Transhybrid loss

$$h_{44} = \frac{2 \quad Z_T \quad (Z_L - Z_B)}{(Z_T + Z_L)(Z_T + Z_B)} \qquad h_{44} = 0 \text{ if } Z_L = Z_B$$

- 2-wire — 4-wire insertion loss

$$h_{24} = \frac{2 \quad Z_T}{Z_L + Z_T} \qquad h_{24} = 1 \text{ if } Z_L = Z_T$$

whereby Z_L is the line impedance, Z_B the balancing impedance and Z_T the termination impedance or 2-wire AC input impedance.

Impedances scaled by a factor k are attached at the low voltage end of the SLIC to realize Z_T and Z_B. Thus a high capacitance value required for high voltage part could be reduced by k.

The scaling factor k is

$$k = \frac{R_{T3}}{4R_F}$$

and is normally set to k = 100. The minimum value for k is 25, the maximum value is 400, however the recommended value is between 50 to 200.

The desired 2-wire input impedance can now be set by externally attaching the impedance k ($Z_T - 4R_F$) between Z_T pin and Z_{IN} pin. Additionally, impedance k · Z_T is connected between Z_{TH} pin and Z_B pin, and impedance k · Z_B is connected between Z_B pin and ground.

The termination impedance Z_T and balancing impedance Z_B used in different countries can be represented by the circuitry in figure 3.

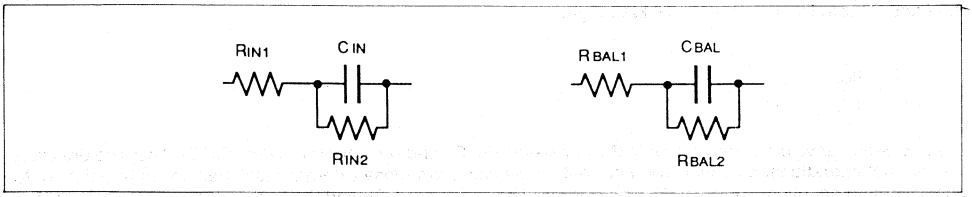


Figure 3. Termination and Balancing Network

The external components can now be determinated as follows:

$C_T = C_{IN}/K$

$R_{T3} = 4K\ R_F,\ R_F = 50\ \Omega$

$R_{T1} = k \cdot R_{IN1} - R_{T3}$

$R_{T2} = k \cdot R_{IN2}$

$C_{TH} = C_{IN}/K$

$R_{TH} = k \cdot R_{IN1}$

$R_{TH2} = k \cdot R_{IN2}$

$C_B = C_{BAL}/K$

$R_{B1} = k \cdot R_{BAL1}$

$R_{B2} = k \cdot R_{BAL2}$

For $R_{IN2} ---> \infty$ R_{T2} and R_{TH2} should be set to

$$R_{T2} = \frac{1}{2\pi f_C C_T}$$

and

$$R_{TH2} = \frac{1}{2\pi f_C C_{TH}}$$

with a cut off frequency fc = 10 . . . 12 Hz. This is to prevent oscillations in a low-frequency band from several Hz to several 10 Hz.

Example 1:

$Z_T = Z_B$ with $R_{IN1} = R_{BAL1} = 600\ \Omega$

$R_{IN2} ==> \infty$, $R_{BAL2} ==> \infty$

$C_{IN} = C_{BAL} = 2.16$

With k = 100 we get

$C_T = C_{TH} = C_B = 0.022\ \mu F$

$R_{T3} = 20\ k\Omega$

$R_{T1} = 40\ k\Omega$

$R_{TH1} = R_{B1} = 60\ k\Omega$

$R_{T2} = R_{TH2} = 600\ k\Omega$

$R_{B2} ==> \infty$

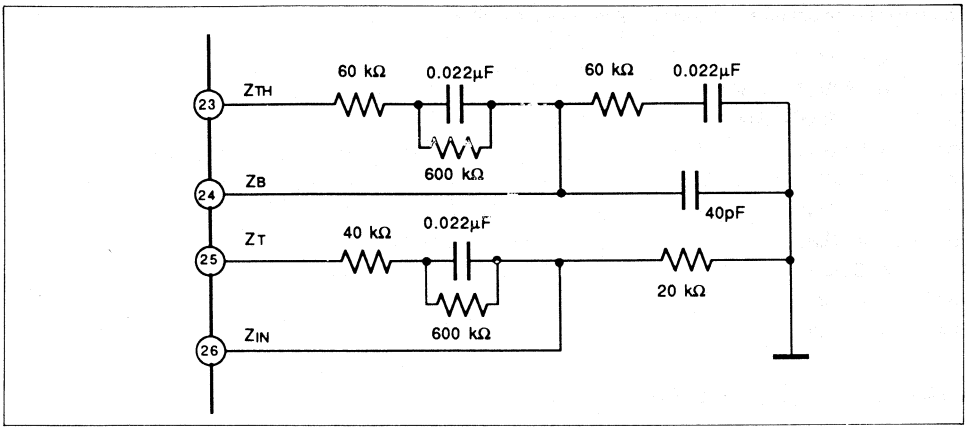


Figure 4. Example 1

Example 2:

$Z_T = Z_B$ with $R_{IN1} = R_{BAL1} = 220\ \Omega$

$R_{IN2} = R_{BAL2} = \Omega$

$C_{IN} = C_{BAL} = 115$ nF

$C_T = C_{TH} = C_B = 1.8$ nF $==> k = C_{IN}/C_T = 63.5$

$R_{T3} = 12.7\ k\Omega$

$R_{T1} = 1.3\ k\Omega$

$R_{TH1} = R_{B1} = 14\ k\Omega$

$R_{T2} = R_{TH2} = R_{B2} = 52.3\ k\Omega$

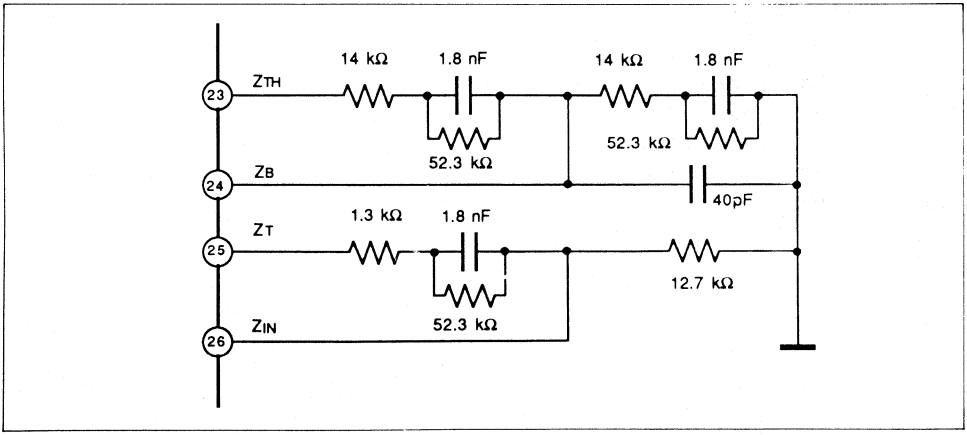


Figure 5. Example 2

Proportional Spacing by Using the μPD7220A Graphics Display Processor

Contents:

Author: Peter Westerdorf
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD7220A	Data Sheet
μPD7220A	Product Description
μPD7220A	Application Note
μPD41264	Data Sheet

Related Products

μPD7220A	Graphics Display Controller (6 MHz)	NMOS
μPD7220A-1	Graphics Display Controller (7 MHz)	NMOS
μPD7220A-2	Graphics Display Controller (8 MHz)	NMOS
μPD72020	Graphics Display Controller (8 MHz)	CMOS

1. Introduction

More and more printers are able to print with proportional spacing. Some high resolution matrixprinters as well as laserprinters can also combine text and graphics. Also the variety of text and text/graphics evaluation programs is increasing rapidly. Some of these programs, like textprocessors, provide proportional spacing capability for printouts, but unfortunately cannot display this on the personal computer screen. The character display electronics normally display the characters in a fixed size (e.g. in an 8 by 8 pixel matrix). The distance between two succeding characters is always the same and figures can not be displayed at all. So the demand for display driving electronics that are able to provide proportional spacing and graphics capability is increasing.

To solve this problem the conventional character display must be substituted by a full graphics display driving logic, which allows both characters and figures to be mixed and displayed in any desired combination. Here, even characters are "drawn" into the image memory, which means there is basically no difference between characters and figures.

There are different ways to develop such a display control logic. In a full bit mapped system where no graphics controller is implemented, the character printing and positioning, as well as the calculations for figures likes lines or circles, are done by the systems microprocessor. The calculation for drawing takes a lot of time and decreases the drawing speed.

The use of a Graphics Display Controller is the better solution. Such a controller, e.G. the μPD7220A, need only a few parameters and a command to do, by itself, all drawings to the separate display memory.

Moreover, this controller generates the video sync timing needed for the monitor and refreshes the dynamic RAM image memory. A realistic system for a color monitor of 8 colors with a resolution of 640 dots/line by 400 lines would need at least 768,999 bits equivalent to 96 KBytes image memory.

This application note describes a software of a system using the μPD7220A graphics display controller (= GDC).

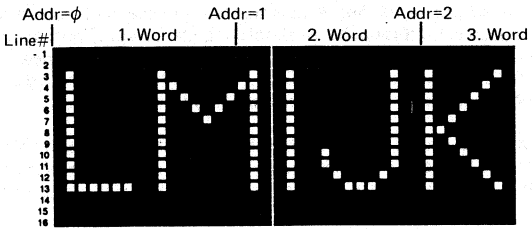
2. How to Program the GDC for Proportional Spacing

The problem which comes up when implementing proportional spacing into a display control software driver, is that characters must be placed in the video memory even if these would cross address-word boundries of the image memory. The image memory for a μPD7220A is normally arranged in succeeding words (one word = 2 bytes) to store the bits. For instance, a 640 by 400 in 8 colors memory arrangement would consist of 3 color plans of 32 Kbytes each. The first color plane would start at word-address 0, the second at address 16000, and the last at address 32000.

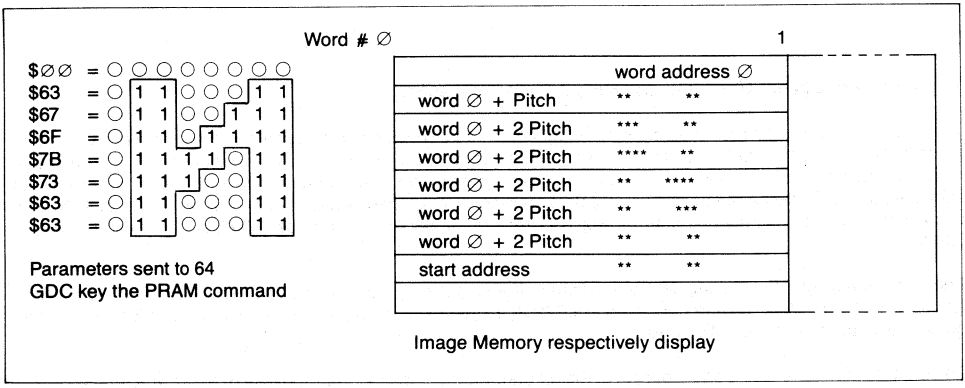
The appropriate line offset taking care of separating the succeeding words in the image memory into display lines can be done by the PITCH command of the μPD7220A. In the above exemple, the value would be 40. (Pitch = dots per line/dots per word = 640/16 = 40)

The characteristic of proportional spacing is the different width of display characters (e.g. the character "M" takes more horizontal dots for displaying than the character "I").

It can happen that only one, or even three or more characters would fit in only one word if regarding one horizontal line of the characters only. The following figure will show this more detailed.



The above figure should represent the upper left cursor of the display respectively of the image memory. As can be seen, all characters consist of a variable length in horizontal direction, but with a fixed length in vertical direction. Here the array of the "I" covers only 3 horizontal pixels by 16 lines. Moreover it can be seen that the first word-array (word-array = 26 words with a distance of 40 words between two succeeding words) stores the "L" completely but only part of the "M". The rest of the "M" is written into the second word, so here the "M" crosses a word boundary. The μPD7220A Graphics Display Controller provides the feature to write over a word boundary. Moreover it is possible to load an 8 by 8 array (= 8 bytes) into the μPD7220A which then will be placed and written into the image memory without further intervention of the system microprocessor. This feature is called "Graphics Character Drawing" and can be initiated in the following manner: The 8 bytes of a data array are sent to the GDC by using the PRAM command (PRAM = Parameter RAM). Then the desired start position is specified by CURS (= Cursor Set), and the FIGS (Figure Specify) command is used to specify that one Graphics Character should be written in direction to the right side. The following GCHRD (Graphics Character Draw) command will then start in execution.



As can be seen in the above example, the drawing direction is to the right and upper side, which means the GDC writes the appropriate bytes in direction to the top. So the start position must be specified to be at least 8* Pitch or multiples of it. The example also shows that the GDC reflects the bits written by the PRAM command. This is done to the fact that the GDC also starts with the last significant bit on the left side of a word.

Normally, the 8 by 8 resolution for an character array is not good enough, particularly high end applications in proportional spacing. So in the following, there is a description which uses a 16 by 16 character array.

Here it is necessary to use a special method of programming because the GDC only supports character array printing up to 8 by 8. First it is necessary to have an alphabet where the biggest character (mostly the "M" or "W") fits into the 16 by 16 matrtix.

This requires that the single characters must be coded into 16 words (one word = two bytes) written consecutively into the memory. One additional word should be written in front of the 16 words storting the horizontal dot count. The vertical amount of lines is always fixed at 16. (Note: When dividing the amount of lines themonitor can display by 16, this will yield the amount of character lines which are displayable. For example, 400/16 = 25.)

Using the ASCII standard, all printable characters are written one after the other in their 17 word equivalent into the memory. Normally, the first printable character is the Blank (= 32.), and the last . . . (= 125.).

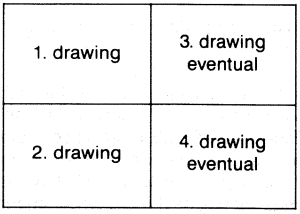
The arrangement of the character in the memory is then as follows:

Blank: (Length horizontal) ; (1. Word) ; (2. Word) ; (3. Word) ; ..... (16. Word)

: (Length horizontal) ; (1. Word) ; (2. Word) ; (3. Word) ; ..... (16. Word)

If now a character is printed which is normally given in its ASCII representation, it is easy to find the appropriate words in the alphabet table.

The addresses for reading the table are easy to calculate. The ASCII number of the character to be printed, minus 32, and the result, are multiplied by 17. They will point to the startposition in the character table. Now the PRAM command is used to print the left upper "sub matrix" of the normally 16 by 16 matrix into the image memory. Then the cursor is moved 8 lines below the just written 8 by 8 matrix, and the PRAM command is used once again, but now to print the lower left "sub matrix" of this character. Now the word used to store the horizontal length of the character is tested to verify if it contains more than 8 (dots). If yes, the similar procedures are done to draw the right side of the character, if not, the character drawing is terminated. So generally, two or four times the PRAM command is activated. Therefore, the horizontal length check saves a lot of drawing time.



A 16 by 16 character matrix

The arrangement of the several bytes of a character in the character table should be done in a way that the bytes to build up the right part of the character are stored just after their horizontal length word following each other, and then the right half bytes should follow. In case that the horizontal length of a character is less then 9 dots, the 16 right side bytes are senseless and are only used to have a constant offset for accessing the following characters. This arrangement of the bytes speeds up and simplifies the reading of the character table. The following figure will show this more detailed.

relative byte address:		0	relative byte address:		16
"	"	1	"	"	17
"	"	2	"	"	18
"	"	3	"	"	19
"	"	4	"	"	20
"	"	5	"	"	21
"	"	6	"	"	22
"	"	7	"	"	23
"	"	8	"	"	24
"	"	9	"	"	25
"	"	10	"	"	26
"	"	11	"	"	27
"	"	12	"	"	28
"	"	13	"	"	29
"	"	14	"	"	30
"	"	15	"	"	31

Arrangement of bytes of a character

horizontal length	
character byte 0	
character byte 1	
character byte 2	
: :	one character
: :	of 33 bytes
character byte 15	
character byte 16	
: :	
: :	
character byte 31	
horizontal length	
character byte 0	
: :	another character
: :	

Arrangement of characters in the character table

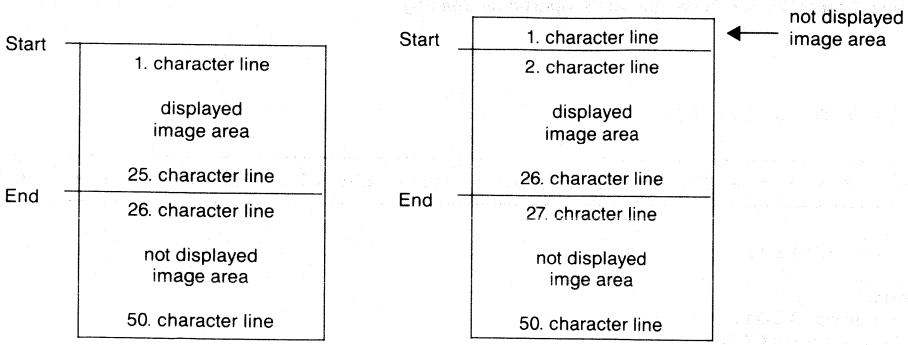
The next item to be discussed is now several characters are printed on the screen. Because of the different horizontal length of the characters, there are two software pointers necessary to put succeeding characters together. As the GDC cursor set command is allowed to specify the start word and the start bit within the word, these two pointers will take over this task.

The length value written in the front of each byte is added to the current start bit position of the current word. If this yields a value greater than 15, then the new start bit position is this value minus 16, and the current word address is incremented by one. So the succeeding character print start addresses in a line can be selected and the character can be printed. In case that this calculation yields a word-address value greater than 39 (amount of horizontal words (16 bit words) inclusive word 0), then a character line counter is incremented by one, and the stored word address and the start bit within that word, are reset to zero. The same must be done if the character to be printed is carriage return and/or line feed. Here, the assumption is made that carriage return and line feed will always occur together.

This is done as long as the character line count has not reached the value 25. (Note: This counter starts with a zero for the first time.) In case that the 25 is reached, then the start word address and the start bit address within that word are reset to zero.

When the display is filled up, the following lines can be displayed by first clearing the entire screen, and start printing again in the upper left corner.

Another more sophisticated method would be to implement a scroll function. This can be done by the help of the PRAM command which specifies the display memory start address and the amount of display lines. Normally the start address is zero and the amount of display lines is 400, which is equivalent to 25 character lines. Moreover, the image memory must be able to content an entire screen twice. Here the image memory must consist of 400 lines, which is equivalent to 50 character lines. If the screen is filled with 25 character lines, the following line is written into the not visible character line 26. After this has been done the PRAM command is used to set the actual start position to second character line, but the length remains the same value. This scrolls up the display by one line, meaning the first character of line 26 occurs on the screen at the bottom.



Scrolling up one line

This procedure is now done as long as the PRAM start position will not point onto the 27. character line. In case that a PRAM start position counter points to 26, then the next character line is written to the position of the first character line. Now the length of the first display position specified by PRAM is reduced by one character line and the PRAM command is now used to specify the start address of the second display area to character line one and with a length of one character line. This reduction of the first display area and the enlargement of the second display area is done with each following display line. This is done until the second display area also displays 25 character lines, and the second one displays none.

If then, still more character lines should be printed, the PRAM command is used to change the two display areas against each other. This restores the start position for the scrolling and must be done because the first display area is of higher priority. Otherwise, the upper and lower display port areas would be arranged in opposite succession.

3. Conclusion

What has been described up to now is a simple but powerful control program for proportional spacing, which can also be seen as terminal emulation program.

To complete this emulation program all the other important control characters like CTRL M, J, . . ., TAB, etc. can be implemented accordingly. Also, the extra display character set starting at ASCII 128, and ending at 255, can be implemented in the character alphabet.

Moreover, it would be possible to use escape sequences to change the colors of the characters. In this case, a character must be printed one, two or three times into the normally available three color planes (e.g. yellow = red + green). Also, the direct access of any startposition in the display area can easily be implemented with the GDC. This is called x-y-terminal, or direct cursor addressable terminal.

In "Appendix", a simple Pascal program can be found which reads a certain file from a disk and displays the content on a 640 by 400 monitor using proportional spacing. Here, the drawing is done in one color only, and all control signals except carriage return/line feed are ignored. Moreover, escape sequences and the scrolling have not been implemented yet. Each time a display has been filled up with 25 character lines, the pressing of carriage return clears the entire screen and the following display page is written.

Appendix: Pascal Source Code Listing "Proportional Spacing"

```
{ date : 5. July 1986 }

{-------------------------------------------------------------------------}
{* * * * * * * * GDC proportional spacing the NEC APC  * * * * * * * * }
{-------------------------------------------------------------------------}

program gdctst;

  const
    cmdreg=$301;
    paramreg=$300;
    statreg=$300;
    readreg=$301;

    charsel:array[0..3167] of byte = (
$04,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$04,$00,$00,$04,$04,$04,$04,$04,$04,$04,$04,$00,$04,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$06,$00,$00,$12,$12,$12,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$50,$50,$FC,$28,$28,$7E,$14,$14,$00,$00,$00,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$07,$00,$08,$1C,$2A,$0A,$0A,$0C,$18,$28,$28,$2A,$1C,$08,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0B,$00,$EC,$12,$12,$92,$4C,$20,$90,$48,$44,$42,$82,$00,$00,$00,$00,
     $00,$03,$02,$01,$00,$00,$00,$01,$02,$02,$02,$01,$00,$00,$00,$00,$00,
$0B,$00,$18,$24,$24,$14,$08,$14,$22,$42,$82,$44,$38,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$01,$01,$00,$01,$02,$00,$00,$00,$00,$00,
$04,$00,$04,$02,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$06,$08,$04,$04,$02,$02,$02,$02,$02,$02,$02,$04,$04,$08,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$06,$04,$08,$08,$10,$10,$10,$10,$10,$10,$10,$08,$08,$04,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$44,$28,$10,$FE,$10,$28,$44,$00,$00,$00,$00,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$10,$10,$10,$FE,$10,$10,$10,$00,$00,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$04,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$04,$04,$04,$02,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$00,$00,$00,$00,$00,$7E,$00,$00,$00,$00,$00,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$03,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$02,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$07,$00,$00,$20,$20,$10,$10,$08,$08,$04,$04,$02,$02,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$38,$44,$82,$82,$82,$82,$82,$82,$82,$44,$38,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$10,$18,$10,$10,$10,$10,$10,$10,$10,$10,$10,$00,$00,$00,$00,
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
```

```
$09,$00,$38,$44,$82,$80,$80,$40,$20,$10,$08,$04,$FE,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$FE,$40,$20,$10,$38,$40,$80,$80,$80,$42,$3C,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$40,$60,$50,$48,$44,$42,$FE,$40,$40,$40,$40,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$FE,$02,$02,$02,$3E,$40,$80,$80,$80,$42,$3C,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$38,$04,$02,$3A,$46,$82,$82,$82,$82,$44,$38,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$FE,$80,$80,$40,$40,$20,$20,$10,$10,$10,$10,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$38,$44,$44,$44,$38,$44,$82,$82,$82,$44,$38,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$38,$44,$82,$82,$82,$82,$C4,$B8,$80,$40,$38,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$03,$00,$00,$00,$00,$00,$02,$00,$00,$00,$00,$00,$02,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$04,$00,$00,$00,$00,$00,$04,$00,$00,$00,$00,$00,$04,$04,$04,$02,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$07,$00,$00,$00,$10,$08,$04,$02,$04,$08,$10,$00,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0B,$00,$00,$00,$00,$00,$FC,$00,$C0,$FC,$00,$00,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$01,$00,$00,$01,$00,$00,$00,$00,$00,$00,$00,
$07,$00,$00,$00,$02,$04,$08,$10,$08,$04,$02,$00,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$3C,$42,$40,$40,$20,$10,$08,$08,$08,$00,$08,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0C,$00,$00,$00,$F8,$04,$E2,$92,$92,$92,$62,$04,$78,$00,$00,$00,$00,
    $00,$00,$00,$01,$02,$04,$04,$04,$04,$03,$00,$00,$00,$00,$00,$00,
$0B,$00,$20,$20,$20,$50,$50,$88,$88,$FC,$04,$02,$02,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$01,$01,$02,$02,$00,$00,$00,$00,
$09,$00,$3E,$42,$42,$42,$3E,$42,$82,$82,$82,$42,$3E,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$78,$84,$02,$02,$02,$02,$02,$02,$02,$84,$78,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$3E,$42,$82,$82,$82,$82,$82,$82,$82,$42,$3E,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$7E,$02,$02,$02,$02,$3E,$02,$02,$02,$02,$7E,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$7E,$02,$02,$02,$02,$3E,$02,$02,$02,$02,$02,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0A,$00,$78,$84,$02,$02,$02,$E2,$02,$02,$02,$84,$78,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$01,$01,$01,$01,$00,$00,$00,$00,$00,$00,
$09,$00,$82,$82,$82,$82,$82,$82,$FE,$82,$82,$82,$82,$82,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$04,$00,$04,$04,$04,$04,$04,$04,$04,$04,$04,$04,$04,$04,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$80,$80,$80,$80,$80,$80,$80,$82,$82,$44,$38,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$82,$42,$22,$12,$0A,$06,$0A,$12,$22,$42,$82,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$02,$02,$02,$02,$02,$02,$02,$02,$02,$02,$7E,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
```

```
$0B,$00,$02,$06,$8A,$52,$22,$02,$02,$02,$02,$02,$02,$00,$00,$00,$00,
    $00,$02,$03,$02,$02,$02,$02,$02,$02,$02,$02,$02,$00,$00,$00,$00,
$0A,$00,$06,$06,$0A,$0A,$12,$22,$22,$42,$82,$82,$02,$00,$00,$00,$00,
    $00,$01,$01,$01,$01,$01,$01,$01,$01,$01,$01,$01,$00,$00,$00,$00,
$0A,$00,$78,$84,$02,$02,$02,$02,$02,$02,$02,$84,$78,$00,$00,$00,$00,
    $00,$00,$00,$01,$01,$01,$01,$01,$01,$01,$01,$00,$00,$00,$00,$00,
$09,$00,$3E,$42,$82,$82,$82,$42,$3E,$02,$02,$02,$02,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0A,$00,$78,$84,$02,$02,$02,$02,$02,$02,$42,$84,$78,$00,$00,$00,$00,
    $00,$00,$00,$01,$01,$01,$01,$01,$01,$01,$01,$00,$01,$00,$00,$00,$00,
$0A,$00,$3E,$42,$82,$82,$82,$42,$3E,$22,$42,$82,$02,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$01,$00,$00,$00,$00,
$09,$00,$38,$44,$82,$02,$04,$38,$40,$80,$82,$44,$38,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$FE,$10,$10,$10,$10,$10,$10,$10,$10,$10,$10,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0A,$00,$02,$02,$02,$02,$02,$02,$02,$02,$02,$02,$84,$78,$00,$00,$00,$00,
    $00,$01,$01,$01,$01,$01,$01,$01,$01,$01,$01,$00,$00,$00,$00,$00,$00,
$0B,$00,$02,$02,$04,$04,$88,$88,$50,$50,$20,$20,$20,$00,$00,$00,$00,
    $00,$02,$02,$01,$01,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0D,$00,$02,$02,$44,$44,$44,$A8,$A8,$A8,$10,$10,$10,$00,$00,$00,$00,
    $00,$08,$08,$04,$04,$04,$02,$02,$02,$01,$01,$01,$00,$00,$00,$00,$00,
$09,$00,$82,$82,$44,$44,$28,$10,$28,$44,$44,$82,$82,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$07,$00,$82,$82,$44,$44,$28,$28,$10,$10,$10,$10,$10,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$FE,$80,$40,$20,$20,$10,$08,$08,$04,$02,$FE,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$05,$0E,$02,$02,$02,$02,$02,$02,$02,$02,$02,$02,$02,$0E,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$00,$04,$04,$08,$08,$10,$10,$20,$20,$40,$40,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$05,$0E,$08,$08,$08,$08,$08,$08,$08,$08,$08,$08,$08,$0E,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$07,$08,$14,$22,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$FF,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$01,$00,$00,$00,$00,
$04,$02,$04,$08,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$00,$00,$00,$3C,$42,$78,$44,$42,$42,$62,$5C,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$02,$02,$02,$3A,$46,$82,$82,$82,$82,$46,$3A,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$00,$00,$00,$38,$44,$02,$02,$02,$02,$44,$38,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$80,$80,$80,$B8,$C4,$82,$82,$82,$82,$C4,$B8,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$38,$44,$82,$FE,$02,$02,$44,$38,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$06,$00,$18,$04,$04,$1E,$04,$04,$04,$04,$04,$04,$04,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$D8,$C4,$82,$82,$82,$82,$C4,$D8,$80,$40,$38,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
```

```
$08,$00,$02,$02,$02,$3A,$46,$42,$42,$42,$42,$42,$42,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$04,$00,$04,$00,$00,$06,$04,$04,$04,$04,$04,$04,$04,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$06,$00,$08,$00,$00,$0C,$08,$08,$08,$08,$08,$08,$08,$08,$08,$06,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$02,$02,$02,$42,$22,$12,$0A,$16,$22,$42,$82,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$05,$00,$06,$04,$04,$04,$04,$04,$04,$04,$04,$04,$04,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0D,$00,$00,$00,$00,$3A,$C6,$42,$42,$42,$42,$42,$42,$00,$00,$00,$00,
    $00,$00,$00,$00,$07,$08,$08,$08,$08,$08,$08,$08,$00,$00,$00,$00,
$08,$00,$00,$00,$00,$3A,$46,$42,$42,$42,$42,$42,$42,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$38,$44,$82,$82,$82,$82,$44,$38,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$3A,$46,$82,$82,$82,$82,$46,$3A,$02,$02,$02,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$B8,$C4,$82,$82,$82,$82,$C4,$B8,$80,$80,$80,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$07,$00,$00,$00,$00,$3A,$46,$02,$02,$02,$02,$02,$02,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$00,$00,$00,$3C,$42,$02,$3C,$40,$40,$42,$3C,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$07,$00,$04,$04,$04,$1E,$04,$04,$04,$04,$04,$24,$18,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$00,$00,$00,$42,$42,$42,$42,$42,$42,$62,$5C,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$82,$82,$44,$44,$28,$28,$10,$10,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$0B,$00,$00,$00,$00,$22,$22,$24,$54,$54,$88,$88,$88,$00,$00,$00,$00,
    $00,$00,$00,$00,$02,$02,$01,$01,$01,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$82,$44,$28,$10,$10,$28,$44,$82,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$09,$00,$00,$00,$00,$82,$82,$44,$44,$28,$28,$10,$10,$08,$08,$04,$04,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$08,$00,$00,$00,$00,$7E,$40,$20,$10,$08,$04,$02,$7E,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$05,$00,$08,$04,$04,$04,$04,$02,$04,$04,$04,$04,$04,$08,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$03,$00,$02,$02,$02,$02,$02,$02,$02,$02,$02,$02,$02,$02,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$05,$00,$02,$04,$04,$04,$04,$08,$04,$04,$04,$04,$04,$02,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,
    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00);

  type
    line=string[40];
```

```
  var
    hor_word:integer;
    in_word,lineoffset:integer;
    characters:text;

    gdcstatus:integer;
    f,aw,vs,hs,hfp,ph,hbp,vfp,al,vbp,il,pitchval,RMWopmode:integer;
    zoomvalue,patvalue,maskvalue,color,code,de:integer;
    figure,direction,DC,D,D2,D1,DM,octant,dotaddr,EAD:integer;
    xa,ya,savecol:integer;
    a1start,a1length,a2start,a2length:integer;

    lettersel:array[1..5,1..8] of byte;

    ende:boolean;

{-------------------------------------------------------------------------------}
{This procedure writes commands into the GDC's command register}

procedure outcommand(command:integer);

  begin
    repeat
    until (port[statreg] and $04) > 0;       { if > 0 then FIFO empty }
    port[cmdreg]:=command;
  end;

{-------------------------------------------------------------------------------}

{This procedure writes parameters into the GDC's FIFO}

procedure outfifo(parameter:integer);

  begin
    port[paramreg]:=parameter;
  end;

{-------------------------------------------------------------------------------}

{This procedure will set predeterminated initial parameters}

procedure initvalue;

  begin
    aw:=80;
    al:=400;
    hfp:=8;
    hs:=8;
    hbp:=16;
    vfp:=7;
    vs:=8;
    vbp:=25;
    f:=0;                  {f:=0 => drawing also during active display time}
    pitchval:=64;          {pitch=words in horizontal direction}
    RMWopmode:=3;          {RMW operation mode = SET mode }
```

```
    patvalue:=$ffff;
    maskvalue:=$ffff;
    il:=0;                  {il=0 => InterLaced mode not active}
    zoomvalue:=$00;
    ph:=0;                  {ph => ninth PITCH bit}
  end;

{-------------------------------------------------------------------------------}
{  #  #  #  #  GDC commands  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  }
{-------------------------------------------------------------------------------}

{This procedure resets the GDC, activates the external sync and can
 initialize the GDC's sync generator if desired}

procedure reset1;

  begin
    port[cmdreg]:=$00;                          {RESET 1 command }
    outfifo((f*16)+4+2 + il*(8+1) );            {mode byte}
    outfifo(aw-2);
    outfifo(((vs and $07)*32)+(hs-1));
    outfifo(((hfp-1)*4)+((vs and $18) div 8));
    outfifo(128+((ph and $0100)*64)+(hbp-1));
    outfifo(vfp);
    outfifo((al and $00ff));
    outfifo((vbp*4)+((al and $0300) div 256));
  end;

{-------------------------------------------------------------------------------}
{ % % % % %   Demo-programs utilizing GDC commands   % % % % % % % % % % % % % }
{-------------------------------------------------------------------------------}

{This procedure can fill a certain memory area }

procedure fillarea;

  var
    charset:set of char;
    answer:char;

  begin
    outcommand($46);                    {ZOOM cmd to GDC / no magnification}
     outfifo($00);

    outcommand($49);                    {CURS}
     outfifo($00);
     outfifo($00);
     outfifo(color);

    outcommand($4a);                    {MASK}
     outfifo($ff);
     outfifo($ff);

    outcommand($78);                    {PRAM (RA8-RA15) }
     outfifo($ff);
     outfifo($ff);
```

```
    outfifo($ff);
    outfifo($ff);
    outfifo($ff);
    outfifo($ff);
    outfifo($ff);
    outfifo($ff);

   outcommand($4c);                    {FIGS}
    outfifo($10);
    outfifo(pitchval-1);               {words/line to be affected}
    outfifo($00);
    outfifo( lo(al) );                 {al=lines to be affected}
    outfifo( hi(al and $0300) );

   outcommand($20+RMWopmode);          {WDAT /0=replace;1=complement; }
                                       {      2=reset;3=set           }

   outcommand($68);                    {GCHRD}

  end;
{-------------------------------------------------------------------------}

{This procedure clear the display memory area }

procedure cleararea(color:integer);

  begin
    outcommand($46);                   {ZOOM cmd to GDC / no magnification}
     outfifo($00);

    outcommand($49);                   {CURS}
     outfifo($00);
     outfifo($00);
     outfifo(color);

    outcommand($4a);                   {MASK}
     outfifo($ff);
     outfifo($ff);

    outcommand($78);                   {PRAM (RA8-RA15) }
     outfifo($ff);
     outfifo($ff);
     outfifo($ff);
     outfifo($ff);
     outfifo($ff);
     outfifo($ff);
     outfifo($ff);
     outfifo($ff);

    outcommand($4c);                    {FIGS}
     outfifo($10);
     outfifo(pitchval-1);             {words/line to be cleared}
     outfifo($00);
     outfifo($ff);                    {lines to be cleared = 1023}
     outfifo($03);
```

```
    outcommand($22);                    {switch to Reset mode}

    outcommand($68);                    {GCHRD}

    outcommand($20+RMWopmode);          {return to former RMWopmode}
  end;

{-------------------------------------------------------------------------}

{This procedure clears the three display memory areas }

procedure screenclear;

  begin
    cleararea(0);
    cleararea(1);
    cleararea(2);
  end;

{-------------------------------------------------------------------------}

procedure test(character:integer);

  var
    in_word_started,hor_word_started,i:integer;

  begin
    in_word_started:=in_word;
    hor_word_started:=hor_word;
    { * * * write left part of character * * * }
    repeat until (port[statreg] and $04) > 0;     { if > 0 then FIFO empty }

    port[cmdreg]:=$78;                            {PRAM (RA8-RA15) }
    for i:=1 to 8 do port[paramreg]:=charsel[((character-32)*33)+i];

    port[cmdreg]:=$4c;                            {FIGS}
    port[paramreg]:=$12;
    port[paramreg]:=7;                            {words/line to be affected

    repeat until (port[statreg] and $04) > 0;     { if > 0 then FIFO empty }
    port[cmdreg]:=$49;                            {CURS}
    port[paramreg]:=lo(lineoffset+hor_word);
    port[paramreg]:=hi(lineoffset+hor_word);
    port[paramreg]:=in_word*16;

    port[cmdreg]:=$68;                            {GCHRD}

    repeat until (port[statreg] and $04) > 0;     { if > 0 then FIFO empty }
    port[cmdreg]:=$78;                            {PRAM (RA8-RA15) }
    for i:=9 to 16 do port[paramreg]:=charsel[((character-32)*33)+i];

    port[cmdreg]:=$4c;                            {FIGS}
    port[paramreg]:=$12;
    port[paramreg]:=7;                            {words/line to be affected
```

```
  repeat until (port[statreg] and $04) > 0;     { if > 0 then FIFO empty }
  port[cmdreg]:=$49;                            {CURS}
  port[paramreg]:=lo(lineoffset+hor_word+8*pitchval);
  port[paramreg]:=hi(lineoffset+hor_word+8*pitchval);
  port[paramreg]:=in_word*16;

  port[cmdreg]:=$68;                            {GCHRD}


{ * * * write right part of character * * * }
  if charsel[(character-32)*33] > 8 then
    begin
      in_word:=in_word + 8;
      if in_word >= 16 then
        begin
          in_word:=in_word-16;
          hor_word:=hor_word + 1;
        end;

      repeat until (port[statreg] and $04) > 0;  { if > 0 then FIFO empty }

      port[cmdreg]:=($78);                      {PRAM (RA8-RA15) }
      for i:=17 to 24 do port[paramreg]:=charsel[((character-32)*33)+i];

      port[cmdreg]:=($4c);                      {FIGS}
      port[paramreg]:=($12);
      port[paramreg]:=(7);                      {words/line to be affected}

      repeat until (port[statreg] and $04) > 0;  { if > 0 then FIFO empty }
      port[cmdreg]:=($49);                      {CURS}
      port[paramreg]:=(lo(lineoffset+hor_word));
      port[paramreg]:=(hi(lineoffset+hor_word));
      port[paramreg]:=(in_word*16);

      port[cmdreg]:=($68);                      {GCHRD}

      repeat until (port[statreg] and $04) > 0;  { if > 0 then FIFO empty }
      port[cmdreg]:=($78);                      {PRAM (RA8-RA15) }
      for i:=25 to 32 do port[paramreg]:=charsel[((character-32)*33)+i];

      port[cmdreg]:=($4c);                      {FIGS}
      port[paramreg]:=($12);
      port[paramreg]:=(7);                      {words/line to be affected}

      repeat until (port[statreg] and $04) > 0;  { if > 0 then FIFO empty }
      port[cmdreg]:=($49);                      {CURS}
      port[paramreg]:=(lo(lineoffset+hor_word+8*pitchval));
      port[paramreg]:=(hi(lineoffset+hor_word+8*pitchval));
      port[paramreg]:=(in_word*16);

      port[cmdreg]:=($68);                      {GCHRD}

    end;
```

```
      in_word:=in_word_started+charsel[((character-32)*33)+0];
      if in_word >= 16 then
        begin
          in_word:=in_word-16;
          hor_word:=hor_word_started + 1;
        end;
  end;

-------------------------------------------------------------------------------}

procedure readfile;

var
  character,x,i,j:integer;
  daten:array[1..30000] of char;

begin
  (* read a file form disk *)
  assign(characters,'prop21.pas');
  reset(characters);
  j:=0;
  while not eof(characters) do
    begin
      j:=j+1;
      read(characters,daten[j]);
    end;

  write(char(7));

  (* write the file with proportional spacing *)
  outcommand($23);                                    { set write mode }

  hor_word:=0;
  in_word:=0;
  lineoffset:=1024;
  for x:=1 to j do
    begin
      character:= ord(daten[x]);
      if character = 13 then
        begin
          lineoffset:=lineoffset+16*pitchval;
          hor_word:=0;
          in_word:=0;
        end;
      if lineoffset >= ((25+il*13)*16*pitchval) then
        begin
          gotoxy(1,1);
          writeln('Press Carriage Return to proceed');
          readln;
          screenclear;
          hor_word:=0;
          in_word:=0;
          lineoffset:=1024;
        end;
```

```
  { write(char(character)); }
    if character = 10 then character:=13;
    if character > 125 then character:=13;
    if character <> 13 then
      begin
        (* write the writable characters *)
        test(character);
        if hor_word >= 39 then
          begin
            lineoffset:=lineoffset+16*pitchval;
            hor_word:=0;
            in_word:=0;
          end;
        if lineoffset >= ((25+il*13)*16*pitchval) then
          begin
            screenclear;
            hor_word:=0;
            in_word:=0;
            lineoffset:=1024;
        end;
      end;
  end;
nd;

{-------------------------------------------------------------------------}
 ////////////////////////////////////////////////////////////////////////}
{****************** Start of main program ********************************}

 egin
    ClrScr;                              {clear control monitor screen}
    initvalue;

    port[statreg+2]:=256-aw;             { active word counter value }
    port[statreg+3]:=256-(hbp-2);        { HBP - (4*2xWCLK) counter value }

    port[cmdreg]:=$00;                   {RESET cmd to GDC / do not check}
                                         {the FIFO Empty bit before!!!!!!}

    delay(100);                          {Give the GDC some time to  }
                                         { execute the reset command }

    reset1;                              {RESET cmd to GDC / do not check}
                                         {the FIFO Empty bit before!!!!!!}

    outcommand($6f);                     {VSYNC to set master mode}

    outcommand($4b);                     {CCHAR cmd to GDC / no cursor}
    port[paramreg]:=($00);
    port[paramreg]:=($80);               {for interlaced mode bit7 and/or
                                          bit6 must be set/in non interlaced
                                          mode this has no effect}

    outcommand($47);                     {PITCH cmd}
    port[paramreg]:=(pitchval);
```

```
  outcommand($70);                    {PRAM cmd to GDC to specify displ.}
   port[paramreg]:=($00);                    {area 1 and its length. Here:
}
   port[paramreg]:=($00);                    {start = 0000h / length = 1032
}
   port[paramreg]:=($03);                    {bit0=1 and bit1=1 must be }
   port[paramreg]:=($20);

  outcommand($0f);                    {SYNC cmd / display enabled}

  screenclear;
  readfile;
  clrscr;
d.
```

Programming the GPIB Controller μPD7210

Contents:

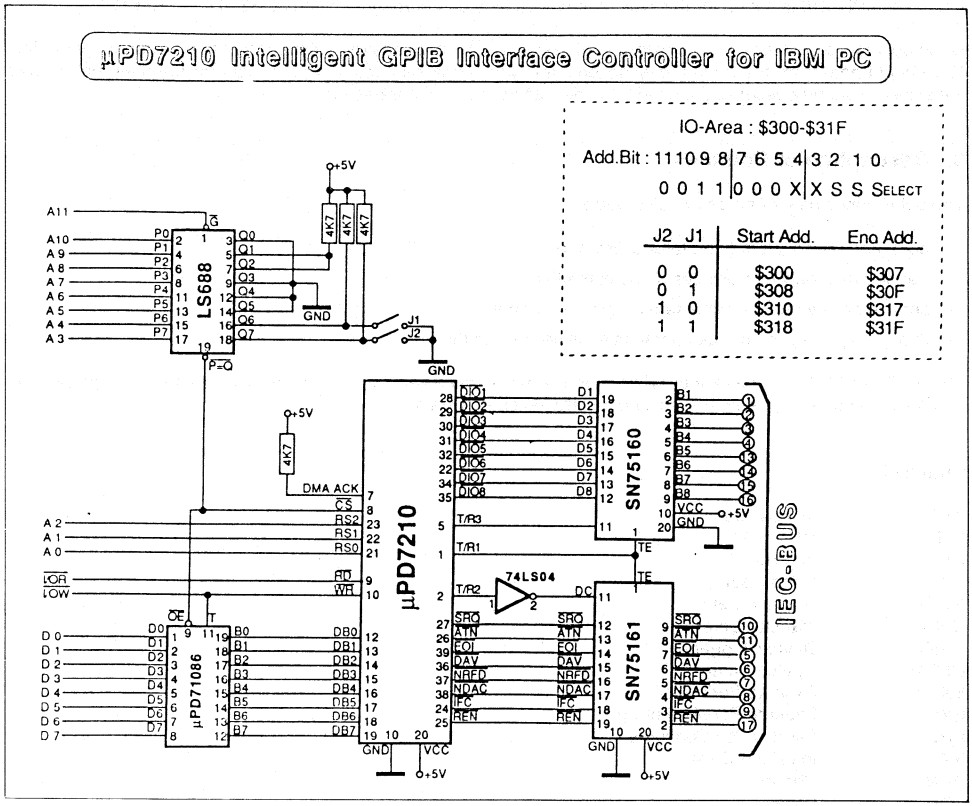
Author: Andreas Kohl
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD7210 Data Sheet
μPD7210 Product Description

1. Introduction

The μPD7210 allows implementation of a IEEE-488 (1978) compatible general purpose interface bus (GPIB), with only a minimum amount of hardware. In Europe, the IEEE-488 specification is adapted to the IEC-625 standard with the same electrical features, but different cable and connector. The complete schematic on an IEEE-488 controller board is shown below.



The GPIB bus is very powerful in terms of communication speed and control protocols. This application note provides a set of routines that allows handing of these protocols. All routines are written in μPD8085 assembly language and can be transfered to any other programming language without problem. Each piece of software is well documented so the user can easily extract those routines needed for personal application.

2. Software Considerations

Before any software should be written for a GPIB, the user should decide what IEEE functions are desired. These will define what processor routines are necessary. Second, the user should develop the necessary interrupt and software protocols to suit the users microcomputer system. Lastly, the necessary software will need to be written. At the end of this application note are program listings with descriptions that should help the user to understand what functions are used in this example, and the details of how these functions are implemented. The program flow for the routines included are as follows:

The μPD7210 (Talker/Listener/Controller) interface is activated by calling the INIT routine, thus initializing the μPD7210 and the GPIB. The INT1 routine should be called by any interruption generated because of changing bus status and polling requests. This routine reads the μPD7210 interrupt status and stores it in memory so that your software can check and respond when available.

The INT0 routine is used to speed data and polling transfers, and should be vectored by a different interrupt then INT1. The μPD7210 is set up for this type of operation as a part of routines such as SEND and RECV.

Now your main program can respond to changing status and initiate desired functions by calling a particular routine. You should note from the routine descriptions, that with some routines it is necessary for the main program to initialize its own processor registers with the necessary rooutine parameters.

3. Description of Routines

The subroutines described below allow you to

— configure your system (i. g., talkers, listeners, remote, local, IFC, etc.)
— transfer data between devices or the controller
— service device requests in a serial or parallel format
— finally, trigger synchronous operation of all devices on the bus.

The NEC μPD7210 can make your GPIB interface easier, more versatile, and more powerful than first generation controllers, and in most systems, second generation controllers also.

Routine List

INIT	Initialization
INT0	Interrupt 0
INT1	Interrupt 1
SEND	Send Data
RECV	Receive Data
XFER	Transfer Data
TRIG	Device Trigger
CLEAR	Device Clear
EXSP	Execute Serial Polling
EXPP	Execute Parallel Polling
PPC	Parallel Poll Configure
IFC	Interface Clear
REM	Remote
LOC	Local

3.1 Initialization

The operation mode of the μPD7210 is set in this routine. The first function of the routine is to issue a chip reset, thereby disabling the GPIB interface functions. The next operation is to set the operation modes of the μPD7210, which includes interrupt masks, address mode, device address and EOS code. To finalize this routine, the Immediate Execute pon command is issued, followed by the system interface clear command.

3.2 Interrupt 0

This routine is initiated by the DMA REQ signal which requests a data transfer when sending data, receiving data, or when executing a serial polling system command. Before executing these system commands, source or destina-

tion address of the first data byte is set to the HL register pair (data pointer) of the CPU. In Case of the Send Data Command, the BC register pair is used to hold the number of data bytes to be sent (data counter).

The subroutine's flow is as follows:

The TA (Talker Active) bit in the Address Status register is checked first to decide on the direction of the data transfer. If it is cleared (data input mode), GPIB data is available in the Data In register and is transferred to the memory location pointed by HL. The data pointer is then incremented, preparing for the next byte.

