intel®

APPLICATION NOTE

AP-41

December 1978

Introduction to the UPI-41A™

John Beaston and Robin Jigour
Microcomputer Applications

9800876

Introduction to the UPI-41A

Contents

DECEMBER 1978

INTRODUCTION

Since the introduction in 1974 of the second generation of microprocessors, such as the 8080, a wide range of peripheral interface devices have appeared. At first, these devices solved application problems of a general nature; i.e., parallel interface (8255), serial interface (8251), timing (8253), interrupt control (8259). However, as the speed and density of LSI technology increased, more and more intelligence was incorporated into the peripheral devices. This allowed more specific application problems to be solved, such as floppy disk control (8271), CRT control (8275), and data link control (8273). The advantage to the system designer of this increased peripheral device intelligence is that many of the peripheral control tasks are now handled externally to the main processor in the peripheral hardware rather than internally in the main processor software. This reduced main processor overhead results in increased system throughput and reduced software complexity.

In spite of the number of peripheral devices available, the pervasiveness of the microprocessor has been such that there is still a large number of peripheral control applications not yet satisfied by dedicated LSI. Complicating this problem is the fact that new applications are emerging faster than the manufacturers can react in developing new, dedicated peripheral controllers. To address this problem, a new microcomputer-based Universal Peripheral Interface (UPI-41A) device was developed.

In essence, the UPI-41A acts as a slave processor to the main system CPU. The UPI contains its own processor, memory, and I/O, and is completely user programmable; that is, the entire peripheral control algorithm can be programmed locally in the UPI, instead of taxing the master processor's main memory. This distributed processing concept allows the UPI to handle the real-time tasks such as encoding keyboards, controlling printers, or multiplexing displays, while the main processor is handling non-real-time dependent tasks such as buffer management or arithmetic. The UPI relies on the master only for initialization, elementary commands, and data transfers. This technique results in an overall increase in system efficiency since both processors — the master CPU and the slave UPI — are working in parallel.

This application note presents three UPI-41A applications which are roughly divided into two groups: applications whose complexity and UPI code space requirements allow them to either stand alone or be incorporated as just one task in a "multi-tasking" UPI, and applications which are complete UPI applications in themselves. Applications in the first group are a simple LED display and sensor matrix controllers. A combination serial/parallel I/O device is an application in the second group. Each application illustrates different UPI config-

UPI-41 vs. UPI-41A

The UPI-41A is an enhanced version of the UPI-41. It incorporates several architectural features not found on the "non-A" device:

- Separate Data In and Data Out data bus buffer registers
- User-definable STATUS register bits
- Programmable master interrupts for the OBF and $\overline{\text{IBF}}$ flags
- Programmable DMA interface to external DMA controller.

The separate Data In (DBBIN) and Data Out (DBBOUT) registers greatly simplify the master/UPI protocol compared to the UPI-41. The master need only check IBF before writing to DBBIN and OBF before reading DBBOUT. No data bus buffer lock-out is required.

The most significant nibble of the STATUS register, undefined in the UPI-41, is user-definable in UPI-41A. It may be loaded directly from the most significant nibble of the Accumulator (MOV STS, A). These extra four STATUS bits are useful for transferring additional status information to the master. This application note uses this feature extensively.

A new instruction, EN FLAGS, allows OBF and $\overline{\text{IBF}}$ to be reflected on Port 2 bit 4 and Port 2 bit 5 respectively. This feature enables interrupt-driven data transfers when these pins are interrupt sources to the master.

By executing an EN DMA instruction Port 2 bit 6 becomes a DRQ (DMA Request) output and Port 2 bit 7 becomes $\overline{\text{DACK}}$ (DMA Acknowledge). Setting DRQ requests a DMA cycle to an external DMA controller. When the cycle is granted, the DMA controller returns DACK plus either RD (Read) or WR (Write). DACK automatically forces $\overline{\text{CS}}$ and A0 low internally and clears DRQ. This selects the appropriate data buffer register (DBBOUT for DACK and RD, DBBIN for DACK and WR) for the DMA transfer.