If the TA bit is set (data output mode), then the data byte pointed by HL is transferred to the Byte Out register, sending the data byte onto the GPIB. The data pointer is then incremented and the data counter (BC) is decremented. If the contents of the data counter (BC) become 1, a send EOI command issued, thereby alerting the μPD7210 to send END message with the next byte (Last byte). After the last byte is sent (BC = 0), the DMAO bit of the μPD7210 is reset to inhibit any further data transfer. Finally, a subroutine return is implemented.

3.3 Interrupt 1

This routine is initiated by the INT signal, caused by a SRQI (service request interrupt) or the CO (command output request) bit being set. The interrupt status is read from the Interrupt Status 2 register of the μPD7210 and is kept in memory because it is automatically cleared by the read. The interrupt status byte in memory will be read and cleared, bit by bit, by the routine which uses it.

3.4 Send Data

This routine sends data from the microcomputer system to one or more devices via the GPIB using the HL, BC, and DE register pairs of the CPU as data pointer, data counter and device address pointer respectively. Each device with its device address listed in an address table pointed by the DE register pair is assigned to receive data (a listener). Then all of the data in the data buffer specified by the HL and BC register pairs is transferred.

This routine assumes that device address of the microcomputer system is not involved in the address table. The last device address in the tableis assumed to be followed by a delimiter having a value which is greater than 30.

3.5 Receive Data

This routine is used to input data from a device whose device address is in the B register of the CPU. The system is configured such that the microcomputer is the only listener and receives data until an END or EOS (End of String) message is received. The data is stored in the data buffer pointed by the HL register pair. The talker device is assumed not to be the microcomputer system itself.

3.6 Transfer Data

This routine is used to transfer data from a device whose device address is in the B register to one or more devices whose device address(es) is (are) listed in the address table pointed by the DE register pair.

The microcomputer system does not receive the data, but lets itself be a listener so it can detect an END or EOS message from talker. On detection data transfer is terminated. The microcomputer system does not participate in the data transfer itself.

3.7 Device Trigger

This routine issues a GET (Group Execute Trigger) message onto the GPIB after fist addressing devices listed in the address table pointed by the DE register pair. This routine is intended to start operation in the listed devices.

3.8 Device Clear

This routine issues a SDC (Selected Device Clear) message onto the GPIB after first addressing devices listed in the address table pointed by the DE register pair. This routine is intended to clear or initialize these devices. If there is no device address listed, a DCL (Device Clear) message is issued instead of a SDC message thus clearing all devices.

3.9 Execute Serial Polling

This routine executes serial polling according to the sequence of device address in the address table pointed by the DE register pair. One byte of status from each device is received and stored in the data buffer pointed by the HL register pair in the same order as in the address table.

The device address of the microcomputer system is assumed not to be involved in the address table.

3.10 Execute Parallel Polling

This routine executes parallel polling and returns one byte of status, Parallel Poll Response, in the A register of the CPU.

3.11 Parallel Poll Configure

This routine configures one or more devices to respond to parallel polling assuming each device can implement the PP1 interface function. The device address is listed in the address table pointed by the DE register pair and the configuration information is stored in a buffer pointed by the HL register pair in the same order as in the address table.

The least significant five bits of the configuration byte are assumed to have the same format as PPE (Parallel Poll Enable), or the PPD (Parallel Poll Disable) message defined by the IEEE Standard 488.

3.12 Interface Clear

This routine activates the GPIB.s IFC line for 100 microseconds, causing the interface functions of all devices to go to known state. This routine is also executed at the end of the initialization routine.

3.13 Local

This routine issues a GTL (Go To Local) message after addressing the devices listed in the address table pointed by the DE register pair. This causes the addressed devices to go to the local state. If there are no device addresses listed in the address table, the GPIB's REN line is inactivated thus letting all of the devices go to local state. This routine is intended to enable the devices to receive local data.

3.14 Remote

This routine activates the GPIB's REN line enabling each device to go to remote state when addressed to listen.

Appendix: Software Listing

```
;
;*****************************************************************************
;*
;* UTILILTY ROULTINES FOR uPD7210 GPIB CONTROLLER
;*
;*****************************************************************************
;
; Write Registers
; ===============
;
BO&REG          EQU     00H
INT&M1          EQU     01H
INT&M2          EQU     02H
ADR&MODE        EQU     04H
AUX&MODE        EQU     05H
ADR&REG         EQU     06H
EOS&REG         EQU     07H
;
; Read Registers
; ==============
;
DI&REG          EQU     00H
INT&ST1         EQU     01H
INT&ST2         EQU     02H
ADR&ST          EQU     04H
CPT&REG         EQU     05H
;
; GENERAL EQUATES
; ===============
;
MY&ADR          EQU     04H
EOS&CODE        EQU     0DH
FREQ            EQU     08H
AUX&A           EQU     00H
;
; INITIALISATION
; ==============
;
INIT:   MVI     A,02H           ;CHIP RESET
        OUT     AUX&MODE
        XRA     A
        OUT     INT&M1          ;DISABLE INTERRUPTS OTHER THAN SRQI & CO
        MVI     A, 58H
        OUT     INT&M2          ;ENABLE DMAI,DISABLE DMAO
        MVI     A,31H
        OUT     ADR&MODE        ;SET T/R MODE 3,SET ADDRESS MODE1
        LDA     MY&ADR
        OUT     ADR&REG         ;SET MY ADDRESS 0-30 TO ADDRESS 0 REGISTER
        MVI     A,0E0H
        OUT     ADR&REG         ;DISABLE ADDRESS 1 REGISTER
        MVI     A,EOS&CODE
        OUT     EOS&REG         ;SET EOS CODE
        MVI     A, FREQ OR 20H
        OUT     AUX&MODE        ;SET FREQUENCY OF CLOCK INPUT 0-8 MHz
```

```
        MVI     A, AUX&A OR 80H
        OUT     AUX&MODE            ;SET AUXILIARY MODE A
        MVI     A,0A6H
        OUT     AUX&MODE            ;SET AUX.MODE B
                                    ;(SEND EOI IN SPAS,HIGH SPEED T1)
        XRA     A
        STA     INT&ST1             ;CLEAR WORKING AREA FOR INTERRUPT STATUS
        OUT     AUX&MODE            ;IMMEDIATE EXECUTE PON
        CALL    IFC
        JMP     WAIT&CO             ;RETURN ON ENTERING INTO CACS
;
;INTERRUPT 0
;===========
;
INT0:   PUSH    PSW
        IN      ADR&ST
        ANI     02H                 ;TA = 1?
        JNZ     DATA&OUT
        IN      DI&REG              ;DATA IN
        MOV     M,A                 ;STORE GPIB DATA
        INX     H                   ;INCREMENT DATA POINTER
RETURN: POP     PSW
        EI
        RET
DATA&OUT: MOV   A,M                 ;DATA OUT
        OUT     BO&REG              ;LOAD GIPB DATA
        INX     H                   ;INCREMENT DATA POINTER
        DCX     B                   ;DECREMENT DATA COUNTER
        XRA     A
        ORA     B
        JNZ     RETURN              ;RETURN IF (BC) IS GREATER THAN 2
        INR     A
        CMP     C
        JC      RETURN              ;RETURN IF (BC)IS GREATER THAN 2
        MVI     A,6H                ;(BC) = 0 OR 1
        OUT     AUX&MODE            ;SEND EOI WITH THE NEXT BYTE
        JZ      RETURN              ;RETURN IF (BC) = 1
        MVI     A,58H               ;(BC)= 0
        OUT     INT&M2              ;DISABLE DMA0 INTERRUPT
LOOP:   IN      INT&ST1
        ANI     2H                  ;DO = 1?
        JZ      LOOP                ;WAIT UNTIL HANDSHAKE IS FINISHED
        MVI     A,11H               ;TCA (TAKE CONTROL ASYNCHRONOUSLY) CODE
        OUT     AUX&MODE            ;ISSUE TCA TO 7210
        POP     PSW
        EI
        RET
;
;INTERRUPT 1
;===========
;
INT1:   PUSH    PSW
        PUSH    H
        IN      INT&ST2             ;READ INTERRUPT STATUS 2 REGISTER
```

```
        LXI     H,INT&ST1
        ORA     M
        MOV     M,A             ;SET INTERRUPT STATUS BYTE IN MEMORY
        POP     H
        POP     PSW
        EI
        RET
;
;UTILITY SUBROUTINES
;====================
;
WAIT&CO: PUSH   H
        LXI     H,INT&ST1
        MOV     A,M             ;LOAD INTERRUPT STATUS BYTE
        ANI     8H              ;CO = 1?
        JZ      WAIT&CO+1       ;WAIT UNTIL CO BIT IS SET
        DI
        XRA     M
        MOV     M,A             ;CLEAR CO BIT
        POP     H
        EI
        RET
;
;UNLISTEN
;========
;
UNLTN:  MVI     A,3FH           ;UNL (UNLISTEN)CODE
        OUT     BO&REG          ;ISSUE UNL ONTO GPIB
        JMP     WAIT&CO         ;RETURN WHEN HANDSHAKE IS FINISHED
;
;ADDRESS LISTENERS (DEVICE ADDRESS POINTER = DE)
;===============================================
;
ADR&L:  CALL    UNLTN           ;ISSUE UNL ONTO GPIB
LOOP1:  LDAX    D               ;LOAD DEVICE ADDRESS
        CPI     31              ;DELIMITER?
        RNC                     ;RETURN IF DELIMITER
        ORI     20H             ;FROM LISTEN ADDRESS
        OUT     BO&REG          ;ISSUE LISTEN ADDRESS ONTO GPIB
        INX     D               ;INCREMENT LISTEN ADDRESS POINTER
        CALL    WAIT&CO         ;WAIT UNTIL HANDSHAKE IS FINISHED
        JMP     LOOP1           ;REPEAT
;
;ADDRESS TALKER (TALK ADDRESS = B)
;=================================
;
ADR&T:  MOV     A,B
ADR&T1: ORI     40H             ;FROM TALK ADDRESS
        OUT     BO&REG          ;ISSUE TALK ADDRESS ONTO GPIB
        JMP     WAIT&CO         ;RETURN WHEN HANDSHAKE IS FINISHED
```

```
;
;SEND DATA (DATA POINTER = HL,DATA COUNTER=BC,DEVICE ADDRESSPOINTER=DE)
;=====================================================================
;
SEND:   CALL    ADR&L           ;ADDRESS LISTENERS
        IN      ADR&REG         ;READ MY ADDRESS
        ORI     40H             ;FROM MY TALK ADDRESS
        OUT     BO&REG          ;ISSUE MTA ONTO GPIB
        CALL    WAIT&CO         ;WAIT UNTIL HANDSHAKE IS FINISHED
        MVI     A,78H           ;INTERRUPT MASK 2
        OUT     INT&M2          ;ENABLE DMAO INTERRUPT
        MVI     A,10H           ;GTS (GO TO STANDBY)CODE
        OUT     AUX&MODE        ;ISSUE GTS TO 7210
        JMP     WAIT&CO         ;RETURN ON DATA CYCLE TERMINATION
;
;RECIEVE DATA(DATA POINTER = HL,TALK ADDESS = B)
;==============================================
;
RECV:   CALL    ADR&T           ;ADDRESS TALKER
        CALL    UNLTN           ;ISSUE UNL (UNLISTEN) ONTO GPIB
        MVI     A,13H           ;LTN (LISTEN) CODE
RECV1:  OUT     AUX&MODE        ;ISSUE LTN OR LTNC TO 7210
        MVI     A,1AH           ;TCSE(TAKE CONTROL SYNCHRONOUSLY ON END)CODE
RECV2:  OUT     AUX&MODE        ;ISSUE TCSE OR TCS TO 7210
        MVI     A,10H           ;GTS (GO TO STANDBY) CODE
        OUT     AUX&MODE        ;ISSUE GET TO 7210 TO INITIATE DATA CYCLE
        JMP     WAIT&CO         ;RETURN ON DATA CYCLE TERMINATION
;
;TRANSFER DATA (DEVICE ADDRESS POINTER = DE,TALK TALK ADDRESS = B)
;================================================================
;
XFER:   CALL    ADR&L           ;ADDRESS LISTENERS
        CALL    ADR&T           ;ADDRESS TALKER
        MVI     A,1BH           ;LTNC(LISTEN WITH CONTINUOUS MODE)CODE
        JMP     RECV1           ;ISSUE LTNC TO 7210,START DATA TRANSFER
                                ;AND RETURN ON DATA TRANSFER TERMINATION
;
;DEVICE TRIGGER (DEVICE ADDRESS POINTER = DE)
;============================================
;
TRIG:   CALL    ADR&L           ;ADDRESS LISTENERS
        MVI     A,8H            ;GET(GROUP EXECUTE TRIGGER) CODE
        OUT     BO&REG          ;ISSUE GET ONTO GPIB
        JMP     WAIT&CO         ;RETURN WHEN HANDSHAKE IS FINISHED
;
;DEVICE CLEAR (DEVICE ADDRESS POINTER = DE)
;==========================================
;
CLEAR:  LDAX    D               ;LOAD DEVICE ADDRESS
        CPI     31              ;NO DEVICE ADDRESS?
        MVI     A,14H           ;DCL(DEVICE CLEAR)CODE
        JC      CLEAR1
        CALL    ADR&L           ;ADDRESS LISTENERS
        MVI     A,4H            ;SDC(SELECTED DEVICE CLEAR)CODE
```

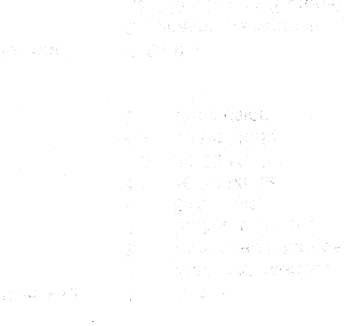
```
CLEAR1: OUT     BO&REG          ;ISSUE DCL OR SDC ONTO GPIB
        JMP     WAIT&CO         ;RETURN WHEN HANDSHAKE IS FINISHED
;
;EXECUTE SERIAL POLLING(DATA POINTER = HL,DEVICE ADDRESS POINTER = DE)
;=====================================================================
;
EXSP:   MVI     A,18H           ;SPE(SERIAL POLL ENABEL)CODE
        OUT     BO&REG          ;ISSUE SPE ONTO GPIB
        CALL    WAIT&CO         ;WAIT UNTIL HANDSHAKE IS FINISHED
        CALL    UNLTN           ;ISSUE UNL (UNLISTEN) ONTO GPIB
        MVI     A,13H           ;LTN (LISTEN) CODE
        OUT     AUX&MODE        ;ISSUE LTN TO 7210
LOOP2:  LDAX    D               ;LOAD DEVICE ADDRESS
        CPI     31              ;DELIMITER?
        JC      CONTN           ;CONTINUE IF NOT DELIMITER
        MVI     A,19H           ;SPD(SERIAL POLL DISABLE)CODE
        OUT     BO&REG          ;ISSUE SPD ONTO GPIB
        JMP     WAIT&CO         ;RETURN WHEN HANDSHAKE IS FINISHED
CONTN:  CALL    ADR&T1          ;ADDRESS TALKER
        MVI     A,12H           ;TCS(TAKE ONTROL SYNCHRONUOUSLY)CODE
        CALL    RECV2           ;ISSUE TCS AND GTS TO 7210
                                ;INITAITING STB TRANSFER
                                ;RETURN ON RECEPTION OF STB
        INX     D               ;INCREMENT TALK ADDRESS POINTER
        JMP     LOOP2           ;REPEAT
;
;EXECUTE PARALLEL POLLING (PARALLEL POLL RESPONSE = A)
;====================================================
;
EXPP:   MVI     A,1DH           ;EPP(EXECUTE PARALLEL POLLING)CODE
        OUT     AUX&MODE        ;ISSUE EPP TO 7210
        CALL    WAIT&CO         ;WAIT UNTIL LPARALLEL POLLING IS COMPLETED
        IN      CPT&REG         ;READ PPR FROM 7210
        RET
;
;PARALLEL POLL CONFIGURE (DEVICE ADDRESS POINTER=DE
;=========================  SECONDARY COMMAND POINTER = HL)
;
PPC:    LDAX    D               ;LOAD DEVICE ADDRESS
        CPI     31              ;DELIMITRE?
        RNC                     ;RETURN IF DELIMITER
        CALL    UNLTN           ;ISSUE UNL(UNLISTEN)ONTO GPIB
        LDAX    D               ;RELOAD DEVICE ADDRESS
        ORI     20H             ;FROM LISTEN ADDRESS
        OUT     BO&REG          ;ISSUE LISTEN ADDRESS ONTO GPIB
        CALL    WAIT&CO         ;WAIT UNTIL HANDSHAKE IS FINISHED
        MVI     A,5             ;PPC(PARALLEL POLL CONFIGURE)CODE
        OUT     BO&REG          ;ISSUE PPC ONTO GPIB
        CALL    WAIT&CO
        MOV     A,M             ;LOAD SECONDARY COMMAND
        ORI     60H             ;FORM SECONDARY COMMAND
        OUT     BO&REG          ;ISSUE PPE OR PPD ONTO GPIB
        INX     D               ;INCREMENT POINTERS
        INX     H
```

```
        CALL    WAIT&CO
        JMP     PPC             ;REPEAT
;
;INTERFACE CLEAR
;===============
;
IFC:    MVI     A,1EH           ;SIFC(SET IFC) CODE
        OUT     AUX&MODE        ;ISSUE SIFC TO 7210 ACTIVATING IFC GPIB LINE
        CALL    WAIT&100        ;WAIT 100 MICROSECONDS
        MVI     A,16H           ;RIFC(RESET IFC)CODE
        OUT     AUX&MODE        ;ISSUE RIFC TO 7210 INACTIVATING IFC
        RET
;
;LOCAL(DEVICE ADDRESS POINTER = DE)
;==================================
;
LOC:    LDAX    D               ;LOAD ADDRESS
        CPI     31              ;NO ADDRESS?
        JC      LOC1
        MVI     A,17H           ;RREN(RESET REN)CODE
        OUT     AUX&MODE        ;ISSUE RREN TO 7210 INACTIVATING GPIB LINE
        RET
LOC1:   CALL    ADR&L           ;ADDRESS LISTENERS
        MVI     A,1             ;GTL(GO TO LOCAL)CODE
        OUT     BO&REG          ;ISSUE GTL ONTO GPIB
        JMP     WAIT&CO         ;RETURN WHEN HANDSHAKE IS FINISHED
;
;REMOTE
;======
;
REM:    MVI     A,1FH           ;SREN(SET REN)CODE
        OUT     AUX&MODE        ;ISSUE SREN TO 7210 ACTIVATING REN GPIB LINE
        RET
;
WAIT&100: RET                   ; DUMMY DELAY ROUTINE
;
        END
```

128K Byte Memory Expansion for Slave Mode

Contents:

Author: K. Grohe
Application Department
NEC Electronics (Europe) GmbH



1. Outline

For some video applications the external memory size may not be sufficiently large. The following pages show how to expand local memory in slave mode by using external circuits.

2. Standard Interface

In slave mode, local memory can be accessed by 77230 via a 8 bit parallel bus. The memory size is limited to 4k Byte high speed and 4k Byte low speed memory. Figure 1 shows a circuit diagram using standard 8k Byte SRAM.

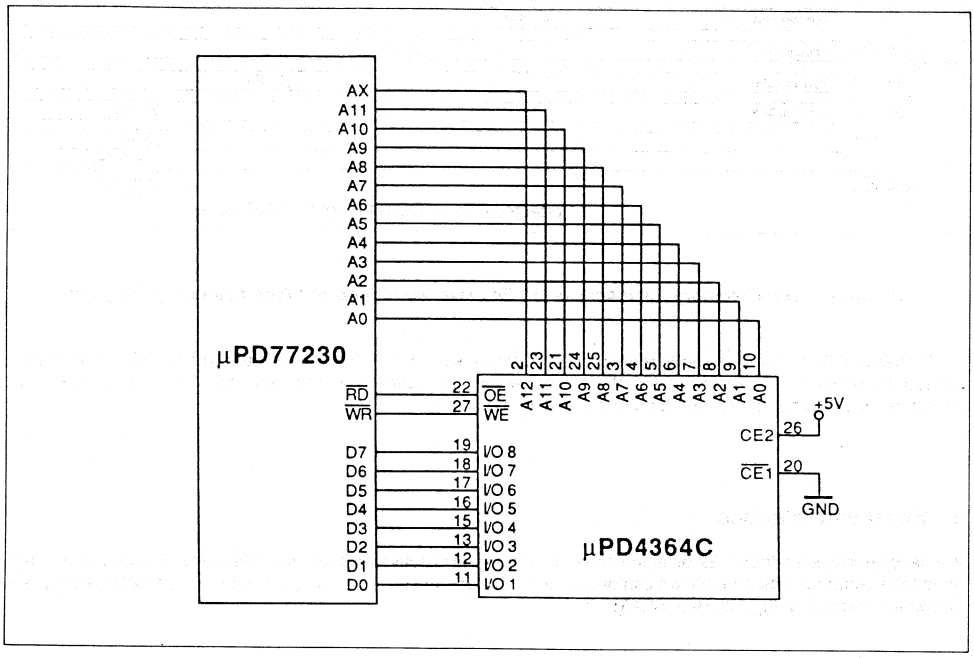


Figure 1. 8K Byte Local Memory Expansion (Slave Mode)

The data format of the local memory can be configured by presetting the DF Bits in the status register to 8, 16, 24 or 32 B it format. If a format of 32 Bit has been chosen, the RD (resp. WR) signal is activated 4 times and 4 consecutive Bytes are transferred between local memory and the internal DR register (see figure 2). After each byte, the positioning of the external 8 Bit bus to the internal DR register is updated due to the selected format. Both actions are running automatically driven only by the DF bits inside the Status register. Additionally, an internal ready flag is set after the RD (WR) instructions and reset after the 4th RD (WR) pulse. The ready flag can be tested by using the conditional jump instruction "JRDY". Besides these actions, the address register must be controlled by program. In other words, after reading one byte the AR register has to be updated by software using the INCAR (or DECAR) statement. Note that every AR register modification during RD or WR instruction is closely related to the timing requirements of 77230.

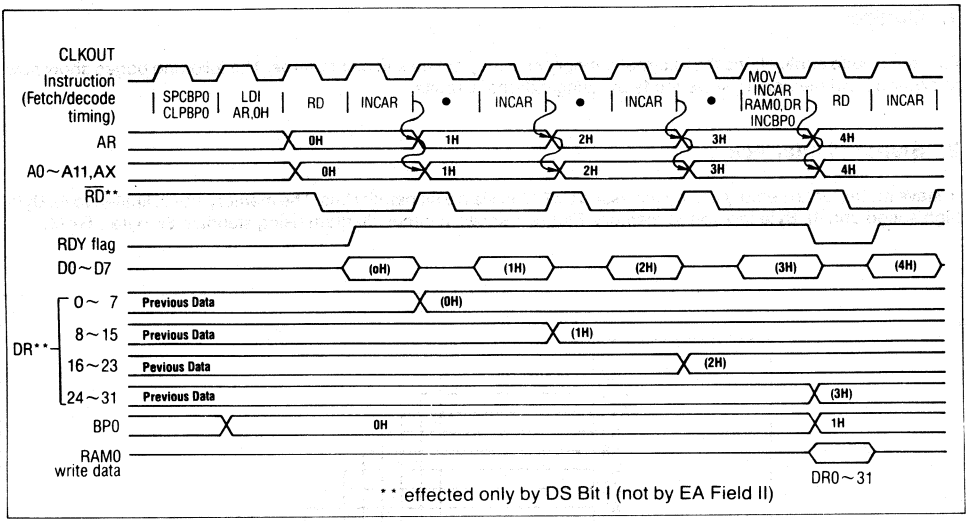


Figure 2. Local Memory Read Access (32 Bit Slave Mode) Address Modification by Program

For high speed RAM, one intermediate instruction must be inserted after every modification of the AR register is performed. In the case of low speed RAM, three intermediate instructions must be inserted. These may be NOP instructions or any other instruction for effective use of the pipeline.

3. Expanded Interface

We have seen above, that in 32 bit mode, the RD (WR) signal is activated 4 times. If the RD (WR) signal is now applied to an external 2 bit counter, we can generate 2 more addresses expanding the memory to 32k Byte! Figure 3 shows an example using 32k Byte SRAM.

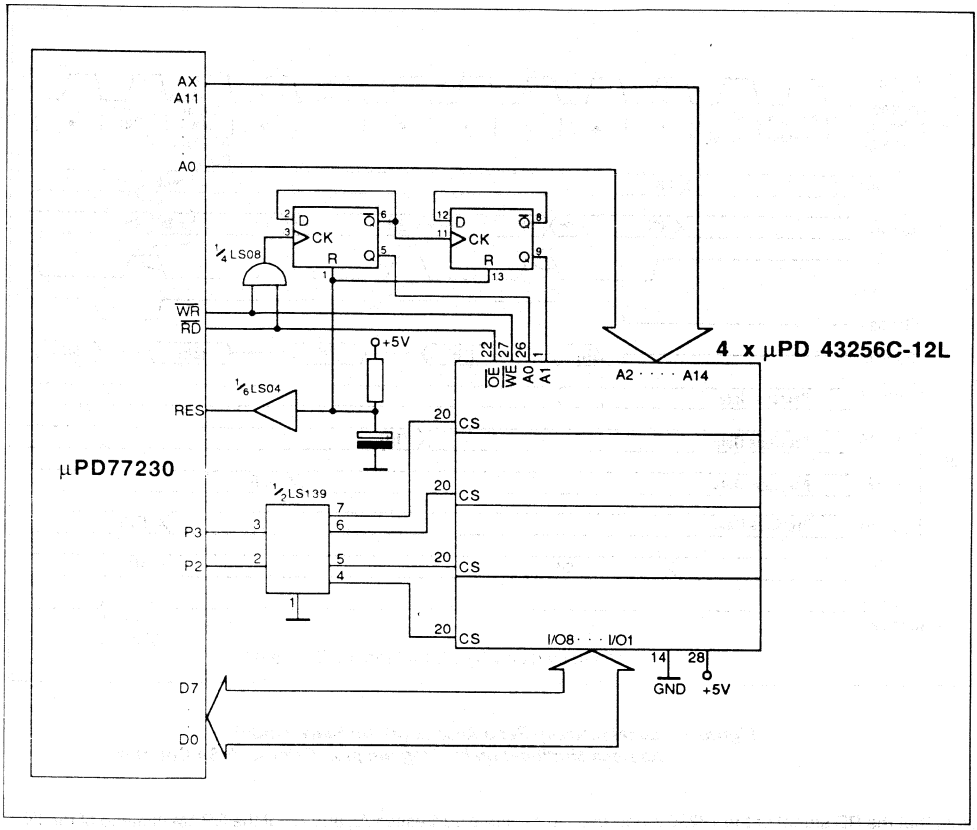


Figure 3. 128K Byte Local Memory Expansion (Slave Mode)

The positive transients of RD (resp. WR) signal are clocking a 2 bit binary counter using 2 D-Type flip flops. Their Q outputs are driving address lines A_0 and A_1 of the SRAMS. Additionally, a demultiplexer connected to portlines P_2 and P_3 are used to expand external memory to 128k Byte. Figure 4 shows the timing for such an interface.

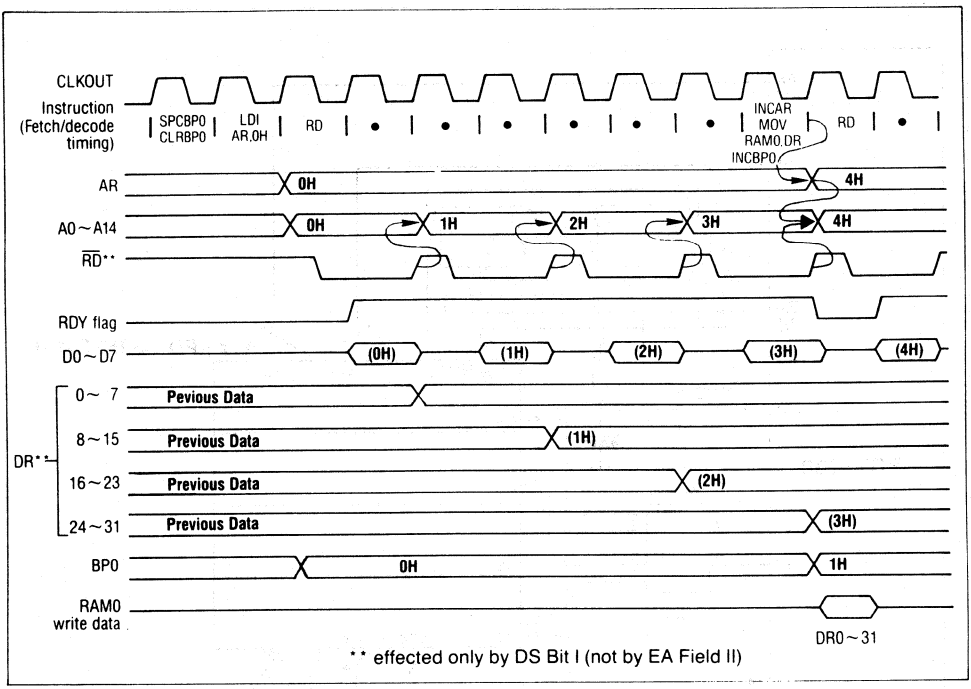


Figure 4. Local Memory Read Access (32 Bit Slave Mode) Address Modification by Program plus External 2 Bit Counter

Note that the RD signal of the 77230 is now used to generate addresses A_0 and A_1 of the SRAM (instead of the INCAR instructions!). A_0 to A_{11}, and A_X of the 77230 are connected to addresses A_2 through A_{14} of the SRAM. The INCAR instruction is now executed only once per 32 bit read cycle preceeding the next RD instruction. Note that only 2 processor cycles out of 8 cycles for a complete 32 bit read are affected by instructions. 6 cycles may be filled with other instructions, for example, a BIQUAD tap. In other words, such transfers may take place as a background task behind other processes or tasks.

4. Software Drivers

The initial condition of the counter is set by applying the same reset signal to both the 77230 and the 2 bit counter. Precautions must be taken to synchronize the external 2 bit counter to the internal program in other modes than 32 bit mode. In these cases, memory addressing becomes more complicated, because under certain addresses more than one word may be accessed!

4.1 8 Bit Mode:

In this mode the 77230 finds 4 words under the same address. The words may only be read sequentially. In monitoring the 2 bit counter state for synchronization, it is helpful to apply blocktransfer routine, but not to access a single word sequentially. Below some sample blocktransfer routines for data read and write are shown:

```
/*************************************************************
 *                                                           *
 *              8bit blockread                               *
 *              (reads 4 8Bit words)                         *
 *                                                           *
 *************************************************************/

        ldi     sr,8bitmode                                  ;
        ldi     ar,startaddress                              ;
        ldi     lc,blocksize                                 ;

rloop8: rd                                                   ;
                mov     ram0,dr         incbp0               ;
        rd                                                   ;
                mov     ram0,dr         incbp0               ;
        rd                                                   ;
                mov     ram0,dr         incbp0               ;
        rd                              declc                ;
        jmp     rloop8                                       ;
                mov     ram0,dr         incbp0  incar        ;
        ret                                                  ;
        nop                                                  ;


/*************************************************************
 *                                                           *
 *              8bit blockwrite                              *
 *              (writes 4 8Bit words)                        *
 *                                                           *
 *************************************************************/

        ldi     sr,8bitmode                                  ;
        ldi     ar,startaddress                              ;
        ldi     lc,blocksize                                 ;

                mov     dr,ram0         incbp0               ;
wloop8: wr                                                   ;
                mov     dr,ram0         incbp0               ;
        wr                                                   ;
                mov     dr,ram0         incbp0               ;
        wr                                                   ;
                mov     dr,ram0         incbp0  incar        ;
        wr                              declc                ;
        jmp     wloop8                                       ;
                mov     dr,ram0         incbp0               ;
        ret                                                  ;
        nop                                                  ;
```

4.2 16 Bit Mode

The 77230 is now finding 2 words under the same address. This could be real and imaginary 16 bit data. Again, these words can only be read sequentially. Applying the same blocktransfer technique results in the following programs:

```
/*************************************************************
 *                                                           *
 *              16bit blockread                              *
 *              (reads 2 16Bit words)                        *
 *                                                           *
 *************************************************************/

        ldi     sr,16bitmode                                 ;
        ldi     ar,startaddress                              ;
        ldi     lc,blocksize                                 ;

rlop16: rd                                                   ;
        nop                                                  ;
        nop                                                  ;
                mov     ram0,dr         incbp0               ;
        rd                                                   ;
        nop                             declc                ;
        jmp     rlop16                                       ;
                mov     ram0,dr         incbp0  incar        ;
        ret                                                  ;
        nop                                                  ;


/*************************************************************
 *                                                           *
 *              16bit blockwrite                             *
 *              (writes 2 16Bit words)                       *
 *                                                           *
 *************************************************************/

        ldi     sr,8bitmode                                  ;
        ldi     ar,startaddress                              ;
        ldi     lc,blocksize                                 ;

                mov     dr,ram0         incbp0               ;
wlop16: wr                                                   ;
        nop                                                  ;
        nop                                                  ;
                mov     dr,ram0         incbp0               ;
        wr                                                   ;
        nop                             declc                ;
        jmp     wlop16                                       ;
                mov     dr,ram0         incbp0               ;
        ret                                                  ;
        nop                                                  ;
```

4.3 24 Bit Mode

Addressing 24 bit words is more tricky. In order to achieve correct synchronization between program and external hardware, a 4 word blocktransfer using 3 consecutive addresses on the local data memory is used. The programs follow:

```
/*************************************************************
*                                                            *
*              24bit blockwrite                              *
*              (writes 4 24Bit words)                        *
*                                                            *
*************************************************************/

        ldi     sr,24bitmode                                ;
        ldi     ar,startaddress                             ;
        ldi     lc,blocksize                                ;

                mov     dr,ram0         incbp0              ;
wlop24: wr                                                  ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
                mov     dr,ram0         incbp0              ;
        wr                                                  ;
        nop                             incar               ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
                mov     dr,ram0         incbp0              ;
        wr                                                  ;
        nop                                                 ;
        nop                                                 ;
        nop                             incar               ;
        nop                                                 ;
                mov     dr,ram0         incbp0              ;
        wr                                                  ;
        nop                                                 ;
        nop                                                 ;
        nop                             declc               ;
        jmp     wlop24                                      ;
                mov     dr,ram0         incbp0  incar       ;
        ret                                                 ;
        nop                                                 ;
```

```
/*************************************************************
 *                                                           *
 *              24bit blockread                              *
 *              (reads 4 24Bit words)                        *
 *                                                           *
 *************************************************************/

        ldi     sr,24bitmode                                ;
        ldi     ar,startaddress                             ;
        ldi     lc,blocksize                                ;

rlop24: rd                                                  ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
                mov     ram0,dr         incbp0              ;
        rd                                                  ;
        nop                             incar               ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
                mov     ram0,dr         incbp0              ;
        rd                                                  ;
        nop                                                 ;
        nop                                                 ;
        nop                             incar               ;
        nop                                                 ;
                mov     ram0,dr         incbp0              ;
        rd                                                  ;
        nop                                                 ;
        nop                                                 ;
        nop                             declc               ;
        jmp     rlop24                                      ;
                mov     ram0,dr         incbp0  incar       ;
        ret                                                 ;
        rop                                                 ;
```

4.4 32 Bit Mode

32 bit addressing is rather straight forward. Comparing code against that of the Standard Interface, shows that 3 IN-CAR subinstructions can be eliminated resulting in a decreasing load of the processor's pipeline.

```
/************************************************************
 *                                                          *
 *              32bit blockread                             *
 *              (reads 1 32Bit word)                        *
 *                                                          *
 ************************************************************/

        ldi     sr,32bitmode                                ;
        ldi     ar,startaddress                             ;
        ldi     lc,blocksize                                ;

rlop32: rd                                                  ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
        nop                                     declc       ;
        jmp     rlop32                                      ;
                mov     ram0,dr         incbp0  incar       ;
        ret                                                 ;
        nop                                                 ;


/************************************************************
 *                                                          *
 *              32bit blockwrite                            *
 *              (writes 1 32Bit word)                       *
 *                                                          *
 ************************************************************/

        ldi     sr,32bitmode                                ;
        ldi     ar,startaddress                             ;
        ldi     lc,blocksize                                ;

                mov     dr,ram0         incbp0              ;
wlop32: wr                                                  ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
        nop                                                 ;
        nop                                     declc       ;
        jmp     wlop32                                      ;
                mov     dr,ram0         incbp0              ;
        ret                                                 ;
        nop                                                 ;

        end
```

5. Conclusion

This paper has illustrated that by applying a single 2 bit counter, the addressable memory size for 77230 in slave mode may be increased by a factor of 4. Another positive effect is the reduced software overhead achieved results in a reduced load of the processor's pipeline. It was recommended to perform memory I/O as a background task, with foreground processing leading to another dimension of pipelining, the pipelining of tasks.

External Memory Banking in the Master Mode for the μPD77230

Contents:

Author: C. Kellerhoff
Application Department
NEC Electronics (Europe) GmbH

1. Introduction

The following application note outlines the expansion and management of external memory in the master mode. A technique to increase the memory size without use of additional address lines is described. Any number of memory banks may be allocated with of a minimal amount of external hardware expense. Bank selection requires only one external data read instruction. High and low speed access areas of the external memory may be duplicated several times, and no restrictions are imposed on the content of each bank in the hich speed area, ie. instruction code and/or data. As an example, the technique is described using a four bank expansion in the high speed area.

2. The Master Mode

In contrast to the slave mode, the external ports are not available to "page" additional memory into the system. The master mode is characterised by an external 32 bit wide data bus and a 12 bit address bus. The maximum 8K length of external memory is divided into 4K slow speed and 4K high speed areas. Selection between the two is determined by the AX pin, whose state is automatically set when an external address is specified. The high speed area can contain either instructions or data, whereas the low speed area is restricted for the storage of data only. To select external instruction memory, specification of any address within the range 1000H—1FFFH, by a branch instruction, automatically resets the AX pin and causes the program counter PC0—PC11 to be latched by the address port (AP). The external instruction is in turn latched by the Data Port (DP) and directly decoded. On the other hand, data is accessed from external memory by direct designation of an address via the 13 bit address register (AR). This register may also be automatically incremented or decremented using the EA bits of the control field of an instruction. Figure 1 shows the complete memory map of the master mode without banking.

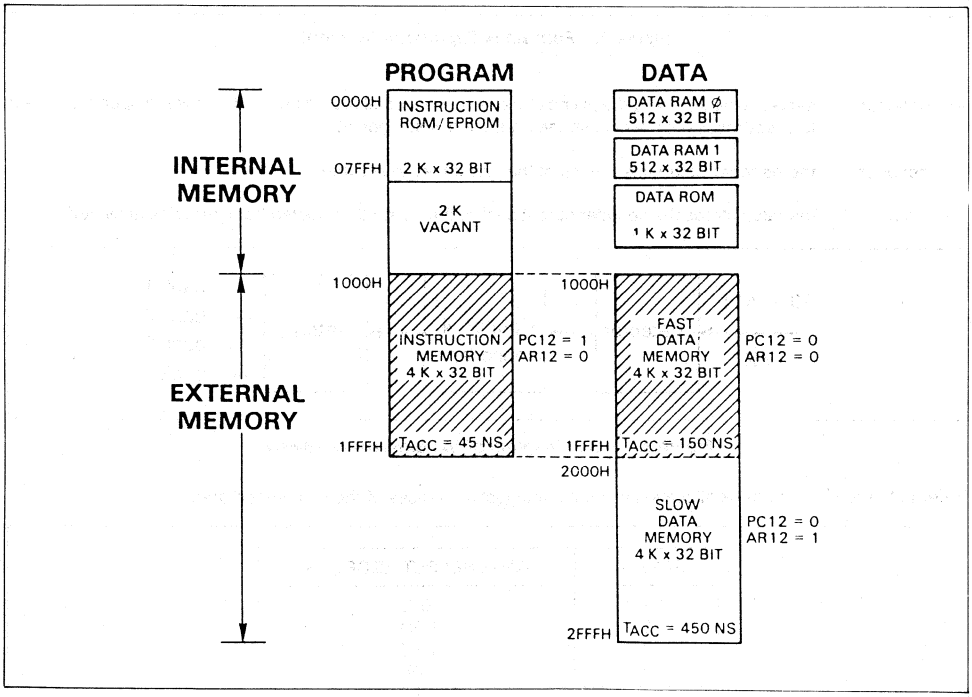


Figure 1. Master Mode Memory Map

3. Banking without the Use of Additional Address Lines

To toggle from one bank to the next without the use of extra address lines, a "soft switching" scheme is implemented. Figure 2 shows the basic concept of a four bank expansion. The block diagram in figure 3 illustrates the required sections.

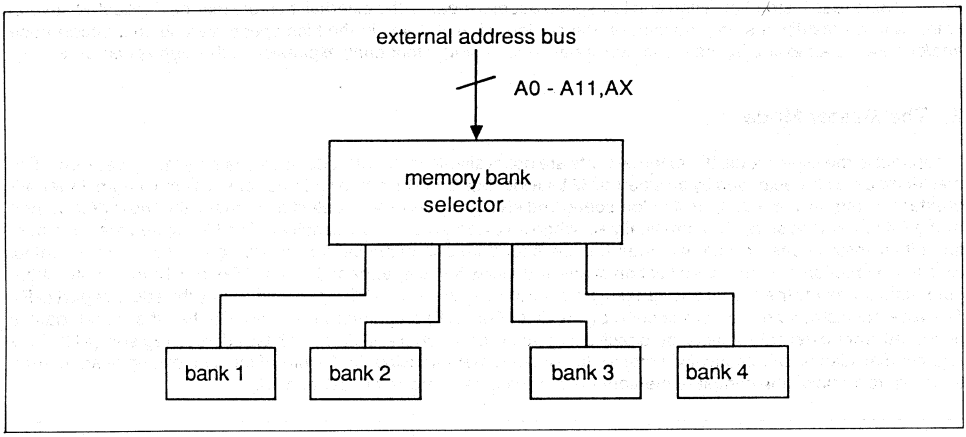


Figure 2. Four Bank Expansion Concept

— detector: serves to watch the address lines for one of the four allocated bank addresses to appear (the last four addresses of the high speed area are a wise choice)

— decoder: the decoder determines which of the four banks to activate

— latch: the latch stores the decoder output so that the currently selected bank remains activated

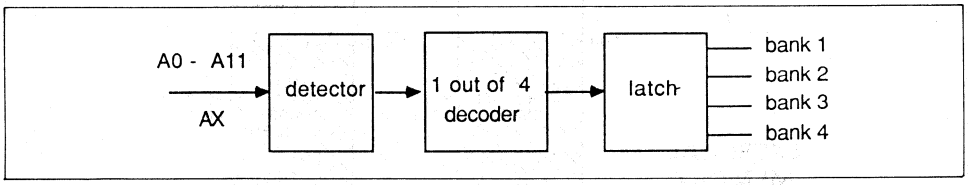


Figure 3. Block Diagram of the External Hardware

Refer to figure 4 for the bank allocation using the topmost addresses of the high speed area.

BANK Nr.	BANK SELECT ADDRESS
1	FFC
2	FFD
3	FFE
4	FFF

Figure 4. Bank Select Address Designation

Figure 5 shows the actual hardware setup for the memory bank expansion of the high speed area. The low speed area was purposely not included for reasons of clarity. The 175 latch is clocked by the falling edge of the /RD pulse, which is only gated through when one address of the bank select address range is detected. Thus, the decodes output is only stored when one of these addresses is output on the address bus. It should be noted that no to-be-processed data or instructions are to be stored in the bank select address range, as paging actually takes place during the low phase of the /RD pulse. To avoid collisions on the data bus as a result of one bank switching off while the other is being switched on, the OR gates block off the /RD pulse during this phase.

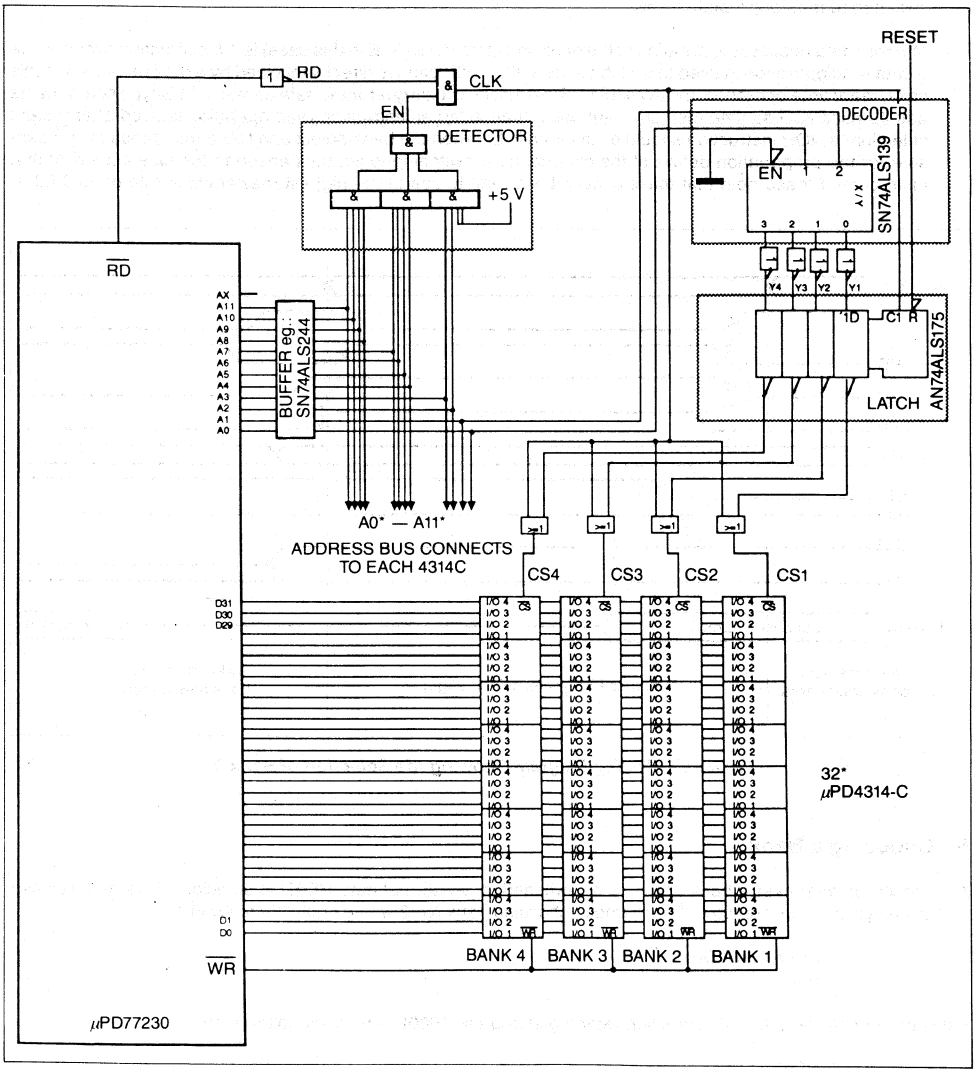


Figure 5. External Memory Banking Circuitry

4. Timing Considerations:

The timing diagram in figure 6 points out the critical timing intervals during and after a bank selection. The example shows the third bank being switched on (address FFE — see figure 4), thereby disabling bank 1. Timing intervals to watch out for are:

— Y1—Y4 decoder/inverter output settling time: before being latched by the CLK signal, the outputs must have switched to their appropiate levels.

— Memory data access: the duration of the bank select address (FFE in this case) is 1,5 instruction cycles since a data read operation is used to switch banks. If this instruction is directly followed by a data (of bank 3 in this case) read/write operation, the available 1,5 instruction cycles guarantee safe access if ALS-type components and 45-nsec access time memories are used. If an external instruction read operation is used after a bank selection, a NOP instruction should be inserted in between the bank-select and CALL instruction to compensate for the propagation delays of the external bank-switching logic, thus ensuring the safe access of that instruction. Please note that these observations refer to use of the highest master clock rate of 13.3 MHz.