Like the "non-A", the UPI-41A is available in both ROM (8041A) and EPROM (8741A) Program Memory versions. This application note deals exclusively with the UPI-41A since the applications use the "A"s enhanced features.

urations and features. However, before the application details are presented, a section on the UPI/master proto col requirements is included. These protocol requirements are key to UPI software development. It is suggested that the reader not already familiar with the architecture and instruction set of the UPI-41A read the "Intel UPI-41 User's Manual" before proceeding with this document. For convenience, the UPI block diagram and instruction set summary are reproduced in Figures 1 and 2.

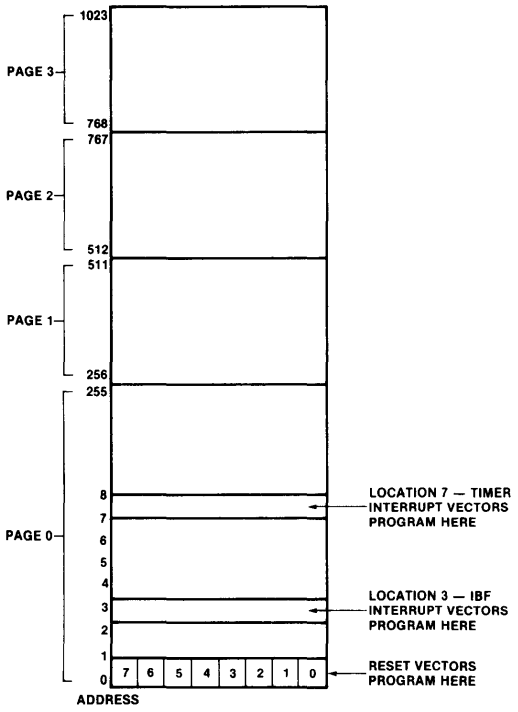


Figure 1A. Program Memory Map

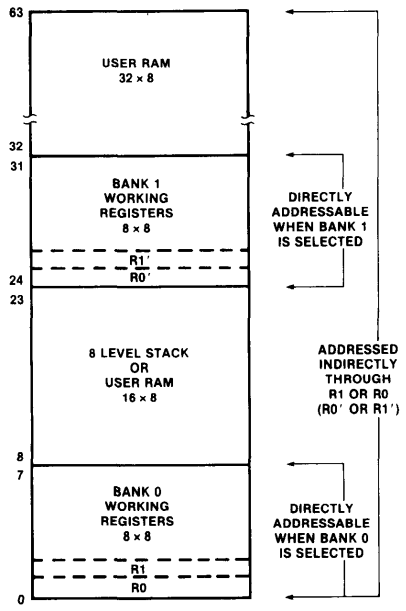


Figure 1B. Data Memory Map

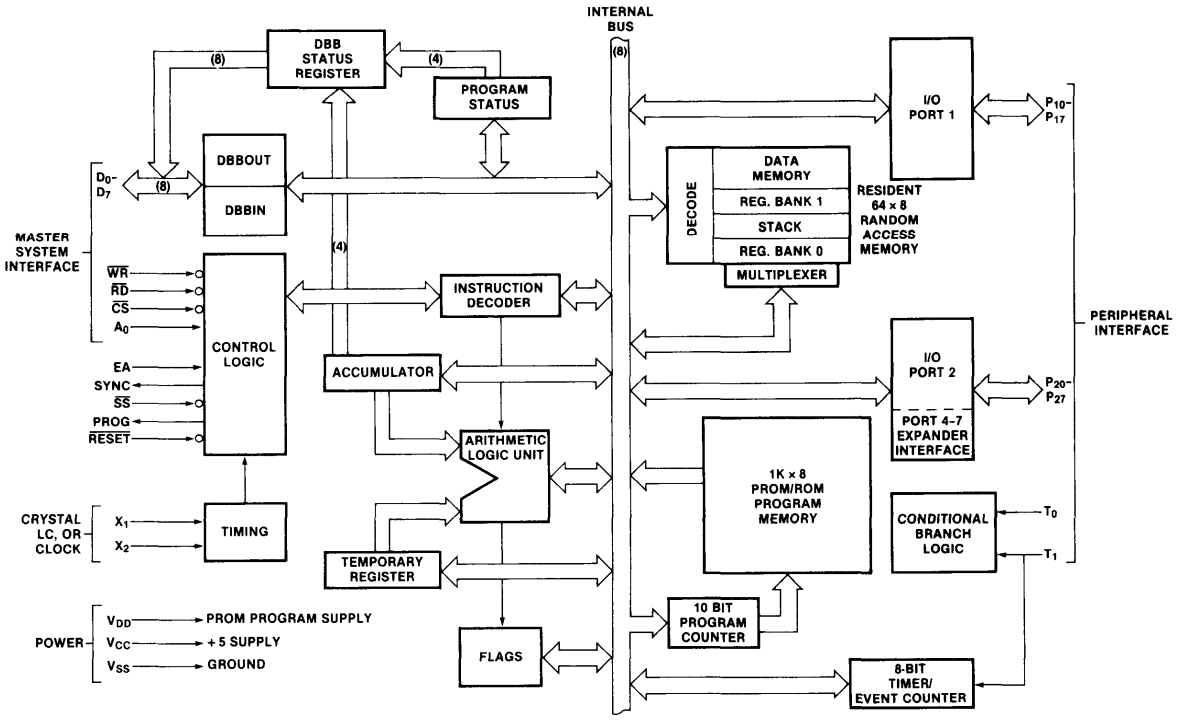


Figure 1C. UPI-41A Block Diagram

UPI INSTRUCTION SET

Mnemonic	Description	Bytes	Cycles
ACCUMULATOR			
ADD A,Rr	Add register to A	1	1
ADD A,@Rr	Add data memory to A	1	1
ADD A,#data	Add immediate to A	2	2
ADDC A,Rr	Add immed. to A with carry	1	1
ADDC A,@Rr	Add immed. to A with carry	1	1
ADDC A,#data	Add immed. to A with carry	2	2
ANL A,Rr	AND register to A	1	1
ANL A,@Rr	AND data memory to A	1	1
ANL A,#data	AND immediate to A	2	2
ORL A,Rr	OR register to A	1	1
ORL A,@Rr	OR data memory to A	1	1
ORL A,#data	OR immediate to A	2	2
XRL A,Rr	Exclusive OR register to A	1	1
XRL A,@Rr	Exclusive OR data memory to A	1	1
XRL A,#data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap digits of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
INPUT/OUTPUT			
IN A,Pp	Input port to A	1	2
OUTL Pp,A	Output A to port	1	2
ANL Pp,#data	AND immediate to port	2	2
ORL Pp,#data	OR immediate to port	2	2
IN A,DBB	Input DBB to A, clear IBF	1	1
OUT DBB,A	Output A to DBB, set OBF	1	1
MOVD A,Pp	Input Expander port to A	1	2
MOVD Pp,A	Output A to Expander port	1	2
ANLD Pp,A	AND A to Expander port	1	2
ORLD Pp,A	OR A to Expander port	1	2
DATA MOVES			
MOV A,Rr	Move register to A	1	1
MOV A,@Rr	Move data memory to A	1	1
MOV A,#data	Move immediate to A	2	2
MOV Rr,A	Move A to register	1	1
MOV @Rr,A	Move A to data memory	1	1
MOV Rr,#data	Move immediate to register	2	2
MOV @Rr,#data	Move immediate to data memory	2	2
MOV A,PSW	Move PSW to A	1	1
MOV PSW,A	Move A to PSW	1	1
XCH A,Rr	Exchange A and register	1	1
XCH A,@Rr	Exchange A and data memory	1	1
XCHD A,@Rr	Exchange digit of A and register	1	1
MOVP A,@A	Move to A from current page	1	2
MOVP3, A,@A	Move to A from page 3	1	2
TIMER/COUNTER			
MOV A,T	Read Timer/Counter	1	1
MOV T,A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1