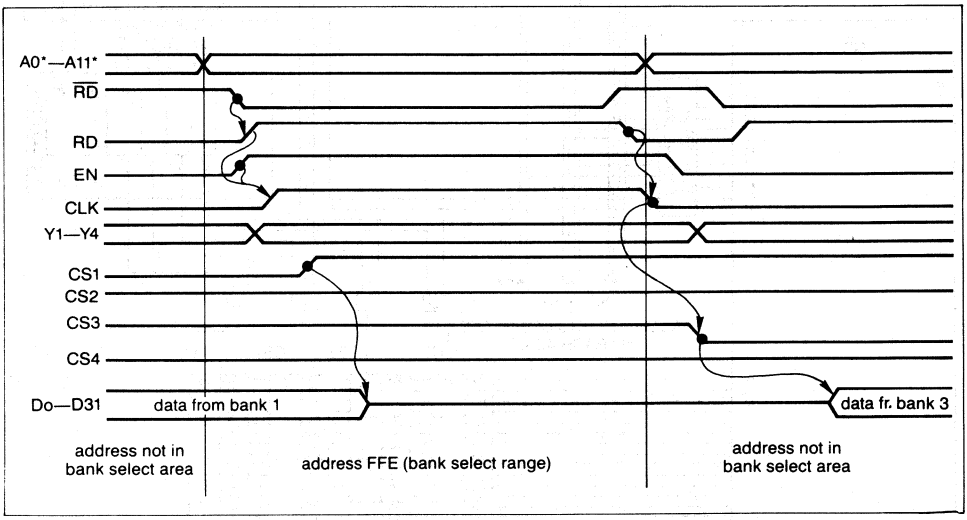


Figure 6. Timing Diagram Showing the Selection of Bank 3

5. Choosing a Bank:

After reset, or whenever a bank needs to be switched, one external data READ is sufficient. This requires two instructin cycles to be executed. For example, choosing bank Nr. 2 would imply the following:

```
LDI AR,1FFDH;
RD;
```

If this bank contains external instructions starting at address 1000H, one could follow with:

```
NOP;
CALL 1000H;
```

Or if data has been allocated to this bank which one wishes to transfer to the internal RAM, the following would suffice:

```
SPCBP0 CLRBP0;
LDI AR,0H;
RD;
INCAR INCBP0
MOV RAM0,DR;
RD;
INCAR INCBP0
MOV RAM0,DR;
etc.
```

6. Conclusion:

The four bank expansion example shows the ease with which the external memory size may be increased. It is intended as a suggestion and any number of banks can be added by following appropiate design steps. The slow speed area may also be utilized simply by including the AX pin to the memory chip select decoding. Furthermore, the addition of further external control hardware is worth considering. It will enhance the system so that deselected banks may be filled with new data/instructions from another system while the signal processor is busy with the external data/instructions of the activated bank.

Stand Alone full Duplex analog to ADPCM Interface using Speech Encoder Devices μPD7730/1

Contents:

Author: K. Grohe
Application Department
NEC Electronics (Europe) GmbH

1. Outline

This paper gives a proposal for a full duplex analog to ADPCM interface using NEC's Speech encoder/decoder 7730/1 (SED) devices.

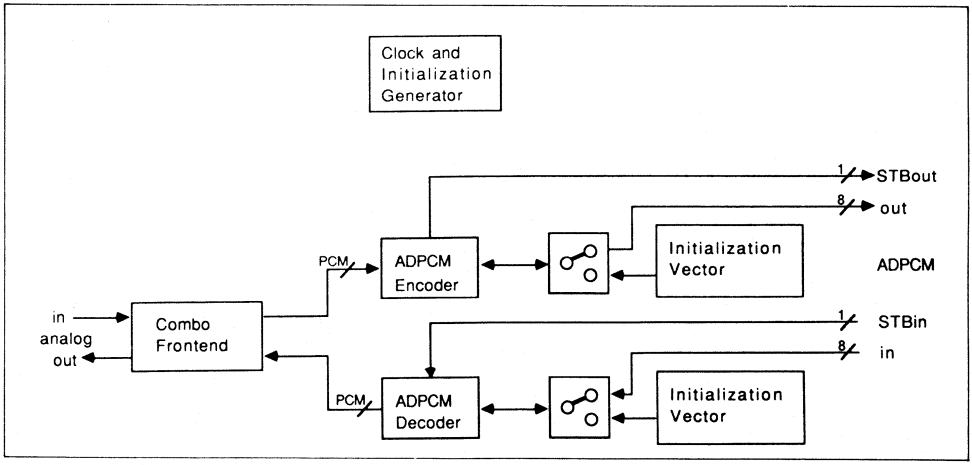
The goal is to by-pass an additional initialization processor for the SEDs, with the initialization being done using discrete devices.

2. Block Diagram

After power up, reset of the clock, and initialization, the generator feeds the combo and the 2 SEDs with its initial vectors. One SED supports the encoder function. Three vectors must be loaded as "mode setting", "threshold data — high part", and "threshold data — low part". The other SED supports the decoder function and needs only one vector a mode setting vector. The initialization procedure ends enabling the ADPCM channels.

Analog data is now digitized inside a combo. The PCM stream is fed to the 1st SED that encodes PCM to ADPCM data. Two ADPCM data words of 4 bit size are packed into one 8 bit word and send continously on to the ADPCM encoding bus every 250 μs. An encoder strobe signal ESTB controls the outgoing ADPCM stream.

On the other hand, 8 bit packed ADPCM data are strobed into the SED performing decoder function with DSTB control signal. The cycle time of the incoming ADPCM stream is 250 μs. The SED generates the adequate PCM stream and feeds the combo transmit section. The combo converts PCM data to an analog band limited signal.



Block Diagram: Stand Alone Analog to ADPCM Interface

3. Interface

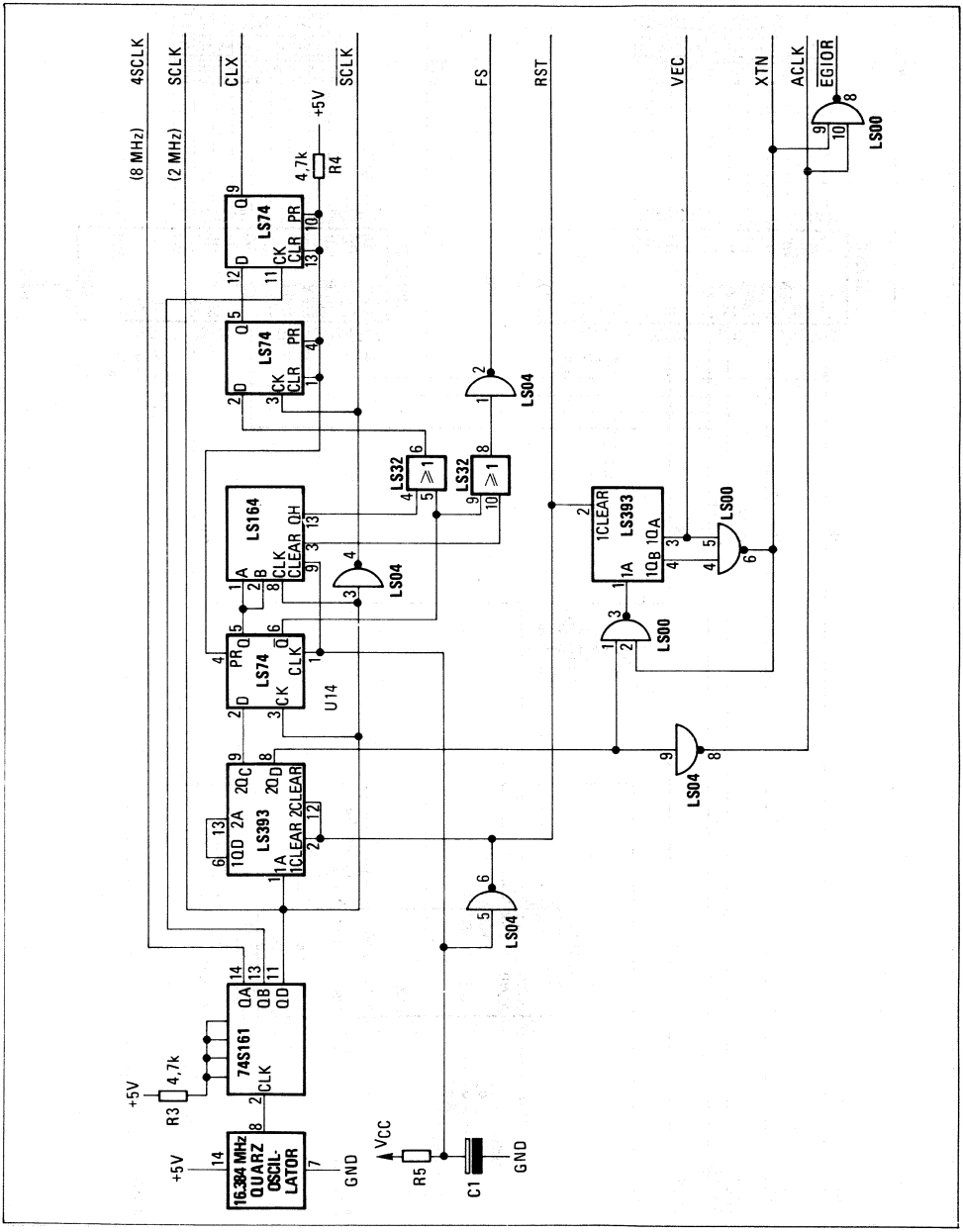
The analog interface uses a CMOS combo 95xx (see NEC telecom data book). The ADPCM interface consists of a 8 bit data bus and one control line for each section. Two ADPCM data words are packed into one 8 bit word in encoder mode. In decoder mode, one 8 bit word is unpacked into two ADPCM data words. The strobe signal ESTB and DSTB provide the correct timing to synchronize between the analog to ADPCM interface and the connected system. The cycle time of ADPCM data is 250 μs. With small hardware modifications, the external ADPCM busses could be taylored from parallel to serial interface.

4. Clock and Initialization Generator

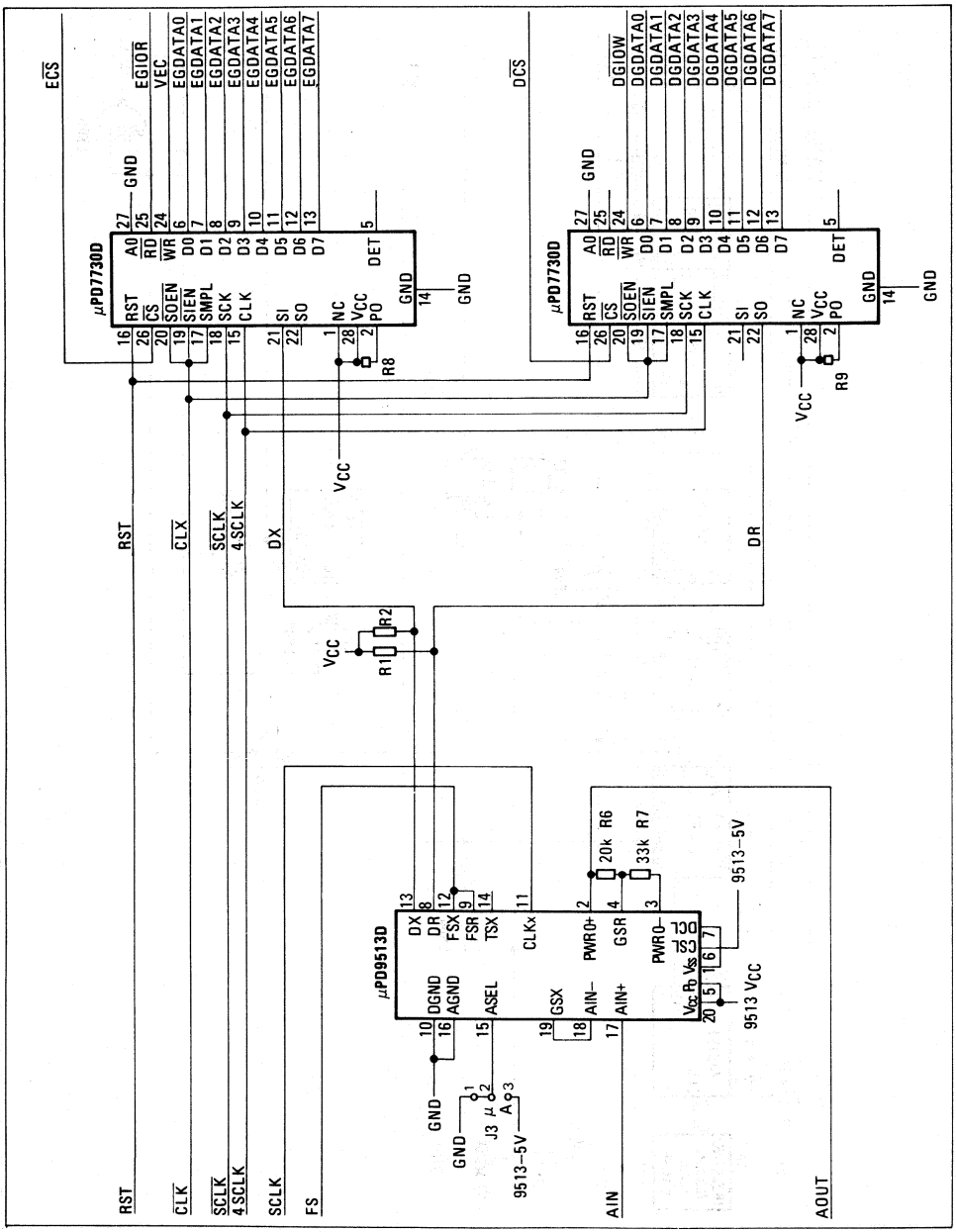
4 clock signals are provided by the clock generator, as are a 8.2 MHz master clock for the SEDs a PCM shift clock of 2048 KHz, a 8 KHz sampling clock for PCM framing, and a frame synchronization clock for the combo. This clock generator is part of the SED documentation and is well tested. The "power on reset" is performed by a RC combination. A 4 bit counter (LS393) is used to generate the initialization vectors. After power on reset, the external ADPCM busses are disconnected from the SEDs via XTN. The first 4 KHz transition on the clock input increments the counter and QA output becomes 1. The QA output is now spread over the encoder bus and the D6, D5 of the decoder bus, which gives the first vectors as FF for encoder, and 6x for decoder mode setting. Decoder initialization is now completed but the encoder needs 2 more vectors. The next 4 KHz transition sets QA to 0 thus giving an initialization vector of 00 for encoder (threshold data high part). The following 4 KHz transition resets QA again, and gives the last encoder vector as FF (threshold data low part). The clock input of the 4 bit counter is also deactivated. The counter will keep this status until power fails, at which time a new power on procedure takes places. As the counter is now deactivated, the external ADPCM busses are activated via XTN and normal operation starts.

5. Circuit Diagram

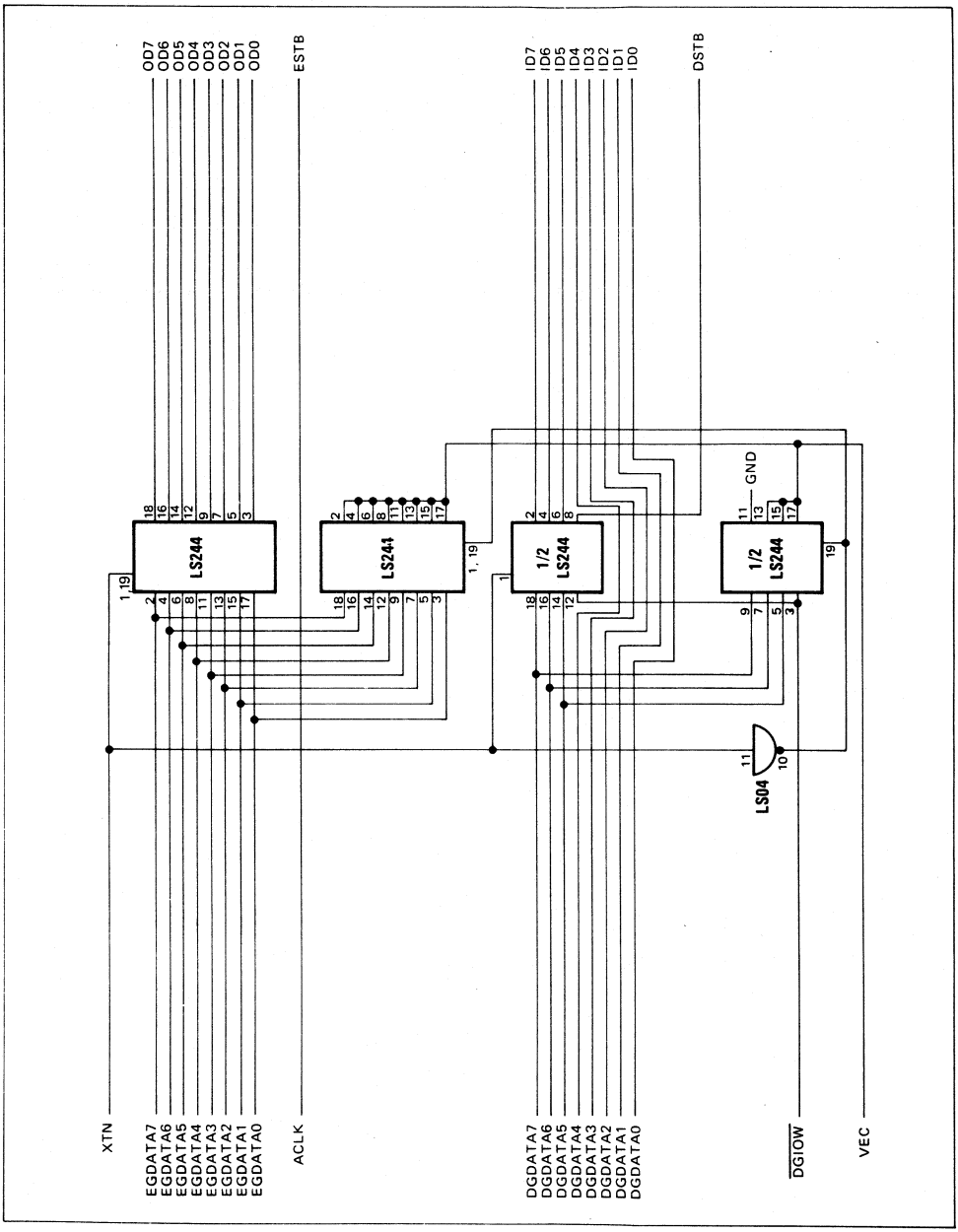
Attached is a preliminary circuit diagram of the interface requiring the Combo, 2 SEDs and 9 TTLs. This circuit diagram displays the realization of such an interface. Further reductions may be possible using higher integrated HCMOS instead of TTL devices.



Clock and Initialization Generator



Analog to ADPCM frontend



ADPCM Bus Switches

Speech Synthesis using Multiple μPD7756's

Contents:

Author: Andreas Kohl
Application Department
NEC Electronics (Europe) GmbH

Related Documentation

μPD775x	Family User's Manual
μPD775x	Data Sheet
μPD775x	Application Note

Related Products

μPD7755	ADPCM Speech Synthesizer	(92k ROM)
μPD7756	ADPCM Speech Synthesizer	(256k ROM)
μPD7759	ADPCM Speech Synthesizer	(ext. ROM)

1. Configuration of a μPD7756 Speech Synthesis System

A typical Speech Synthesis Module which is applicable for medical- and telecommunication equipment, cars, toys, cashregisters and other types of systems that can be improved by a natural speech output, is shown in figure 1.

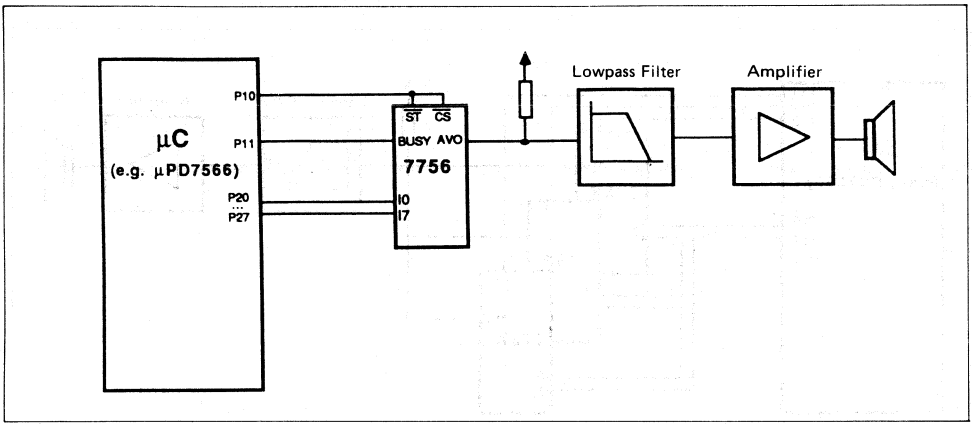


Figure 1. μPD7756 System Configuration

The speech synthesizer μPD7756 outputs messages selected through the inputs I0 to I7. As the output of the μPD7756 is already analog, only a simple lowpassfilter and an amplifier are needed to connect with a loudspeaker. The system is controlled by a single chip Microcomputer (e.g. 4-bit Processor like μPD7766) which generates the message numbers (I0 . . . I7) and selects the μPD7756 (CS). Synthesis of a message is begun upon receipt of a START pulse (ST) from the Microprocessor. The μPD7756 will then activate a Busy signal for as long as the message is synthesized. In all following examples, it is assumed that CS and ST are connected together. Figure 2 shows a typical signal at the analog output of μPD7756.

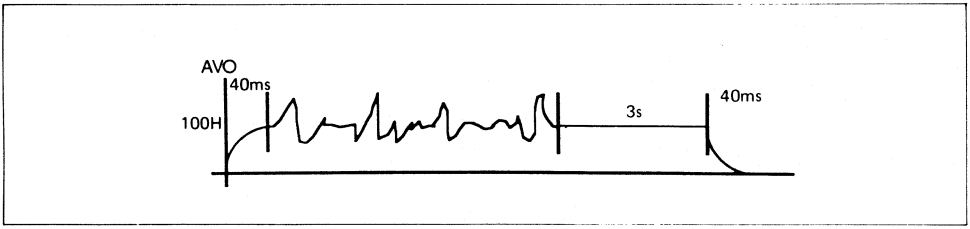


Figure 2. Analog Output Signal of μPD7755/56/59

After activating CS and ST, the speech synthesizer awakes from standby mode, gradually increases the analog output, and starts synthesis. After output of the message, the μPD7756 waits approximately three seconds before returning to standby mode. If the Microcomputer requests synthesis of another message within these three seconds, the μPD7756 outputs it immediately.

Using this feature, one can concatenate phrases to complete sentences. This allows to create a large, flexible vocabulary from a fixed pool of phrases stored in the speech memory.

2. Multi-μPD7756 Configuration with Separated Chip Selects

In applications where a large set of messages have to be synthesized, a single μPD7756 will not be able to store all speech phrases. In that case one has the choice of using a μPD7759 with external memory or to use multiple μPD7756's.

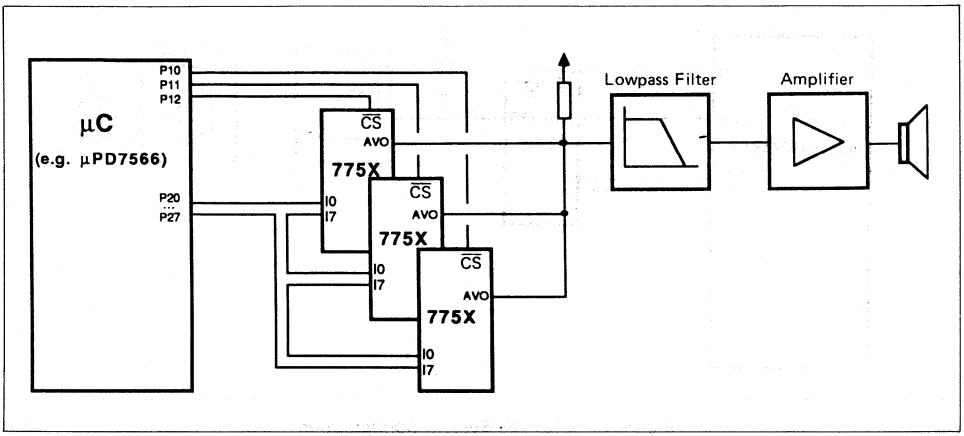


Figure 3. A Multi-μPD7756 System with Separate CS

Above configuration allows easy to expand speech synthesis capacity. The system control is almost identical to the single-μPD7756 configuration because each speech synthesizer is individually selected by CS. The number of messages that can be handled by such system is n times 256 (n = # of μPD7756).

Care must be taken not to activate two (or more) speech synthesizers simultaneously. In that case, the output signals would superimpose resulting in a distorted synthesis.

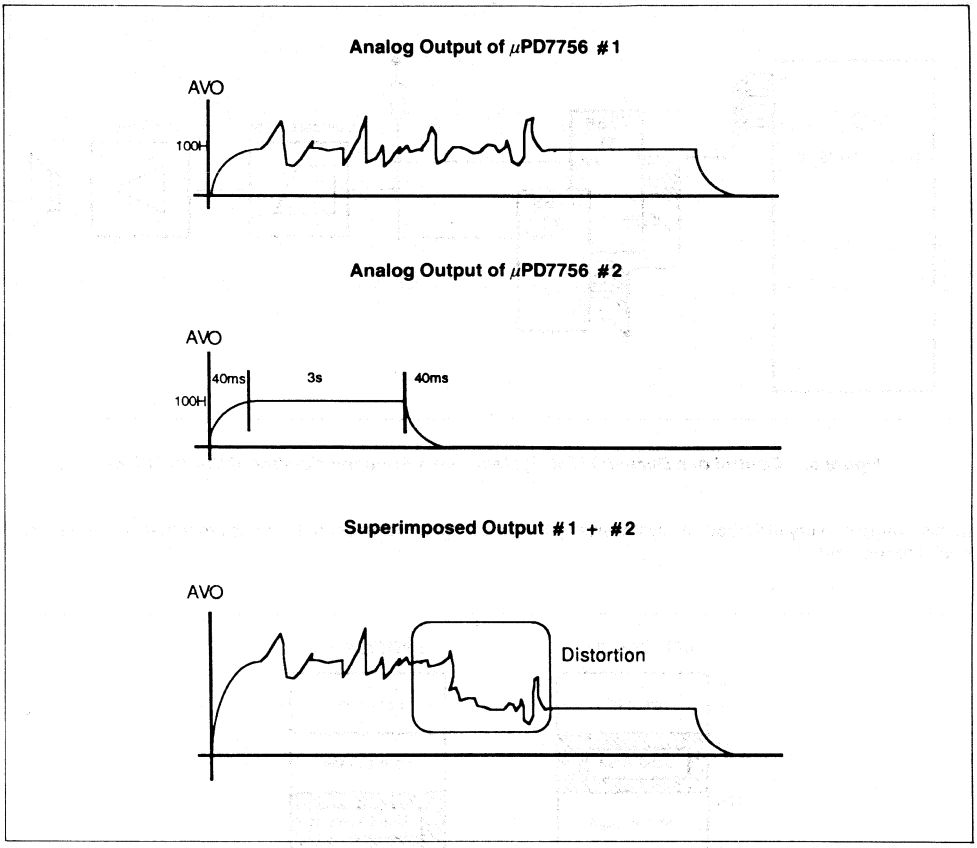


Figure 4. Overlapping of two Output Signals

Even if only one μPD7756 outputs a speech signal and all others output at the same time a silence signal or "nothing" (no message is stored for that specific message number), the output result is faulty, as can be seen from figure 4. The reason is, that every μPD7756 which is activated through CS awakes from standby mode and gradually increases the D/A converter output to one half of the full scale.

Consequently, concatenation is restricted to concatenation of phrases which reside in the same μPD7756 speech memory. If two messages from different μPD7756's must be concatenated then a pause of three seconds must be inserted after synthesis of the first message before the next message is started. This ensures that the first μPD7756 has gone to standby and is thereby not driving the output anymore. However, this is not a significant restriction, as up to 256 phrases in each connected μPD7756 may be concatenated. The average number of messages per μPD7756, in most applications, is 10 to 30.

Taking this into account, the number of microprocessor parts used to control the speech output system may be further reduced.

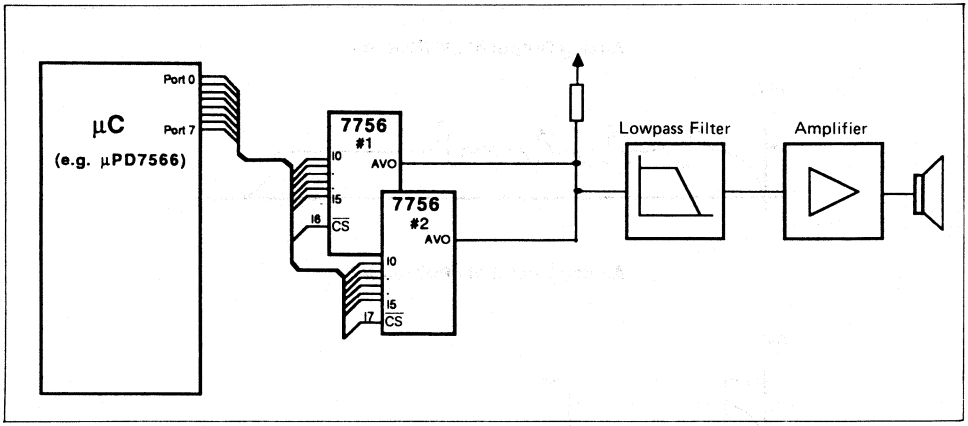


Figure 5. Control of a Dual-µPD7756 System with a Reduced Number of Control Lines

In this example, every µPD7756 can synthesize up to 64 messages. No other control ports other than the "message bus" are required.

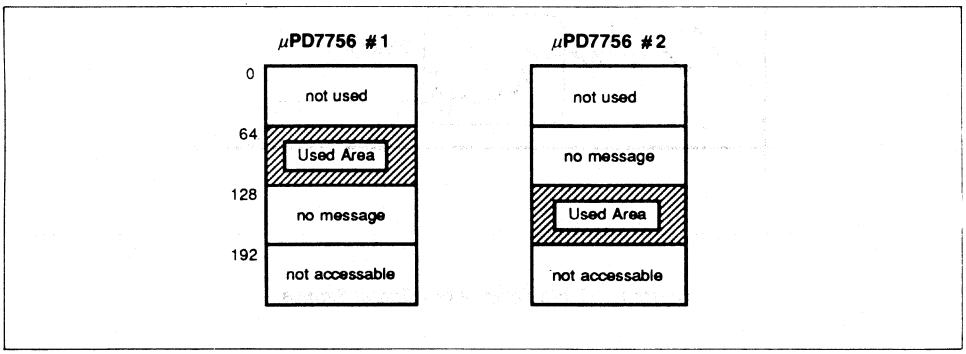


Figure 6. Speech Memory Organization

This figure shows the memory organization for above system. Addresses 0 to 63 are prohibited because in this range both µPD7756's would be activated and result in the described analog output conflict.

3. Multi-μPD7756 Configuration with Common Chip Selects

Analog output contention may be avoided by adding only the AC part of the speech. In this way it is possible to simultaneously activate two (or more) μPD7756's with a common CS. Only one of the activated synthesizers should output a valid message. All others will output a silence period.

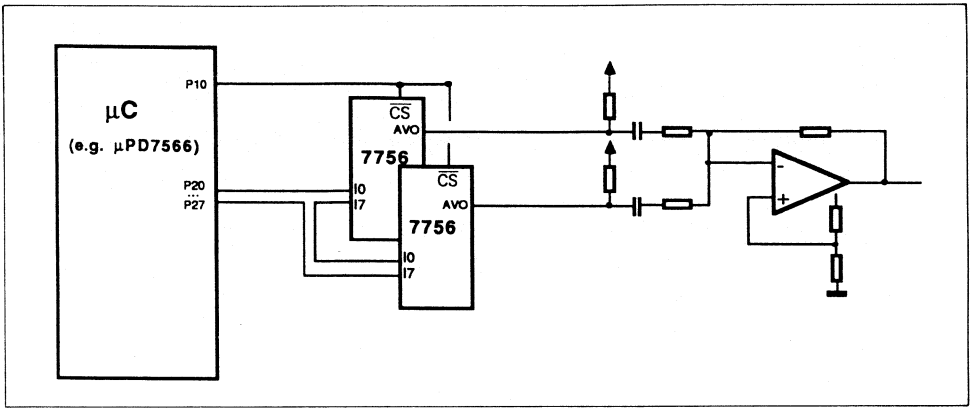


Figure 7. Two μPD7756's with Analog Outputs, AC Coupled

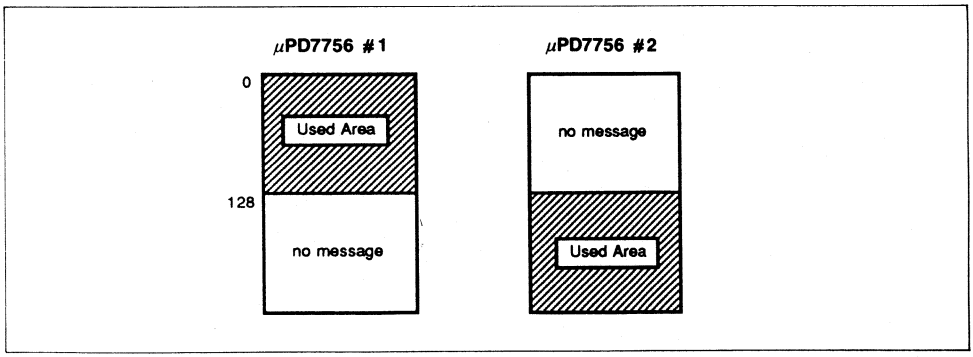


Figure 8. Speech Memory Configuration

Using this configuration, concatenation of phrases from different μPD7756's is possible. This solution uses very few control ports from the microprocessor (as common CS is implemented). In above example, 128 messages are available for each μPD7756 (which is more than enough in almost all applications).

An SLD Interface for NEC's Signal-Processors μPD7720, μPD77C25, μPD77230 and Related Products

Outline: The following pages show briefly how to integrate NEC's signal-processors and speech related products, such as the μPD7730 and the μPD7761 into ISDN applications. The interface circuit adapting DSPs, to the SLD bus, is discussed in general. Furthermore, a concrete example how to adapt the μPD7730 to a Feature COMBO (29C48) providing SLD interface is given.

Contents:

Author: K. Grohe
Application Department
NEC Electronics (Europe) GmbH

1. The SLD Interface

Figure 1 shows the ISDN reference model as standardized by CCITT I.411. On the subscriber side, 2 of 8 terminals (TE1) can be activated for ISDN communication over the S_0 interface. Non-ISDN terminals (TE2) may be connected via a terminal adapter (TA) to ISDN. The network terminator (NT1, NT2) interfaces the ISDN terminals to the PBX. Now, where to find the SLD interface?

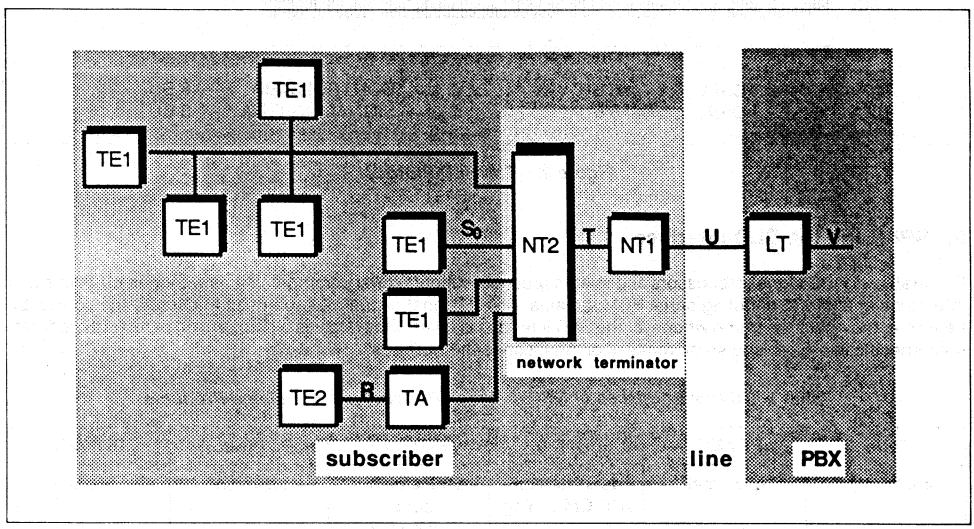


Figure 1. The ISDN Reference Model

The SLD interface is used with ISDN terminals as well as with PABX/PBXes. It serves half duplex communication with 512 kbps. Each frame is split into two parts, a 256 kbps receive, and a 256 kbps transmit. 128 kbps of each part are used for two B channels of 64 kbps each, one D channel of 16 kbps, and several other control and status informations.

The SLD interface uses three signals; a bidirectional data line (SLD), a control line (SDIR) that determines the direction of the data line, and the sampling clock (SCL) of 512 KHz. The SLD data are set up at the rising edge of the SCL signal and stored on its falling edge. The SDIR signal is switched every 125 μs so that a transmission of 32 bits from master to slave are followed by a 32 bit transmitted from slave to master. Each 32 bit stream consists of channels B1 and B2, a 8 bit feature control channel F, and a signalling channel S.

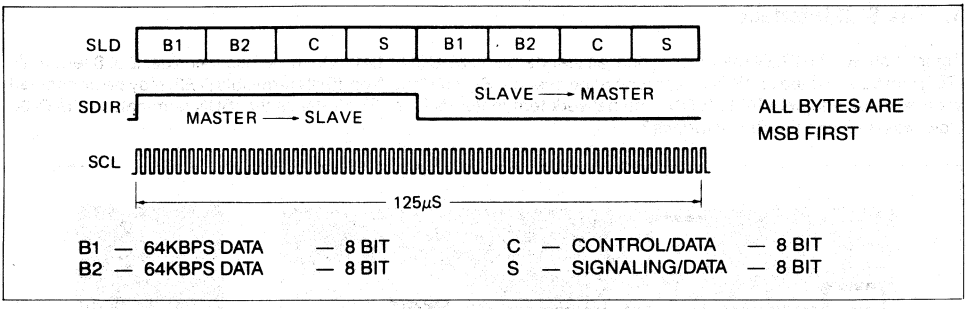


Figure 2. The SLD Protocol

2. SPI's and the SLD Interface

Generally, all NEC signal-processors and related products support serial communications with their on-chip serial interfaces capable of handling serial PCM streams. An additional circuit must be used to capture the correct bit pattern at the right time frame whenever the SLD interface is used. Table 1 gives a comparison of related products, their specific interface, and some remarks on adaption to the SLD bus.

Table 1. Serial Interfaces of NEC's Signal-Processors and Related Products

Type	μPD7720 [1]	μPD7730 [1]	μPD7761 [1]	μPD77C25 [1]	μPD77230
application	universal	dedicated for ADPCM coding	dedicated for speech recognition	universal	universal
number of serial interfaces input output	 1 1	 1 [2] 1 [2]	 1 1	 1 1	 1 1
serial clock	common external	common external	—	common external	common internal or separate external
multi channel operation	yes	no	no	yes	yes
max. number of servable channels 8 bit mode 16 bit mode 24 bit mode 32 bit mode	 1 2	 1 2	 1	 1 2	 1 2 3 4
IO framing relations	independant	the same timeslot	requires input only	independant	independant

Note: 1) Apply framing signals (SIEN, SOEN carefully. Otherwise noise may occur due to overdriven serial I/O shift registers. Synchronization can be reestablished activating the reset pin only.

2) Half duplex only. Full duplex cannot be served with one chip.

The standard products, μPD7720, μPD77C25, and μPD77230, allow practically independent input and output frames. The input/output relation of such devices is almost free, and only fixed due to a common clock (except μPD77230, if external serial clock is used). Due to the internal algorithm of the μPD7730, the relation for that device is more restricted. Serial input and output do require the same framing signals.

The maximum number of servable channels (B1, B2, F, S) can easily be increased on a software basis by configuring the serial interfaces to 16 bit, 24 bit or 32 bit mode. The channel to be served must be specified by hardware (depending on how the framing signals are treated).

3. A Universal Master/Slave Interface Circuit

Depending on the application, it may be necessary to provide the signals SDIR and SCL by external hardware. For instance, a pure voice mail system may not require a bypass of the voice channel, therefore, the peripheral board controller (PBC) may be omitted. However, the SDIR and SCL must be provided by the SLD interface circuit. Figure 3 shows the block diagram of a typical voice mail application in an ISDN terminal.

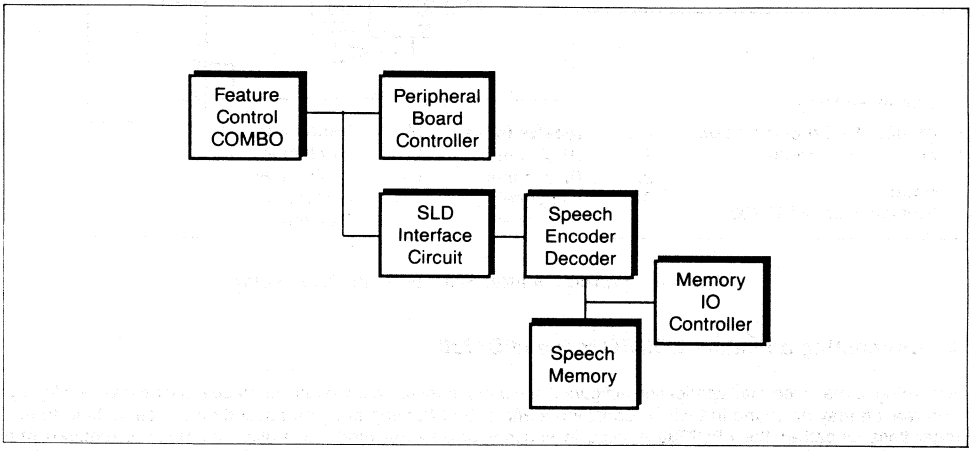


Figure 3. Block Diagram of a Voice Mail Application in an ISDN Terminal

Figure 4 shows a possible solution for a SLD interface circuit. The interface may be configured as a master providing active SCL and SDIR signals with jumpers J1 and J2 shortcut, or as a slave with SCL and SDIR signals coming from the peripheral board controller (J1 and J2 left open). In the last case, IC3 may be omitted. Additionally, the clock generator IC1 can drive the μPD77xx master clock. Furthermore, jumpers J3 and J4 allow for selection of one of the 4 channels (B1, B2, F, S) or the receive and transmit sections independently. Before the system is invoked, the RST signal must be released by the controlling host processor.

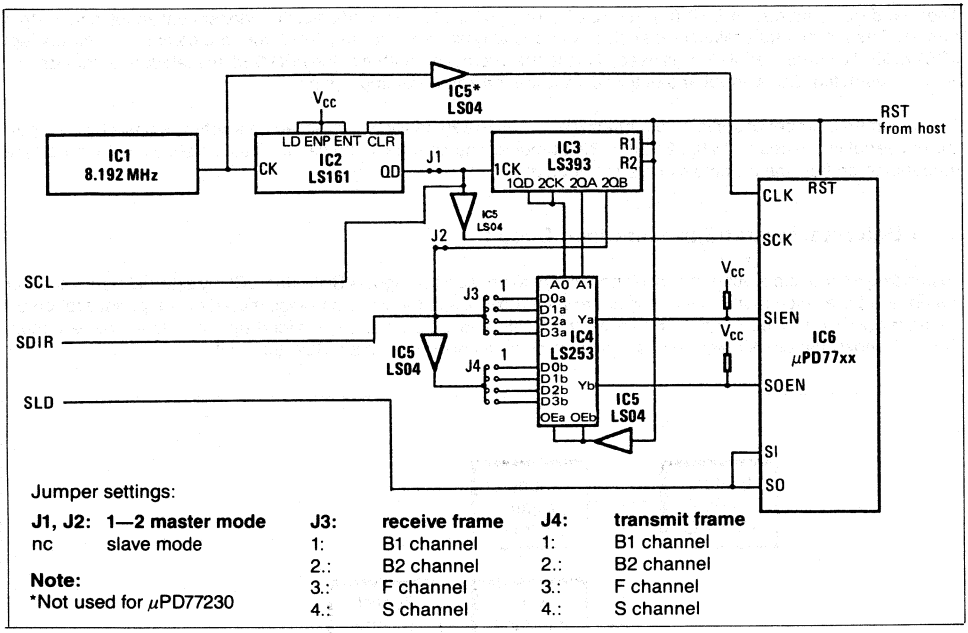


Figure 4. Universal Master/Slave SLD Interface Circuit

4. Connecting a Feature COMBO to the μPD7730

Returning to the voice mail application of figure 3, the afore mentioned universal interface must be revised slightly. The reason may be found in table 1. Serial input and output framing must be applied on the same time frame, regardless wheather the μPD7730 is used in encoder or decoder mode. If a slave interface is required (the peripheral board controller generates SCL and SDIR signals), the interface would now appear as in figure 5. This interface circuit is now jumpered to serve channel B1 on transmit as well as on receive frame. If SDIR and /SDIR are connected to other input pins of the DEMUX (IC4), other channels can be selected easily. Normally it won't be necessary to change the channel to be served. Figure 6 shows an interface capable of serving channel B1 on receive and transmit frames by using gates only.

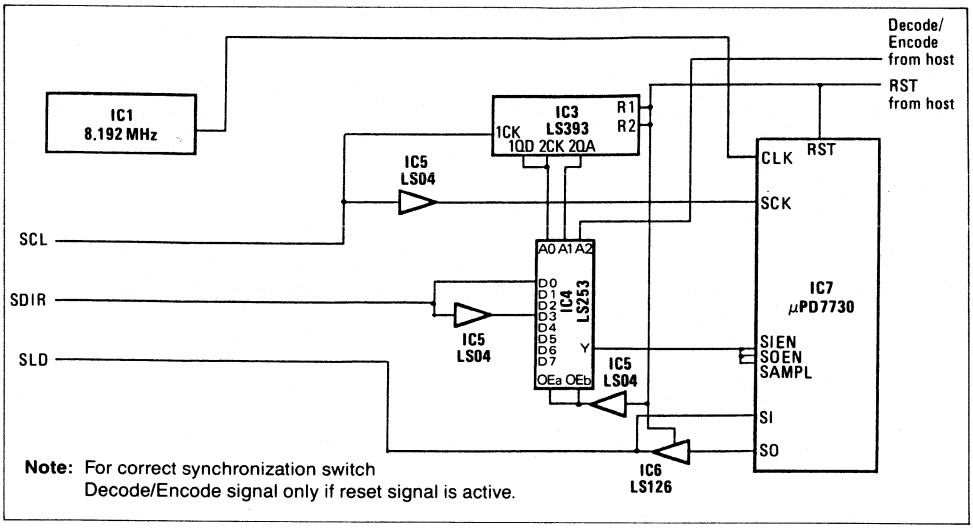


Figure 5. Universal SLD Interface Switched to B1 Channel Serving and Framing Signals SIEN, SOEN tied together for μPD7730 Applications

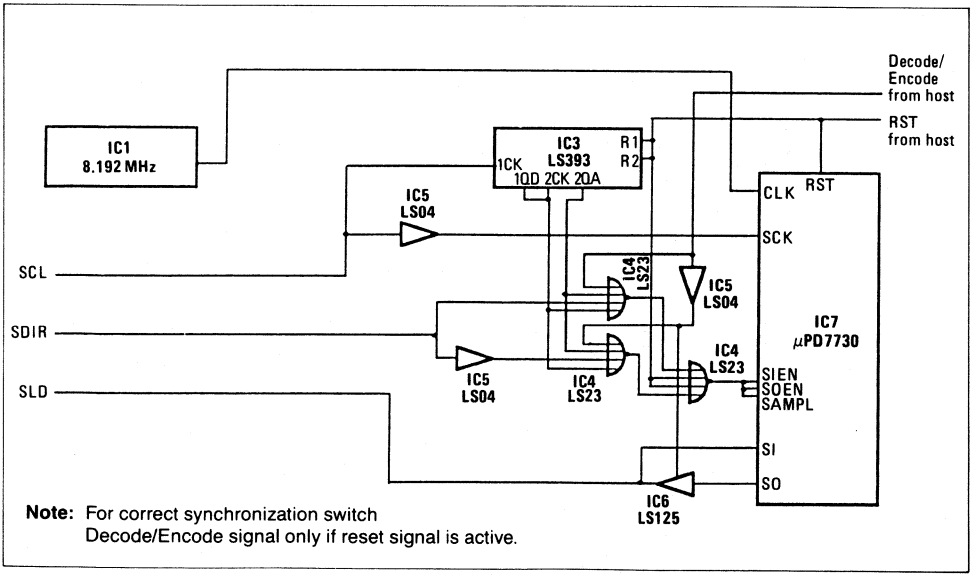


Figure 6. Slave SLD Interface Switched to B1 Channel Serving and Framing Signals SIEN, SOEN tied together for μPD7730 Applications

Source Code Compatibility of the μPD77C25 with the μPD7720

Contents:

Author: N. Kellerhoff
Application Department
NEC Electronics (Europe) GmbH

1. 77C25 Features and Comparison to the 7720

μPD77C25/77P25

MAIN FEATURES

16-bit CMOS signal processor
CMOS EPROM version available 3Q87

Compatibility with 7720
>> Pin-for-pin compatible
>> Software compatible (see note)

Higher processing speed
122 nsec instruction cycle (8.192 MHz clock)

Larger memory size

>> Instruction ROM:	2048 x 24B
>> Data ROM:	1024 x 16B
>> Data RAM:	256 x 16B

CMOS 1.6 μm technology

APPLICATIONS

Modems
Speech compression
Robotics control
Audio

POSITIONING OF 77C25 IN NEC's DSP FAMILY

Improved version of 7720 family

Note: 7720 source program is applicable for 77C25 assembler.