Mnemonic	Description	Bytes	Cycles
CONTROL			
EN DMA	Enable DMA Handshake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Disable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
REGISTERS			
INC Rr	Increment register	1	1
INC @Rr	Increment data memory	1	1
DEC Rr	Decrement register	1	1
SUBROUTINE			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
FLAGS			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
MOV STS, A	A_4-A_7 to Bits 4-7 of Status	1	1
BRANCH			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R,addr	Decrement register and skip	2	2
JC addr	Jump on Carry = 1	2	2
JNC addr	Jump on Carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 Flag = 1	2	2
JF1 addr	Jump on F1 Flag = 1	2	2
JTF addr	Jump on Timer Flag = 1, Clear Flag	2	2
JNIBF addr	Jump on IBF Flag = 0	2	2
JOBF addr	Jump on OBF Flag = 1	2	2
JBb addr	Jump on Accumulator Bit	2	2

Figure 2. UPI-41A Instruction Set Summary

UPI/MASTER PROTOCOL

As in most closely coupled multiprocessor systems, the various processors communicate via a shared resource. This shared resource is typically specific locations in RAM or in registers through which status and data are passed. In the case of a master processor and a UPI-41A, the shared resource is 3 separate, master-addressable, registers internal to the UPI. These registers are the STATUS register (STATUS), the Data Bus Buffer Input register (DBBIN), and the Data Bus Output register (DBBOUT). [Data Bus Buffer direction is relative to the UPI]. To illustrate this register interface, consider the 8085A/UPI system in Figure 3.

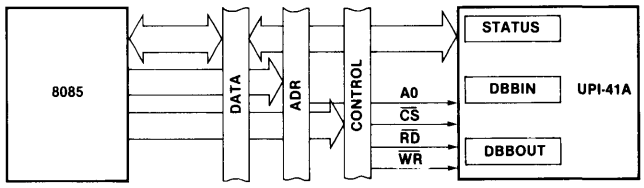


Figure 3. Register Interface

Looking into the UPI from the 8085A, the 8085A sees only the three registers mentioned above. If the 8085A wishes to issue a command to the UPI, it does so by writing the command to the DBBIN register according to the decoding of Figure 4. Data for the UPI is also passed via the DBBIN register. (The UPI differentiates commands and data by examining the A0 pin. Just how this is done is covered shortly.) Data from the UPI for the 8085A is passed in the DBBOUT register. The 8085A may interrogate the UPI's status by reading the UPI's STATUS register. Four bits of the STATUS register act as flags and are used to handshake data and commands into and out of the UPI. The STATUS register format is shown in Figure 5.

Bit 0 is OBF (Output Buffer Full). This flag indicates to the master when the UPI has placed data in the DBBOUT register. OBF is set when the UPI writes to DBBOUT and is reset when the master reads DBBOUT. The master finds meaningful data in the DBBOUT register only when OBF is set.

The Input Buffer Full (IBF) flag is bit 1. The UPI uses this flag as an indicator that the master has written to the DBBIN register. The master uses IBF to indicate when the UPI has accepted a particular command or data byte. The master should examine IBF before outputting anything to the UPI. IBF is set when the master writes to DBBIN and is reset when the UPI reads DBBIN. The master must wait until IBF = 0 before writing new data or commands to DBBIN. Conversely, the UPI must ensure IBF = 1 before reading DBBIN.

The third STATUS register bit is F0 (Flag 0). This is general purpose flag that the UPI can set, reset, and test. It is typically used to indicate a UPI error or busy condition to the master.

Flag 1 (F1) is the final dedicated STATUS bit. Like F0 the UPI can set, reset, and test this flag. However, in addition, F1 reflects the state of the A0 pin whenever the master writes to the DBBIN register. The UPI uses this flag to delineate between master command and data writes to DBBIN.

$\overline{CS}$	A0	$\overline{RD}$	$\overline{WR}$	REGISTER
0	0	0	1	READ DBBOUT
0	1	0	1	READ STATUS
0	0	1	0	WRITE DBBIN (DATA)
0	1	1	0	WRITE DBBIN (COMMAND)
1	X	X	X	NO ACTION

Figure 4. Register Decoding

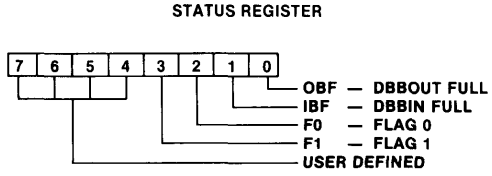


Figure 5. Status Register Format

The remaining four STATUS register bits are user definable. Typical uses of these bits are as status indicators for individual tasks in a multitasking UPI or as UPI generated interrupt status. These bits find a wide variety of uses in the upcoming applications.

Looking into the 8085A from the UPI, the UPI sees the two DBB registers plus the IBF, OBF, and F1 flags. The UPI can write from its accumulator to DBBOUT or read DBBIN into the accumulator. The UPI cannot read OBF, IBF, or F1 directly, but these flags may be tested using conditional jump instructions. The UPI should make sure that OBF is reset before writing new data into DBBOUT to ensure that the master has read previous DBBOUT data. IBF should also be tested before reading DBBIN since DBBIN data is valid only when IBF is set. As was mentioned earlier, the UPI uses F1 to differentiate between command and data contents in DBBIN when IBF is set. The UPI may also write the upper 4-bits of its accumulator to the upper 4-bits of the STATUS register. These bits are thus user definable.

The UPI can test the flags at any time during its internal program execution. It essentially "polls" the STATUS register for changes. If faster response is needed to master commands and data, the UPI's internal interrupt structure can be used. If IBF interrupts are enabled, a master write to DBBIN (either command or data) sets IBF which generates an internal CALL to location 03H in program memory. At this point, working register contents can be saved using bank switching, the accumulator saved in a spare working register, and the DBBIN register read and serviced. The interrupt logic for the IBF interrupt is shown in Figure 6. A few observations concerning this logic are appropriate. Note that if the master writes to DBBIN while the UPI is still servicing the last IBF interrupt (a Return (RETR) instruction has not been executed), the IBF Interrupt Pending line is made high which causes a new CALL to 03H as soon as the first RETR is executed. No EN I (Enable Interrupt) instruction is needed to rearm the interrupt logic as is needed in an 8080 or 8085A system; the RETR performs this function. Also note that executing a DIS I to disable further IBF interrupts does not clear a pending interrupt. Only a CALL to location 03H or RESET clears a pending IBF interrupt.

Keeping in mind that the actual master/UPI protocol is dependent on the application, probably the best way to illustrate correct protocol is by example. Let's consider using the UPI as a simple parallel I/O device. (This is a trivial application but it embodies all of the important protocol considerations.) Since the UPI may be either interrupt or non-interrupt driven internally, both cases are considered.

Let's take the easiest configuration first; using the UPI Port 1 as an 8-bit output port. From the UPI's point-of-view, this is an input-only application since all that is required is that the UPI input data from the master. Once the master writes data to the UPI, the UPI reads the DBBIN register and transfers the data to Port 1. No testing for commands vs data is needed since the UPI "knows" it only performs one task — no commands are needed.