μPD77C25/77P25 detailed comparison with 77C20A

	77C20A/77P20	77C25/77P25
TECHNOLOGY	CMOS/NMOS	CMOS/CMOS
INSTRUCTION CYCLE	244 ns	122 ns
INSTRUCTION ROM	512 x 23b	2048 x 24b
DATA ROM	510 x 13b	1024 x 16b
DATA RAM	128 x 16b	256 x 16b
FIXED POINT MULTIPLIER	16b x 16b −> 31b	16b x 16b −> 31b
ALU	16b FIX. PT.	16b FIX. PT.
ACCUMULATOR	2 x 16b	2 x 16b
HOST CPU INTERFACE	8 BIT BUS	8 BIT BUS
SERIAL INTERFACE	INPUT/OUTPUT 1 CHANNEL EACH 2 MHz	INPUT/OUTPUT 1 CHANNEL EACH 4 MHz
TEMPORARY REGISTER	ONE	TWO

ADDITIONAL INSTRUCTIONS	—	JDPLN0 JDPLNF
	—	MODIFICATION OF RAM COLUMN DATA POINTER M8—MF
DMA MODE	—	BIDIRECTIONAL
PACKAGE	28 PIN DIP 44 PIN PLCC	28 PIN DIP 44 PIN PLCC
POWER SUPPLY	5 V	5 V
POWER CONSUMPTION	40 mA (MAX) 8.192 MHz	40 mA (MAX) 8.192 MHz

2. 77C25 Assembler Package Configuration

The source code of the 77C25 is compatible with the 7720 source code. Only a few minor adjustments are necessary to get 77C25 hex object code from the 7720 source code using the 77C25 assembler package. These changes will be outlined in the following report.

Figure 1 shows the configuration of the μPD77C25 assembler package. In contrast to the 7720, the 77C25 assembler is relocatable and includes most of the features that the Advanced Signal Processor 77230 assembler package contains.

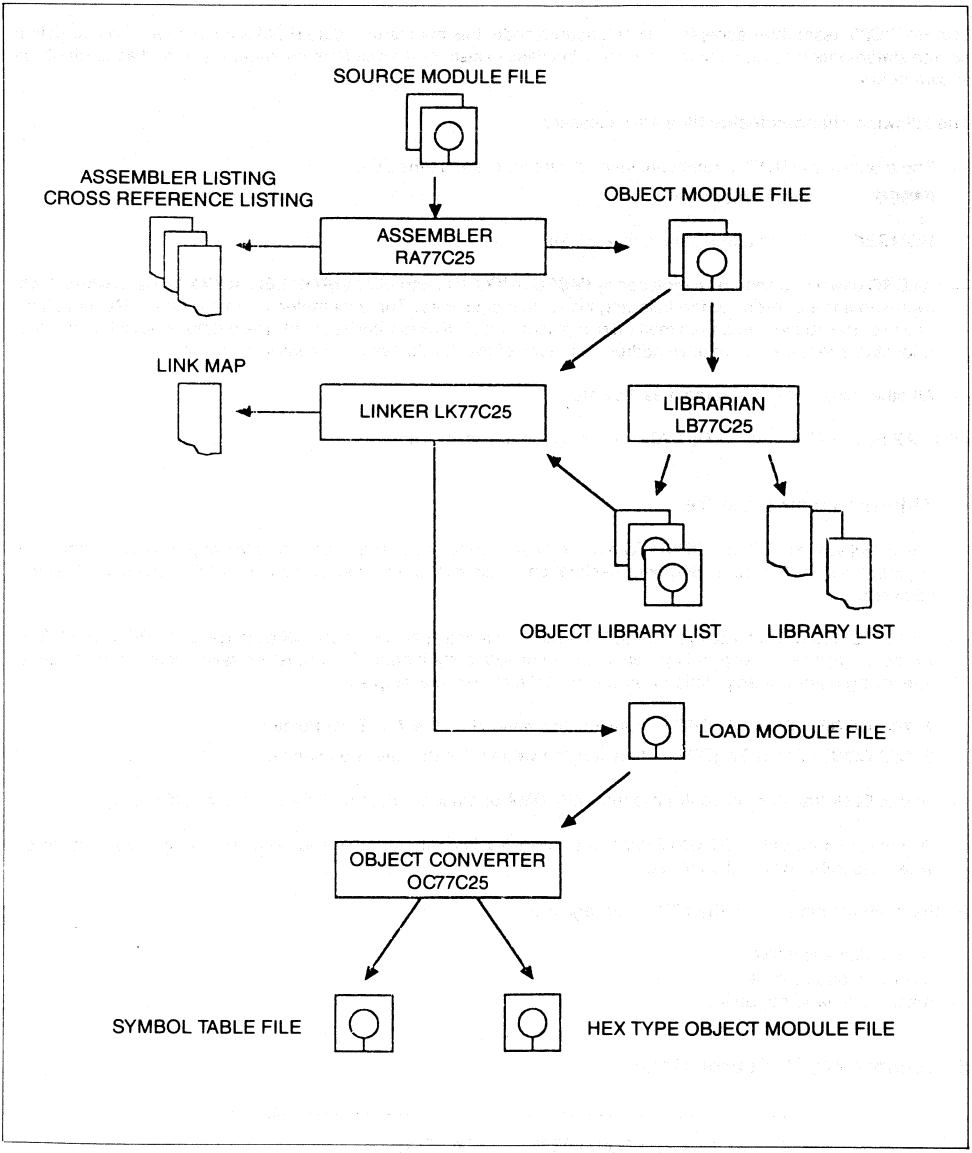


Figure 1. Assembler Package Configuration

3. Necessary Changes to the 7720 Code

The **file names extensions** of the 7720 code need not be changed. However, remember to use the same extension when invoking the assembler.

The RA77C25 assembler accepts data and source code; this means that data ROM source code and instruction source statements may be placed in one file. Directives to distinguish one from the other are added accordingly as shown below.

The following **changes inside files** are necessary:

— The directive MPROG is removed. Instead segments are defined by:

IMSEG (for instructions)
or
ROMSEG (for data to reside in data ROM)

— All **ORG** directives should be replaced by **IMSEG at XXX** for instructions (**ROMSEG at XXX** for data) where XXX represents the address of the following instruction/data word. The assembler will accept the ORG directive, however the above mentioned method of replacement of ORG directives avoids the pitfalls associated with the relocatable feature and relative addressing mode of the 77C25 assembler when modifying 7720 code.

— All other directives may remain as they are.

All **TYPE** names (OP, CALL, LDI, DATA, etc.) may remain as they are.

4. Things to watch out for

— The directives IMSEG and ROMSEG need to be preceeded by a label (see example program) which defines a segment name. For each Segment declaration, a distinct label must be allocated (no repetition of labels allowed).

— As the ROM and RAM size in the 77C25 has been increased (INSTR. ROM: 4X as large, DATA ROM and RAM: 2X as large) the corresponding pointers are subsequently longer. Care must be taken with the 7720 code specifying pointer manipulations, as shown in the following example:

7720 CODE: OP MOV §DP,FF; transfers the value 7F to the 7 bit Data Pointer

77C25 CODE: OP MOV §DP,FF; transfers the value FF to the new 8 bit pointer

— Use of DMA transfer: in contrast to the 7720, DMA operations with the 77C25 affect the RQM flag.

— Do not run comments in **EQUATE** directives onto a new line without a line end delimiter. This generates an error when assembling with the RA77C25.

Further features included in the 77C25 package are:

— relocatable assembler
— extensive directive set
— macrocommand capability

5. Sample 77C25 Source Code

```
/*-----------------------------------------------*/
/*  BIQUAD FILETER EXECUTION SUBROUTINE          */
/*-----------------------------------------------*/
```

```
        NAME    BIQUAD
/* */
        PUBLIC  BIQD
/* */
BIQ     IMSEG
/* */
BIQD:   OP      XOR     ACCA,IDB        /* A=0 */
                MOV     @KLR,A          /* K=W(I),L=A(0) */
                RPDEC                   ;
        OP      ADD     ACCA,M          /* A=W(I)*A(NO) */
                MOV     @KLR,MEM        /* K=W(I-1),L=B(1) */
                RPDEC                   ;
        OP      ADD     ACCA,M          /* A=W(I)*A(0)+W(I-1)*B(1) */
                MOV     @B,K            /* SAVE W(I-1) (NEW W(I-2) */
                M1                      ; /* DP4 = NOT DP4 */
        OP      ADD     ACCA,M          /* A=W(I)*A(0)+2*W(I-1)*B(1) */
                MOV     @KLR,MEM        /* K=W(I-2),L=B(2) */
                RPDEC                   ;
        OP      ADD     ACCA,M          /* A=W(I)*A(0)+2*W(I-1)*B(1)
                                            +W(I-2)*B(2) (NEW W(I-1) */
                MOV     @L,R0           /* L=A(2) */
                RPDEC                   ;
        JNOVA1  BIQD1                   ;
/* */
        OP      MOV     @A,SGN          ;
BIQD1:  OP      ADD     ACCA,M          /* A=NEW W(I-1) + W(I-2)*A(2) */
                MOV     @TR,A           /* SAVE NEW W(I-1) */
                M1                      ; /* DP4 = NOT DP4 */
        OP      MOV     @KLR,MEM        /* K=W(I-1),L=A(1) */
                RPDEC                   ;
        OP      ADD     ACCA,M          /* A=NEW W(I-1)+W(I-2)*A(2)
                                            +W(I-1)*A(1) */
                MOV     @MEM,TR         /* SET NEW W(I-1) */
                M1                      ; /* DP4 = NOT DP4 */
        OP      ADD     ACCA,M          /* A=NEW W(I-1)+W(I-2)*A(2)
                                            +2*W(I-1)*A(1)*/
                MOV     @MEM,B          /* SET NEW W(I-2) */
                M1                      /* DP4 = NOT DP4 */
                DPDEC
                RET                     ;
/* */
        END
```

Frequency Synthesis Macros for NEC's Advanced Signal-Processor μPD77230 Volume 1 The Primitives

Outline: The following pages describe some easy-to-digest macros for often used basic functions like noise and tone generation. Besides a pseudo random noise generator different types of sinewave generators are introduced including sample programs. Much emphasis was put onto an effective programming structure resulting in speed optimized code and comfortable user interface.

Contents:

Author: K. Grohe
Application Department
NEC Electronics (Europe) GmbH

1. Why Macros

Comfortable assemblers allow to write relocatable code and do support macros. A macro can be a single instruction or a sequence of instructions which gets a new name. After the macro once has been defined (macro definition) it can be called by name within a program like an additional mnemonic (macro reference). Whenever the assembler finds a macro reference within a program it expands the code by the predefined code (macro expansion). Macros can have parameters and it is possible to refer inside a macro to another one (macro nesting).

In contrast to subroutines a macro exists within a certain program as many times as there are macro references. The disadvantage of more code compared to subroutines is reflected by higher speed. Other advantages of macros:

- have a better programming structure and readability of source code
- create your own instruction set and have a virtual machine
- use it as a fast programming technique to test new algorithms

```
                        Macro Definition:
            square      MACRO           reg__no                 ;
            mov         k,wr&reg__no                            ;
            mov         l,wr&reg__no    clr     wr&reg__no      ;
            addf        wr&reg__no ,m                           ;
            endm

                        Macro Reference:
squaresum:  square      0                                       ;
            square      1                                       ;
            mov         non,wr0         addf    wr1,ib          ;
```

Figure 1. Sample Macro

Additionally these macros support flexible software development as working registers are not fixed by macro definition. Only at macro expansion the working registers will be tied to an actual register. Compared to subroutine programming techniques the macro technique is much quicker and allows software development somewhat between high level language compiler and assembler if the programmer creates a clever interface structure.

2. Noise Generation

Generation of pseudo random signals can be realized with some shift register having an exored feedback. The length of the shift register and intermediate bits that are fed back determine the performance resp. periodicity of such noise generator. A 4 bit shift register for instance can at least run through 16 different states until another period starts. To do so 2 conditions must be fulfilled. First the initial value of the shift register must be set to any value except zero otherwise the output will not change from zero. Secondly the maximum length of period will not be achieved if bits 3 and 4 are exored and fed back. Table 1 shows the different states of such a 4 bit noise generator.

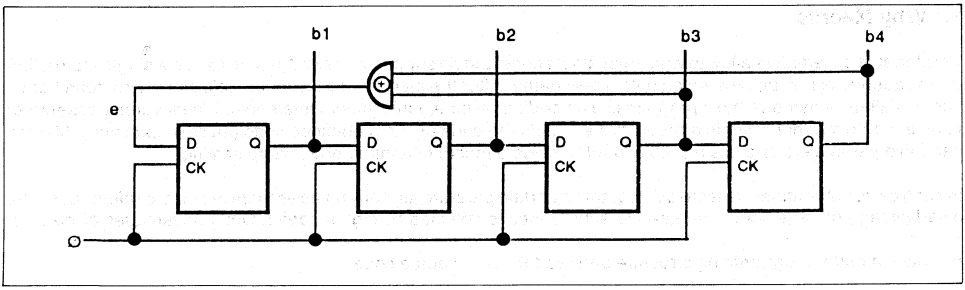


Figure 2. Block Diagram of a 4 Bit Pseudo Noise Generator

Table 1. States of a 4 Bit Shift Registers with Feedback of Bit 3 and 4

ø	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
b1	1	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1
b2	0	1	0	0	1	1	0	1	0	1	1	1	1	0	0	0
b3	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0	0
b4	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0
e	0	0	1	1	0	1	0	1	1	1	1	0	0	0	1	0

Of course a 4 bit noise generator may in most cases not give acceptable performance. But note that the periodicity of such a noise generator will increase rather fast if the length if the shift register is increased. The maximum periodicity can be determined by: $N = 2^{n}-1$ where n is the length of the shift register. Now which bits have to be exored and coupled back for maximum periodicity? The calculation is rather complicated. At least the LSB has to be fed back. Table shows the results.

Table 2. Feedback Bits vs different Shift Register Length

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	2	3	3	5	4	7	5	7	9	11	10	13	14	14	14	11	18	17
						5				8	6	8		13			17	
						3				6	4	4		11			14	

The attached macro is an implementation of a 20 bit type random noise generator having a periodicity of $2^{20}-1$ which should result in acceptable performance. By exchanging the parameters BIT2027 and FB21 in the source code other feedbacks can be achieved. Note that this macro supports 2 bit feedbacks only.

3. Tone Generation

Tone generation can be reduced to sinewave generation. Several solutions are possible beginning with simple table output up to fourier synthesis. The best compromize between speed and flexibility is a solution well known from analog computers used for flight simulation, etc. It's the implementation of the differential equation of a sinewave.

$$\text{Equation 1:} \quad d^2/dt^2\, y(t) + \Omega_0^2\, y(t) = 0$$

A sinewave can be approximated by 2 integrators with weighted feedback where the weighting factor denotes the frequency ratio of generated vs. sampling frequency. Figure 2 shows the block diagram of the sinewave generator and the equations that have to be figured out for each sample period.

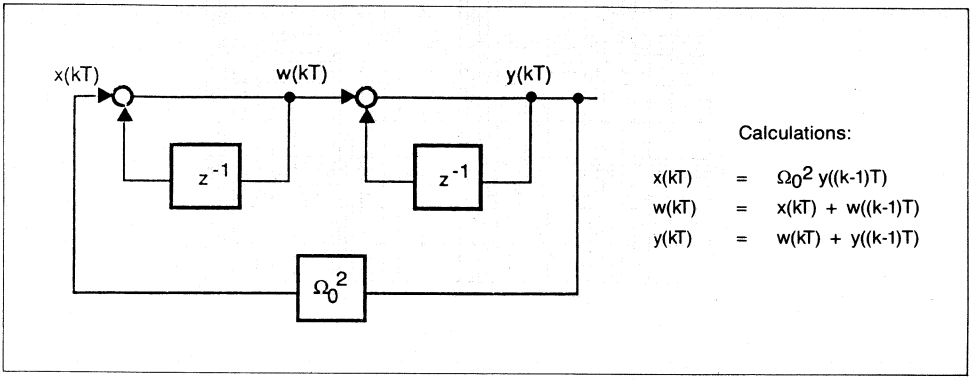


Figure 3. Digitized Model of the differential Equation and its Calculations

Some notes on performance of the generator:

- As the poles of the closed loop are lying within the unit circle the generator cannot become instable.
- The parameters amplitude and frequency are independent of each other. The amplitude is set by the initial value w(0) resp y(0) with y(0) resp w(0) set to 0. The normalized frequency is set by the feedback factor Ω_0^2.
- w(kT) and y(kT) have π /2 phase difference supplying a quadrature reference.
- The generator starts directly without any non-stationary behaviour at π n resp. π /2(2n+1) with n as integer.
- Due to the nature of the model sinewaves no measurable harmonic disturbance is achieved for frequency ratios below approx. 1/10. Bigger ratios cause some distortion due to the nonlinear character. A maximum value of —40 dB for second harmonic was measured at a frequency ratio of 1/3 (without any reconstruction filter).
- The analytic relation between generated frequency f_0, sampling frequency f_S and feedback factor Ω_0^2 can be approximated with:

$$\text{Equation 2:} \quad f_0 = 1/(2\pi)\, f_S\, \Omega_0^2$$

- Additionally the relation of normalized frequency f_0/f_S vs feedback factor Ω_0^2 is given in figure 4 over the full range of possible values Ω_0^2.

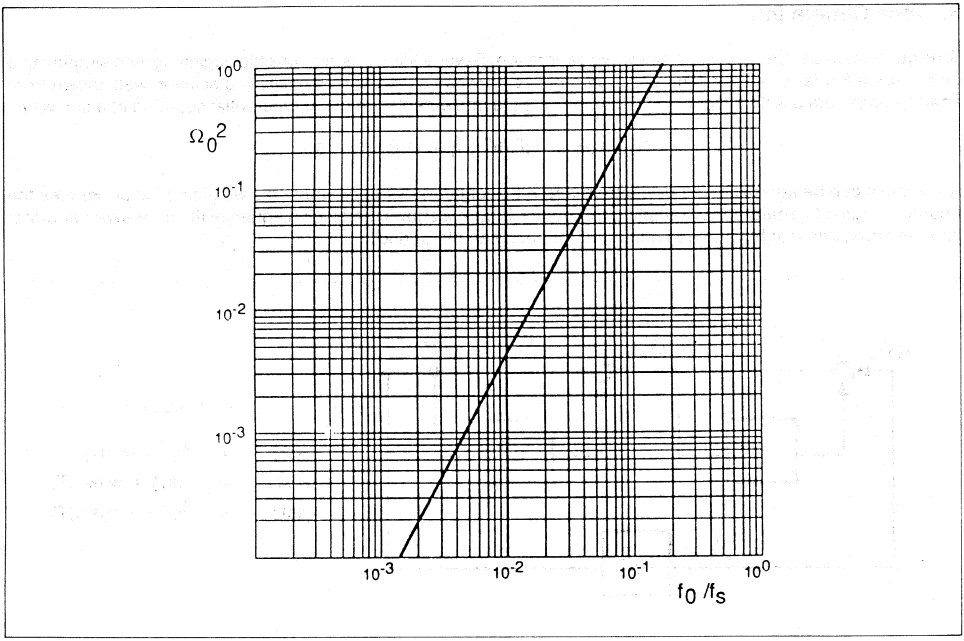


Figure 4. Normalized Frequency vs Feedback Factor

4. Conclusion

The implementation of both generators demonstrates the excellent performance of NEC's advanced signal-processor μPD77230. Especially sinewave generation based on the implementation of the differential equation results in state of the art benchmark as is 750 ns for two independent sinewaves! Again dual sinewave generation gives some idea how to pipeline tasks. Besides that some good training effect is achieved to get familiar with the processors pipeline. The structure of macros shows the flexible use of these programs supporting a complete class of programs where actual registers are selected at macro reference time but not before. All programs have been tested and contain direct executable code for NEC's EB-77230 FFT hardware with Combo or 12 bit analog front end.

5. Source Code for RA77230 Relocatable Assembler Package

```
;-------------------------------------------------------------------------
;
;
;       NEC Advanced Signal-Processor uPD77230 Macro Library
;
;                       Frequency Synthesis Macros
;
;                               Volume 1
;
;                           The Primitives
;
;                               V1.10
;
;
;       Contents:                                                   page
;
;       1.    OUTLINE ..................................................  2
;       1.1   jump table ..............................................  2
;       1.2   list of useful constants ...............................  2
;       1.3   i/o macros and subroutines ..............................  3
;
;       2.    NOISE GENERATION ........................................  5
;       2.1.  macro noise .............................................  5
;       2.2.  noise output to COMBO or DAC.............................  5
;
;       3.    TONE GENERATION .........................................  6
;       3.1   macro sinewave (register based) .........................  6
;       3.2   sinewave output to COMBO or DAC(1 kHz) ..................  6
;       3.3   macro dual-tone sinewave (register based) ..............  7
;       3.4   pushbutton tone generation with dual-tone generator
;             (digit 3, r1/c1 = 697Hz/1209Hz) ........................  7
;       3.5   macro sinewave (memory based) ..........................  8
;       3.6   subsampled sinewave output to COMBO or DAC(2 KHz).....  8
;       3.7   macro multi-tone sinewave (memory based) ..............  9
;       3.8   3-tone output to COMBO or DAC(600Hz/800Hz/1kHz)........ 10
;       3.9   multi-tone output to COMBO or DAC ..................... 10
;
;
;-------------------------------------------------------------------------

imseg   at      0h                                              ;
jmp     1000h                                                   ;
nop                                                             ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       1. outline                                                        ;
;                                                                         ;
;       Note: This programs can be tested with NEC's EBIBM-77230-TOOL     ;
;             board using a standard COMBO interface or a linear DAC.     ;
;             Output to the COMBO is performed by the processors serial   ;
;             interface whereas DAC output requires the parallel data     ;
;             bus of the processor. Selection of one of these analog      ;
;             frontends can be made by replacing the macros "parout"      ;
;             against "serout" and assemblying new with "RA77230"         ;
;             relocatable assembler package. All frequency parameters     ;
;             are based on a sampling frequency of 8 kHz. Whenever the    ;
;             sampling frequency has to be adjusted refer to the          ;
;             documentation:                                              ;
;                                                                         ;
;                          Frequency Syntesis Macros                      ;
;                 for NEC's Advanced Signal-Processor uPD77230            ;
;                                 Volume 1                                ;
;                             The Primitives                              ;
;                                                                         ;
;-------------------------------------------------------------------------;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       1.1 jump table                                                    ;
;                                                                         ;
;       jumps to direct executable code!                                  ;
;                                                                         ;
;-------------------------------------------------------------------------;
```

```
imseg   at      1000h                   ;

ldi     sr,50h                          ; init COMBO interface
di                                      ; init EBIBM-77230-TOOL
jmp     xnoise                          ; noise generator
nop                                     ;
jmp     xsine                           ; sinewave of 1 kHz
nop                                     ;
jmp     xdial3                          ; dialtone pushbutton 3
nop                                     ;
jmp     xsub                            ; subsampling 2 kHz
nop                                     ;
jmp     xtri                            ; superpos. 600,800,1000 Hz
nop                                     ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       1.2 list of useful constants                                      ;
;                                                                         ;
;       The attached frequency parameters are related to a standard       ;
;       combo interface using a sampling frequency of 8 kHz. The          ;
;       parameters are splitted into two sections.                        ;
;       Parameters of section 1 can be used together with tone            ;
;       generation macros directly.                                       ;
;       The maximum achievable frequency is limited to 1/6 sampling       ;
;       frequency. Applying a subsampling technique the maximum           ;
;       achievable frequency can be increased to 1/3 sampling frequency.  ;
;       Parameters of section 2 are related to this subsampling           ;
;       technique. In that case call the tone generation macro twice      ;
;       before ouput to the combo                                         ;
;       A detailed description of the relation parameter value,           ;
;       generated frequency, and sampling frequency is given in the       ;
;       documentation "Frequency Syntesis Macros Vol.1 - The Primitives"  ;
;                                                                         ;
;-------------------------------------------------------------------------;


; section 1:
o600Hz  set     -0.6                    ;
o800Hz  set     -0.65                   ;
onekHz  set     -0.7                    ;
row1    set     -0.2922058              ;
row2    set     -0.3546753              ;
row3    set     -0.4312439              ;
row4    set     -0.5217285              ;
col1    set     -0.8359375              ;
col2    set     -1                      ;

; section 2:
twokHz  set     -0.7                    ;
cols1   set     -0.2211914              ; use 2x subsampling !!
cols2   set     -0.2689819              ;   "         "
cols3   set     -0.3690491              ;   "         "
cols4   set     -0.3968201              ;   "         "
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       1.3 i/o macros and subroutines                                    ;
;                                                                         ;
;       Macros:                                                           ;
;         serout: Outputs a linear value from register r0 to a PCM        ;
;                 combo; the value is first changed to ulaw PCM data      ;
;                 and then output; this program checks the empty flag     ;
;                 of the serial output shift register and therefore       ;
;                 comes back after the external hardware has fetched      ;
;                 data completely; subroutine "lme" is used!              ;
;         parout: Prepares output of a linear value from register r1      ;
;                 to the DAC of NEC's EBIBM-77230-TOOL board. The value   ;
;                 is first formatted to adapt to the DAC. After that it   ;
;                 waits for INT signal. The output itself is done         ;
;                 within the "intser" routine.                            ;
;                                                                         ;
;       Subroutines:                                                      ;
;         thru: PCM loop through 77230; 8 bit input data are output       ;
;               without changes; use it for hardware debugging!           ;
;                                                                         ;
;         lme: This program converts 14 bit linear values into ulaw       ;
;              compressed PCM data for later output to a PCM combo        ;
;              the rom segment "u148" is used for internal                ;
;              calculations! (see end of this source!)                    ;
;                                                                         ;
;         intser: This interrupt service routine outputs a value          ;
;                 prepared by the "parout" macro to the DAC of NEC's      ;
;                 EBIBM-77230-TOOL board and returns to that macro        ;
;                 skipping the "parou0" loop inside "parout".             ;
;                                                                         ;
;-------------------------------------------------------------------------;

        serout  macro   r0                      ;
                                                ;
        mov     wr0,wr&r0                       ;
        call    lme                             ; convert to ulaw data
        nop                                     ;
        mov     so,wr0                          ;
serou1: jneso   serou1                          ; output via serial interface
        nop                                     ;
        endm                    ;
```

```
        parout  macro   r0,r1                          ;
        mov     wr&r0,wr&r1     setsvr  0ch            ; shift to correct position
        shram   wr&r0                                  ;
        nop                                            ;
        mov     dr,wr&r0        clrbm                  ;
        call    parou0                                 ; stack return address
        nop                                            ;
parou0: jmp     parou0                                 ; wait for interrupt
        ei                                             ;
        endm                            ;

;-------------------------------------------------------------------------------;
        imseg   at      1100h                          ;
        dacadr  set     1ff8h                          ;
        timadr  set     1ff9h                          ;
        ldrwre  set     37bh                           ;
;-------------------------------------------------------------------------------;
intser: ldi     ar,dacadr                              ; output to DAC
        call    ldrwre                                 ;
        ldi     wr0,2                                  ; calculate return address
        mov     non,stk add     wr0,ib                 ;
        nop                                            ;
        mov     stk,wr0                                ;
        ret                                            ; .. and exit
        nop                                            ;

;-------------------------------------------------------------------------------;
thru:   mov     so,si                                          ;
        nop                                                    ;
thru0:  jneso   thru0                                          ;
        nop                                                    ;
        jmp     thru                                           ;
        nop                                                    ;

;-------------------------------------------------------------------------------;
lme:    and   wr0 ,ib     mov non ,wr0          ;set sign flag bit
        nop                                     ;
        jns0 posi6                              ;if sign +,goto posi
        clr   wr2                               ;store sign for positive
        neg   wr0                               ;wr0 2's cmplement
        nop                                     ;
        jnv0 posi6                              ;if wr0 is not 80..0H,goto
        ldi   wr2 ,800000H                      ;store sign for negative
        clr   wr0         mov ix1 ,wr6          ;
        ret                                     ;
        nop                                     ;return
```

```
posi6:  ldi   rp  ,U148RM
        add   wr0 ,ib     mov non ,rom    incrp       ;rom pointer set
        nop                                           ;wr0 <- wr0+8400H
        jnv0  nov16                                   ;
        clr   wr1                                     ;if not overflow,goto $+3
        ldi   wr0 ,7FFFFFH                            ;
nov16:                    mov tr  ,wr0                ;overflow correction
                          mov tre ,rom    incrp       ;
                          mov wr0 ,tr                 ;tr <- exp(=07H)+wr0
;nov16: or    wr0 ,ib     mov non ,rom    incrp       ;a  <- exp(=07H)+wr0
        norm  wr0                         trnorm      ;
                                          wrbel8      ;exp <- segment number
        shlm  wr1         mov wr1 ,wr0    setsvl 43   ;exp -> low 8bit of mantissa
        shrm  wr0                         setsvr 2    ;wr1 <- wr0 43bit 1.sft
                                          setsvl 0    ;2bit right shift
        and   wr0 ,ib     mov non ,rom    wrbord      ;
        or    wr0 ,ib     mov non ,wr1                ;detect step num. ordinary
        or    wr0 ,ib     mov non ,wr2                ;wr0 <- segment+step
        not   wr0                                     ;wr0 <- sign+segment+step
        ret               mov ix1 ,wr6                ;wr0 1's cmplement
        nop                                           ;
                                                      ;

;-------------------------------------------------------------------------------;
        romseg

U148RM: DW      8400H
        DW      07H
        DW      0F0000H


        imseg   at      1200h                         ;
```

2. noise generation

2.1 noise

The following source code supplies a pseudo-random noise generator tap. Noise generation is based on a 20 bit shift register with xored feedback of bit 17 and bit 20. To initialize register r0 must be set to any value except 0. The noise generator outputs a 20 bit fractional value with periodicity of 2**20 in register r0. The lower 12 bits have to be masked off if necessary. r1 contains the 20 bit predecessor.

Benchmarks:

max. sampling frequency	555 kHz
max. computation speed per tap	1.8 us

Initialization: set r0 to any value except 0 !
Used registers: r0, r1, TR, PSW0
Output: r0 (or r1 for predecessor)
Output format: 20 bit fraction (mask off 12 LSBs if needed!)

```
        noise   macro   r0, r1          ;

BIT2017 set     10010000000000000000000b        ; xor bit mask
FD21    set     11111111111111111110111b        ; feedback value

        mov     wr&r1,wr&r0     spcpsw0         ; keep old value
        ldi     tr,BIT2017                      ; load feedback bitmask
        and     wr&r0,ib        mov non,tr      ; mask feedback bits
        nop                                     ;
        jz0     case00                          ; jump if both zero
        xor     wr&r0,ib        mov non,tr      ; check ones
        nop                                     ;
        jz0     case11                          ; jump if both ones
        ldi     wr&r0,FD21                      ; generate feedback for case00
        not     wr&r0           jblk    nend    ; .. resp for case01 and case10
        or      wr&r0,ib        mov non,wr&r1   ; .. and put together
case00: ldi     wr&r0,FD21                      ; generate feedback for case11
case11: and     wr&r0,ib        mov non,wr&r1   ; .. and put together
nend:   rol     wr&r0                           ; shift for next value
        nop                                     ;
        endm                            ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       2.2 Example: noise output to combo interface                      ;
;                                                                         ;
;-------------------------------------------------------------------------;

xnoise: ldi     wr4,000010h             ; set start value

noise0: noise   4,5                     ;; generate noise using wr4,wr5

        parout  2,4                     ;; output to combo

        jmp     noise0                  ; jump to next sample
        nop                             ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       3. tone generation                                                ;
;                                                                         ;
;       The following source code supplies signal generation macros as    ;
;       are sinewaves, dual-tone sinewave and multiple-tone sinewaves.    ;
;       Tone generation is based on a recursive 2nd order tap with        ;
;       feedback. The parameters frequency and amplitude can be choosen   ;
;       independently without any couplings. Each generator consists      ;
;       of an initialization part with am parameters and a computational  ;
;       part that has to be called continuously in a loop for continuous  ;
;       tone generation. The macros are using 24 bit fractions for        ;
;       calculations.                                                     ;
;                                                                         ;
;       Benchmarks:                                                       ;
;         max. sampling frequency (dual-tone)          1.33 MHz           ;
;         max. frequency vs sampling frequency                            ;
;          (without subsampling)                       1:6                ;
;         min. frequency vs sampling frequency         1:8000             ;
;         max. computation speed per tap (dual-tone)   375 ns             ;
;         max. number of waves (multiple-tone)         1/(fs*750 ns)      ;
;                                                                         ;
;               Note: with fs sampling frequency!                         ;
;                                                                         ;
;-------------------------------------------------------------------------;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       3.1 sinewave (register based)                                     ;
;                                                                         ;
;       The following source code supplies a register based sinewave      ;
;       generator. No memory is used. To initialize use macro initr.      ;
;       The generator outputs a 24 bit fraction to register r0. Re-       ;
;       gister k may not be destroyed by other programs.                  ;
;                                                                         ;
;       Benchmarks:                                                       ;
;         max. sampling frequency                        1.33 MHz         ;
;         min. computation speed per tap                 0.75 us          ;
;                                                                         ;
;       Initialization: use macro initr !                                 ;
;       Used registers: r0, r1, K, L                                      ;
;       Output:         r0 (or r1)                                        ;
;       Output format:  24 bit fraction                                   ;
;                                                                         ;
;-------------------------------------------------------------------------;

        initr   macro   freq,ampl,r0,r1                 ;
        ldi     k,freq                                  ; set frequency
        ldi     wr&r0,ampl                              ; set amplitude
        clr     wr&r1                                   ; clear Wn
        endm                            ;


        regsine macro   r0,r1                           ;
        mov     1,wr&r0                                 ; Xn = k * Yn-1
        add     wr&r1,m                                 ; Wn = Wn-1 + Xn
        nop                                             ;
        add     wr&r0,ib        mov non,wr&r1           ; Yn = Yn-1 + Wn
        nop                                             ;
        endm                            ;


;-------------------------------------------------------------------------;
;                                                                         ;
;       3.2 Example: sinewave output to combo interface                   ;
;                    freq = 1 kHz, ampl = fullscale                       ;
;                                                                         ;
;-------------------------------------------------------------------------;

xsine:  initr   <-0.99>,<0.4>,3,4               ;;

rsine0: regsine 3,4                             ;; generate sine using WR3, WR4

        parout  2,3                             ;; output to combo

        jmp     rsine0                          ; jump to next sample
        nop                                     ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       3.3 dual-tone sinewave (register based)                           ;
;                                                                         ;
;       The following source code supplies a register based dual-tone     ;
;       sinewave generator. The two tasks are running in parallel. No     ;
;       additional computation time is required. Two memory locations     ;
;       are used to keep the frequency values. To initialize use macro    ;
;       initd. The generator outputs 24 bit fractions to register r0      ;
;       and r2. Macro add_sin can be used to mix both frequencies.        ;
;       No overflow protection is used for that case. Register k may      ;
;       not be destroyed by other programs.                               ;
;                                                                         ;
;       Benchmarks:                                                       ;
;         max. sampling frequency                          1.33 MHz       ;
;         min. computation speed per tap                   0.75 us        ;
;                                                                         ;
;       Initialization: use macro initd !                                 ;
;       Used registers: r0, r1, r2, r3, K, L, BP0                         ;
;       Used mem. loc.: 2 (pointed at by BP0)                             ;
;       Output:         r0  and r2 (or r1 and r3)                         ;
;                       resp r4 (if macro add_sin is used)                ;
;       Output format:  24 bit fractions                                  ;
;                                                                         ;
;-------------------------------------------------------------------------;

initd   macro   freq1,freq2,ampl1,ampl2,r0,r1,r2,r3     ;
ldi     bp0,0                                   ;
spcbp0                                          ;
ldi     tr,freq1                                ;
mov     ram0,tr         incbp0                  ;
ldi     tr,freq2                                ;
mov     ram0,tr         decbp0                  ;
ldi     wr&r2,ampl1                             ;
clr     wr&r3                                   ;
ldi     wr&r0,ampl2                             ;
clr     wr&r1                                   ;
endm                            ;


dualsin macro   r0,r1,r2,r3                     ;
mov     lkr0,wr&r0      incbp0                  ; Xun =ku*Yun-1
add     wr&r1,m         mov  lkr0,wr&r2
                        decbp0                  ; Wun =Wun-1+Xun, Xvn =kv*Yvn-1
add     wr&r3,m                                 ; Wvn =Wvn-1+Xvn
mov     non,wr&r1       add  wr&r0,ib           ; Yun =Yun-1+Wun
mov     non,wr&r3       add  wr&r2,ib           ; Yvn =Yvn-1+Wvn
endm                            ;


add_sin macro   r0,r1,r2,r3,r4                  ;
mov     wr&r4,wr&r0                             ;
mov     non,wr&r2       add  wr&r4,ib           ; X = Yun + Yvn
nop                                             ;
endm                            ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       3.4 Example: pushbutton tone generation with dual-tone            ;
;                    generator output to combo interface                  ;
;                                                                         ;
;                  row-tone = 1209 Hz                                     ;
;                  column-tone = 697 Hz                                   ;
;                  preemphasis = 3 dB                                     ;
;                                                                         ;
;-------------------------------------------------------------------------;

xdial3: initd   row1,col1,<-0.5>,<-0.35>,3,4,5,6 ;; set parameters

dsine0: dualsin 3,4,5,6                          ;; generate dual-tone

        add_sin 3,4,5,6,0                        ;; mix tones

        parout  2,0                              ;; output to combo

        jmp     dsine0                           ; jump to next sample
        nop                                      ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       3.5 sinewave (memory based)                                       ;
;                                                                         ;
;       The following source code supplies a memory based sinewave        ;
;       generator using 3 memory locations. To initialize use macro       ;
;       initm. The generator outputs a 24 bit fraction to register        ;
;       r0 and to RAM1 with IX1 as memory pointer. One location of        ;
;       RAM0 is used to keep the frequency value. BP0 is used as          ;
;       pointer.                                                          ;
;                                                                         ;
;       Benchmarks:                                                       ;
;         max. sampling frequency                          1.33 MHz       ;
;         min. computation speed per tap                   0.75 us        ;
;                                                                         ;
;       Initialization: use macro initr !                                 ;
;       Used registers: r0, r1, K, L                                      ;
;       Output:         r0 (or r1) and RAM1(IX1)                          ;
;       Output format:  24 bit fraction                                   ;
;                                                                         ;
;       Memory map:                                                       ;
;                                                                         ;
;       RAM0:                              RAM1:                          ;
;        freq_ptr (BP0) --> freq            ampl_ptr (IX1)  --->   0      ;
;                                                                  ampl   ;
;                                                                         ;
;-------------------------------------------------------------------------;
```

```
        initm   macro   freq_ptr,ampl_ptr       ;
        spcbp0  spcix1                          ;
        ldi     bp0,freq_ptr                    ;
        ldi     ix1,ampl_ptr                    ;
        endm                            ;
```

```
memsine macro   r0                                      ;
add     wr&r0,ram1 mov lkr0,ram1 incix1 ; Xn = k * Yn-1
add     wr&r0,m mov wr&r0,ram1   decix1 ; Wn = Wn-1 + Xn
add     wr&r0,ram1       incix1         ; Yn = Yn-1 + Wn
mov     ram1,wr&r0       decix1         ;
clr     wr&r0   mov ram1,wr&r0          ;
endm                                ;
```

```
;-------------------------------------------------------------------------------
;
;       3.6 Example: subsampled sinewave output
;
;                    freq = 440 Hz, ampl = fullscale
;
;-------------------------------------------------------------------------------
```

```
xsub:   initm   <10>,<10>                       ;; initialize pointers

        ldi     tr,0.25                         ;
        mov     ram1,tr         incix1          ;
        ldi     tr,0                            ;
        mov     ram1,tr         decix1          ;
        ldi     tr,-0.75                        ;
        mov     ram0,tr                         ;
        clr     wr5                             ;

msine0: memsine 5                               ;; generate sine ..

        memsine 5                               ;;  .. twice !!!

        parout  2,5                             ;;  .. but output only once !!!

        jmp     msine0                          ; jump to next sample
        nop                                     ;
```

```
;
;   3.7 Multi-tone sinewave (memory based)
;
;   The following source code supplies a memory based sinewave
;   generator that can be cascaded to a multi-tone generator.
;   Each tone uses 3 memory locations. To initialize memory
;   pointers use macro inits. Each generator outputs a 24 bit
;   fraction to RAM1 with IX1 as memory pointer. Additionally
;   the output is accumulated to register r1 (accu) using floating
;   point addition. When concatenating this macros register r1
;   contains already the superposition of all sines that have been
;   generated. One location of RAM0 is used to keep the frequency
;   value. BP0 is used as pointer.
;
;   Benchmarks:
;     max. sampling frequency                  1.33 MHz per tone
;     min. computation speed per tap           0.75 us  per tone
;
;   Initialization: use macro inits !
;   Used registers: r0, r1, K, L
;   Output:         r0 (or r1) and RAM1(BP1)
;   Output format:  24 bit fraction
;
;   Memory map:
;
;   RAM0:                              RAM1:
;    freq_ptr (BP0) --> freq1           ampl_ptr (IX1)  -->   0
;                                                            ampl1
;                                                              0
;                       freq2                                ampl2
;                                                              "
;                         "                                    "
;                       freqn                                  0
;                                                            ampln
;
```

```
inits   macro   freq_ptr,ampl_ptr,r0,r1 ;
ldi     bp0,freq_ptr                    ;
ldi     ix1,ampl_ptr                    ;
ldi     bp1,2                           ;
clr     wr&r0           spcbp0  spcbi1  ;
clr     wr&r1                           ;
endm                            ;


sines   macro   r0,r1                   ;
add     wr&r0,ram1 mov lkr0,ram1 incix1 ; Xn = k * Yn-1
add     wr&r0,m    mov wr&r0,ram1 decix1 ; Wn = Wn-1 + Xn
add     wr&r0,ram1                incix1 ; Yn = Yn-1 + Wn
addf    wr&r1,ib mov ram1,wr&r0   decix1 ; accumulate Yn
clr     wr&r0    mov ram1,wr&r0   stix1
                                  incbp0 ;
endm                            ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       3.8 Example: 3-tone output to combo interface                     ;
;                                                                         ;
;                    freq1 = 600 Hz,  ampl1 = 0.25 fullscale              ;
;                    freq2 = 800 Hz,  ampl2 = 0.25 fullscale              ;
;                    freq3 = 1 kHz,   ampl3 = 0.25 fullscale              ;
;                                                                         ;
;-------------------------------------------------------------------------;

a1      set     0.25                                    ;
a2      set     0.25                                    ;
a3      set     0.25                                    ;

xtri:   inits   <10>,<10>,5,7                           ;; initialize pointers

        ldi     tr,0                                    ; set freq vs ampl table
        ldi     wr0,a1                                  ;
        mov     ram1,wr0        incix1                  ;
        mov     ram1,tr         incix1                  ;
        ldi     wr0,o600Hz                              ;
        mov     ram0,wr0        incbp0                  ;
        ldi     wr0,a2                                  ;
        mov     ram1,wr0        incix1                  ;
        mov     ram1,tr         incix1                  ;
        ldi     wr0,o800Hz                              ;
        mov     ram0,wr0        incbp0                  ;
        ldi     wr0,a3                                  ;
        mov     ram1,wr0        incix1                  ;
        mov     ram1,tr         incix1                  ;
        ldi     wr0,onekHz                              ;
        mov     ram0,wr0        incbp0                  ;

sines0: inits   <10>,<10>,5,7                           ;; reset pointers and accu

        sines   5,7                                     ;; generate sine 1

        sines   5,7                                     ;; generate sine 2

        sines   5,7                                     ;; generate sine 3

        parout  2,7                                     ;; output accu to combo

        jmp     sines0                                  ; jump to next sample
        nop                                             ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       3.9  multi-tone output to COMBO or DAC                            ;
;                                                                         ;
;                                                                         ;
;                 Note:                                                   ;
;                 No initialization is performed! Frequency and           ;
;                 amplitude values must be set by host processor!         ;
;                 Refer to the memory map of sines for initialization.    ;
;                 Additionally location ampl_ptr contains the  number     ;
;                 of sines. All other RAM0 locations are shifted to +1    ;
;                                                                         ;
;-------------------------------------------------------------------------;

mtpsin: mov     lc,ram0         incbp0          ;  load number of sines

        inits   <10>,<10>,5,7                   ;; reset pointers and accu

mtpsi0: sines   5,7                             ;; generate sine i

        declc                                   ;
        jmp     mtpsi0                          ;

        parout  2,7                             ;; output accu to combo

        jmp     mtpsin                          ; jump to next sample
        decbp0                                  ;

end
```

Fast Vector Matrix Calcultions for NEC's Advanced Signal-Processor μPD77230 Volume 1

Outline: The following pages contain macros for fast floating point calculations of scalar product, product of matix, and vector and product of two matrices. The macro approach allows maximum flexibility and speed. Examples are given for vectors with 6 elements and matrices of size 6 x 6. Other sizes may easily be accomodated for.

Contents:

Author: K. Grohe
Application Department
NEC Electronics (Europe) GmbH

1. Introduction

Fast matrix calculations are still a domain of superminis. Technological progress in the field of VLSI chip design now allows such calculations to run on a monolithic floating point signal-processor. NEC's 77230 advanced signal-processor, the first floating point candidate on the market, is powerful enough to achieve speeds up to 13.3 MFLOPS opening new applications. Modern control theory and 3D graphics require such a powerful and accurate engine. Behind all theoretical background, such as state space analysis and transformations etc., realsystems may be reduced in many cases to simple matrix calculations.

The goal of the following approach is to achieve optimum calculation speed (as opposed to care of universality or program length). Thus, linear coding is a must. Speed requires the maximum possible calculations to be handled internally. As a compromise, vectors/matrices would be stored in the external memory because they are only involved in external processes.

2. Scalar Product

Scalar product of two vectors x, y of size n is defined as:

$$a = x^T y = \sum_{i=1}^{n} x_i \, y_i$$

Suppose x is located in RAM0 and vector y in RAM1, then the scalar product resulting is a in RAM0. Base pointer 0 addresses the first element of x, base pointer 1 that of y and the result is placed where index-register 0 points at. Then a macro scalar-product and its reference looks like:

Macro Definition:

```
scalar_product      macro       r0,r1
clr     wr&r0       mov     lkr0,ram1       incbp0      incbp1
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1      spcix0
addf    wr&r0,m     mov     ram0,wr&r1      incix0      spcbp0
endm
```

Macro Reference:

```
scalar_product      0,1
nop                                         spcix0
                    mov     ram0,wr0        spcbp0
```

Note: Subinstructions written *italic* are used for further calculations only!

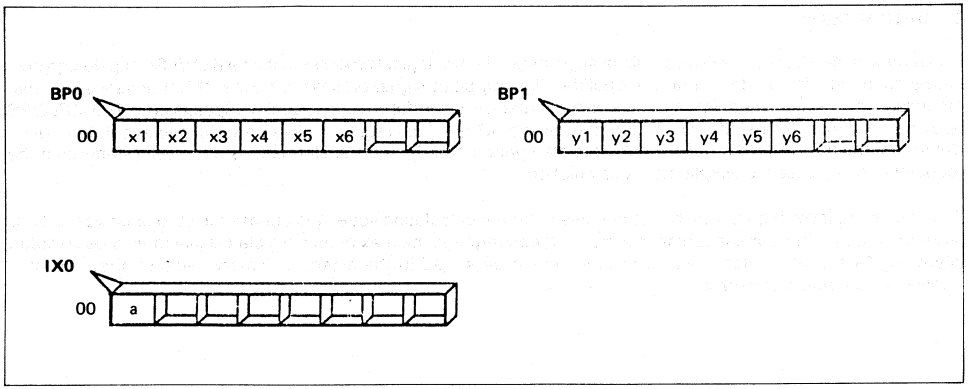


Figure 1. Memory Map

3. Vector Matrix Product

Vector product is defined as:

$$\underline{x} = \underline{B}\,\underline{y}$$

Each element of vector $\underline{x}$ is given by

$$x_k = \sum_{i=1}^{n} b_{ki}\, y_i$$

Suppose matrix $\underline{B}$ is located in RAM0 vector $\underline{y}$ in RAM1 the vector product $\underline{x}$ in RAM0. Base pointer 0 addresses the first element of $\underline{B}$ base pointer 1 that of $\underline{y}$ and the result is placed where index-register 0 points at. Then the macro looks like:

Macro Definition:

```
vector_product      macro       r0,r1                                   ;
initial_product     r0,r1                                               ;
scalar_product      r1,r0                                               ;
scalar_product      r0,r1                                               ;
scalar_product      r1,r0                                               ;
scalar_product      r0,r1                                               ;
last_product        r1,r0                                               ;
endm                                                                    ;
```

where initial_product is:

```
initial_product     macro       r0,r1                                   ;
clr     wr&r0       mov     lkr0,ram1       incbp0      incbp1          ;
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1          ;
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1          ;
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1          ;
addf    wr&r0,m     mov     lkr0,ram1       incbp0      incbp1          ;
addf    wr&r0,m     mov     lkr0,ram1       incbp0      clrbp1          ;
addf    wr&r0,m                                                         ;
endm
```

and last product:

```
last__product       macro       r0,r1                                          ;
clr     wr&r0       mov   lkr0,ram1     incbp0      incbp1                     ;
addf    wr&r0,m     mov   lkr0,ram1     incbp0      incbp1                     ;
addf    wr&r0,m     mov   lkr0,ram1     incbp0      incbp1                     ;
addf    wr&r0,m     mov   lkr0,ram1     incbp0      incbp1                     ;
addf    wr&r0,m     mov   lkr0,ram1     incbp0      incbp1                     ;
addf    wr&r0,m     mov   lkr0,ram1     incbp0      incbp1      spcix0         ;
addf    wr&r0,m     mov   ram0,wr&r1    stix1       clrbp0      spcbp0         ;
                                        incix0      clrbp1                     ;
endm
```

Macro Reference:

```
vector__product     0,1                                                        ;
                                        spcix0                                 ;
                    mov   ram0,wr1      spcbp0                                 ;
```

Note: Subinstructions written *italic* are used for further calculations only!

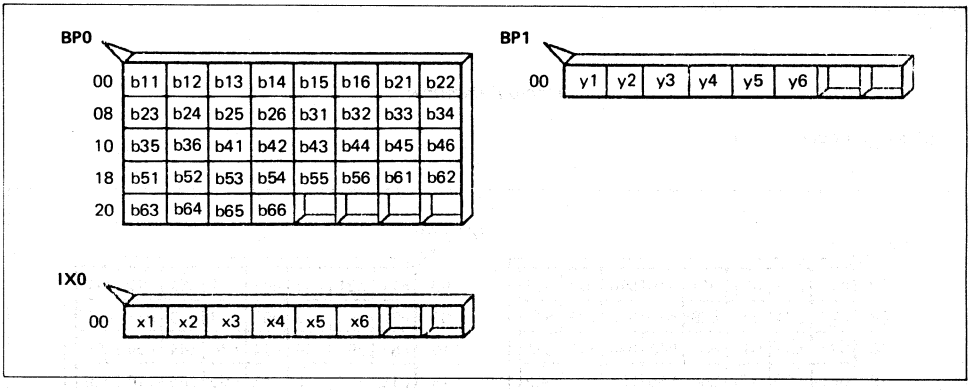


Figure 2. Memory Map

4. Product of two Matrices

The product of two mactrices is defined as $\underline{A} = \underline{B}\,\underline{C}$

Each element of matrix A is given by: $a_{ij} = \sum_{z=1}^{n} b_{iz}\, c_{zj}$

Suppose matrix B is located in RAM0 and matrix C in RAM1, then the product of both matrices is A in RAM0. Base pointer 0 addresses the first element of B, base pointer 1 that of C, and the result is placed where index-register 0 points. The macro looks like:

Macro Definition:

```
matrices_product    macro       r0,r1                                   ;
vector_product      r0,r1                                               ;
next_product        r0,r1                                               ;
next_product        r0,r1                                               ;
next_product        r0,r1                                               ;
next_product        r0,r1                                               ;
next_product        r0,r1                                               ;
nop                                                     spcix0          ;
                    mov     ram0,wr&r1                  spcbp0          ;
endm                                                                    ;
```

where next_product is defined as:

```
next_product        macro       r0,r1                                   ;
scalar_product      r0,r1                                               ;
scalar_product      r1,r0                                               ;
scalar_product      r0,r1                                               ;
scalar_product      r1,r0                                               ;
scalar_product      r0,r1                                               ;
last_product        r1,r0                                               ;
endm                                                                    ;
```

Macro Reference:

```
matrices_product    0,1
```

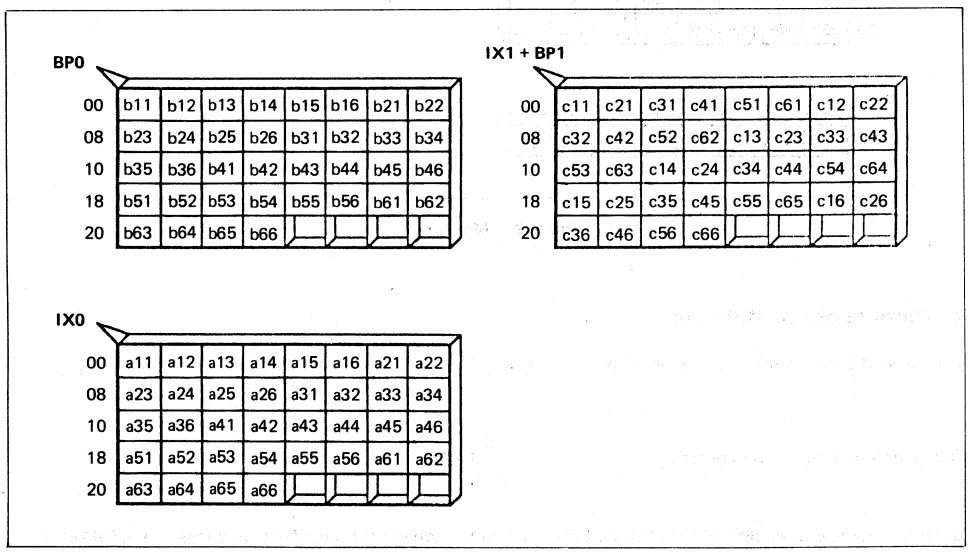


Figure 3. Memory Map

5. Adapting other Vector/Matrix Sizes

So far, macros have been defined with vector/matrix size of 6 resp. 6 x 6. Other sizes of 'n' may be achieved if the following instructions are taken into consideration.

I. Macro scalar_product:

1. Copy line 2 of macro body (n — 3) times

II. Macro vector_product:

1. Copy line 2 of macro body (n — 3) times
2. Reverse the order or registers r0,r1 on every odd line through the end of the macro (including the last_product!!)
3. Change macros scalar product, initial product and last product due to instruction I.

III. Macro matrices_product

1. Copy line 2 of macro body (2 — 2) times
2. If size 'n' is odd, reverse the order or registers r0 and r1 on every odd line through the end of the macro.
3. Change macros next-product and vector-product due to instruction II
4. Change macros initial product, scalar-product and last product due to instruction I
5. Change register of the final mov instruction to r0 if matrix size is odd resp. to r1 and matrix size is even.

For all 3 cases the related memory maps can be derived analog to the above shown memory maps.

6. Fast I/O Routines

Whenever vectors/matrices from external memory are accessed the following transfer macros can be used: Again, fast routines require linear coding.

Macro Definition:

```
load_vector     macro     no                              ;
rd                                                        ;
mov  ram&no,dr  incar     incbp&no                        ;
rd                                                        ;
mov  ram&no,dr  incar     incbp&no                        ;
rd                                                        ;
mov  ram&no,dr  incar     incbp&no                        ;
rd                                                        ;
mov  ram&no,dr  incar     incbp&no                        ;
rd                                                        ;
mov  ram&no,dr  incar     incbp&no                        ;
rd                                                        ;
mov  ram&no,dr  incar     incbp&no                        ;
endm                                                      ;

load_matrix     macro     no                              ;
load_vector     macro     no                              ;
load_vector     macro     no                              ;
load_vector     macro     no                              ;
load_vector     macro     no                              ;
load_vector     macro     no                              ;
load_vector     macro     no                              ;
endm                                                      ;
```

Macro Reference:

```
ldi         ar,A_mat
ldi         bp0,factor1
spcbp0
load_matrix         0
```

Direct output to the external memory may be achieved by the following instruction:

1. Replace the "mov, ram0,wr&r1" sub-instruction of all macros by the following sub-instructions:

   mov dr,wr&r1 incar

2. Insert another instruction after the one under 1. with:

   wr

3. Delete all index-register 0 related sub-instructions.

4. Before referencing to any macro, make sure that the address register 'ar' has been loaded instead of the index-register 0.

7. Benchmarks

Benchmark: fast floating point scalar product					
Device: μPD77230 (master mode) Parameter: number of instructions					
I/O	factor 1	internal	internal	internal	external
	factor 2	internal	internal	external	external
	product	internal	external	external	external
vector size	3 x 1	6	6	12	18
	6 x 1	9	9	21	33
	n x 1	n + 3	n + 3	3n + 3	5n + 3

Benchmark: fast floating point scalar product					
Device: μPD77230 (master mode at 13.3 MHz) Parameter: computational speed					
I/O	factor 1	internal	internal	internal	external
	factor 2	internal	internal	external	external
	product	internal	external	external	external
vector size	3 x 1	0.9 μs	0.9 μs	1.8 μs	2.7 μs
	6 x 1	1.35 μs	1.35 μs	3.15 μs	4.95 μs
	n x 1 (multiply by 150 ns)	n + 3	n + 3	3n + 3	5n + 3

Benchmark: fast floating point vector product					
Device: μPD77230 (master mode) Parameter: number of instructions					
I/O	factor 1	internal	internal	internal	external
	factor 2	internal	internal	external	external
	product	internal	external	external	external
vector size	3 x 1	13	15	21	27
	6 x 1	43	48	60	72
	n x 1	$n^2 + n + 1$	$n^2 + 2n$	$n^2 + 4n$	$n^2 + 6n$

Benchmark: fast floating point vector product					
Device: μPD77230 (master mode at 13.3 MHz) Parameter: computation speed					
I/O	factor 1	internal	internal	internal	external
	factor 2	internal	internal	external	external
	product	internal	external	external	external
vector size	3 x 1	1.95 μs	2.25 μs	3.15 μs	4.05 μs
	6 x 1	6.45 μs	7.2 μs	9 μs	10.9 μs
	n x 1 (multiply by 150 ns)	$n^2 + n + 1$	$n^2 + 2n$	$n^2 + 4n$	$n^2 + 6n$

Benchmark: fast floating point product of 2 matrices					
Device: μPD77230 (master mode) Parameter: number of instructions					
I/O	factor 1	internal	internal	internal	external
	factor 2	internal	internal	external	external
	product	internal	external	external	external
matrix size	3 x 3	38	47	53	59
	6 x 6	260	296	368	440
	n x n	$n^3 + n^2 + n+2$	$n^3 + 2n^2 + n+2$	$n^3 + 4n^2 + n+2$	$n^3 + 6n^2 + n+2$

Benchmark: fast floating point product of 2 matrices					
Device: μPD77230 (master mode at 13.3 MHz) Parameter: computational speed					
I/O	factor 1	internal	internal	internal	external
	factor 2	internal	internal	external	external
	product	internal	external	external	external
matrix size	3 x 3	5.7 μs	7.05 μs	7.95 μs	8.85 μs
	6 x 6	39 μs	44.4 μs	55.2 μs	66 μs
	n x n	$n^3 + n^2 + n+2$	$n^3 + 2n^2 + n+2$	$n^3 + 4n^2 + n+2$	$n^3 + 6n^2 + n+2$

```
;-------------------------------------------------------------------------
;
;
;       NEC Advanced Signal-Processor uPD77230 Macro Library
;
;
;                   Fast Vector/Matrix Calculation Macros
;
;                               Volume 1
;
;                               V1.10
;
;
;       Contents:                                                   page
;
;       1.    OUTLINE ..................................................  1
;       1.1   jump table ...............................................  2
;       1.2   list of test matrices and their result.....................  2
;       1.3   demochip subroutines ....................................  3
;       1.4   equates ..................................................  3
;
;       2.    SCALAR-PRODUCT ...........................................  3
;       2.1   macro definition .........................................  4
;       2.2   macro reference: calculate a = x y .......................  4
;
;       3.    VECTOR-PRODUCT ...........................................  5
;       3.1   macro definition .........................................  5
;       3.2   macro reference: calculate z = x y .......................  6
;
;       4.    PRODUCT OF 2 MATRICES ....................................  6
;       4.1   macro definition .........................................  7
;       4.2   macro reference: calculate A = B C .......................  7
;
;
;-------------------------------------------------------------------------

imseg   at      0h                                                      ;
jmp     1200h                                                           ;
nop                                                                     ;

;-------------------------------------------------------------------------
;
;       1. Outline
;
;       Note: The following programs are tailored to vectors of size 6
;             and matrices of size 6x6. This programs can be tested
;             with NEC's EBIBM-77230-TOOL board. For additional
;             information especially adaption to other vector/matrix
;             sizes refer to the document:
;
;                     Fast Vector/Matrix Calculations
;                   for NEC's Advanced Signal-Processor uPD77230
;                               Volume 1
;
;-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       1.1 jump table                                                    ;
;                                                                         ;
;       jumps to direct executable code!                                  ;
;                                                                         ;
;-------------------------------------------------------------------------;

        imseg   at      1200h                   ;

        jmp     scl_pd                          ; scalar_product
        nop                                     ;
        jmp     vec_pd                          ; vector_product
        nop                                     ;
        jmp     mat_pd                          ; matrices_product
        nop                                     ;
halt:   jmp     halt                            ; processor hang-up
        nop                                     ; insert user action!

;-------------------------------------------------------------------------;
;                                                                         ;
;       1.2 list of test matrices and their result .                      ;
;                                                                         ;
;       The attached test matrices of size 6x6 are used for demo          ;
;       purposes. Additionally their results are given calculated with    ;
;       TURBO PASCAL's floating point routines and 8087 coprocessor.      ;
;                                                                         ;
;-------------------------------------------------------------------------;

        exmseg  at      800h

B_mat:  dw      1.23456,    2.34567,    3.45678,    4.56789,    5.67890,    6.12345
        dw      -0.7896,    789.001,    76.5430,    -12.567,    0.00015,    -567.78
        dw      1234567.,   -98.789,    -234345.,   102.987,    -30e30,     0.789654
        dw      4444444.,   -89e-15,    987.456,    -654321.,   -12345e8,   555555.
        dw      32123e2,    0.4e-3,     123e-7,     567.987,    434.9875,   32999.67
        dw      321.876,    67.9876,    4.76569,    555e-5,     45e4,       555.890

C_mat:  dw      765.987,    6666666.,   -0.0987,    9.87654,    123.876,    987654.
        dw      -98765.,    -45321.,    3.4543,     6.4356,     2.67,       4.56789
        dw      671234.,    6.234e-1,   1234.56,    1.9876e2,   5.,         1.23456
        dw      98.12e4,    654321.,    -999e-4,    -9876e3,    987.344e-20,    9e-4
        dw      4.567e-2,   987654.,    876543.,    675432.,    45.45454,       0
        dw      3.56781,    432.7697,   453e-5,     0,          8.989898,   -543287.
```

```
; result A = B C should be:
;
;  2.1687E+07  -2.282E+05  8.3389E+05  -4.237E+07  8.4323E+06  -3.326E+06
;
;  4.6992E+09  -3.568E+07  -4.382E+05  6.3960E+08  8.3787E+08  3.0881E+08
;
;  -3.716E+33  -8.010E+31  -1.500E+32  -2.950E+14  -1.364E+33  -2.697E+32
;
;  -1.524E+14  -3.735E+12  -3.189E+12  1.0823E+13  -5.655E+13  -1.140E+13
;
;  3.5053E+10  -3.173E+11  2.1562E+12  3.1463E+12  3.8380E+08  -1.792E+10
;
;  1.0583E+09  -3.367E+07  2.1831E+08  3.6026E+08  9.1784E+07  -2.979E+08
;

        imseg   at      1300h   ;

;-------------------------------------------------------------------------;
;                                                                         ;
;       1.3 demochip subroutines (addresses of 77230R-002)                ;
;                                                                         ;
;                                                                         ;
;       Subroutines:                                                      ;
;                                                                         ;
;         htra2e:  loads data from high speed memory to RAM0              ;
;         htra4e:  loads data from high speed memory to RAM1              ;
;         htra2oe: stores data from RAM0 to high speed memory             ;
;                                                                         ;
;       For more details refer to Application Note 2 of 77230!            ;
;                                                                         ;
;-------------------------------------------------------------------------;

        htra2e  set     38h                             ;
        htra4e  set     56h                             ;
        htra2oe set     2c9h                            ;

;-------------------------------------------------------------------------;
;                                                                         ;
;       1.4 equates                                                       ;
;                                                                         ;
;       The attached equates allow adaptions to other vector/matrix       ;
;       sizes. Note that all macros must be adapted as well! Refer to     ;
;       chapter 5! The internal pointers "factor1" and "factor2" cannot   ;
;       be changed to values except 0 in the case of "matrix_product".    ;
;                                                                         ;
;-------------------------------------------------------------------------;

        size_n  set     6                               ; #vector elements
        sqrsize set     36                              ; #matrix elements

        factor1 set     0                               ; internal pointers
        factor2 set     0                               ;
        product set     40h                             ;

        A_mat   set     8a0h                            ; external pointers
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       2. scalar_product                                                 ;
;                                                                         ;
;       The following source code calculates the scalar product of 2      ;
;       vectors x,y. Vector x is located in RAM0 vector y in RAM1.        ;
;       Before calling the macro basepointer 0 must be set to the         ;
;       start address of vector x (element x[1,1]) and basepointer 1      ;
;       to the start address of vector y (element y[1,1]). The result a   ;
;       is stored in RAM0 with index-register 0 as pointer.               ;
;                                                                         ;
;       Benchmarks:                                                       ;
;         max. computation speed per tap (size n = 6)    1.35 us          ;
;                                                                         ;
;       Initialization: set bp0 -> x[1,1]                                 ;
;                       set bp1 -> y[1,1]                                 ;
;                       set ix0 -> a                                      ;
;                       set elements of vectors x,y in RAM0 and RAM1      ;
;                       according to the memory map                       ;
;       Used registers: r0, r1, K, L, M, BP0, BP1, IX0                    ;
;       Output:         r0                                                ;
;       Output format:  32 bit ASP floating point format                  ;
;                                                                         ;
;-------------------------------------------------------------------------;

;-------------------------------------------------------------------------;
;                                                                         ;
;       2.1 macro definition                                              ;
;                                                                         ;
;-------------------------------------------------------------------------;

        scalar_product  macro   r0,r1                       ;

        clr   wr&r0     mov lkr0,ram1   incbp0  incbp1  ; a = 0
        addf  wr&r0,m   mov lkr0,ram1   incbp0  incbp1  ; a = a + x[1]*y[1]
        addf  wr&r0,m   mov lkr0,ram1   incbp0  incbp1  ; a = a + x[2]*y[2]
        addf  wr&r0,m   mov lkr0,ram1   incbp0  incbp1  ; a = a + x[3]*y[3]
        addf  wr&r0,m   mov lkr0,ram1   incbp0  incbp1  ; a = a + x[4]*y[4]
        addf  wr&r0,m   mov lkr0,ram1   incbp0  clrbp1  spcix0  ; a=a+x[5]*y[5]
        addf  wr&r0,m   mov ram0,wr&r1  incix0  spcbp0  ; a = a + x[6]*y[6]
        endm                                            ;

;-------------------------------------------------------------------------;
;                                                                         ;
;       2.2 macro reference: calculate a = x y                            ;
;                                                                         ;
;-------------------------------------------------------------------------;

scl_pd: ldi     lc,size_n                               ; load vector x=B[i,1]
        ldi     ix0,size_n-1                            ;
        ldi     ar,B_mat                                ;
        spcix0                                          ;
        call    htra2e                                  ; call demochip routine
        nop                                             ;

        ldi     lc,size_n                               ; load vector y=C[1,i]
        ldi     ix1,size_n-1                            ;
        ldi     ar,C_mat                                ;
        spcix1                                          ;
        call    htra4e                                  ; call demochip routine
        nop                                             ;
```

```
        ldi     bp0,factor1                     ; set BP0 to B[1,1]
        ldi     ix1,factor2                     ; set IX1 to C[1,1]
        ldi     ix0,product                     ; set IX0 to A[1,1]
        clrbp1  spcbp0  spcbi1                  ; initiate pointers

        scalar_product  0,1                     ;; calculate a = y z
                        spcix0                  ;
        mov ram0,wr0    spcbp0                  ;

        ldi     lc,size_n                       ; store scalar a=A[i,1]
        ldi     ix0,product + size_n-1          ;
        ldi     ar,A_mat                        ;
        spcix0                                  ;
        call    htra2oe                         ; call demochip routine
        nop                                     ;
        jmp     halt                            ; stop calculations
        nop                                     ;
```

```
;-------------------------------------------------------------------;
;                                                                   ;
;       3. vector_product                                           ;
;                                                                   ;
;       The following source code calculates the vector product of 2;
;       vectors x,y. Vector x is located in RAM0 vector y in RAM1.  ;
;       Before calling the macro basepointer 0 must be set to the   ;
;       start address of vector x (element x[1]) and basepointer 1  ;
;       to the start address of vector y (element y[1]) awa index-  ;
;       register 0 pointing at the result vector z element z[1] in  ;
;       RAM0.                                                       ;
;                                                                   ;
;       Benchmarks:                                                 ;
;         max. computation speed per tap (size n = 6)   6.6 us      ;
;                                                                   ;
;       Initialization: set bp0 -> x[1]                             ;
;                       set bp1 -> y[1]                             ;
;                       set ix0 -> z[1]                             ;
;                       set elements of vectors x,y in RAM0 and RAM1;
;                       according to the memory map                 ;
;       Used registers: r0, r1, K, L, M, BP0, BP1, IX0              ;
;       Output:         r0                                          ;
;       Output format:  32 bit ASP floating point format            ;
;                                                                   ;
;-------------------------------------------------------------------;
```

```
;-------------------------------------------------------------------;
;                                                                   ;
;       3.1 macro definition                                        ;
;                                                                   ;
;-------------------------------------------------------------------;
```

```
        vector_product  macro   r0,r1           ;

        initial_product r0,r1                   ;; calculate z[1]
        scalar_product  r1,r0                   ;; calculate z[2]
        scalar_product  r0,r1                   ;; calculate z[3]
        scalar_product  r1,r0                   ;; calculate z[4]
        scalar_product  r0,r1                   ;; calculate z[5]
        last_product    r1,r0                   ;; calculate z[6]
        endm                                    ;
```

```
        initial_product macro   r0,r1                                    ;

        clr   wr&r0       mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   clrbp1   ;
        addf  wr&r0,m                                          ;
        endm


        last_product      macro   r0,r1                                 ;

        clr   wr&r0       mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   ;
        addf  wr&r0,m     mov 1kr0,ram1    incbp0   incbp1   spcix0  ;
        addf  wr&r0,m     mov ram0,wr&r1   stix1    clrbp0   spcbp0  ;
                                           incix0   clrbp1   ;
        endm                                                   ;

;-------------------------------------------------------------------------;
;                                                                         ;
;       3.2 macro reference: calculate z = x y                            ;
;                                                                         ;
;-------------------------------------------------------------------------;

vec_pd: ldi     lc,size_n                                  ; load vector x=B[i,1]
        ldi     ix0,size_n-1                               ;
        ldi     ar,B_mat                                   ;
        spcix0                                             ;
        call    htra2e                                     ; call demochip routine
        nop                                                ;

        ldi     lc,size_n                                  ; load vector y=C[1,i]
        ldi     ix1,size_n-1                               ;
        ldi     ar,C_mat                                   ;
        spcix1                                             ;
        call    htra4e                                     ; call demochip routine
        nop                                                ;

        ldi     bp0,factor1                                ; set BP0 to B[1,1]
        ldi     ix1,factor2                                ; set IX1 to C[1,1]
        ldi     ix0,product                                ; set IX0 to A[1,1]
        clrbp1  spcbp0  spcbi1                             ; initiate pointers

        vector_product  0,1                                ;; calculate x = y z
                        spcix0                             ;
        mov ram0,wr1    spcbp0                             ;

        ldi     lc,size_n                                  ; store vector z=A[i,1]
        ldi     ix0,product + size_n-1                     ;
        ldi     ar,A_mat                                   ;
        spcix0                                             ;
        call    htra2oe                                    ; call demochip routine
        nop                                                ;
        jmp     halt                                       ; stop calculations
        nop                                                ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       4. matrices_product                                               ;
;                                                                         ;
;       The following source code calculates the product of 2 matrices    ;
;       B,C. Matrix B is located in RAM0 matrix C in RAM1. Their          ;
;       product is stored to RAM0. Before calling this macro base-        ;
;       pointer 0 must be set to the start address of matrix B (element   ;
;       B[1,1]) and basepointer 1 to the start address of matrix C        ;
;       (element C[1,1]) awa index-register 0 pointing at the result      ;
;       matrix A element a[1,1].                                          ;
;                                                                         ;
;       Benchmarks:                                                       ;
;         max. computation speed per tap (size n = 6)     39 us           ;
;                                                                         ;
;       Initialization: set bp0 -> B[1,1]                                 ;
;                       set ix1 -> C[1,1]                                 ;
;                       set ix0 -> A[1,1]                                 ;
;                       clear bp1 and specify ix0 and ix1 + bp1           ;
;                       set elements of matrices B,C in RAM0 and RAM1     ;
;                       according to the memory map                       ;
;       Used registers: r0, r1, K, L, M, BP0, BP1, IX0, IX1               ;
;       Output:         RAM0                                              ;
;       Output format:  32 bit ASP floating point format                  ;
;                                                                         ;
;-------------------------------------------------------------------------;


;-------------------------------------------------------------------------;
;                                                                         ;
;       4.1 macro definition                                              ;
;                                                                         ;
;-------------------------------------------------------------------------;

        matrices_product        macro   r0,r1           ;

        vector_product  r0,r1                           ;; calculate A[i,1]
        next_product    r0,r1                           ;; calculate A[i,2]
        next_product    r0,r1                           ;; calculate A[i,3]
        next_product    r0,r1                           ;; calculate A[i,4]
        next_product    r0,r1                           ;; calculate A[i,5]
        next_product    r0,r1                           ;; calculate A[i,6]
        nop                                     spcix0  ;
                mov ram0,wr1                    spcbp0  ;
        endm                                            ;


        next_product    macro   r0,r1           ;

        scalar_product  r0,r1                                   ;;
        scalar_product  r1,r0                                   ;;
        scalar_product  r0,r1                                   ;;
        scalar_product  r1,r0                                   ;;
        scalar_product  r0,r1                                   ;;
        last_product    r1,r0                                   ;;
        endm                                    ;
```

```
;-------------------------------------------------------------------------;
;                                                                         ;
;       4.2 macro reference: calculate A = B C                            ;
;                                                                         ;
;-------------------------------------------------------------------------;

mat_pd: ldi     lc,sqrsize                      ; load matrix B
        ldi     ix0,sqrsize-1                   ;
        ldi     ar,B_mat                        ;
        spcix0                                  ;
        call    htra2e                          ; call demochip routine
        nop                                     ;

        ldi     lc,sqrsize                      ; load matrix C
        ldi     ix1,sqrsize-1                   ;
        ldi     ar,C_mat                        ;
        spcix1                                  ;
        call    htra4e                          ; call demochip routine
        nop                                     ;

        ldi     bp0,factor1                     ; set BP0 to B[1,1]
        ldi     ix1,factor2                     ; set IX1 to C[1,1]
        ldi     ix0,product                     ; set IX0 to A[1,1]
        clrbp1  spcbp0  spcbi1                  ; initiate pointers

        matrices_product        0,1             ;; calculate A = B C

        ldi     lc,sqrsize                      ; store matrix A
        ldi     ix0,product + sqrsize-1         ;
        ldi     ar,A_mat                        ;
        spcix0                                  ;
        call    htra2oe                         ; call demochip routine
        nop                                     ;
        jmp     halt                            ; stop calculations
        nop                                     ;

end
```

Synchronization at the μPD7220A Graphics Display Controller

Contents:

Authors: P. Westerdorf / U. Schäfer
Application Department
NEC Electronics (Europe) GmbH

1. Outline

The external vertical synchronization with the GDC may be easily achieved, and with the help of some additional hardware, horizontal synchronization is also possible.

When running the μPD7220A with external vertical synchronization in an interlaced mode, a special initialization sequence must be followed.

In case of a different amount of active display lines between a master (e.g. a camera) and a slave (= GDC), the μPD7220A may be programmed to perform vertical synchronization with each new frame.

The line pairing error of the GDC, in the interlaced mode, will be dependent on the quality of the utilized display monitor and may be totally eliminated with some extra hardware.

These items will be explained more in detail below.

2. External Vertical Synchronization

In cases using one GDC as a master and a second GDC as slave, it may occur that after RESET and specification of Master and Slave that they run synchronously, or that they are shifted apart by one 2xWCLK clock period. Each falling edge of the Master VSYNC pulse will reset the internal synchronization generator of the slave GDC. The ALE (= Address Latch Enable) signals of the slave are synchronized to the master and simultaneous frame synchronization is done.
(Note: One ALE signal will occur within two 2xWCLK clock cycles.)

This synchronization is stopped upon sending the START command.

In case where the Master (e.g. Video Tape Recorder) processes less display lines that the Slave (= GDC), the slave must perform a continuous line synchronization, even during the active display. If the master then generates a VSYNC pulse, the slave will also be forced to start a new frame. Here it is important to note that the CE-Bit of the CHAR command which is used to select this operation mode must not be specified before sending the START command.

3. External Vertical Synchronization in Interlaced Mode

In selecting the interlaced mode for master and slave, an additional field synchronization is required. This is necessary to prevent having the wrong fields of Master and Slave synchronized after RESET. The μPD7220A will always generate non-interlaced video even if interlaced has been specified. In the interlaced mode, the GDC will start with the first field. To do this, first the Slave and then the Master are reset by RESET 1. Then Bit 5 (VSYNC) of the Master status register is checked for the transition from 1 to 0. This is the point when the master enters the second field and the slave the first field. Then this transition is checked a second time and after this the Master and the Slave enter the first field. Lastly, the START command is sent to the Slave and Master and ends the field synchronization.

4. Horizontal Synchronization

Horizontal synchronization is not as easy to implement as vertical synchronization, but may be achieved with the help of a PLL (= Phase Locked Loop). This PLL compares the HSYNC frequencies of Master (e.g. Video Tape Recorder) and Slave (= GDC), and in case of differences, a VCO (Voltage Controlled Oscillator) is adjusted accordingly. The output frequency of this VCO is used as DOT frequency, which in turn generates the 2xWCLK for the GDC, as well as closing the loop. A schematic may be found in the appendix.

5. The Line Pairing Error

Because the GDC is not able to add all horizontal Parameters (HFP, HS, HBP, AW) and to divide these by two to find the exact middle position of a line, another approximation is used. The μPD7220A uses active words, divided by two minus 1.5, as the middle position. The cause of the line pairing error depends largely on the quality of the utilized display monitor.

The greater the relationship between AW (= Active Word) on HB (= Horizontal Blank = HFP + HS + HBP) is, the smaller the error. Usually, better monitors have a high AW value compared to HB. This line pairing error can be eliminated by using extra hardware. A schematic may be found in the appendix. In the following, the above items are discussed once again in more detail.

Interlaced Video

Interlaced video (2:1 interlaced) is generated using a two-video field sequence (in which the total line count for both fields is odd, and the vertival sync pulse between the two fields is offset by exactly one-half line time). These conditions cause the display lines of the second field to be displayed between the lines of the first field. In this way, an image may be displayed at a reduced bandwith without flicker. Interlacing is particularly suited to the display of camera- (or computer-) generated images (as differentiated from line drawings and text) because of the spatially band-limited nature of these images. This band-limiting exists along both the horizontal sweep line and vertically across the video lines. Interlaced video is less well suited to typical computer-generated graphics and text displays because there is no spatial band-limiting in the vertical direction. For example, one line in the video raster might be black while the following line is full white. If these lines are scanned in alternate video fields, as they are in using interlaced video, there will be a noticeable flicker and apparent vibration. Long persistence phosphors can minimize the problem.

The GDC may be programmed to generate interlaced video. The separate horizontal and vertical sync signals may be used with any CRT unit which accepts separate sync signals. The total line count of both fields (one video frame) will be one more than twice the line count per field for which the GDC is programmed. The GDC's video sync generator adds this extra line automatically and offsets the second VSYNC pulse by approximately one-half line. In typical applications, the second field will be displayed within 15 % of the ideal position between the lines of the first field.

Transitions of the VSYNC output during interlaced operation occur at two different points with respect to the HSYNC pulse. First, field transitions (both leading and falling edges) occur simultaneously with the leading edge of BLANK. In the second field, the transitions occur three 2xWCLK cycles before the middle of the active words interval, AW. It does not automatically position the VSYNC transitions at the exact midpoint of the line (including the retrace blanking period). The exact relationship is shown in figure 6-1 and the following:

Interval A = 2 (HFP + HS + HBP + AW/2) — 3 (2xWCLK cycles)
Interval B = 2 (AW/2) + 3 (2xWCLK cycles)

Horizontal blanked display cycles, HB:

HB = HFP + HS + HBP

Total number of 2xWCLK cycles during a horizontal line, TC:

TC = HFP + HS + HBP + AW

The line pairing interlacing error percentage, LPE (percentage of a half line of offset):

LPE = ((interval A — Interval B) (TC(2)) x 100 %
= ((2HB + AW — 3) — (AW + 3) / (TC/2)) x 100 %
= ((2HB — 6) / (TC/2)) x 100 %
= ((4HB — 12) / TC) x 100 %

The line pairing error for a typical application:

TC = 100 2xwclk cycles
HB = (1/5 x TC) / 2 blanking percentages of 20 %
LPE = ((4 x 10 — 12) / 100) x 100 % = 28 %

The differences between standard interlaced video and the interlaced video signals produced using the 7220 are elaborated below.

Horizontal Sync. In standard video, equalization pulses are generated during the VSYNC period to keep the receiving television or monitor's horizontal oscillator from drifting, and thus keeping the set in sync. The GDC, however, does not generate these pulses explicity. Equalization pulses independent of standard HS pulses were determined to be unnecessary due to the high quality of monitors built during the last few years.

Line Spacing. During standard interlaced video, VSYNC on odd video fields (1st, 3rd, 5th, etc.) begins and ends with the start of the horizontal front porch. VSYNC on even fields, however, begins at the midpoint of the period consisting of the sum of the horizontal front porch, the horizontal sync period, the horizontal back porch, and the active video period for each line.

The GDC knows the values of horizontal front porch (HFP), horizontal back porch (HBP), horizontal sync period (HS), and active video period (AW) in terms of words of display memory. As it does not have the facilities to add these values together however, it uses another method to generate the timing for the even VSYNC pulses. They are generated three 2xWCLK pulses before the midpoint of AW (figure 6-2).

The resulting line pairing problem is observed as uneven line spacing between adjacent lines. This problem is only observed on low quality monitors and will not be a problem in most applications. The actual spacing between the lines may be calculated as follows:

Field 1 line — field 2 line = AW/2 + 1.5
Field 2 line — field 1 line = HFP + HS + HBP + AW/2 — 1.5

In most systems, the sum of HFP, HBP and HS represents 25 % or less of one horizontal line period. Therefore, the deviation between the two line spacings will be less than 15 %. This difference is not noticeable in the vast majority of designs.

Systems in which this spacing difference is a problem may use the circuit illustrated by figures 6-3 and 6-4 to correct the VSYNC generation.

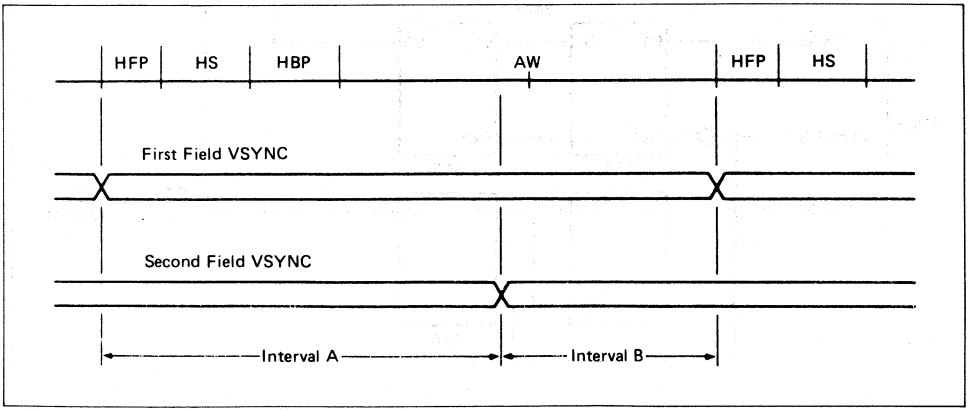


Figure 6-1. Horizontal Timing Relative to Vertical Sync Transitions

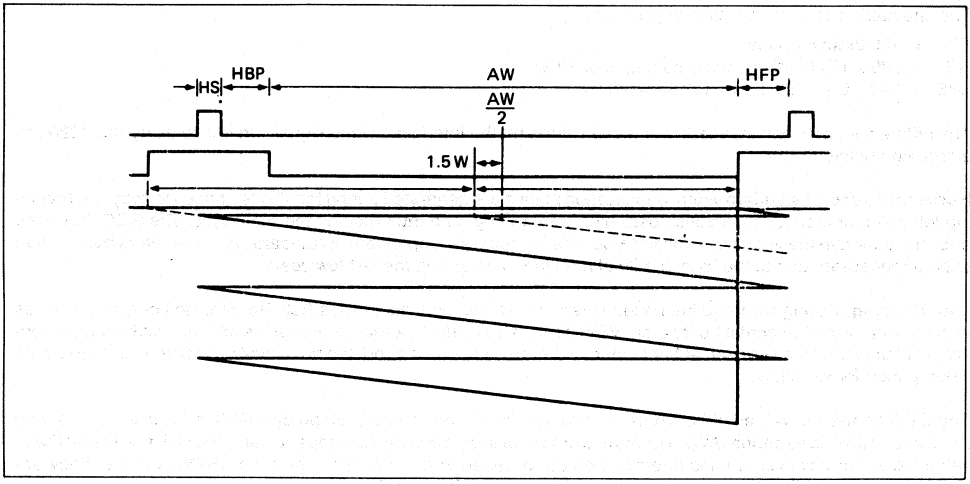


Figure 6-2. GDC-Generated Interlaced Display

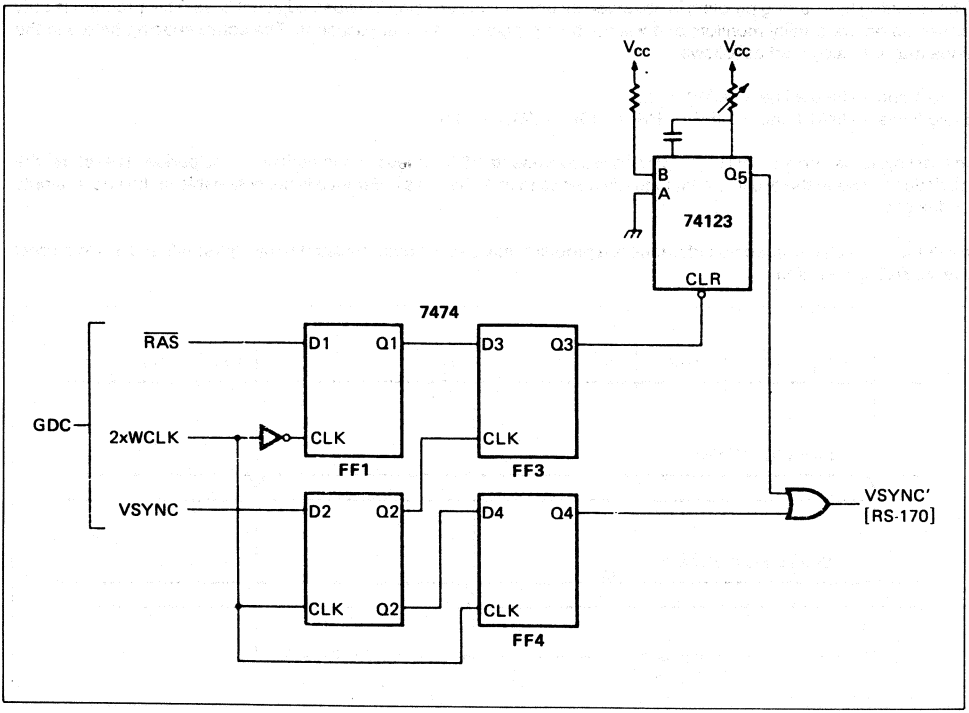


Figure 6-3. VSYNC Correction Circuit

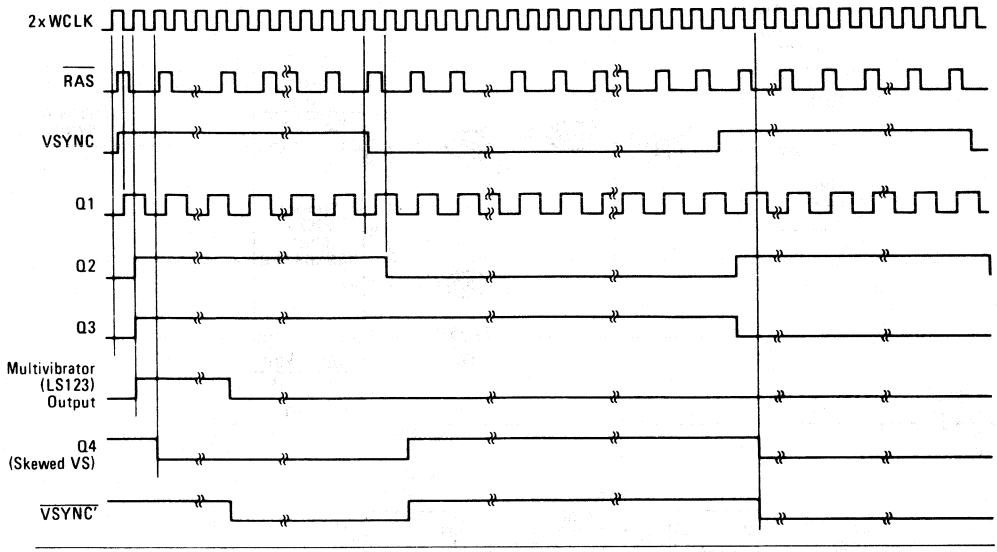


Figure 6-4. Timing Waveforms for VSYNC Correction Circuit

It is possible to use the GDC in a display apparatus to directly record character and graphic data created by the GDC on a video recorder without passing the data through a TV camera or overlaying the data to the video signal from a TV camera, a video disc, etc. This appendix describes a circuit (figure G-1) for synchronizing the GDC with signals conforming to the NTSC standard.

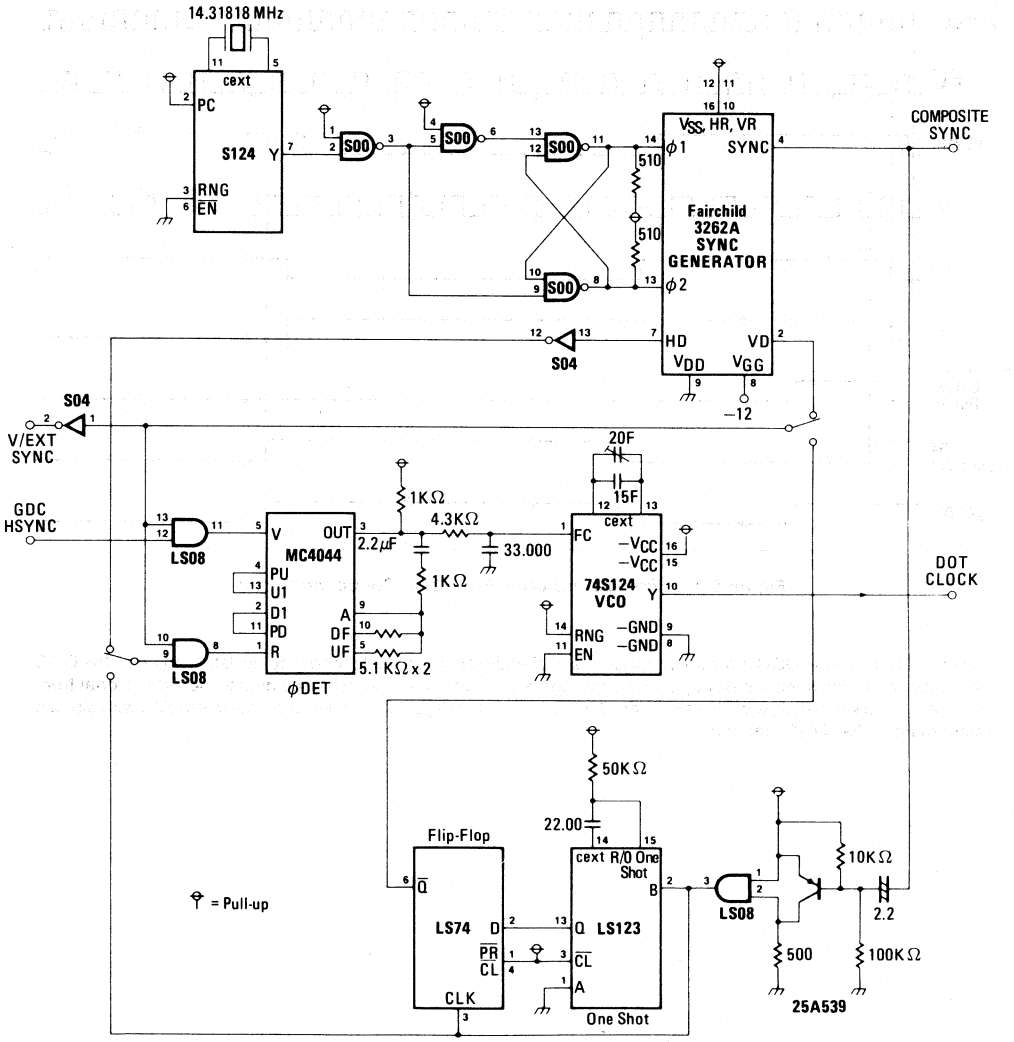


Figure G-1. NTSC Circuit

Synchronization to VSYNC [One Frame]

Vertical synchronization is established by setting the GDC to slave and connecting the VSYNC signal from a video apparatus, such as a TV camera or video disc, to the V/EXT SYNC pin.

When one frame period for the GDC is equal to one frame period of the video apparatus, this vertical synchronization is performed only once (at the beginning of the GDC operation).

Synchronous aberrations after the vertical synchronization operation are corrected by horizontal synchronization with the PLL mentioned below.

Synchronization to HSYCN [One Line]

Horizontal synchronization is established with the following operations:

Locate the original oscillator that generates the dot-clock and provides the GDC with a divided dot-clock in the PLL.

Change the oscillating frequency by detecting the phase difference between two falling edges of the GDC's HSYNC and the HSYNC of the video apparatus. If the horizontal period (1H) of the GDC is equal to the horizontal period of the apparatus, do not change the oscillating frequency. The circuit in figure G-1 uses Fairchild's 3262A sync generator for a master sync signal generator. HD/VD signals are removed and only a composite sync signal is provided.

The 3262A generates sync signals that conform to the NTSC standard by supplying a 14.31818-MHz clock.

The phase-sensitive detector MC4044 detects the phase difference between the falling edges of the GDC's HSYNC and the HD signal from the 3262A and controls the voltage controlled oscillator (VCO), 74S124. The VCO lowers the frequency when advance is detected and raises the frequency when delay is detected.

Since the equalizing pulse generated during the VD signal is active, the input signals of the phase-sensitive detector are gated by the VD signal to remove the effect of the equalizing pulse.

A 25A539 separates the signals associated with the SYNC signal from the composite SYNC signal. The LS123 one-shot multivibrator separates nine horizontal periods after the beginning of VBLANK.

The SYNC signal conforming to NTSC standard is a master signal, and the GDC is slave. Vertical synchronization is roughly performed by the V/EXT SYNC function, and strict horizontal synchronization is performed by the PLL. Therefore, the master VSYNC signal being input to V/EXT SYNC may be a rough signal, of which one period is detectable by the GDC. It is not necessary to provide the master VSYNC signal to the slave GDC every vertical period.

Figure G-2 shows the commands that will allow the 7220 to be synched to an NTSC signal.

Figure 6-2. Slave GDC Synchronization Commands and Parameters

Commands		Parameters	
C	RESET	00	
C	MASTER/SLAVE	6E	(SLAVE)
C	SYNC	0E	
P1		0A	
P2		46	
P3		05	
P4		00	
P5		08	
P6		01	
P7		F4	
P8		24	

Issue the START command after detecting the falling of VSYNC twice.

An Adaptive Echo Canceller for ISDN Applications with NEC's Digital Signal-Processor μPD77C20

Contents:

Author: K. Grohe
Application Department
NEC Electronics (Europe) GmbH

A Proposal

Many efforts have been made to solve the echo cancellation problem of the U-loop link in ISDN applications. Still, single chip interface circuits are rare and do not fully match with target specifications. Meanwhile, interim solutions until dedicated VLSI chips become available are of great interest. The following pages introduce two possible implementations, a standard signal-processor, and an ASIC solution.

1. The U_{K0} Interface Circuit

Two wire U-loop links connect the subscribers network terminator to the local PBX. Although the specifications of the U-reference point are still not in agreement, a high sophisticated echo cancellation technique must be applied in order to achieve full duplex 160 kbps transmission rates over lines up to 8 km length. Figure 1 shows the location of the U-reference point within the ISDN reference model.