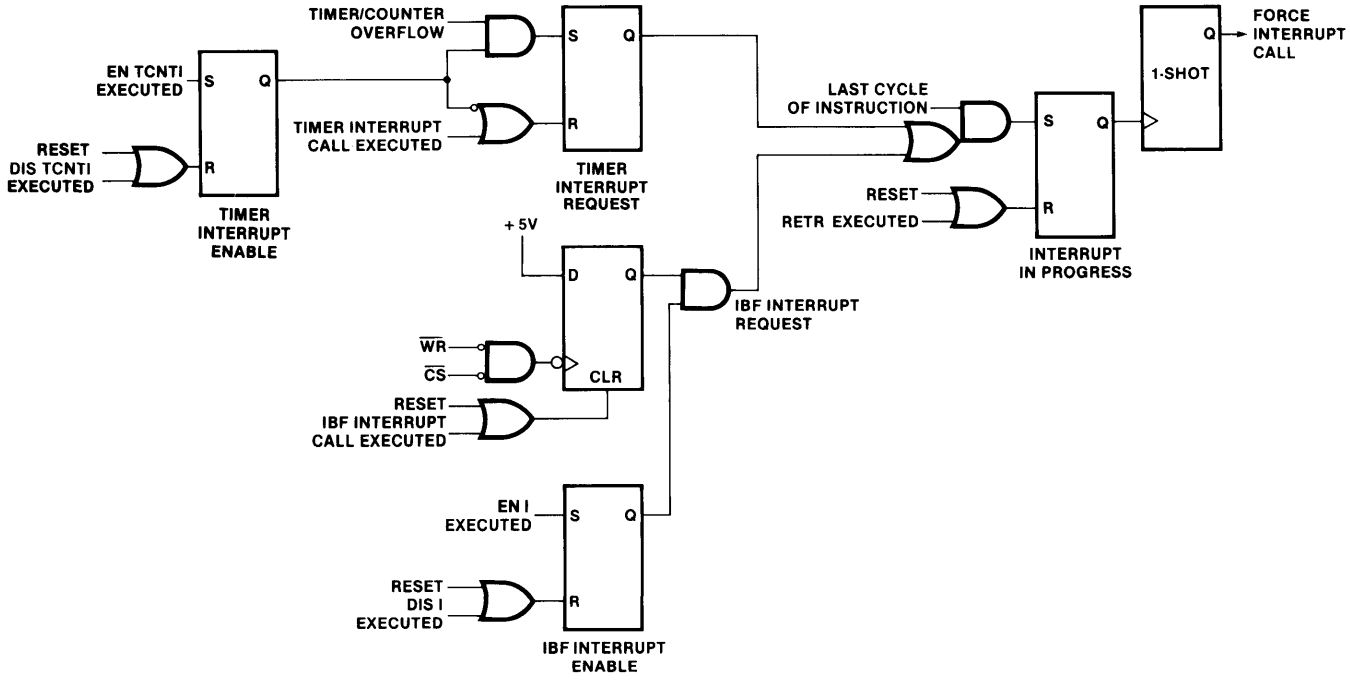


Figure 6. UPI-41A Interrupt Structure

Non-interrupt driven UPI software is shown in Figure 7A while Figure 7B shows interrupt based software. For Figure 7A, the UPI simply waits until it sees IBF go high indicating the master has written a data byte to DBBIN. The UPI then reads DBBIN, transfers it to Port 1, and returns to waiting for the next data. For the interrupt-driven UPI, Figure 7B, once the EN I instruction is executed, the UPI simply waits for the IBF interrupt before handling the data. The UPI could handle other tasks during this waiting time. When the master writes the data to DBBIN, an IBF interrupt is generated which performs a CALL to location 03H. At this point the UPI reads DBBIN (no testing of IBF is needed since an IBF interrupt implies that IBF is set), transfers the data to Port 1, and executes an RETR which returns program flow to the main program.