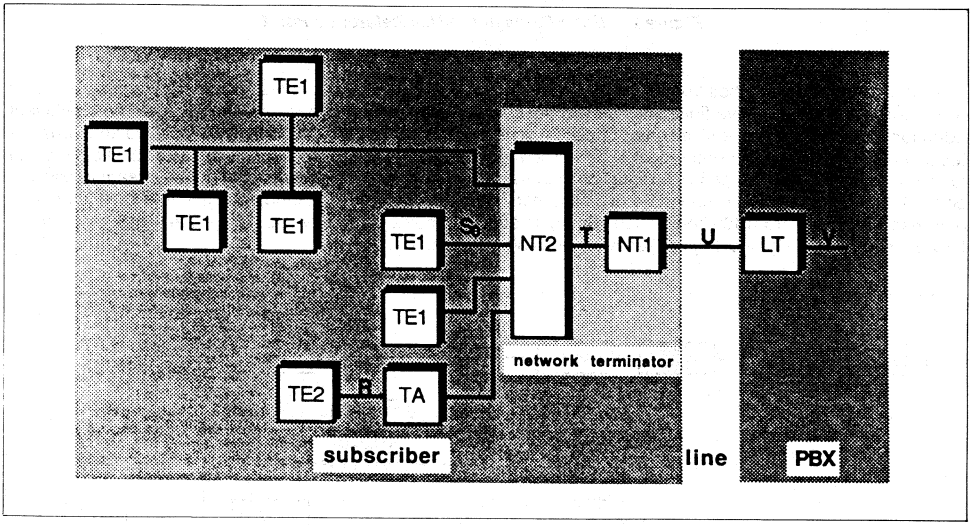


Figure 1. ISDN Reference Model

"Full duplex technique over two wires", implies that the transmitter's output signal couples into the receiver's input signal via certain echo paths. The main echo is generated within the subscriber itself (caused by the difference transformer). Other echo sources arise from electrically inconsistent wires, non matching impedance of the counterpart network terminator, etc. (Figure 2).

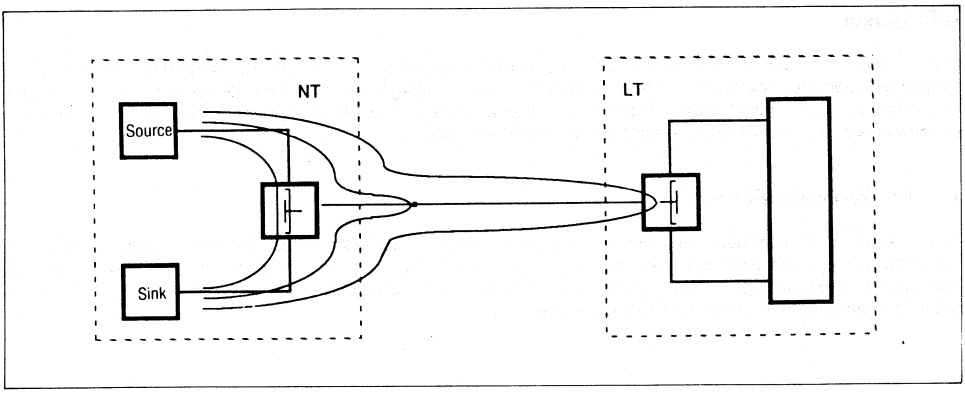


Figure 2. Echo Paths at the U_{K0} Reference Point

To eliminate transmitter echos from the received signal, an echo canceller becomes necessary. Transmitter errors are predicted by an adaptive finite impulse response filter that forms a model of the echo paths. The coefficient adaption algorithm can be realized on a correlation scheme. This works correctly as long as received and transmitted signals are statistically independent. Different scrambler functions in both sections based on a 23 bit polynomial fulfill the statistical independency. A reduction in transmission rate is achieved by coding the bit stream to MMS43 code. A 4 bit binary stream is coded into a 3 bit ternary signal stream (4B/3T coding) resulting in a 25 % reduction of transmission rate (from 160 kbps to 120 kBaud). In addition, the AC characteristics of the line are better adapted for applying ternary signals. Figure 3 shows a possible solution.

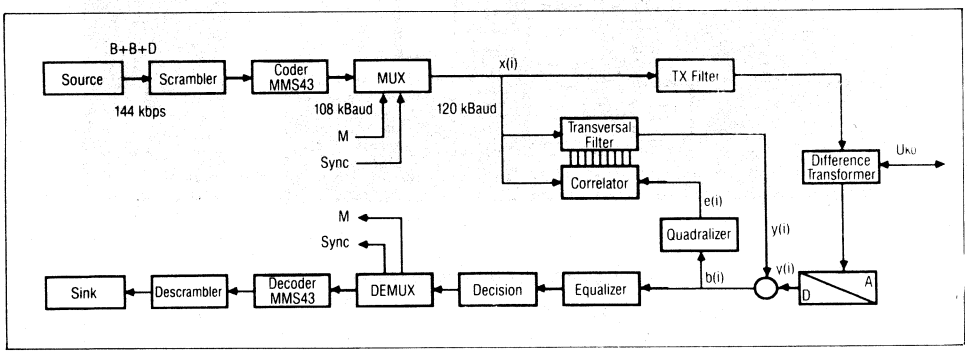


Figure 3. Block Diagram of an U_{K0} Interface Circuit

2. The Echo Canceller

The ternary transmit signal is fed into the echo canceller after shaping onto the transmission line. The echo canceller outputs a model signal y(i) that predicts the echo path of the transmit signal components in the receive signal y(i). The receive signal and the model signal are subtracted. The difference b(i), ideally should match with the pure receive signal, excluding any transmitter signal components. b(i) is quantized forming the error signal e(i).

Error signal and transmitter signal are correlated and accumulated leading to the updated coefficients of the transversal filter. The algorithm to update coefficients minimizes the least mean square of the error signal (LMS).

$$\Sigma\ e(i)^2 = \text{Min !}$$

It is realized in an iterative gradient algorithm. A quantization table allows for the optimal use the echo canceller due to fast convergence and small residual error. Figure 4 shows the block diagram of the transversal filter, including correlator, forming the adaptive FIR filter.

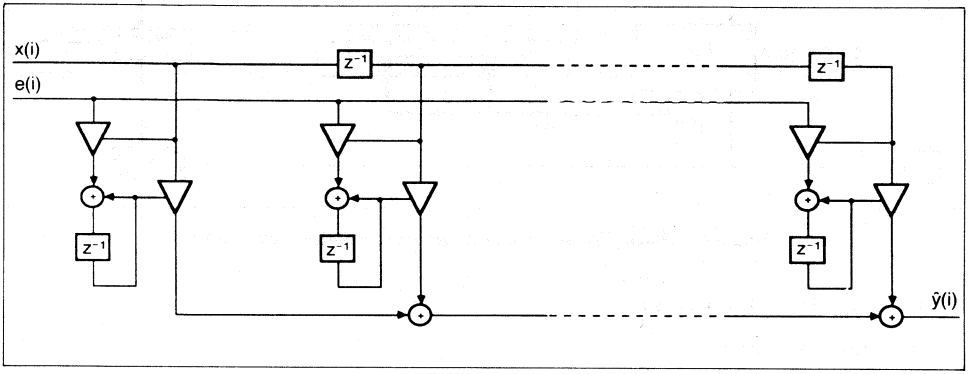


Figure 4. Block Diagram of Adaptive FIR Filter

The following equations must be solved:

1. Transversal Filter Calculation:

$$y(i) = \sum_{K=1}^{N} C_k \cdot x\,(i - k)$$

2. Coefficient Adjustment:

$$C(k + 1) = C\,(k) + e\,(k) \cdot x\,(i - k)$$

Parameter N defines the FIR filter length and is closely related to transmission rate and the max. distance between subscriber and PBX:

$$N > 2\ C_V \text{ distance} \cdot \text{transmission rate with } C_V \approx 300.000 \text{ km/sec.}$$

A distance of 8 km is recommanded leading to a value N:

$$N = 32.$$

3. A Signal-Processor Implementation

Figure 5 shows the U_{K0} interface circuit of figure 3, decomposed into blocks that fit into NEC's signal-processor, the μPD77C20. 32 tap adaptive filter is realized in two cascaded μPD77C20s (AFiR16) running at 16.6 MHz. Transmitter signal, intermediate signal after 16 taps, and modelled signal are linked via the processor's serial interfaces. The dif-

ference between receive signal and modelled signal is encountered in the digital front end quantized and results in an error signal distributed via the parallel bus interface to the AFiR16 processors. The digital front end performs all the other functions shown in figure 3 (scrambling, coding, framing etc.). Again, a μPD77C20 implementation seems realistic. Obviously, the most critical part of the U interface circuit lies in the adaptive filter. Following is a brief explenation of the AFiR16 routine running on μPD77C20.

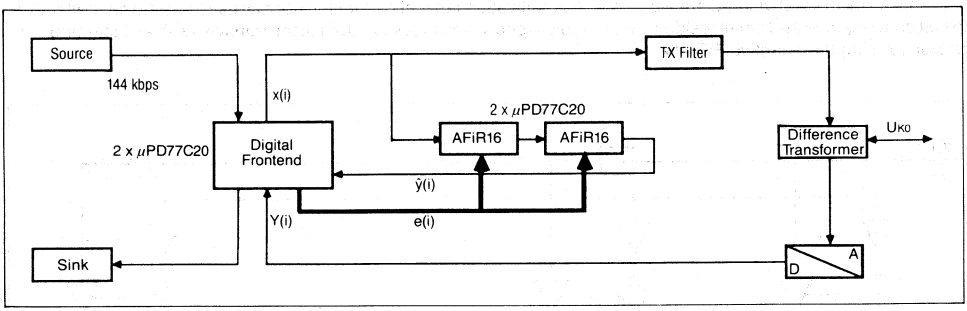


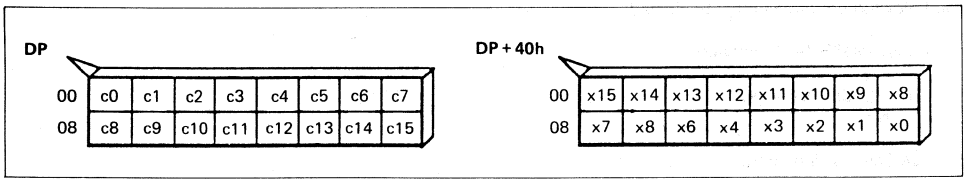
Figure 5. Block Diagram of an UK0 Interface Realized with DSPs

The following tasks must be managed:

a) Task 1 (transversal filter)

equation: $$y(i) = \sum_{k=1}^{N} C_k \cdot x\,(i - k)$$

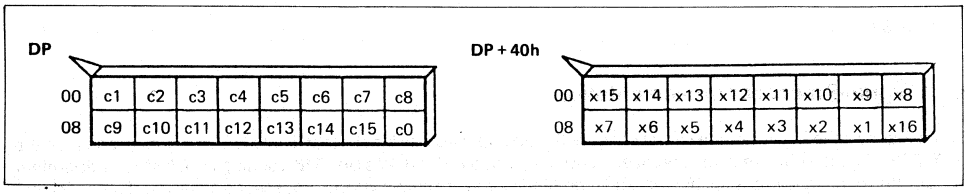
memory allocation at time k:



Note:

The memory allocation changes dynamically with every sampling cycle. Samples of the input signal 'x', which extend the filter length, are overwritten with the actual value. This is done in task 2 (input/output). Additionally the coefficients C must be rotated with every sample period. This is done in task 3 (adaption. The above memory allocation is only correct at the sampling time 'k'. One sample later the allocation looks like this:

memory allocation at time k + 1:



Note:

This algorithm does not include overflow handling. It works correctly only if the output signal does not overflow. That means within each tap, the sum does not exceed 1/n of the 16 bit register length. (For example: filter length = 32 coefficients ternary input signal ——> coeff. length does not exceed 11 bits). For full length after 3 consecutive additions, an overflow check must be done to lower the computational speed.

b) Task 2 (input/output)

4 instructions are needed to link the AFiR16 routine to external signals and to loop to the next sampling period.

c) Task 3 (adaption)

equation: $C(k+1) = C(k) + e(k) \cdot x(i-k)$

Note:

This algorithm requires 4 instruction cycles. The coefficient must be fetched from memory (1 cycle), the new coefficient calculated (2 cycles), and then stored back in the next memory cell (1 cycle). To force this algorithm, the pipeline architecture of the 7720 should be used. That means the coefficients are no longer calculated sequentially, but rather quasi parallel. 2 consecutive coefficients can be calculated in 6 instruction cycles (see below) which spare 1 cycle with each tap.

The complete listing of the AFiR16 routine running within 8.28 µsec per tap at 16.6 MHz master frequency which fits into the 120 kBaud frame follows.

```
;------------------------------------------------------------------------------;
;                                                                              ;
;               16 tap Adaptive Transversal Filter (AFIR16)                    ;
;                                                                              ;
;                                Kernel                                        ;
;                                                                              ;
;               Initialization: set DP                                         ;
;                               set initial coefficient values                 ;
;                               clear input buffer                             ;
;                               clear TR                                       ;
;                                                                              ;
;------------------------------------------------------------------------------;

AFIR 16: OP      MOV @KLM,MEM                        DPINC    ; transversal filter
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M       DPINC    ;
         OP      MOV @KLM,MEM       ADD ACCA,M                ;
         OP      MOV @MEM,SI        ADD ACCA,M                ; get input sample
         OP      MOV @SO,A                                    ; output prev. sample
         OP      MOV @B,MEM                                   ; coefficient adaption
         OP      MOV @KLM,TR                         DPDEC    ;
         OP      MOV @A,MEM         ADD ACCB,M                ;
         OP      MOV @MEM,B                                   ;
         OP      MOV @KLM,TR                         DPDEC    ;
```

```
OP       MOV @B,MEM       ADD ACCA,M                   ;
OP       MOV @MEM,A                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @A,MEM       ADD ACCB,M                   ;
OP       MOV @MEM,B                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @B,MEM       ADD ACCA,M                   ;
OP       MOV @MEM,A                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @A,MEM       ADD ACCB,M                   ;
OP       MOV @MEM,B                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @B,MEM       ADD ACCA,M                   ;
OP       MOV @MEM,A                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @A,MEM       ADD ACCB,M                   ;
OP       MOV @MEM,B                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @B,MEM       ADD ACCA,M                   ;
OP       MOV @MEM,A                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @A,MEM       ADD ACCB,M                   ;
OP       MOV @MEM,B                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @B,MEM       ADD ACCA,M                   ;
OP       MOV @MEM,A                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @A,MEM       ADD ACCB,M                   ;
OP       MOV @MEM,B                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @B,MEM       ADD ACCA,M                   ;
OP       MOV @MEM,A                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @A,MEM       ADD ACCB,M                   ;
OP       MOV @MEM,B                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @B,MEM       ADD ACCA,M                   ;
OP       MOV @MEM,A                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @A,MEM       ADD ACCB,M                   ;
OP       MOV @MEM,B                                    ;
OP       MOV @KLM,TR                        DPDEC      ;
OP       MOV @TR,DR       ADD ACCA,M                   ; read new e(i)
OP       MOV @MEM,A       XOR ACCA,IDB                 ; prepare next sample
JNESO    $                                             ; synchronize
JMP      AFIR16                                        ; loop to next sample
```

4. An ASIC Implementation

Generally, the implementation of DSP algorithms on ASIC fail due to the complexity of a hardware multiplier with sufficient accuracy. A big advantage of the echo canceller is that it works on a ternary transmitter signal. In other words, all multiplications that must be computed are 2 bit by n bit multiplications. A multiplier for ternary signals may easily be implemented as follows:

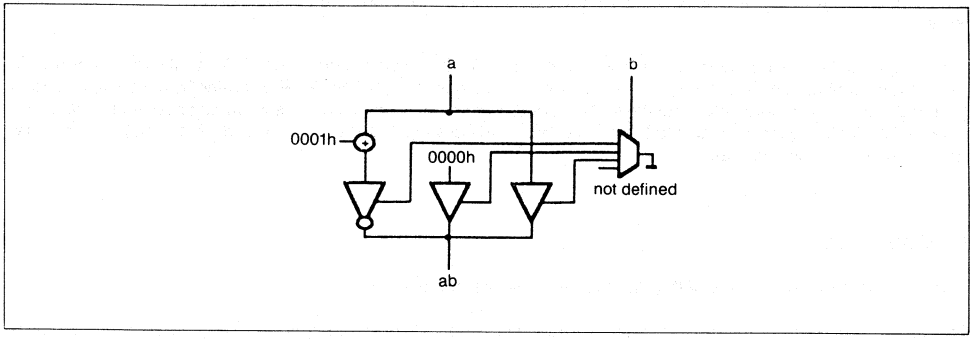


Figure 6. Multiplier for ternary signals

To determine the number of gates, the wordlength of all signals and coefficients is of interest. (1) details wordlength and accuracy. Figure 7 shows a possible solution:

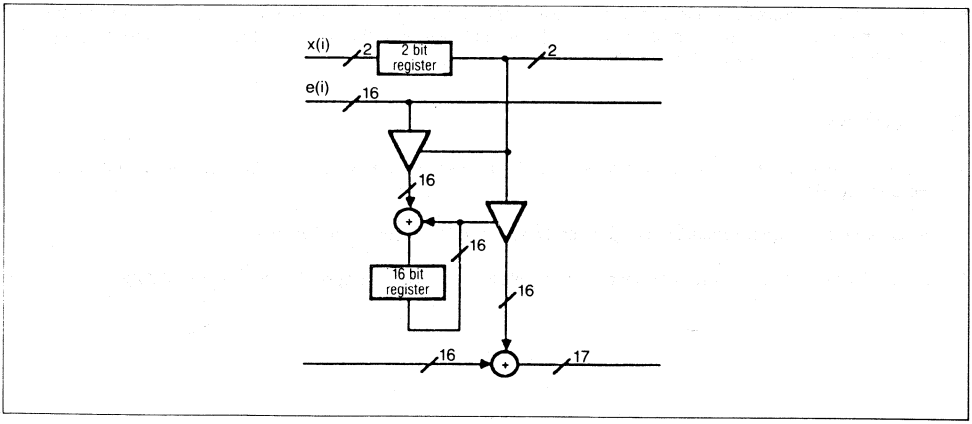


Figure 7. Filter tap and word length

5. Evaluation Notes

Up to now, the basic part of the U$_{K0}$ interface has been discussed and two possible implementations were shown in principle.

Still, a concrete realization needs further investigation. Especially convergence problems, and problems of the residual error which haven't been discussed here.

Besides all theoretical background a realtime evaluation seems promising. For the moment, evaluation may be worked out on the EPROM device of the μPD7725, the μPD77P25.

Whenever wordlength effects need to be analyzed, the floating point signal-processor μPD77230 is an adequate device. No hardware design is necessary if the EB-77230 FFT is used — an evaluation board that fits into IBM PC's.

Finally, if the optimum algorithm has been encountered, a CMOS mask version of the μPD77C20 can be ordered. To reduce the number of different masks, it is useful to put (besides the optimum AFiR16 routine) routines of the digital frontend (scrambler), code, etc.) into the same device. The actual routine a processor must choose may easily be selected by a reset vector initializing the parallel bus of the device. The device itself checks this vector after reset and jumps to its individual task.

6. Conclusion

Two possible implementations of the U$_{K0}$ interface have been presented.

The signal processor solution is split into an adaptive echo canceller requiring two μPD77C20's and a digital frontend. A detailed look into the adaptive echo canceller was presented. The digital frontend needs further investigation, however, most tasks may be implemented easily because they do not require high sophisticated algorithms. Coder/Decoder and quantizer may be realized through lookup talbes, whereas the scrambler/-descrambler requires only simple arithmetic calculations. At least two μPD77C20's should be used to handle the digital frontend. The complete U$_{K0}$ interface will require four signal processors, μPD77C20's.

Regarding the second implementation, a reduction of the number of components requires freezing the algorithm evaluated, with the signal-processor solution, into an application specific IC.DSPing with ASIC. This is normally not a problem because the implementation of a hardware multiplier is rather easy in this case and ternary signals greatly reduce the complexity of the hardware multiplier.

7. Literature

1. P. Kahl, "ISDN das künftige Fernmeldenetz der Deutschen Bundespost", R. v. Deckers Verlag, G. Schenck. Heidelberg 1985

2. M. Bellanger, "Digital Processing of Signals Theory and Practice", John Wiley & Son

3. Digital Signal Processor Product Description by NEC Electronic (Europe) GmbH PDDSP 067V30

Broadband Speech Coding with Standard Devices

Contents:

Author: K. Grohe
Application Department
NEC Electronics (Europe) GmbH

1. Introduction

Transmission and storage of digitized speech has become a standard in telecommunications, and offers all the advantages of digital systems. Existing older analog systems have been digitized without noticable increase in speech quality, and almost every large telecom firm has made research on speech compression algorithms. The aim is twofold; first, to achieve a better speech quality within existing channel capacity, and second, to increase the channel capacity by keeping the standard toll quality.

The transmission sector, ISDN (being the driving force of the 90's), will bring many new services with obviously greater popularity for professional customers. In contrast, the private user generally not requiring additional services besides his telephone set, has fewer advantages with the ISDN concept. A broad and increasing acceptance even from private users, will be achievable if the speech quality of telephone transmissions increase. This is the idea of so called "broadband ISDN". Certain European telecom companies have been working on different speech compression/expansion algorithms for broadband ISDN which are now ready to be standardized by the European PTT's gremium CEPT in Geneva.

On the other hand, there is the market segment of voice store and forward systems. A very young, but rather fast growing market, with applications like automatic answering machines, voice mailing systems and information systems with speech output etc. Applying speech compression makes it possible to reduce memory amount without loss of speech quality. Todays toll quality speech devices are well accepted in most of the applications. But as technological progress makes it possible to increase speech quality, an increasing number of designers are asking for products that can handle bandwidth of 7KHz.

Both sectors require adequate speech coding devices. NEC, as a pioneer on sophisticated speech devices currently offers a complete palette of speech products for almost every application. One candidate is the ADPCM speech encoder/decoder (SED), the μPD7730/μPD7731, available since 1984. In 1986 NEC introduced the CMOS version μPD77C30. The reduced die size and power dissipation of the μPD77C30 allows the design of broadband ADPCM applications with state of the art high speech quality comparable to AM receivers, with just a single chip.

2. The Algorithm

Among several coding schemes NEC has employed, one is a waveform coding scheme named "adaptive differential pulse code modulation" (ADPCM). Compared to its competitors, waveform coding gives the best compromize between achievable bit rate and natural quality of speech. Applying a parametric coding scheme, such as linear predictive coding (LPC), the bit rate can be reduced far beyond that of an ADPCM, but with rather poor speech quality. Therefore, LPC coders are targetted at the toy and military communication markets.

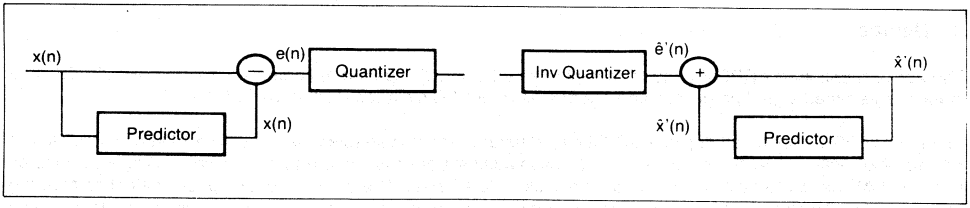


Figure 1. Principle of ADPCM Coding

ADPCM is based on a linear prediction of the signal to be encoded. Figure 1 shows the principle of ADPCM coding. Bit rate reductiin is achieved by transmitting the quantized error signal 'e (n)', instead of the PCM signal 'x (n)'. The actual implementation (figure 2) of the ADPCM algorithm employs some additional improvements such as:

— Prediction on the encoder side is carried out after quantization which results in a reduced quantization distortion.

— The nature of telephne signals allow the reduction of bit rate one stop further by applying adaptive quantizers and predictors.

— Excellent protection against transmission errors is derived by splitting the predictor into a fixed and an adaptive predictor. The fixed predictor contains all the poles of the transfer function within the unit circle and can never be come instable.

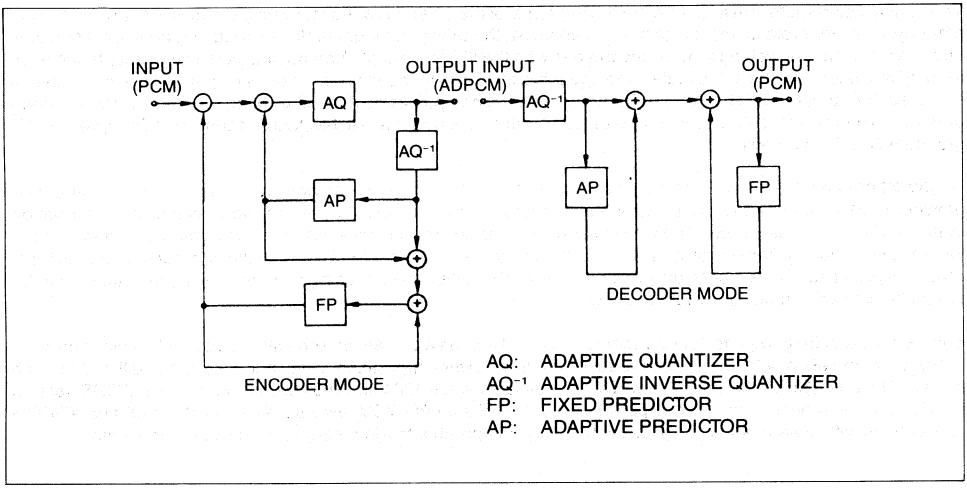


Figure 2. ADPCM Algorithm of μPD77C30

3. Device

The above mentioned ADPCM scheme is implemented in NEC's speech encoder/decoder, the μPD77C30, as a mask programmed signal processor, and is a derivative of NEC's standard DSP, the μPD77C20.

The μPD77C30 covers a half duplex ADPCM encoder/decoder with variable data rates from 16 kbps toll quality, to 64 kbps AM receiver like quality. Excellent protection against transmission errors is achieved. I/O is performed by a serial PCM link, and an 8 bit ADPCM link with packed data format. The device supports pause detection allowing additional speech compression in voice store and forward systems. The following functions are controllable via an 8 bit host interface:

— operating mode

— ADPCM data format 3/4 bit

— PCM data law linear or logarithmic μ- (A-) law

— presettable voice detection threshold

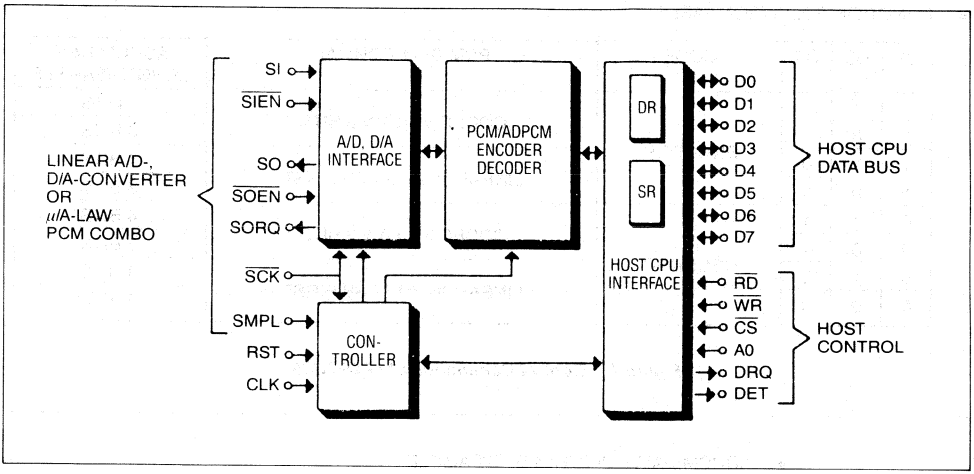


Figure 3. Block Diagram of the μPD77C30

Figure 3 shows the block diagram of the device. The PCM/ADPCM encoder/decoder is surounded by an ADIDA interface which converts the serial PCM input/output stream to parallel data, and vice versa, and to an 8 bit processor interface covering the packed ADPCM streams. Encoder/decoder operation is selected via the host CPU bus effecting the internal controller of the device.

4. Initial Setting

Generally, the μPD77C30 is used as a slave processor controlled via the 8 bit bus. Before speech coding is performed, the device must be initialized as follows:

Encoder Mode:

After releasing the reset signal of the μPD77C30, the device waits for the mode setting control word (figure 4). The host continues with 16 bit threshold data.

Next comes the ADPCM data setting, followed by the start of encoder compression. Figure 5 shows the output format of the word. Input signals with power below the threshold value will be interpreted as pause signals, and the detection pin (DET) will be raised to signal a pause to the host. The link between host and the μPD77C30 may be either interrupt driven or polled by software.

1. CONTROL COMMANDS

- SELECTABLE FUNCTIONS
  - — OPERATING MODE : ENCODER OR DECODER
  - — ADPCM DATA LENGTH : 3 BITS/SAMPLE OR 4 BITS/SAMPLE
  - — A/D, D/A CONVERSION LAW : μ/A-LAW 8 BITS OR LINEAR

- FORMAT

MSB LSB

D7	D6	D5	0	0	0	0	0

- FUNCTION SELECTION TABLE

D7	D6	D5	MODE	PCM DATA FORMAT	ADPCM DATA LENGTH/SAMPLE
1	1	1	ENCODER	CODEC μ/A-LAW 8 BITS	4 BITS
1	0	1			3 BITS
1	1	0		LINEAR 16 BITS LSB FIRST	4 BITS
1	0	0			3 BITS
0	1	1	DECODER	CODEC μ/A-LAW 8 BITS	4 BITS
0	0	1			3 BITS
0	1	0		LINEAR 16 BITS LSB FIRST	4 BITS
0	0	0			3 BITS

Figure 4. Control Commands of μPD77C30

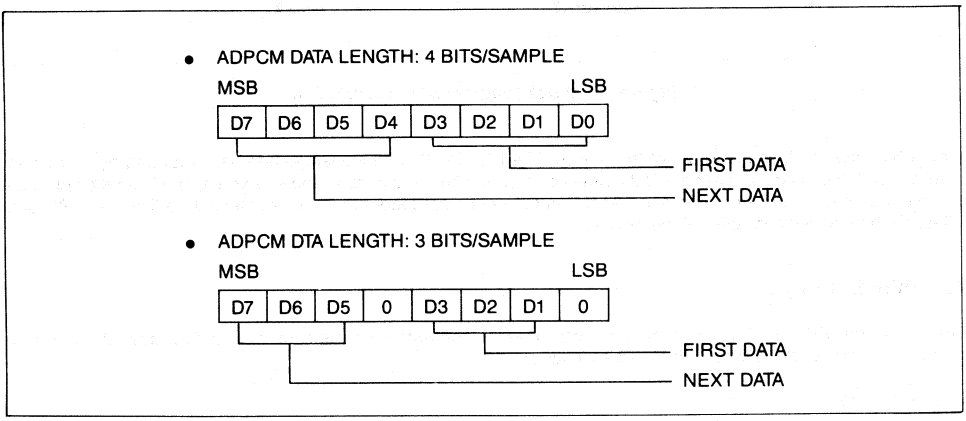


Figure 5. ADPCM Data Format

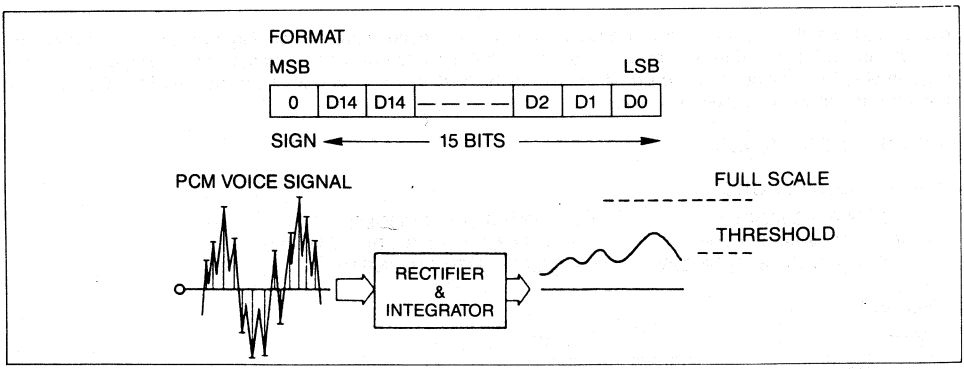


Figure 6. Pause Detection

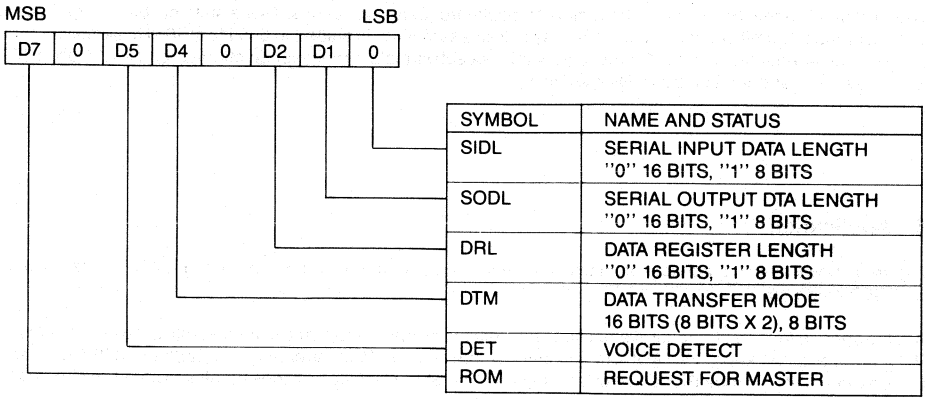


Figure 7. μPD77C30 Status Word Format

5. Gain Tracking at Broadband ADPCM Characteristics

Several tests have been applied to the μPD77C30 with a board supporting a half duplex ADPCM tranceiver with ADPCM loop back capability and Combo interface. Combo and the μPD77C30 run at double speed with 16 kHz sampling frequency. Figure 8 shows the block diagram of gain tracking test, and an excellent frequency response under broadband ADPCM conditions.

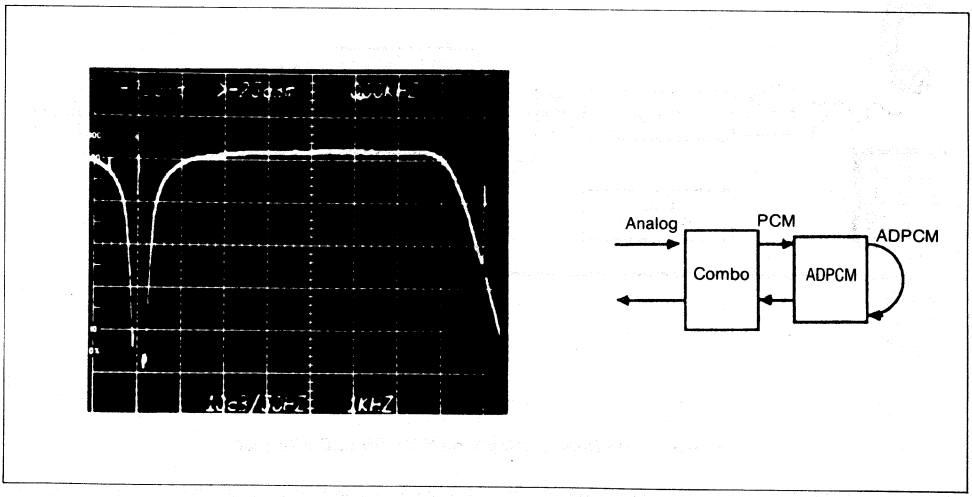


Figure 8. PCM Loop Tracking at 16 kHz Sampling Rate

The lower edge frequency has a weak point caused by the Combo's band pass filter at the receiver section. The bandpass filter cuts frequencies below 600 kHz.

The Combo's internal SC filters are consistent to master frequency variations. As a result, the lower edge frequency of the Combo transmit section, which is 300 Hz at 8 kHz sampling, is shifted to 600 Hz at 16 kHz sampling, thus increasing the attenuation of pitch frequencies within speech signals. Whenever an attenuation of —1.8 db at 400 Hz is acceptable, standard Combos will be sufficient.

6. Application

Figure 9 shows a typical application in telecommunication end equipment of a broadband ADPCM telephone set with SLD interface.

Analog speech from the handset is amplified and digitized by the Combo. At a sampling rate of 16 kHz, the resulting PCM bit rate is 128 kbps. The ADPCM encoder compresses the PCM stream to a broadband ADPCM stream of 64 kbps which fits into one B channel of the SLD bus. The SLD interface provides parallel to serial link interconnection.

On the other hand, broadband ADPCM data from the SLD interface are routed to the decoder μPD77C30 which expand the bit stream into a 128 kbps PCM stream. After DIA conversion and filtering, the signal drives the earphone of the handset. Housekeeping functions, such as dialling, ringing and initializing of the ADPCM devices etc. are realized with an additional controller. The SLD interface may be omitted if the controller is powerful enough to handle real time parallel to serial data conversion.

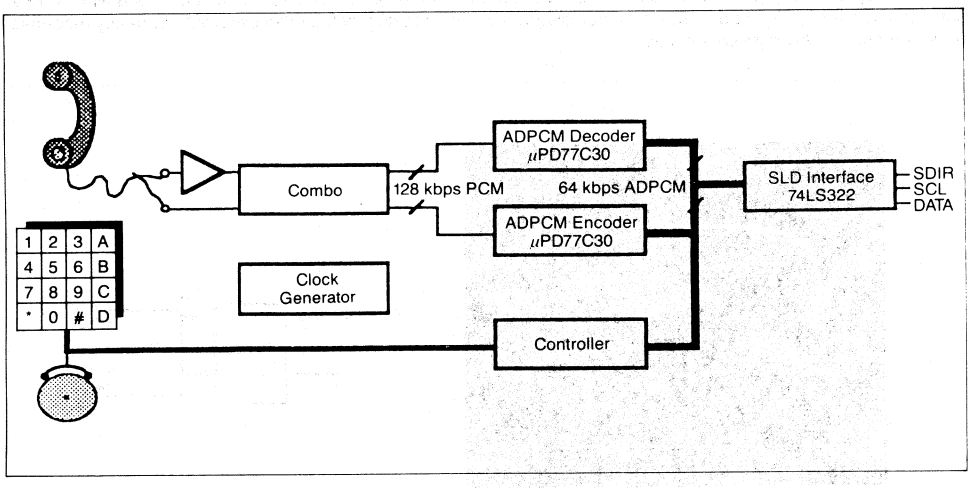
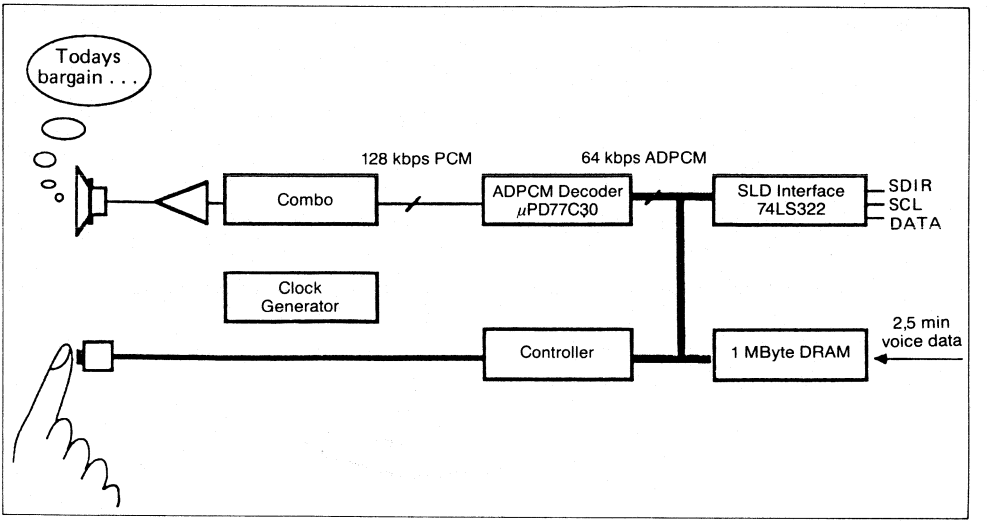


Figure 9. High Quality Telephone Set with SLD Interface

Figure 10 introduces an automatic information system with voice store and forward capability. Generally, these systems require a high speech quality. Standard toll quality is difficult to understand in noisy environments such as department stores and railway stations, etc. By depressing a key, the controller feeds the ADPCM decoder with voice data previously stored in the RAM. After DIA conversion and filtering, it is amplified and output to the loudspeaker. The voice data RAM may be updated via telephone line, however, the calling station must support the same broadband ADPCM scheme.



7. Conclusion

A state of the art, broadband ADPCM coding device, the μPD77C30, has been introduced. Whenever speech quality far above the standard toll quality is required, a design with μPD77C30 offers an elegant and compact solution which reduces the number of chips greatly (to one or two μPD77C30's). Availability and simple interface are another important advantage. A design with the μPD77C30 may be realized with speed and ease.

Technological progress continuous. NEC currently offers a complete Firmware package as a fast full duplex ADPCM algorithm for their DSP μPD77C20 in source code. This speech coder reduces the amount of chips by one half without noticable loss of speech quality. This algorithm may also be implemented onto the μPD77C25 which runs with doubled speech capability and drives two full duplex speech channels simultaneously.

8. Literature

1. M. Bellanger: "Digital Processing of Signals Theory and Practice"

2. P. Kahl: "ISDN das künftige Fernmeldenetz der Deutschen Bundespost", R. v. Deckers Verlag, G. Schneck, Heidelberg 1985

3. T. Nishitani, S. Aiko, T. Arasaki, K. Ozawa, R. Maruta: A 32 kb/s Toll Quality ADPCM Codec using a Single Chip Signal Processor

4. Telecommunication Speech A/D Converter Components, Data Book NEC Electronics (Europe) GmbH

5. K. Grohe: Stand Alone Full Duplex Analog to ADPCM Interface Using Speech Encoder Devices μPD7730/1, NEC Application News HCP 11

Part C
Technical Letters

Part C
Technical Papers

Table of Contents

Part C Technical Letters

Digital Signal Processor
μPD77230

4. • DI-Instruction

   IEEE Format Conversion

5. • Pipelining with Development Tools

6. • Interrupts and Pipelining

μPD77230
Interrupt and IEEE Conversion Limitations

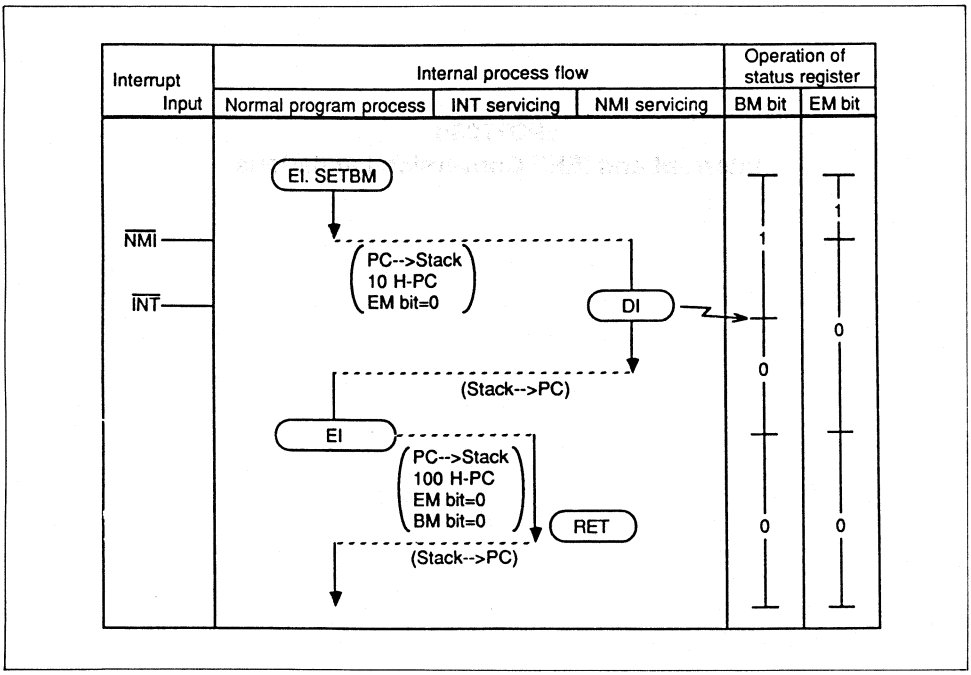
• 77230 "DI" Instruction

PROBLEM:

The "DI" instruction always clears the booked interrupt, regardless of the previous condition (EL = 1 or EL = 0). So if an interrupt occurs during the execution cycle of a "DI" instruction, the interrupt is memorized, but then (unfortunately) cleared!

In orther words, an interrupt that occurs during the execution of a "DI" instruction is guaranteed to be lost.

Example of "DI" Serving



Solutions:

A: Monitor "DI" instruction whenever the loss of an interrupt is critical.

B: Change your hardware so that every interrupt is signalled by two interrupt bursts. (At least one should be detected). Therefore, the "DI" instruction can be used.

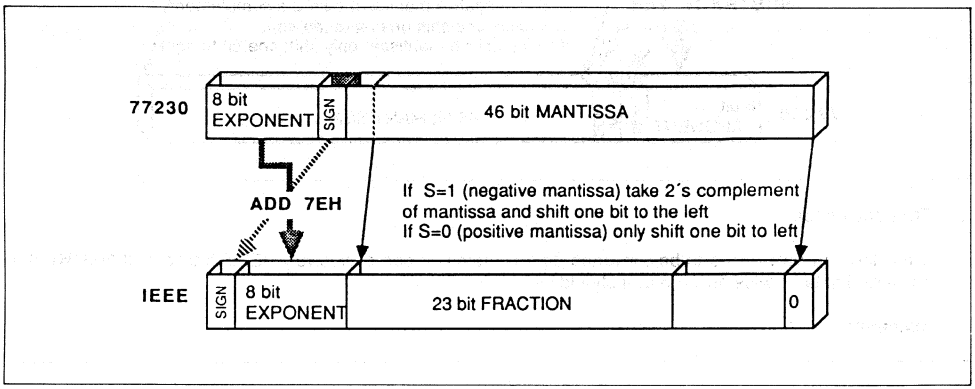
C: If some losses of interrupts are acceptable, every "DI" instruction should be initiated by a "EI" instruction. In other words, before executing the "DI" instruction, presently booked interrupts will be carried out. Only interrupts occuring while the "DI" instruction is executed will be lost. The probability for that case might be acceptably small enough in some applications.

• Note on the Use of the μPD77230 IEEE Conversion Instruction

The CVT instruction of the μPD77230 uses an algorithm that produces an error in some exceptional cases. This applies to the conversion of ASP numbers to IEEE format and vice-versa. This note explains the error and suggests a one-instruction error compensation procedure.

1. 77230-IEEE Conversion

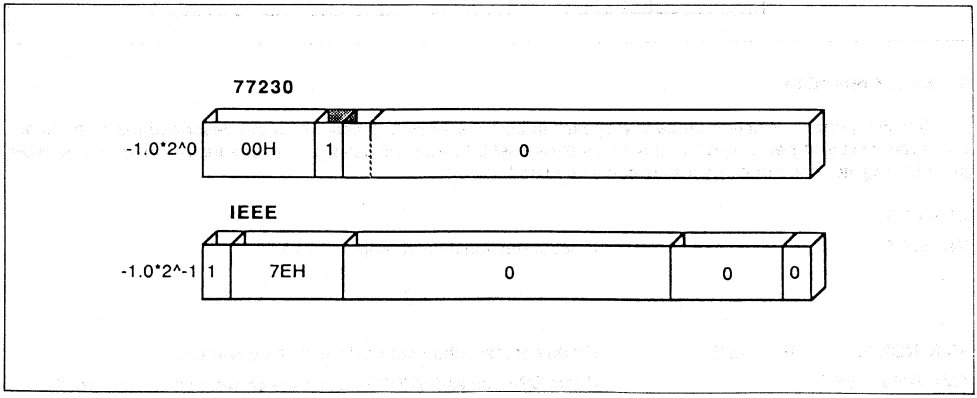
a) Conversion algorithm



b) Error occurence

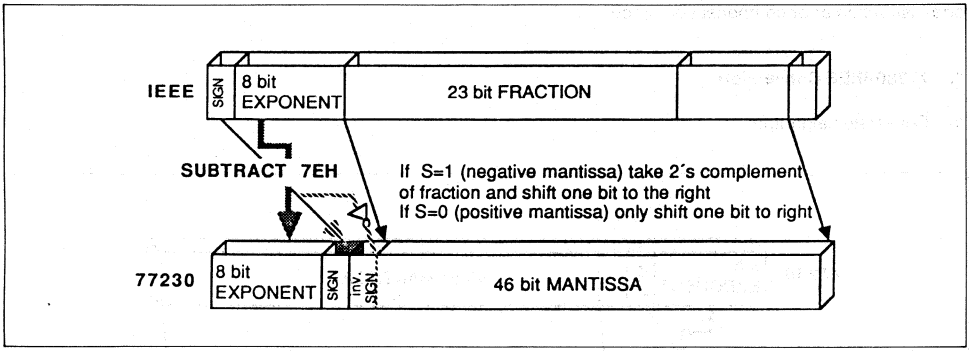
If the mantissa of the to-be-converted data is **—1** (47 bis ASP mantissa: **800000000000H**), the converted number is only half as big as it should be.

Example:



2. IEEE-77230 Conversion

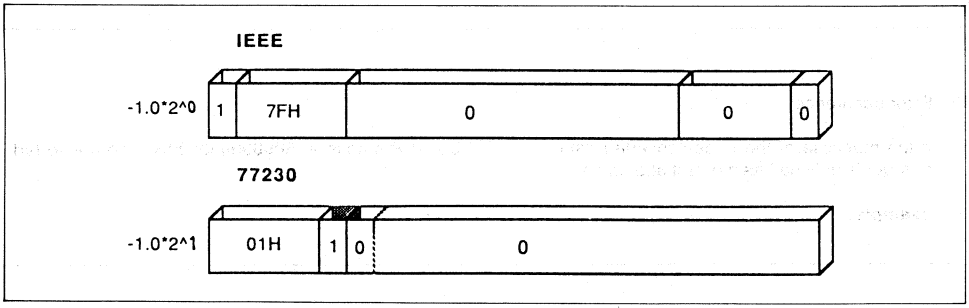
a) Conversion algorithm



b) Error occurence

If the fraction part of the to-be-converted data contains zeroes exclusively (IEEE fraction: **000000H**), the converted data is twice as large as it should be.

Example:



3. Error Correction

The following error correction procedure applies to both conversion directions. In the described case, the to-be-converted data is stored in WR0, and WR7 contains a set LSB which is used to compensate for the error. Any other possible register combinations may of course be used.

```
CLR WR7;
INC WR7;                          /* set correction data at program start */
  •
  •
  •
MOV NON,WR7   OR WR0,IB;          /* add compensation data before each conversion */
CVT WR0   IESP;                   /* use SPIE instead of IESP if opposite conversion is required */
```

4. Conversion Inaccuracies

The compensation procedure described above introduces a small inaccuracy in some cases, as a one is added to the LSB of to-be-converted data. This is however tolerable in the majority of cases.

a) 77230-IEEE conversion inaccuracy

— If the mantissa is possitive (S = 0), the conversion is accurate.

— If the mantissa is negative (S = 1) AND the lower 23 bits of the to-be-converted data (77230 format) are all zero, the conversion result shows an inaccuracy of 2^-23.

All other negative mantissas are converted accurately.

b) IEEE-77230 conversion inaccuracy

— If the IEEE number is positive (S = 0), the conversion is accurate.

— If the IEEE number is negative (S = 1), the conversion result shows an inaccuracy of 2^-46.

Note on the Pipeline Destruction using the Single Step Mode and Breakpoints for the 77230 Evakit and EB-77230 FFT

Introduction

The aim of the following note is to clarify the often misunderstood concept of real-time behaviour of the μPD77230 development tools regarding single-stepping and use of breakpoints. In a few cases the result do clash with true real-time behaviour, but this is consistent with the pipeline nature of the device and not a case of malfunction. In the following note the underlying principles will be summarized in order to give the design engineer a better understanding of what to expect and how to use the advantages of single-stepping and breakpoints.

The μPD77230 Pipeline

The μPD77230 uses a three stage pipelining technique to execute instructions. Various phases of execution overlap each other and are therefore mutually interconnected. Whenever a breakpoint is set, or the single-step mode introduced, the internal processor registers are read out. This task interrupts the normal operational flow because the pipeline is "cleared" with NOPs. These are automatically inserted and replace all following instructions. This is followed by a read-out of the internal processor status. Resuming the emulation may lead to false operational results in some cases. This is caused by the loss of adjacent instruction interconnection (as shown in three specific examples).

The pipeline can be characterised by three distinct phases:

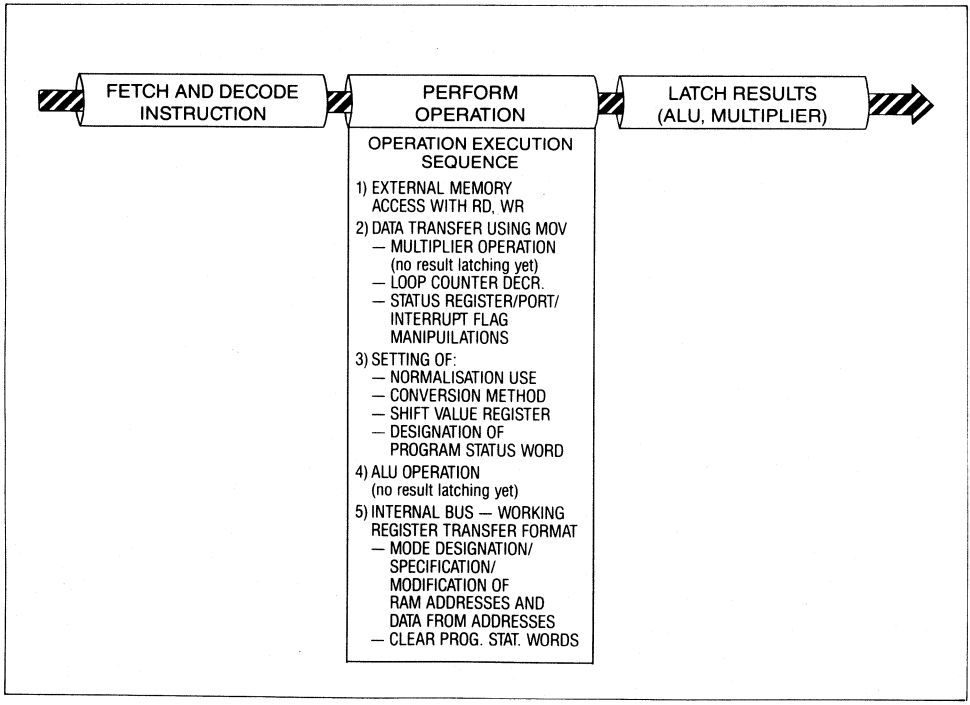


Figure 1. Pipeline Execution Sequence

True pipelining is achieved by overlapping the various execution phases:

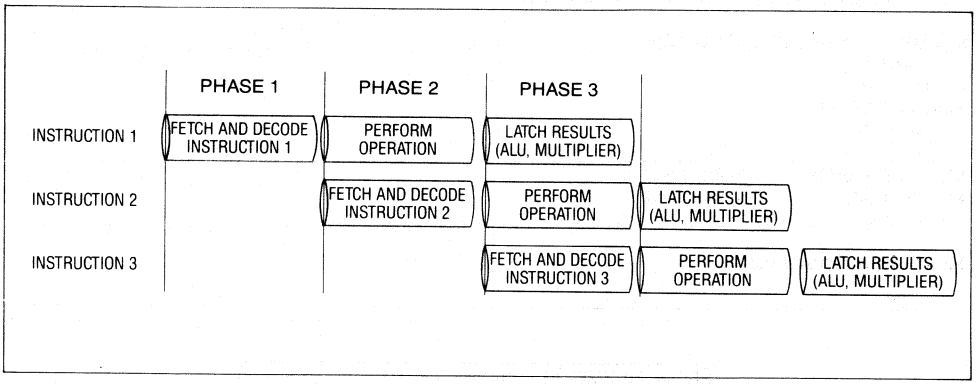


Figure 2. The Pipeline Technique

Pipeline Execution Overlap

In certain cases, the latching of data into a register which serves as a source register for a transfer operation in the same phase can occur. Is the data before or after the latching taken as the transfer data? If the ALU result of instruction 1 is latched to the same register which serves as a source register of a data transfer in instruction 2 (in a MOV instruction), the contents of the source register before being modified by the ALU result will be taken as the source. This is a typical case of a latch-move-overlap in the same pipeline phase. The general rule "**ALU result latch after MOVE transfer**" should be kept in mind.

Example:
```
LDI WR0,AAAAH;
LDI WR1,1111H;

MOV NON,WR0 ADDF WR1,IB;
MOV TR,WR1;
```

The result in the TR register is 1111H and NOT BBBBH.

Breakpoint/Single-Step: True Processor Status or not?

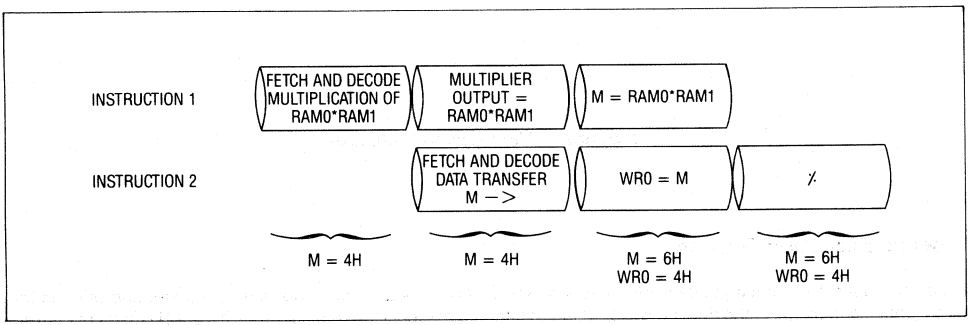
When a breakpoint is set at a particular instruction, a true replica of the internal processor status for that phase of execution is expected. So when a breakpoint is set at an ALU operation instruction, one would expect the program to halt directly after this phase for a status read-out. This would mean one would not see the result of the ALU operation according to Figures 1 and 2, as the program should have been stopped at phase 2. Unfortunately, the processor cannot be "halted", which means that every instruction which has been issued will be performed right to its end. All ensuring instructions are replaced by NOPs. So keep in mind that the true (real-time) processor status at a specific location does not correspond exactly with results seen on the screen. In fact, continuing emulation can lead to discrepancies compared with real-time behaviour. Three following examples explain this phenomena:

Pipeline Destruction Example Nr. 1:

The following instruction sequence is to be executed:

```
SPCBP0 CLRLBP0 SPCBP1 CLRBP1;
LDI RAM0,1H;
LDI RAM1,2H;
MOV LKR0,RAM1,
LDI RAM1,3H;

MOV LKR0,RAM1;      Instruction 1
MOV WR0,M;          Instruction 2
```

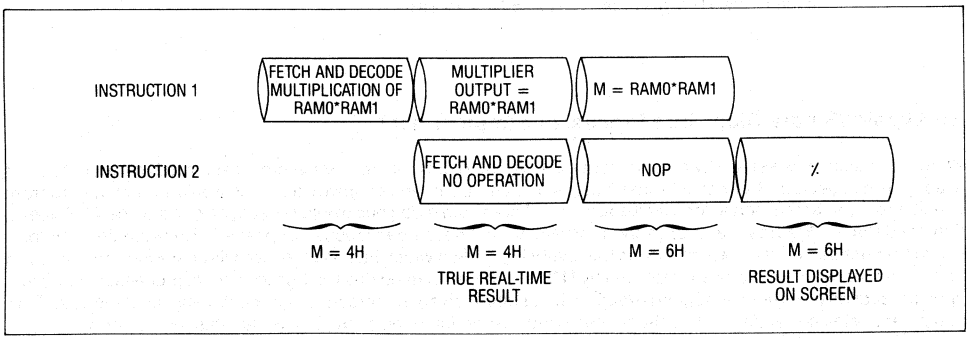
Real-Time Operation



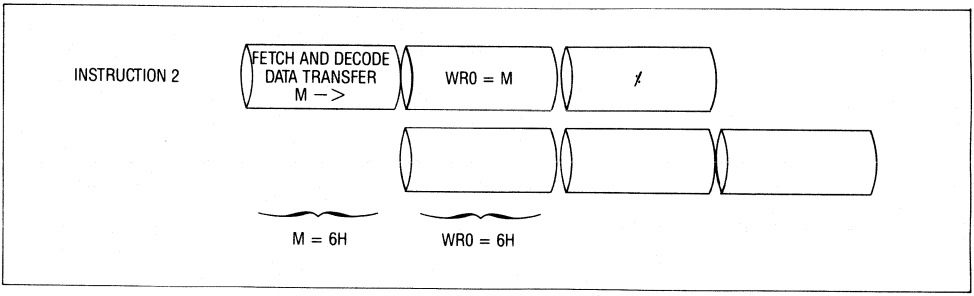
This is the typical case alluded to above (ALU result latch after MOVE transfer).

Now let's look at the sequence of events when a **breakpoint** is inserted at **Instruction 2**. Before reading out internal registers with a subroutine, the pipeline is cleared with NOPs following Instruction 2.

Breakpoint Operation:



After the registers have been read out and the internal processor condition restored, the program is resumed with Instruction 2:



Note the discrepancy in WR0 compared to real-time operation. The exact processor status at the point of interruption in program sequence cannot be restored when resuming the program at the breakpoint.

Pipeline Destruction Example Nr. 2:

As the actual execution of an instruction always takes place in the second phase of the pipeline, another instruction (the following one) will be fetched (and later executed) before the actual execution of the first one has taken place. So a **CALL**, or **BRANCH**, always performs one instruction more before the program actually branches. It is wise to insert a **NOP** after a BRANCH or CALL for an easy understanding of the program. However, the proficient programmer can insert a useful instruction, provided he knows its effect when placed at that location.

The useful instruction placed after a BRANCH or CALL is not executed in the single-step mode if the jump takes place (condition satisfied). When the program is stopped at a BRANCH or CALL, the following instructions are replaced by NOPs to clear the pipeline and the branch address is saved. When the program is resumed, the branch address is taken as the address for continuing the program. The instruction after the BRANCH or CALL is lost. Thus the program will run differently in real-time mode than it will in the single-step mode if an instruction other than a NOP is placed after a BRANCH or CALL.

Pipeline Destruction Example Nr. 3:

In normal real-time operation a **loop counter borrow** after the instruction DECLC means that the following instruction will be automatically replaced by a NOP and thus "skipped". However, in the single step mode the replacement with NOP coincides with the NOPs used to clear the pipeline before internal status read-out and is therefore permanently lost. On program resumption, the next address (being the one that should be replaced by the lost NOP) is falsely executed. The expected skip does not occur.

Conclusion

If these differences/restrictions are kept in mind, no problems will be encountered when implementing breakpoints or running a program in the single-step mode. As a matter of interest: it takes 8 instruction cycles, at the most, to clear the pipeline. This worst case instruction execution delay time occurs when the slow-speed external memory is being accessed and an interrupt arrieves at the same time.

Interrupts and Pipelining for the μPD77230

1. Introduction

Pipeline architecture requires special care from assembler programmers. Especially interrupts in conjunction with pipeline processing which must be treated carefully in order to avoid destroying the pipeline processing. No problems will occur if the programmer writes clean programs; keeping in the mind the architecture and pipelining of the μPD77230. Some examples will give an idea how to write clean programs.

2. A Closer Look at the Architecture

In contrast to the former μPD7720, the μPD77230 uses the 3 stage instruction pipeline completely. The stages are as follows:

— program counter modification

— instruction fetch and decode

— instruction execution

The function of the data pipe becomes more understandable if a slightly modified block diagram of the μPD77230 is used. Figure 1 shows the internal structure of the μPD77230 from the view point of a system-programmer, with clear separation of addresses, data and instructions.

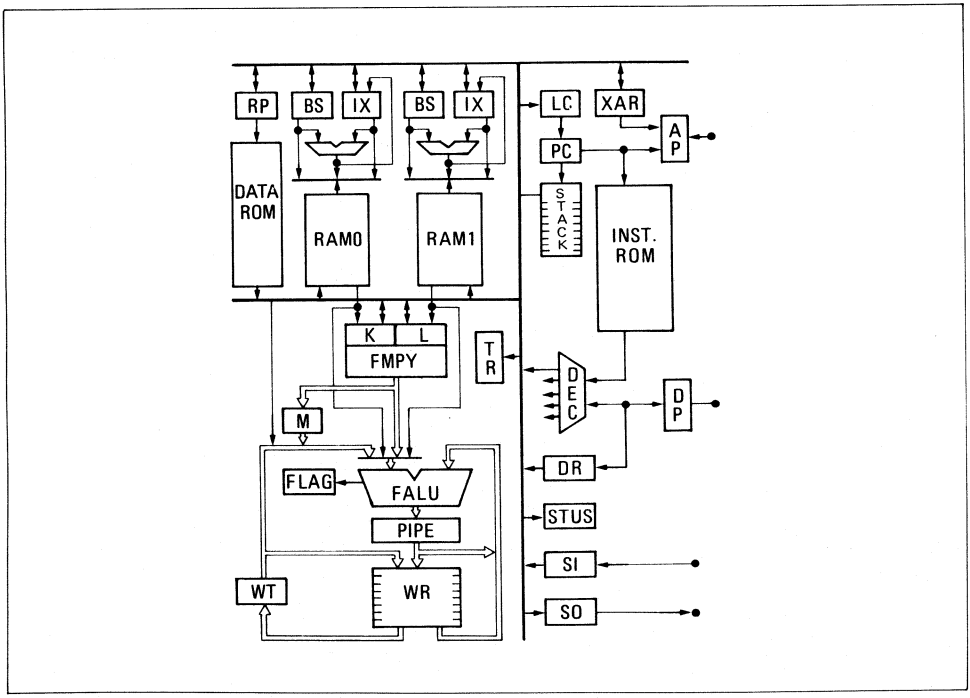


Figure 1. Internal Structure of μPD77230 from the Viewpoint of the System-Programmer

Typical signal-processing algorithms exist from a recursive calculation of mathematical terms, such as "ax + b", in other words of a multiplication and an accumulation. The aim of a chip designer to do both calculations at the same time in parallel and not to waste time storing intermediate results. Both criterias are implemented in the μPD77230. After register K and L have been set up by a double data transfer within one instruction cycle, the result of the multiplication is given to the floating point alu (FALU) at the beginning of the next instruction without any intermediate storage. The multiplier works continuously. That means even if no data transfer to registers K and L happened, the product K · L is available at the output. At the end of the second instruction, the register PIPE contains the term ax + b. PIPE is a virtuel register inserted into the block diagram for better understandability of the data-pipe. Intermediate results are "stored" in PIPE, but exist only latently.

They are used as a source for the following alu operation, without intermediate storage. PIPE can be any register of the 8 working registers WR. After computation of all intermediate results of a pipeline task, ands one additional instruction cycle, the selected working register contains the final result. This implies that every intermediate FALU result going through PIPE will occur one instruction cycle later in the selected working register.

3. A Clean Program and Interrupts

A clean program means that it is interruptable at any time without pipeline struction. Or in other words, intermediate results are neither lost, nor overwritten. Figure 2 shows an example of a recursive filter tap (biquad) that can be interrupted at any time.

Step No.	Mnemonic			Comment
1	MOV	LKR0,	ROM	K ← W_{n-1} L ← $-B_1$
	CKR	WR1		WR1 clear
	INCRP			RP = 1H
	INCBP0		;	BP0 = 1H
2	ADDF	WR0,	M	WR0 = $x_n - B_1 \cdot W_{n-1}$
	MOV	LRK0,	ROM	K ← W_{n-2} L ← $-B_2$
	INCRP		;	RP = 2H
3	ADDF	WR0,	M	WR0 = $x_n - B_1 \cdot W_{n-1} - B_2 \cdot W_{n-2}$
	MOV	LKR0,	ROM	K ← W_{n-2} L ← A_2
	INCRP			RP = 3H
	DECBP0		;	BP0 = 0H
4	ADDF	WR1,	M	WR1 = $A_2 \cdot W_{n-2}$
	MOV	LKR0,	ROM ;	K ← W_{n-1} L ← A_1
4	ADDF	WR1,	IB	WR1 = WR0 (= W_n = New W_{n-1}) + $A_2 \cdot W_{n-2}$
	MOV	RAM0,	WR0	RAM0 ← New W_{n-1}
	INCRP			RP = 4H (for next stage)
	INCBP0		;	BP0 = 1H
6	ADDF	WR1,	M	WR1 = $W_n + A_1 \cdot W_{n-1} + A_2 \cdot W_{n-2}$
	MOV	RAM0,	K	RAM0 ← New W_{n-2}
	INCBP0		;	BP0 = 2H (for next stage)

Figure 2. Biquad Filter Tap

Note that each instruction is linked to its successor by two (or at least one) intermediate value which only exist latenly and are not stored for its successor.

Supose at execution time of this program an interrupt occurs so that step number 2 is still executed before the execution of the interrupt service routine starts. The first interrupt service routine should be a very simple:

```
interrupt:  RET
            NOP
```

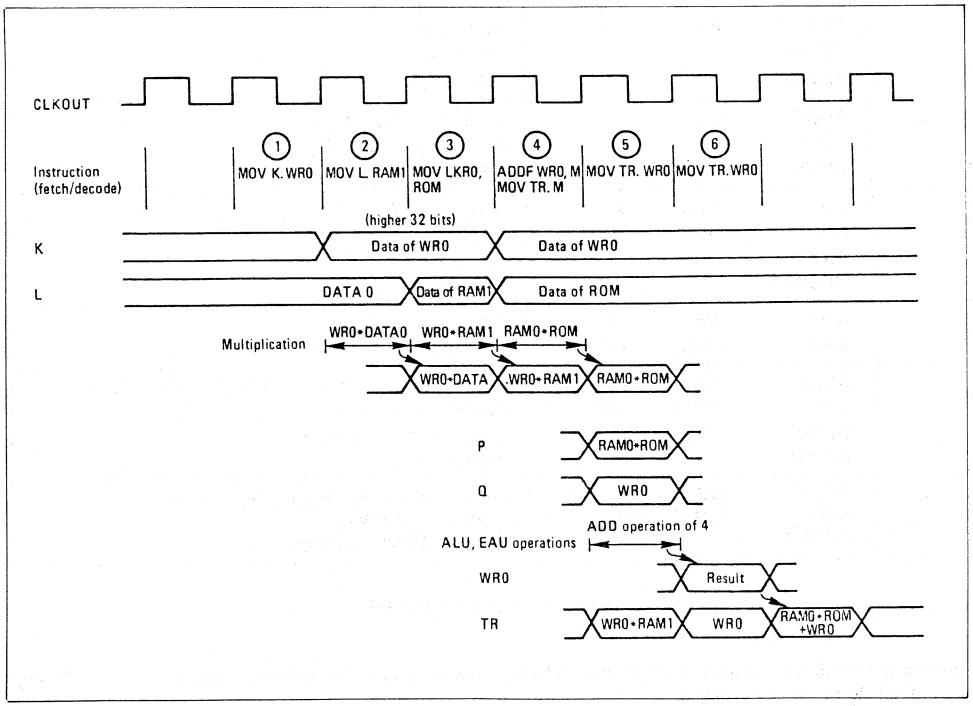
After interruption the pipeline becomes empty. As a consequence, the intermediate results of step number 2 from FMPY and PIPE are stored into registers M and WR0. When the interrupt service routine has been completed, the pipeline restarts with step number 3, but as it has been interrupted, it does not use intermediate values and instead it uses the values stored in register M and WR0. These values are the exact values computed right before the pipeline was been interrupted. No results are lost. The program is therefore interruptable.

Supose the interrupt service routine is somewhat more complicated (i.e., involving multiplier and FALU). In this case cause registers M and WR0 must be pushed onto a heap within the interrupt service routine in order not to loose the results. Before leaving the interrupt service, routine M and WR0 must be recovered again.

4. A Trick and its Restrictions

In some cases, it may be useful to keep intermediate results for later calculations. The pipeline allows to put those intermediate results onto the heap right after (!!) the ongoing computation, with the restriction that such a sequence may not be interrupted. Whenever programmers are using this feature they must be careful of interrupts.

The following pipeline shows two cases related to register M and PIPE (which is WR0) that are not interruptable.



Interrupt between step 3 and step 4 (register M):

Steps 1 through 3 are affecting the multiplier. Step 4 saves the intermediate multiplication result of step 2 to TR. Now, whenever an interrupt occurs taking place at the end of step 3, the intermediate multiplication result of step 2 gets because of the following multiplication at step 3. This value is now stored into TR erronously.

Interrupt between step 4 and 5 (PIPE):

Step 5 saves the value of WR0 of step 1 to TR. Now whenever an interrupt occurs taking place at the end of step 4 the intermediate result of WR0 of step 1 gets lost because of the following addition of step 4 and this value is stored into TR erroneously.

5. Conclusion

It was shown that interrupts and pipelining normally do not conflict. However, two exceptions cause a destruction of the pipeline. One: Whenever an alu or multiply operation precedes a transfer operation. Two: when the destination of the alu/multiply operation equates the source of the transfer operation the pipeline is destructed. In these cases, interrupt must be disabled before executing such a sequence.

Part D
Application News

Table of Contents

Part D Application News

μPD8085 Emulation Restrictions on V20—V50 Microprocessors

The μPD8080 emulation capability of NEC's V-Series standard microprocessors V20—V50, often raises questions about possibility of emulating the μPD8085. This paper outlines the major limitations which should be considered for a possible μPD8085 emulaton.

Contents:

Author: Shyam Gupta
Application Department
NEC Electronics (Europe) GmbH

1. General

The V-Series microprocessors V20, V30, V40, V50 have the capability to emulate the 8-bit microprocessor μPD8080. This emulation feature allows a large software base already existing for the μPD8080 to be executed in 16-bit environment to further improve the performance of the existing software package.

As the μPD8080 and the μPD8085 (a later member of the 8-bit generation) are similar in architecture and the instruction set is nearly the same. Questions are often asked whether it is possible to emulate the μPD8085 on the above mentioned 16-bit V-Series microprocessors as well?

In the following a short description of the dissimilarities between the two microprocessors, the μPD8080 and the μPD8085, is given. This description is an attempt to outline the bottlenecks with μPD8085 emulation on the V-Series microprocessors.

2. μPD8085 Instruction Set

The μPD8085 instruction set is a super set of the μPD8080 instruction set. Except for the extra instructions supported by the μPD8085, the instructions of the two processors are compatible at machine level (atleast at the first sight).

The extra instructions offered by the μPD8085 are:

RIM: Read interrupt mask

SIM: Set interrupt mask

The RIM instruction in the μPD8085 is used for reading the interrupt mask and to read the serial I/O. Further, after a TRAP, it may be necessary to execute the RIM instruction to preserve the status of the IE (Interrupt enable).

The SIM instruction in the μPD8085 is used for setting the interrupt mask as well as to output the serial data.

These RIM and the SIM instructions are not supported by the μPD8080 microprocessor and therefore also not on V-Series microprocessors which can only emulate the μPD8080.

3. Flag Operations

Apart from the two instructions which are different between the two microprocessors, there are certain instructions which although have the same function but treat the flags in a different manner. This is especially true for the AC-flag (Auxillary carry flag).

Following is a list of four instructions where this is true:

Instruction	AC-flag μPD8080	AC-Flag μPD8085
ANA r	*	1
ANA M	*	1
ANI Data	0	1
DATA	?	?

Note(s): r = 8 bit register, M = Memory location
0 = Flag cleared
1 = Flag set
* = Flag affected
? = Flag affected but not necessarily in the same manner for both processors

For normal operation the user may not notice the difference but if after execution of these instructions some follow-up action is desired, the results may be catastrophic.

4. Summary

The V-Series 16-bit microprocessors V20, V30, V40, and V50 can emulate the μPD8080 microprocessor without any limitations, but if an emulation of μPD8085 microprocessor is required the instructions and flag operation should be taken into account. If the user application can make sure that the above mentioned restrictions are not violated, a limited μPD8085 emulation on V-Series standard microprocessor should be possible.

Stand-by Mode for V-Series Standard Peripherals

One advantage of the V-Series standard peripherals is the stand-by mode, which allows a low power consuming state. This document shows the ease of using the mode for the different device types.

Contents:

Author: Rolf Elter
Application Department
NEC Electronics (Europe) GmbH

1. Stand-by Mode at the μPD71051

The stand-by mode may be entered by a hardware or a software reset. This will switch off the feeding of external input clocks, like CLK, TxCLK and RxCLK.

1.1 Entering Stand-by

There are two methods for entering the stand-by mode at the μPD71051:

- With a hardware reset
  The device will enter than the stand-by mode by the falling edge of the Reset signal.
- With a software reset
  Software reset is achieved by writing the appropiate command word. After the rising edge of the WR-signal stand-by mode is entered.

1.2 Releasing Stand-by

Writing a mode word is the only way to escape from the stand-by. The device will go to a normal operation after the rising edge of the WR-signal.

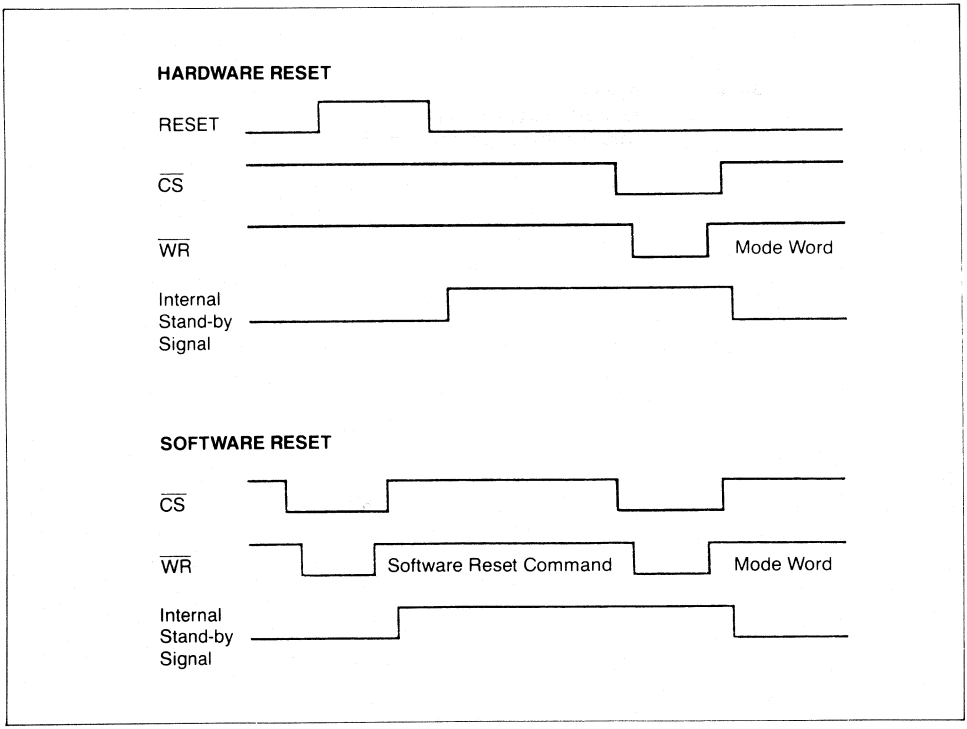


Figure 1. Timing Diagram for μPD71051 Stand-by Function Timing

2. Stand-by Conditions

Contrary to the μPD71051, all other devices of the V-Series standard peripherals have only one possibility to reach stand-by mode. A special level condition on the input pins results in a lower power consumption.

- μPD71054

  All inputs must be on high level and also the CS-signal.

- μPD71055

  The same conditions as mentioned for the μPD71054, but additionally the RESET-pin must be held on low level.

- μPD71059

  The same conditions as mentioned for the μPD71054, but additionally the pins INT0 to INT7 must be kept low.

On-Chip AD Converter of μPD7811/C11/C14

The μPD78C11 On-Chip 8 Bit AD Converter is one of the most interesting peripherals on this microcomputer. In order to guarantee its function and accuracy the designer should observe certain do's and don'ts which are described here.

Contents:

Author: Heiner Tendyck
Application Department
NEC Electronics (Europe) GmbH

1. AD Converter μPD7811/C11 H/W-Design-Rules

The on-chip μPD7811/C11 AD converter must be used with regard to the electrical specifications.

Details:

a) The specification for the reference voltage:

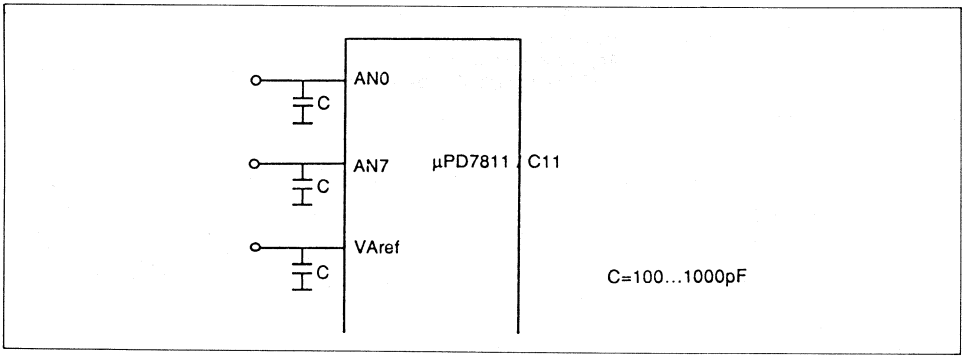
AV_{CC} −0.5V ≦ V_{AVref} ≦ AV_{CC} and AV_{CC} = V_{CC} — for μPD7810/11(H) (linearity 0.6 % ± 1/2 LSB)

4.0V ≦ V_{AVref} ≦ AV_{DD} and V_{DD} −0.5V ≦ AV_{DD} ≦ V_{DD} — for μPD78C10/C11/C14 (linearity 0.6 % ± 1/2 LSB)

3.4V ≦ V_{AVref} ≦ AV_{DD} and V_{DD} −0.5V ≦ AV_{DD} ≦ V_{DD} — for μPD78C10/C11/C14 (linearity 0.8 % ± 1/2 LSB)

b) V_{SS} has to be grounded. The analog and digital ground (V_{SS} and AV_{SS}) and power supply (V_{CC} and AV_{CC}) are separated in order to improve noise immunity of the device.

c) NEC recommends the use of decoupling capacitors of 100—1000pF from every analog input pin (including V_{AVref}) to ground.



d) NEC recommends to apply external Zener diodes to the analog input pins in order to prevent too high/low input voltages. The allowed input range voltage is

$$-0.5 \leqq V_{An} \leqq V_{AVref} + 0.4V$$

e) Timing of AD Conversion

Following is the timing specification of the on-chip AD converter:

— Sampling Time	12(16)x2x3 t_{cyc}
— Wait Time	12(16)x1x3 t_{cyc}
— Conversion Time	12(16)x8x3 t_{cyc}
—	12(16)x1x3 t_{cyc}
	432(576) t_{cyc}

FR-Bit = 1 => 12 ; FR Bit = 0 => 16

FR-Bit = 1 => t_{conv} = 48 μsec ; f_{osc}(max) = 9 MHz

FR-Bit = 0 => t_{conv} = 48 μsec ; f_{osc}(max) = 12 MHz

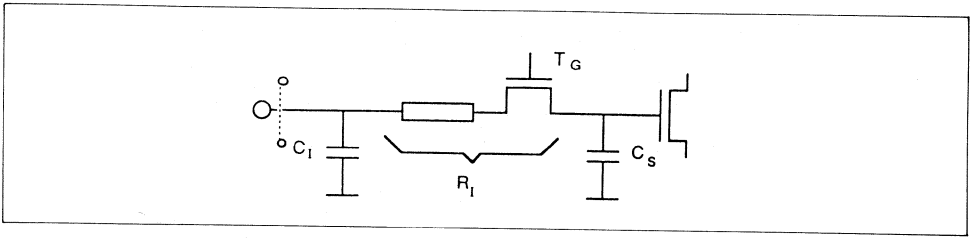
f) Analog Pin Input Impedance R_I

The AD conversion is based on a sample and hold circuit. At the start of conversion the sampling of the analog line is performed for the a.m. time. During this time the analog AC impedance is reduced to about R_I = 10 KΩ.

In all other cases: R_I = 1000 MΩ (nearly no leakage current)

DC impedance stays at: 1000 MΩ

Input Circuit of AD Converter:



C_I ≈ 10pF max

C_S = 10pF max

T_G = Sampling Transistor

g) The μPD78C11/C14 features a STOP Mode to reduce the power supply current at V_{DD} heavily. This is very useful for battery driven systems.

But: The supply current to V_{Aref} is 1.5 mA (typ) also during STOP mode.

In order to prevent the I_{Aref} it is recommended to cut off the supply V_{Aref}. The pin 42 should be open during STOP mode.

If one grounds V_{Aref} during STOP mode no voltage should be applied on the analog inputs AN0—AN7.

2. AD Converter μPD7811/C11 S/W-Design-Guidlines

a) The ANM register can be used to trigger the AD conversion:

After write operation to ANM register, a new conversion period is induced. Using a software time loop it is possible to read the first conversion result after 48 μsec (min), instead of waiting for INTAD occuring after 4x48 = 192 μsec (min).

b) If the analog input voltage is below AV_{SS} = 0V, the AD converter's result will be erroneous.

If the analog input voltage is V_{Aref}, the AD converter's result will be 0FFH.

Easy Interfacing of μPD7508 with μPD7756

A simple demonstration is showing the easy interfacing between an μCOM75 Microcomputer with an μPD7756. To improve the overall impression, the demonstration is implemented in a toy-telefon where the messages stored in the μPD7756 are selectable by the telefon's keyboard.

Contents:

Author:

Wilfred Noll
Application Department
NEC Electronics (Europe) GmbH

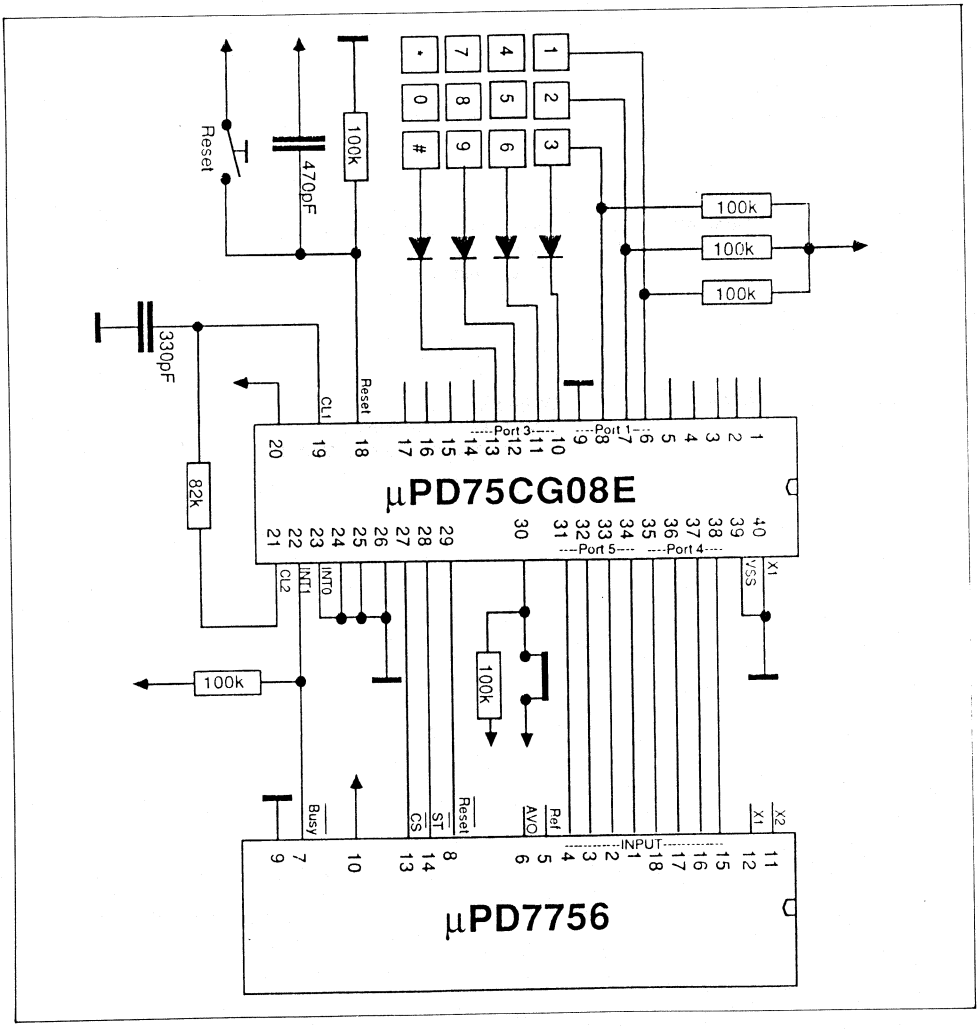
1. Features

Outline: Telephone set

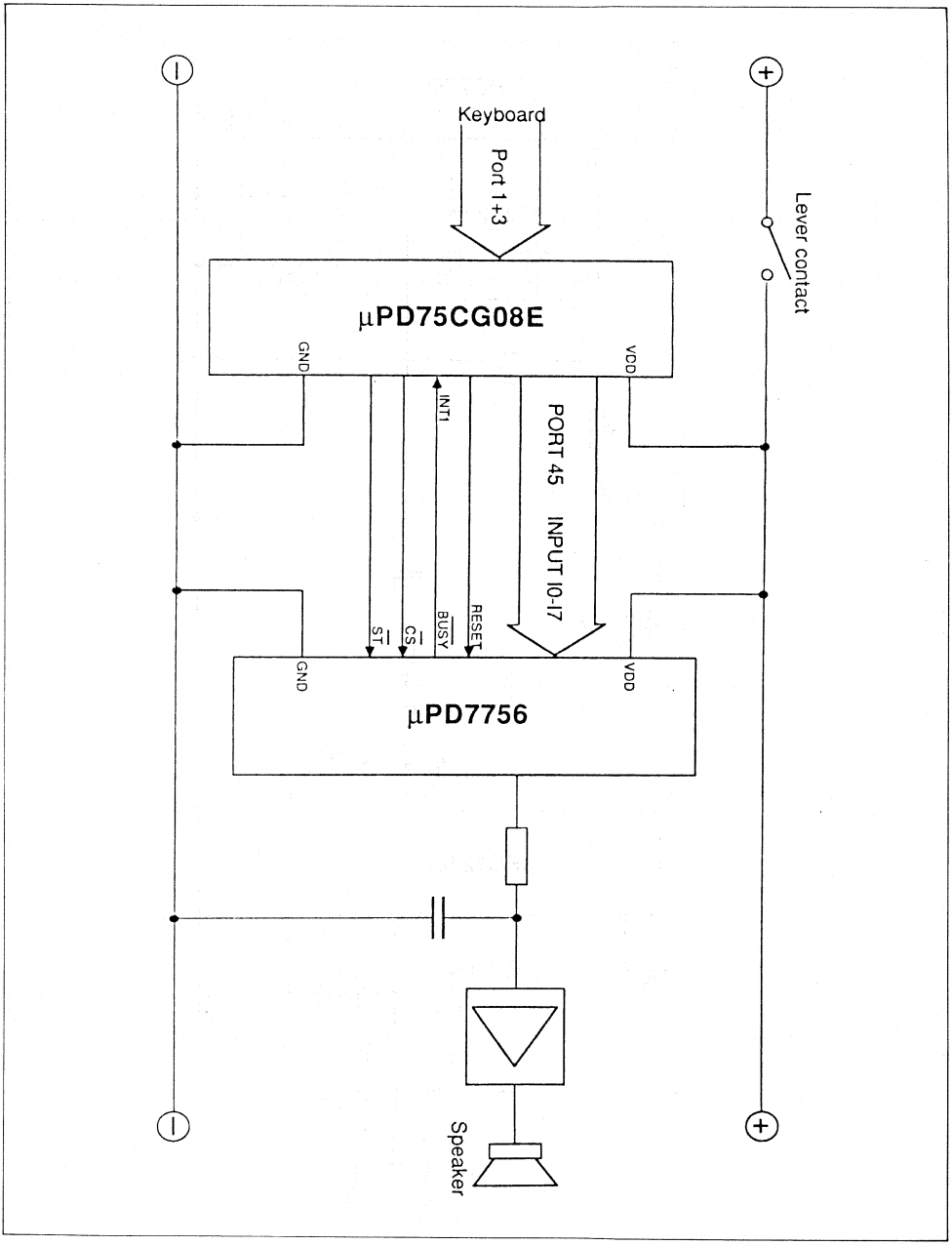
Function: After picking up the phone, "May I help you" sounds out of the speaker. Via the keyboard, several quality levels of speach synthesis can be selected. (Same as available for Demobox.)

When opening the telephone box, special test mode for watching stand-by mode of μPD7756 as well as controlling the analog output, can be selected by jumper setting.

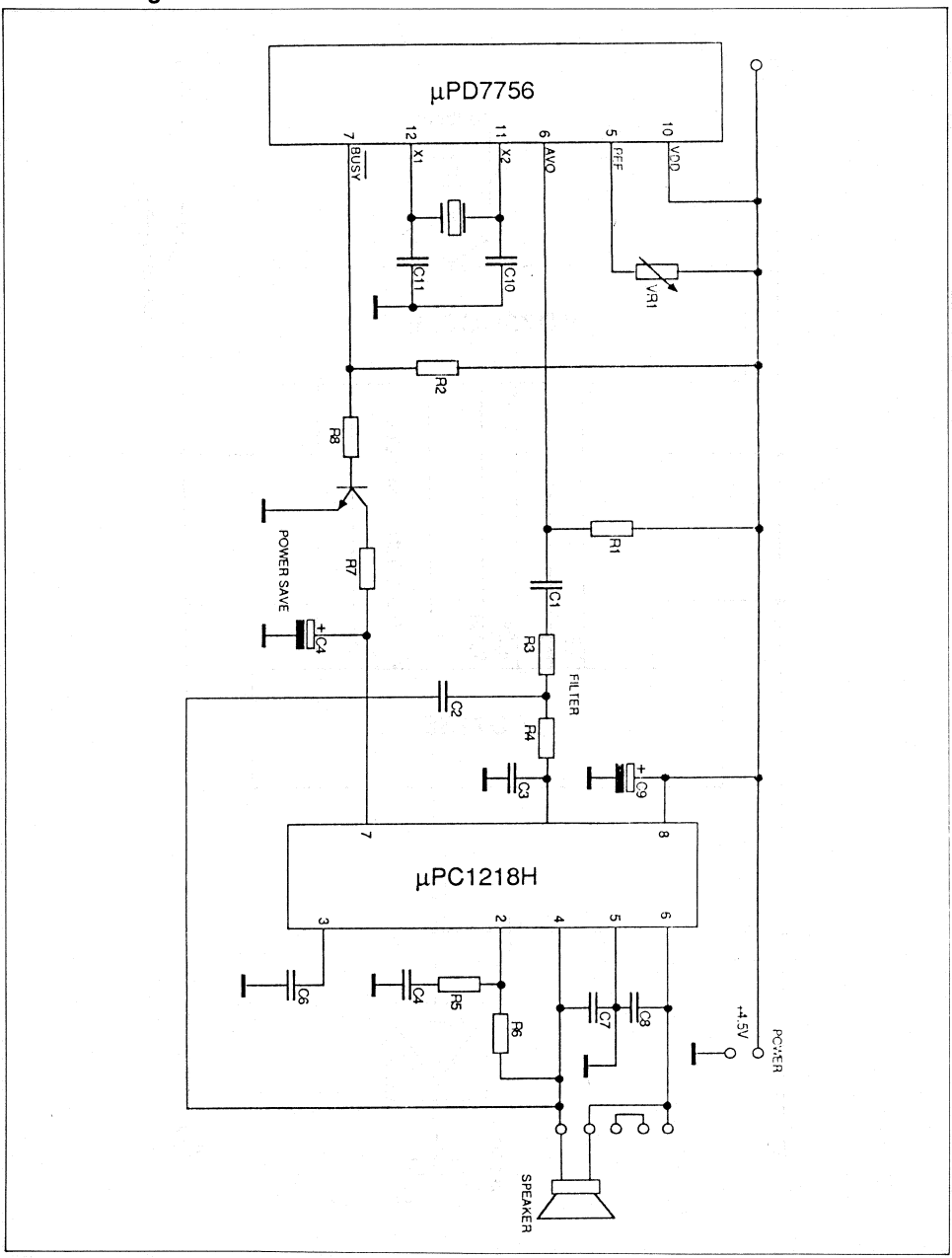
Purpose: This Demoset was made specially for exhibitions and customer demonstrations.



2. Block Diagram



3. Circuit Diagram



μPD7527/37 as Low Cost FIP Driver

μPD7527/37 is the right alternative to drive the alphanumeric FIP's. To highlight μPD7527/37A capabilities, a demo-software was made to convince design engineers in using our four bit microcomputers.

Contents:

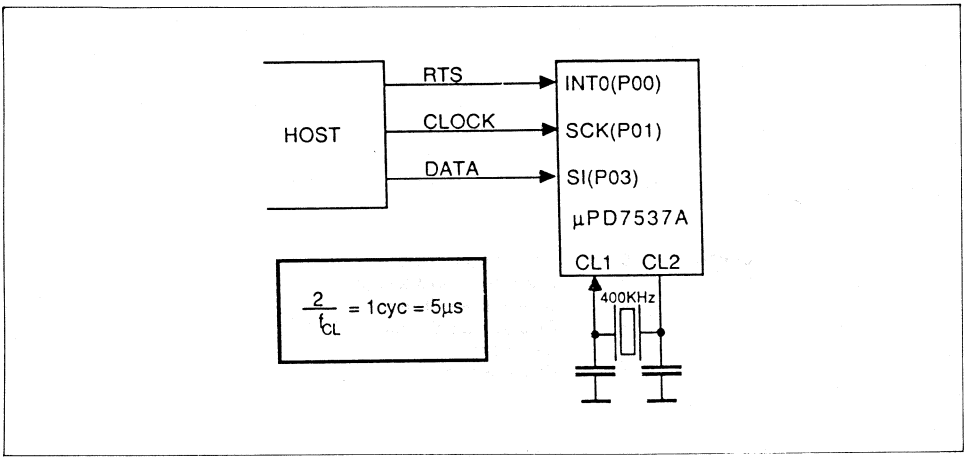
Author: Wilfred Noll
Application Department
NEC Electronics (Europe) GmbH

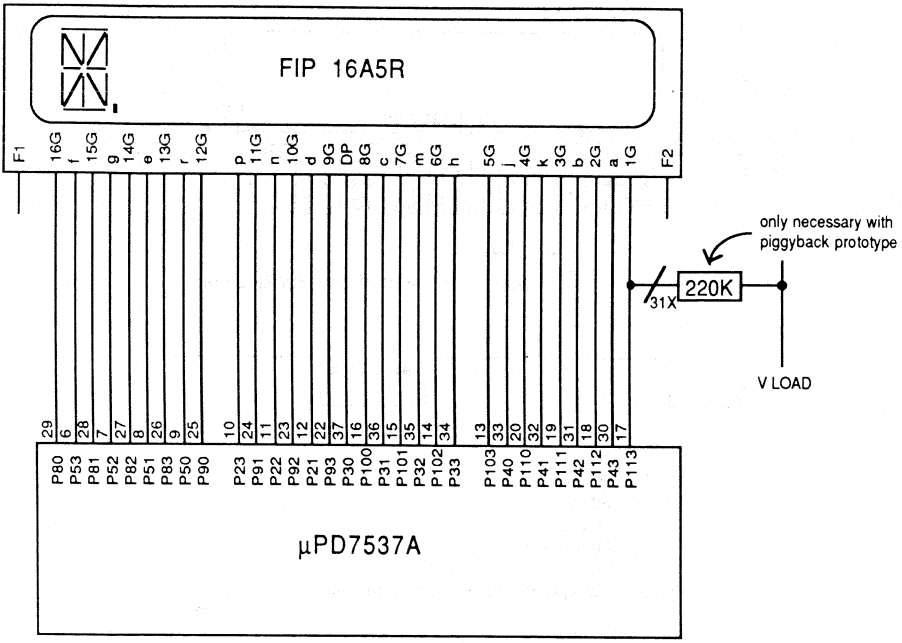
FIP16A5R Driver/Controller Program for μPD7537A

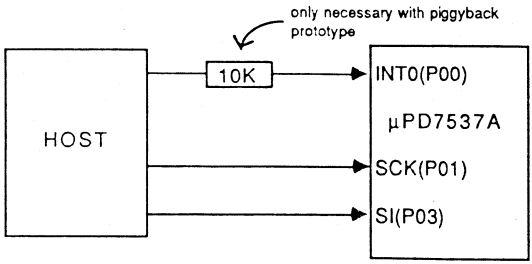
0. Introduction

The software at hand is based on the usage of a 400 kHz ceramic resonator. However, it is possible to choose another system frequency in accordance with specification of the data sheet or to use an external system clock in connection with a software-compatible μPD7527A.

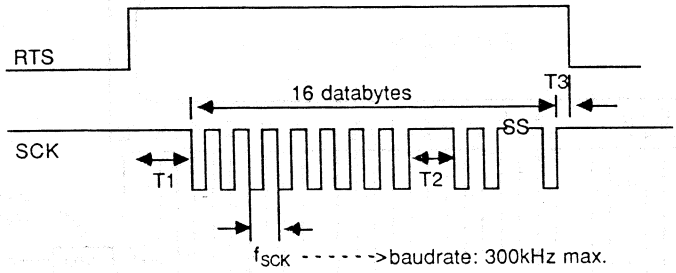
1. Interface to Host System

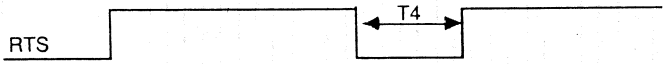



FIP 16A5R
F1
16G
f
15G
g
14G
e
13G
r
12G
p
11G
n
10G
d
9G
DP
8G
c
7G
m
6G
h
5G
j
4G
k
3G
b
2G
a
1G
F2
only necessary with piggyback prototype
220K
31X
V LOAD
29
6
28
7
27
8
26
9
25
10
24
11
23
12
22
37
16
36
15
35
14
34
13
33
20
32
19
31
18
30
17
P80
P53
P81
P52
P82
P51
P83
P50
P90
P23
P91
P22
P92
P21
P93
P30
P100
P31
P101
P32
P102
P33
P103
P40
P110
P41
P111
P42
P112
P43
P113
µPD7537A


only necessary with piggyback prototype
10K
HOST
INT0(P00)
µPD7537A
SCK(P01)
SI(P03)

2. Data Transfer (Timing)



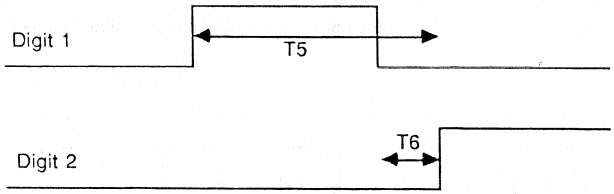


T1 > 60 cyc
T2 > 60 cyc
T3 > 0 cyc
T4 > 800 cyc + (800 cyc / T5) 40 cyc

For 400 kHz systemclock the following values are resulting:

T1 > 300 μs
T2 > 300 μs
T3 > 0 μs
T4 > 6 ms

3. Display Multiplexing



T5 = (n+1)*2cyc T6 = 20 cyc

n = Value of the Timer Modulo Registers (at present 33 hex = 520 μs with 400 kHz)

Due to modification please change as follows:

ADDR 68 Hex 13 low nible (at present 3)
ADDR 6A Hex 13 high nible (at present 3)

Concerning the example, following values are resulting for Display Multiplexing:

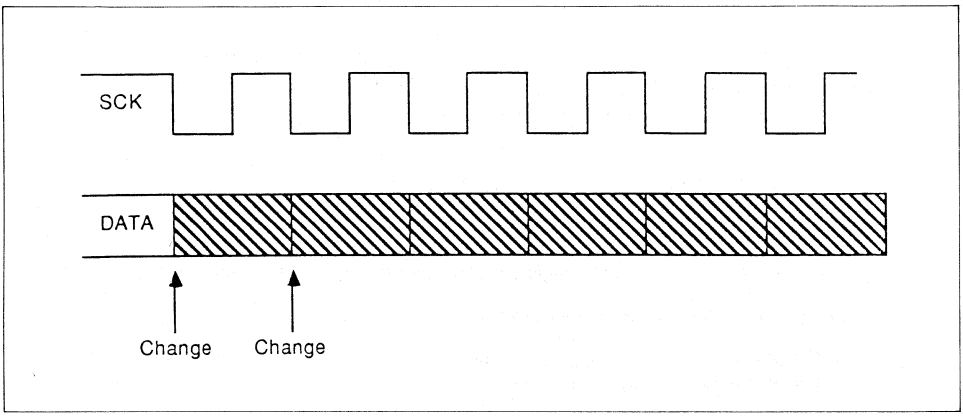
Repeat Frequency: 120 Hz
Blanking Time: 100 μs
Display On Time: 420 μs
Duty: 1/20

4. Data Format

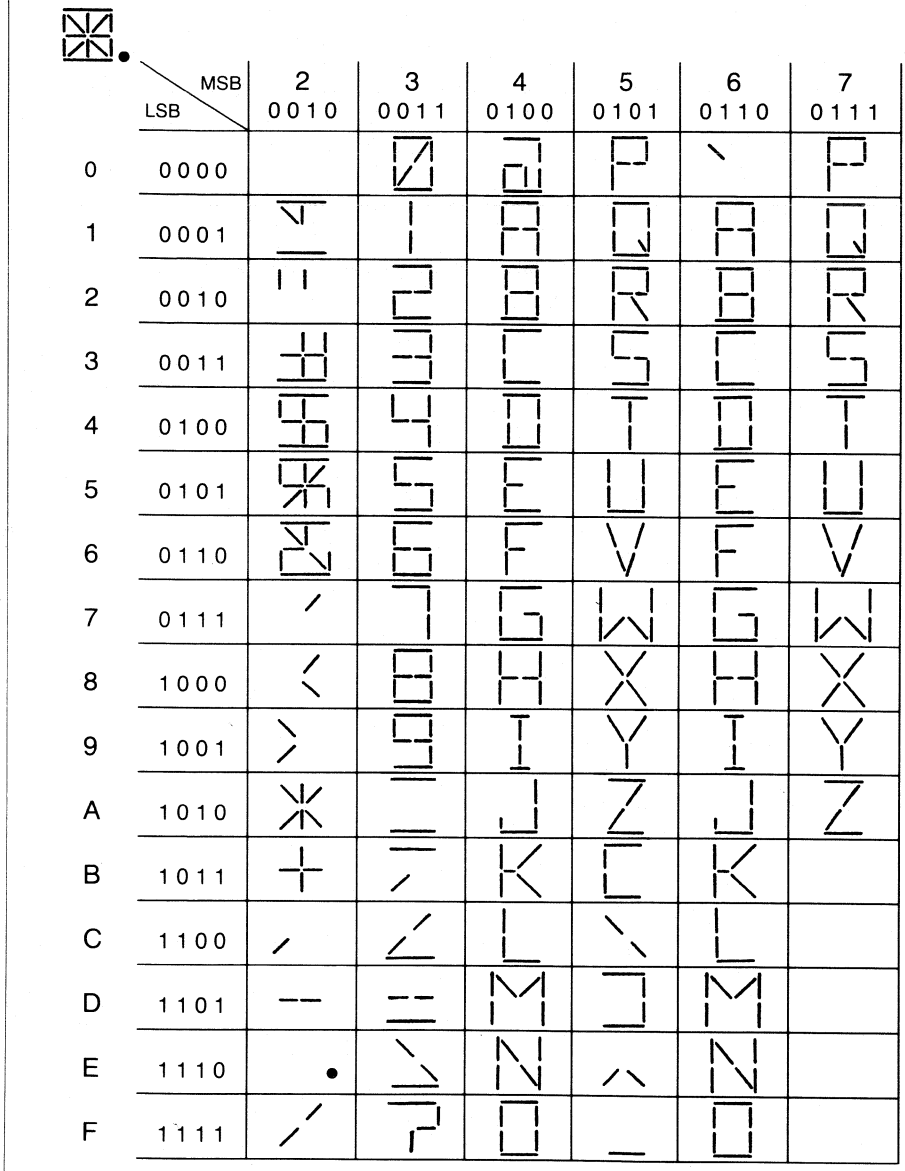
Serial data: MSB First, 8 Bit only

Symbols: a) ASCII, as shown in the enclosed table

b) MSB = 1 —> ASCII, as shown in the enclosed table plus decimalpoint



5. Character Set



Direct LED Drive with μPD7507/08

Although μPD7507/08 4-Bit mocrocomputers are not designed for direct LED drive, there is a possibility of direct connection for a limited number of LEDs. Needless to say that an advanced multiplex procedure optimizes the quality of the display.

Contents:

Author:

Wilfred Noll
Application Department
NEC Electronics (Europe) GmbH

1. Features of Direct LED Drive

As pointed out in the absolute maximum ratings of the concerning devices, the total output current for low output voltage is limited to 25 mA per port group. Since there are two port groups available, total output current in this operation mode is limited to 50 mA.

Assumed that a two digit LED display is required, the available current has to be distributed over 14 LEDs. The standard LED current is around 10 mA, making a multiplexing of the LEDs necessary.

Two different ways of multiplexing are discussed below. Depending upon the LED characteristics, the designer may choose his optimum solution.

2. Hardware Description

To evaluate both multiplex modes, a demoboard and a demosoftware is necessary. In below discussed example, there is the Piggyback version of the μPD7507/08, the μPD75CG08E, mounted on a small board. The multiplex frequency is controlled via a 4 bit Hex-switch, so that in total 16 different multiplex frequencies can be selected by the switch. A wired jumpers is selecting the multiplex mode, and although there is a power up reset circuit, program may be restart again when pressing the RESET key. The RC-oscillator circuit is supplying the systemclock and the crystall oscillator circuit is necessary to generate the exact multiplex frequencies. Fourteen LEDs organised in two digits are controlled digitwise via the ports P20 and P21 (using PNP-transistors) and segmentwise via the ports P30 to P42. Due to different multiplex modes it is necessary to make the resistors, limiting the LED current, exchangeable.

3. Operation

Set the multiplex mode with the jumper. If the jumper is not set, mode 1 (parallel mode) is selected, otherwise the sequential mode 2 is activ. In **mode 1**, complete 7 segmentinfo is supplied to the segment ports P30 to P42. Then the concerning digit is switched on, where the other digit is switched off during that time (multiplexing). Needless to say, that the total current of the 7 LEDs are exceeding the maximum ratings of the port current. The easy multiplexing of the LEDs, with a dutycycle from 50 percent, has to be achieved by limiting the LED current with 360 Ω resistors down to 7 mA (that means 49 mA in total).

In **mode 2** only 3 to 4 LEDs of one digit are switched on at the same time. The current via one LED may be now up to 12 mA (4 x 12 = 48 mA) which is achieved with 180 Ω resistors. But dutycycle is decreased now down to 25 percent.

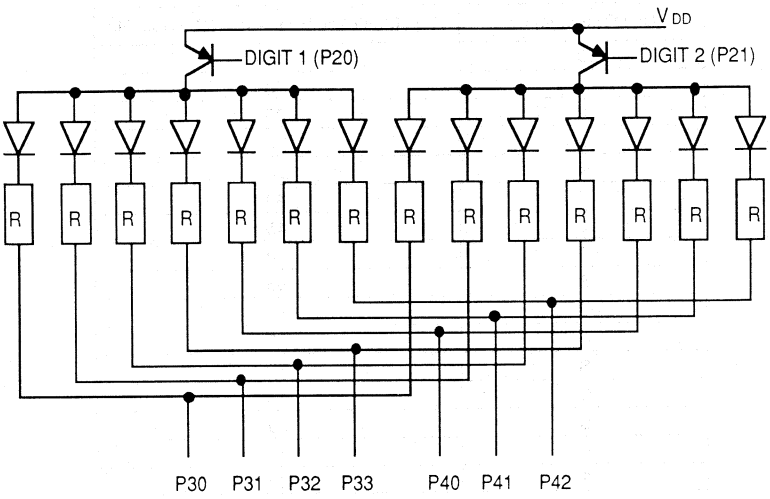
Turning the hex-switch, the multiplex frequency is changing due to the Hex value and in accordance to the selected multiplex mode.

4. Table Data for Multiplex Modes

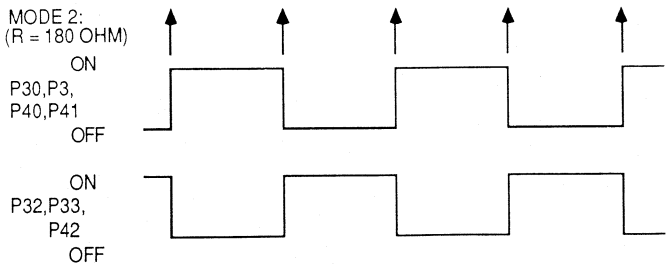
Hex Switch Value	Digit Multiplex Frequency (1/T). Unit: Hz
0	15
1	20
2	25
3	30
4	35
5	40
6	45
7	50
8	55
9	60
A	65
B	70
C	75
D	80
E	85
F	90

Mode 1: R = 360 Ohm
Mode 2: R = 180 Ohm

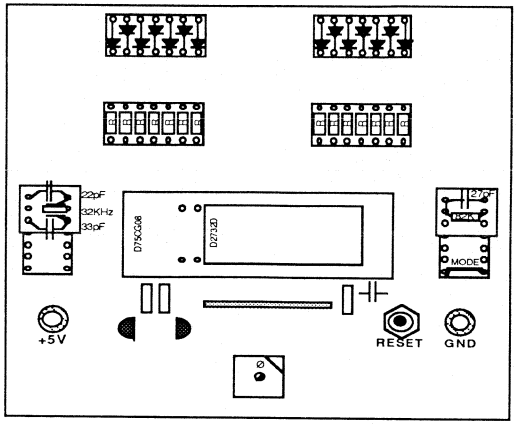
5. LED Multiplex Timing Diagram

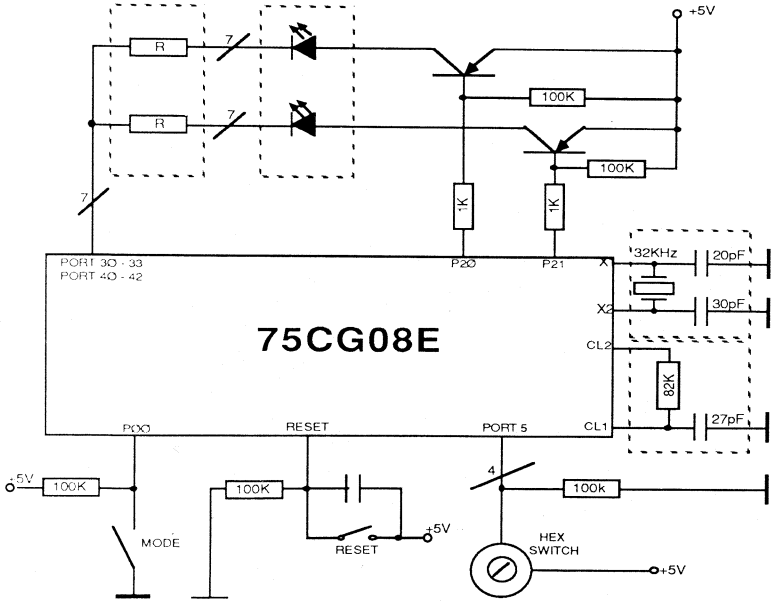


Mode 1: Port P30—Port P42 are always active depending upon segment informations. **Note:** R = 360 Ohm.



6. Board Drawing and Block Diagram





Baud Rate Adaption in Telecommunication Systems

Data Rate adaption techniques are needed in any communication system where multiple data channels with different baud rates are used. This Application News describes a way to solve this problem by software.

Contents:

Author: Andreas Kohl
Application Department
NEC Electronics (Europe) GmbH

1. Problem

Baud rate adaption is always needed in systems where data is received with a fixed data rate and must be transmitted using a different data rate. This occurs, for instance, in a multi-function digital telephone.

Such a telephone allows not only speech communication, but also data transmission. Data is received from the digital line and is sent to an RS232C serial interface or vice versa. In the following example, the data rate of the line is 64 kBit/s and the baud rate of the serial interface is 9.6 kBit/s. Data is transferred from the line to the RS232C interface.

2. Configuration

A block diagram of the system is shown in the following figure.

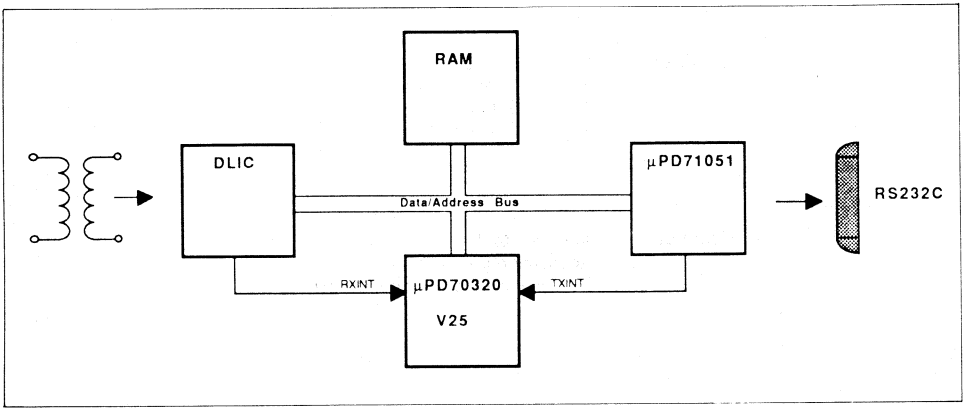


Figure 1. Part of Digital Multifunction Telephone with Ping Pong Interface

This configuration may also be used as a kernel for a wide range of telecom applications like Digital Line Card or ISDN Telephone. In order to adapt to individual interfaces, only the interface controllers need be changed. For an ISDN telephone, for instance, one would have to replace the DLIC (digital line interface controller) by an S-IFC (S-interface controller).

In this example a serial data stream is received from the line. The DLIC interfaces the digital line to the parallel bus of the microprocessor system. It performs the physical connection, which is also known as layer 1 of the OSI (open systems interconnection) model of the International Standardisation Organisation (ISO). We assume that the received data shall be sent to the RS232C interface transparently without further processing. The RS232C is implemented using a serial communication controller such as the μPD71051. However, it is possible to use any other controller or even the **on-chip serial interface of V25**.

3. Solution

The internal memory organization is shown below.

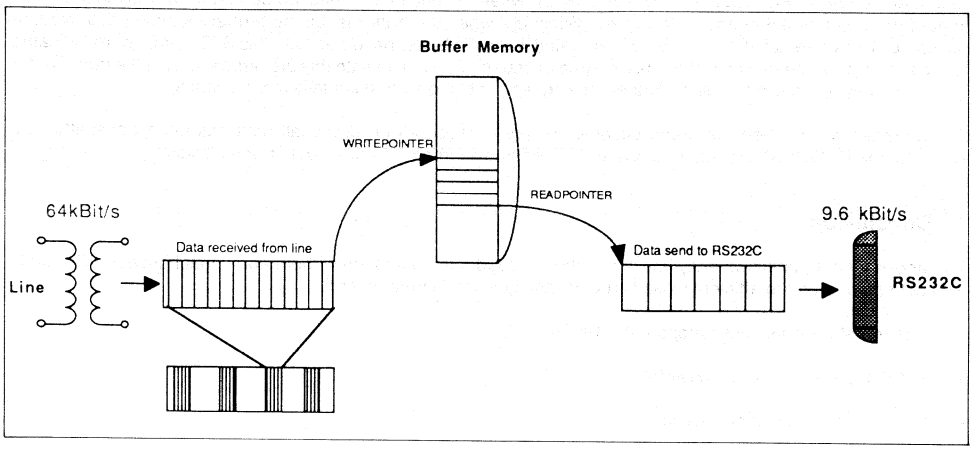


Figure 2. Buffer Memory Organization

Data arriving from the line are serial to parallel converted by the DLIC. For each byte received, the DLIC generates an interrupt (RXINT) to the V25 CPU. In the interrupt service routine for RXINT, the data is fetched from the DLIC and put into memory. A part of the system memory is reserved as buffer for these incoming data. The software organizes this memory as a FIFO (first-in first-out). Each time new data enters the buffer memory, the WRITEPOINTER and the variable "ACTUAL" are incremented. The variable ACTUAL counts the data residing in the buffer.

In parallel to filling the buffer with incoming data, the serial communication controller requests data to be sent to RS232C. Again, this is done by generating an interrupt (TXINT). The interrupt service routine for TXINT reads data from the buffer, increments the READPOINTER and decrements the counter ACTUAL. A simplified flow chart of the interrupt service routine is shown in figure 3.

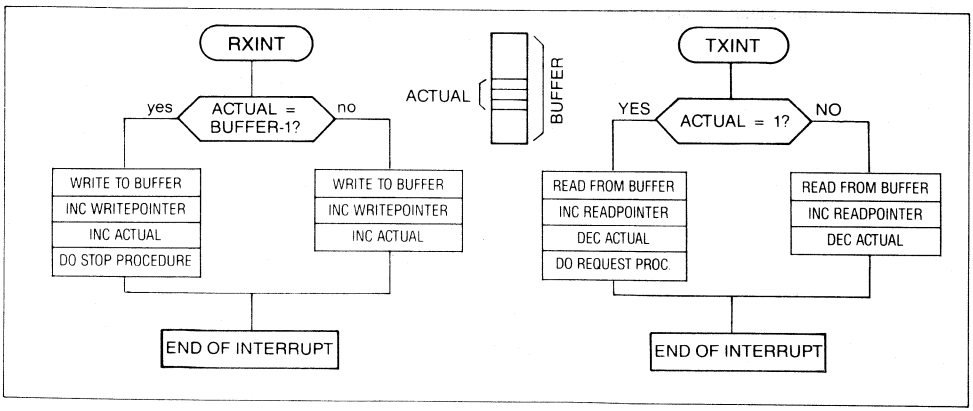


Figure 3. Interupt Service Routine

Special care must be taken in order to avoid buffer over- and underflows.

Firstly, the buffer memory must to be organized as an endless "ring" memory. This may be easily achieved by resetting the WRITEPOINTER and READPOINTER when the end of the reserved buffer memory is reached (this is not further explained in above flow chart). Secondly, when the buffer is full, no further incoming data may be accepted. Therefore, a STOP procedure is activated when the buffer becomes full. This STOP procedure indicated to the sending source that no further data may be accepted. One can initiate this procedure, even if the buffer is not yet completely ful, in order to store further arriving data until the source actually stops sending.

On the other hand, when the buffer becomes empty because all or almost all data has been transmitted via RS232C, the REQUEST procedure activates the sending source to continue data transmission.

4. Conclusion

The above technique allows easy implementation of baud rate adaptation without additional hardware. Currently available resources are efficiently used. The advantages of this method are:

— data transfer works fully transparent in background

— half-/full duplex mode is possible

— multiple channel can be served

To further increase the performance of the above method, the μPD70320 (V25) provides several features which can be used to modify the described algorithm.

One helpful function provided is the on-chip DMA (direct memory access) controller. An application for this is transfering a block of data from the DLIC to the buffer memory via DMA instead of using interrupt. After completion of the block transfer, the STOP procedure is activated.

Another useful V25 feature for this application is the MACRO SERVICE. This is used to transfer data similiar to DMA transfer, while offering additional functions. It allows comparison of incoming data with a predefined value and the transfer to be stopped as soon as coincidence is detected. This can be used to trigger on an end-of-block character which signals the end of data transmission.

As proved above, the μPD70320 provides all functions to efficiently handle baud rate adaptation in all kinds of applications. Data transfer can be supported by Interrupt, macro service and DMA. Multiple channels, as well as full duplex data communication are supported.

4-Bit A/D Converter μPD75104/106/108

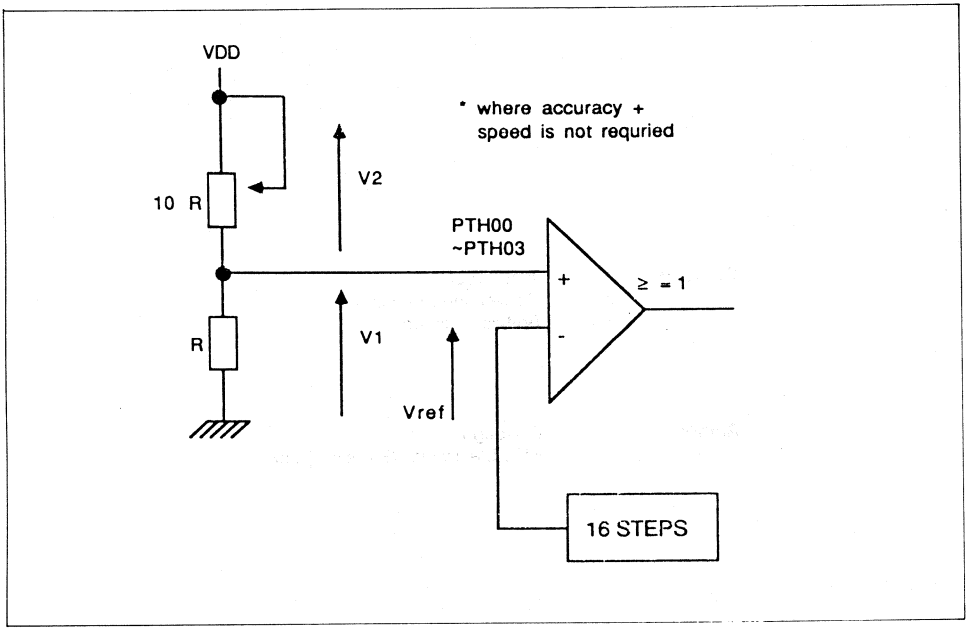
4-channel, 4-bit A/D Converter for the μPD75104/106/108, using the comparator input.

Contents:

Author: B. Gough
NEC Electronics (Europe) GmbH

4-Bit A/D Converter

1. Test Circuit



* 160 mV ~ 4960 mV VOLTAGE RANGE V_{DD} = 5120 mV

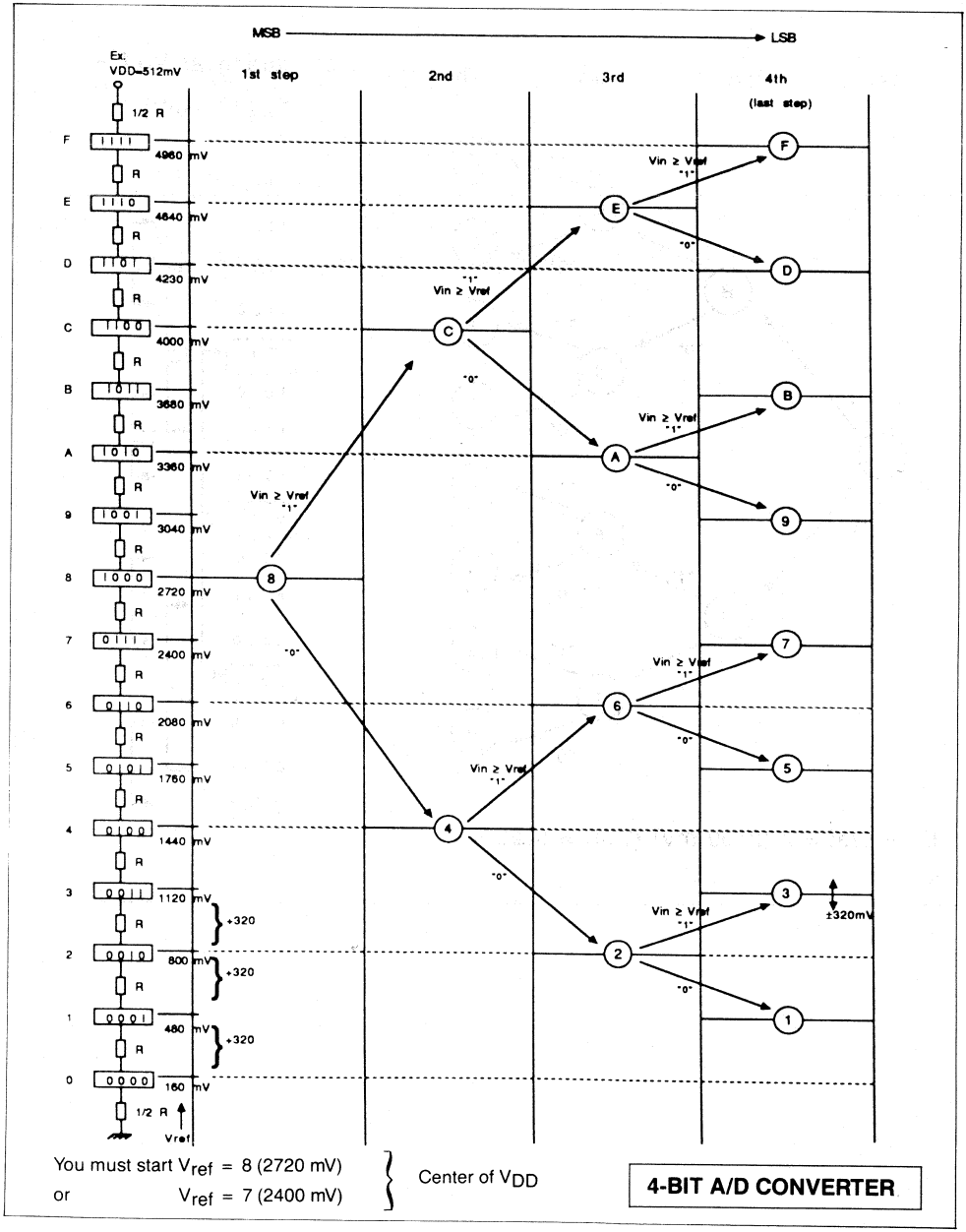
$V_{DD} \times \frac{0.5}{16}$ = MIN

$V_{DD} \times \frac{15.5}{16}$ = MAX

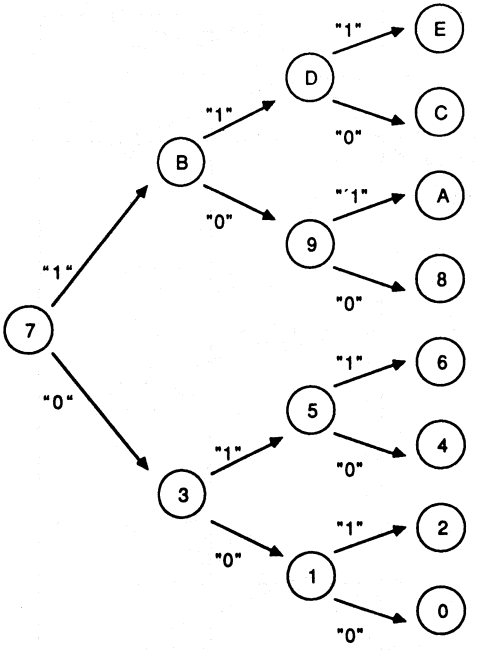
* ACCURACY ± 100 mV OF COMPARATOR

* ACCURACY OF A/D ± 320 mV

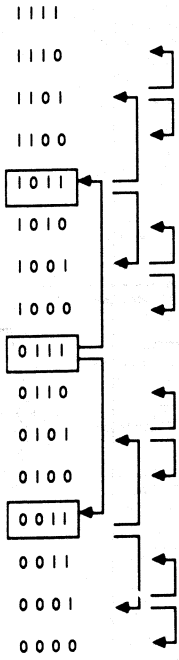
2. Conversion Diagrams



3. Software Listing



or starting at Vref = 7 (2400 mV)



Ex = Vref = 7 (2400 mV) (VDD = 5120 mV)

Software Listing

```
        (COMMAND FILE :

SOURCE STATEMENT


         NAME     NECADCON
;4 BIT A/D CONVERTER PROGRAM WRITTEN FOR 75104/75106/75108
;4 BIT RESULT IN BSB0, UPDATE BY B.GOUGH 4/87
;*****************************************************************
;NAME              :ADCON
;DESC              :4 BIT A/D
;DEVICE            :75104/75106/75108
;MEMORY BANK       :15
;REGISTER BANK     :3
;USED HARDWARE     :COMPARATOR,BSB0
;INPUT             :COMPARATOR PORT BIT 0 (PTH0.0)
;OUTPUT            :-
;DESTORY           :HL,XA,PSW
;CALLS             :-
;*****************************************************************
         PUBLIC   ADCON
         EXTRN    CODE(SELM15,SELR3)

SADSEG   CSEG     SENT

ADCON:   PUSH     BS                  ; MEMORY BANK AND REG BANK
         GETI     SELM15
         GETI     SELR3
         MOV      HL,#0D3H            ;H HAS HIGHER 4 BITS OF PTH0 MB15
                                      ;L HAS BSB0 BIT 3
         MOV      XA,#0C0H            ;X HAS START BIT INFO
         MOV      BSB0,A              ;A HAS RESET VALUE 0000
LOOP:    SET1     BSB0.3L             ;VREF START MIDDLE OF VDD
         MOV      A,BSB0
         MOV      PTHM,XA             ;START CONVERTION
         MOV      A,#04H
WAIT:    INCS     A                   ;WAIT FOR STEP CONVERTION
         BR       WAIT
         MOV1     CY,3H+PTH0.0        ;LOAD STEP RESULT IN BSB0
         MOV1     BSB0.3L,CY
         DECS     L                   ;NEXT STEP
         BR       LOOP
         POP      BS
         RET                          ;AFTER 4 STEPS CONVERTION IS COMPLETE
         END
```

Application Example for a Talking Doll Project (toy) Using μPD7566

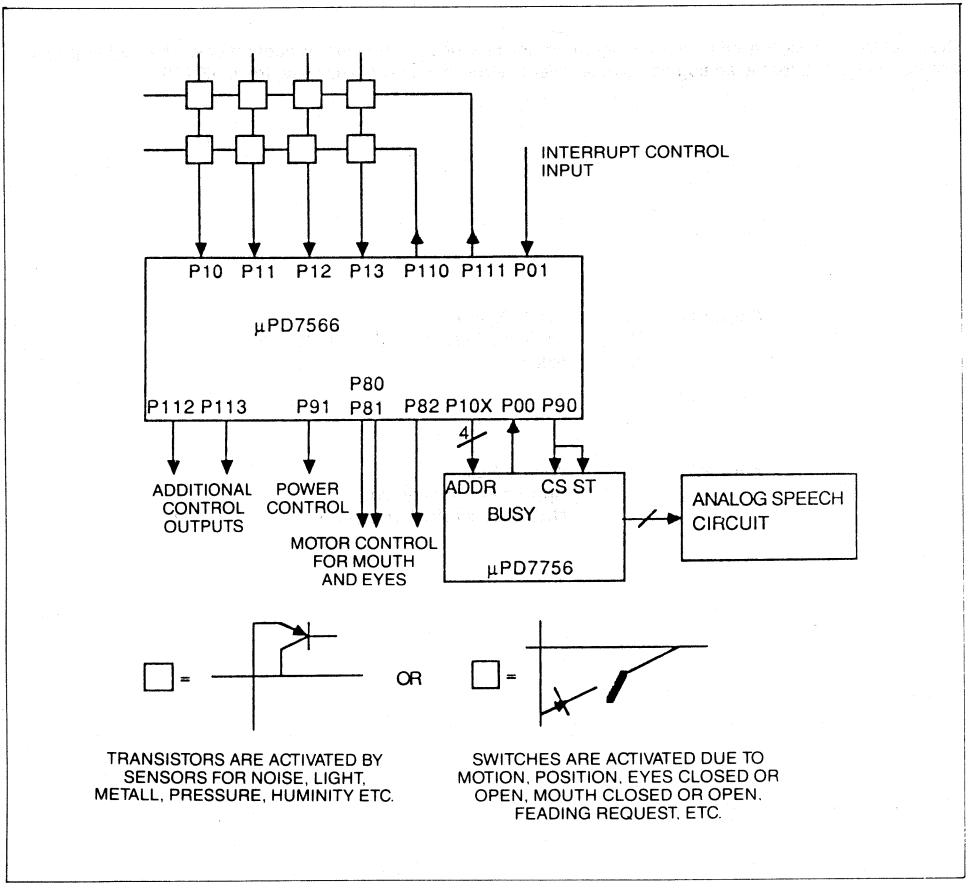
This example shows the minimum configuration for realizing such kinds of applications. The digital part will discussed in detail, for the analog part please refer to application examples using the μPD7756.

Contents:

Author: Wilfred Noll
Application Department
NEC Electronics (Europe) GmbH

1. Block Diagram

An Example showing block diagram for the digital part, of the application.



2. Program Description

In the actual program which can be demonstrated, the following functions are realized:

1) Noise detection
2) Motion detection
3) "Doll is undressed" detection
4) "Combing of Doll" detection
5) "Feeding request" detection
6) Eyes blinking
7) Mouth syncronization by vocals and consonants
8) Depending upon the choosen language, around 10 seconds of speech can be controlled freely by the μCOM.

Program execution:

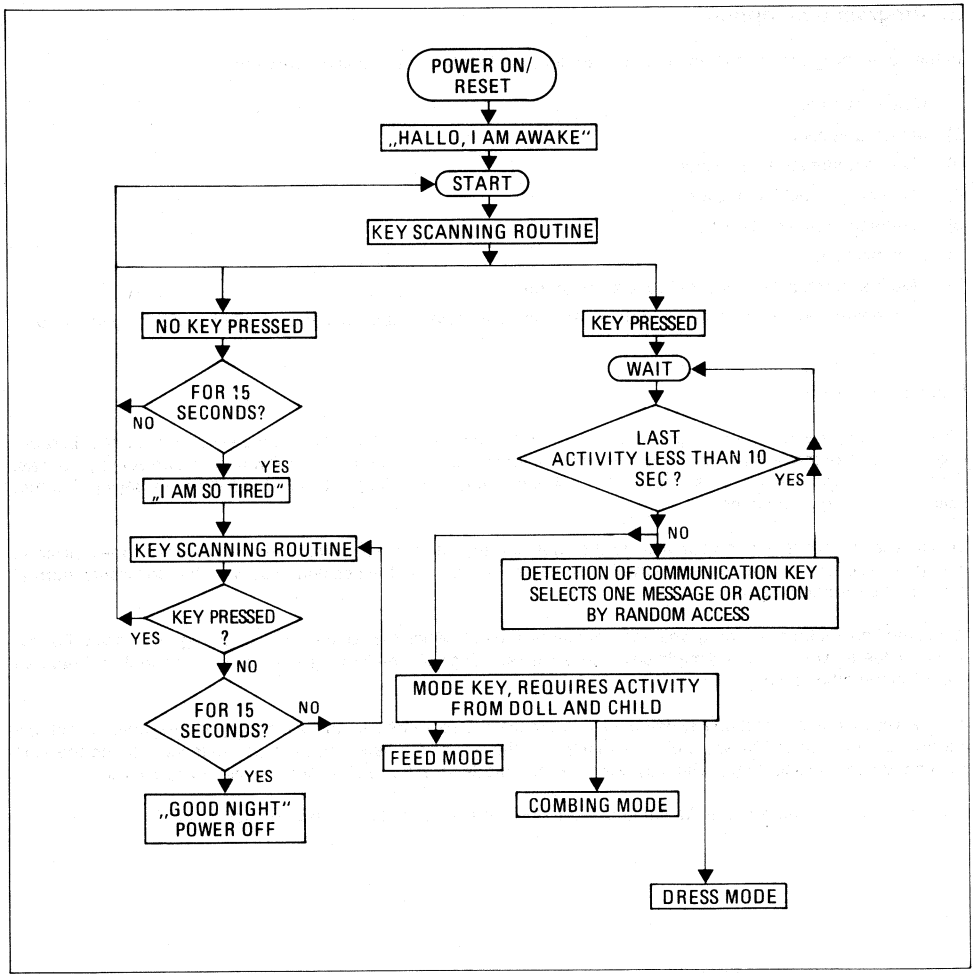
After power on, the doll gives its greetings by saying "Hallo, I am awake". Then all the keys are scanned, in order to get an input for the next activity. Together with a random access generator, the doll will require an activity such as; "feed me", "comb me" or "dress me", or the doll will simply start talking with its "dollmother" by saying "mama, I like you", crying or laughing.

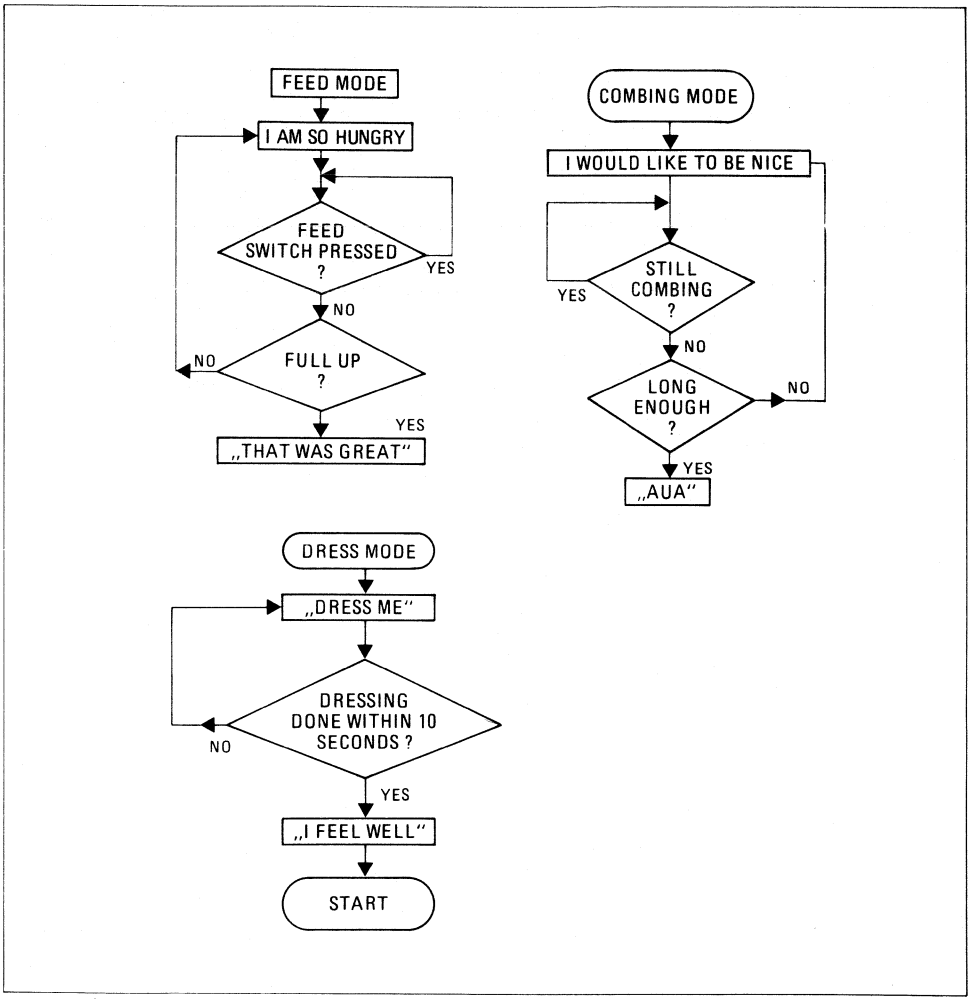
In this case, the doll is requiring an activity, the "dollmother" has to fullfil this requirement, otherwise the doll starts crying and repeats its requirement. If the "dollmother" is still not in the position to do what the doll wants, normal scan mode is entered again.

When the "dollmother" is doing what the doll wants, the doll keeps the communication during this activity. For example, in feeding mode the doll might say "more, I am so hungry" if the feeding time was not sufficient or, "that was great", in the other case.

If there was no noise or motion key detected for more than 15 seconds after the last activity or message, the doll will inform the "dollmother" by saying "I am so tired". If there is now a new "key pressed" detection during the next 15 seconds, the doll continues with key scanning and mode selection will be done same as at power up.

If there is still no input from the "dollmother", the doll will switch itself off by saying "good night".


POWER ON/ RESET
„HALLO, I AM AWAKE"
START
KEY SCANNING ROUTINE
NO KEY PRESSED
FOR 15 SECONDS?
NO
YES
„I AM SO TIRED"
KEY SCANNING ROUTINE
KEY PRESSED ?
YES
NO
FOR 15 SECONDS?
NO
YES
„GOOD NIGHT" POWER OFF
KEY PRESSED
WAIT
LAST ACTIVITY LESS THAN 10 SEC ?
YES
NO
DETECTION OF COMMUNICATION KEY SELECTS ONE MESSAGE OR ACTION BY RANDOM ACCESS
MODE KEY, REQUIRES ACTIVITY FROM DOLL AND CHILD
FEED MODE
COMBING MODE
DRESS MODE



3. Notes:

Above presentes a rough functional description of the program which is available in soft and hardware from NEC-EE.

The program flow or messages will vary between several applications. This as an example of the most cost efficient solution possible.

If more complexibility is required, larger micro (like μPD7533) and larger speech chip (like μPD7757) are needed, and will result in increased system costs.

NEC Customer Services

NEC's Commitment to Information

Our offices throughout Europe are always at your service for comprehensive support. Here are some of the technical services we provide:

- INSECT
- Seminars
- Update service
- Hotline
- Mailing list
- University program

INSECT

INformation **S**ystem and **E**lectronic **C**a**T**alog.

This is an on-line information service. Via a telephone link you can call up the latest data on all VLSI devices available from NEC. This includes enhancements, news and the most recent application know-how.

The service is free to our customers and other interested parties. For a menu-driven guest session, you can dial in to INSECT via the international packet switching network – in Germany this is DATEX-P – using of these numbers
45 21 10 13 020/030
and responding to the request for USERNAME and PASSWORD simply with "customer".

Seminars

No-one is more aware than NEC of the difference that a brief intensive training course can make to your mastery of advanced and often complex devices. We hold regular workshops and seminars at local NEC offices, in our Düsseldorf headquarters, or on customer premises. For information on NEC workshops and seminars, please contact your nearest office.

Update Service

When you buy an evaluation package from NEC, you become automatically entitled to one year's free updates for both hardware and software. All updates reach you fast and reliably via a courier service. In a field where rapid changes are the norm, you can be thus be sure of working with the most up-to-date development tools.

Hotlines

NEC's offices located throughout Europe are responsible for technical support and customer services. On the back cover of this brochure you can check which office is most convenient for you to contact. You are also welcome to contact our European headquarters in Düsseldorf directly.

Mailing list

Our engineering staff produces frequent additions to the available technical documentation in the form of application notes, product news and technical letters. If you would like to be included on our mailing list for this documentation, please inform us.

University Program

Many of our products, because of their complexity and dedicated application support through EB tools, are interesting subjects for graduate studies. NEC is always ready to discuss this possibility.