Software for the master 8085A is included in Figure 7C. The only requirement for the master to output data to the UPI is that it check the UPI to be sure the previous data had been taken before writing new data. To accomplish this the master simply reads the STATUS register looking for IBF = 0 before writing the next data.

```
;
; UPI INPUT ONLY EXAMPLE - PORT 1 USED AS OUTPUT PORT
;       UPI POLLS IBF FOR DATA
;
RESET:  JNIBF   RESET   ; WAIT ON IBF FOR INPUT
        IN      A, DBB  ; INPUT THERE, SO READ IT
        OUTL    P1, A   ; TRANSFER DATA TO PORT 1
        JMP     RESET   ; GO WAIT FOR NEXT DATA
```

Figure 7A. Single Output Port Example — Polling

```
;
; UPI INPUT ONLY EXAMPLE - PORT 1 USED AS OUTPUT PORT
;       DATA INPUT IS INTERRUPT-DRIVEN ON IBF
;
RESET:  EN      I       ; ENABLE IBF INTERRUPTS
        JMP     RESET+1 ; LOOP WAITING FOR INPUT
IBFINT: IN      A, DBB  ; READ DATA FROM DBBIN
        OUTL    P1, A   ; TRANSFER DATA TO PORT 1
        RETR            ; RETURN WITH RESTORE
```

Figue 7B. Single Output Port Example — Interrupt

```
;
; 8085 SOFTWARE FOR UPI INPUT-ONLY EXAMPLE
;       DATA FOR OUTPUT IS PASSED IN REG. C
;
UPIOUT: IN      STATUS  ; READ UPI STATUS
        ANI     IBF     ; LOOK AT IBF
        JNZ     UPIOUT  ; WAIT FOR IBF=0
        MOV     A, C    ; GET DATA FROM C
        OUT     DBBIN   ; OUTPUT DATA TO DBBIN
        RET             ; DONE, RETURN
```

Figure 7C. 8085A Code for Single Output Port Example

Figure 8A illustrates the case where UPI Port 2 is used as an 8-bit input port. This configuration is termed UPI output-only as the master does not write (input) to the UPI but simply reads either the STATUS or the DBBOUT registers. In this example only the OBF flag is used. OBF signals the master that the UPI has placed new port data in DBBOUT. The UPI loops testing OBF. When OBF is clear, the master has read the previous data and UPI then reads its input port (Port 2) and places this data in DBBOUT. It then waits on OBF until the master reads

DBBOUT before reading the input port again. When the master wishes to read the input port data, Figure 8B, it simply checks for OBF being set in the STATUS register before reading DBBOUT. While this technique illustrates proper protocol, it should be noted that it is not meant to be a good method of using the UPI as an input port since the master would never get the newest status of the port.

The above examples can easily be combined. Figure 9 shows UPI software to use Port 1 as an output port simultaneously with Port 2 as an input port. The program starts with the UPI checking IBF to see if the master has written data destined for the output port into DBBIN. If IBF is set, the UPI reads DBBIN and transfers the data to the output port (Port 1). If IBF is not set or once the data is transferred to the output port if it was, OBF is tested. If OBF is reset (indicating the master has read DBBOUT), the input port (Port 2) is read and transferred to DBBOUT. If OBF is set, the master has yet to read DBBOUT so the program just loops back to test IBF.

```
;
; UPI OUTPUT ONLY EXAMPLE - PORT 2 USED AS INPUT PORT
;         PORT DATA IS AVAILABLE IN DBBOUT
;
RESET:  JOBF    RESET   ; LOOP IF OBF = 1 (DATA NOT READ)
        IN      A, P2   ; DBBOUT CLEAR, READ PORT
        OUT     DBB, A  ; TRANSFER PORT DATA TO DBBOUT
        JMP     RESET   ; WAIT FOR MASTER TO READ DATA
```

Figure 8A. Single Input Port Example

```
;
; 8085 SOFTWARE FOR UPI OUTPUT-ONLY EXAMPLE
;         INPUT DATA RETURNED IN REG. A
;
UPIIN:  IN      STATUS  ; READ UPI STATUS
        ANI     OBF     ; LOOK AT OBF
        JZ      UPIIN   ; WAIT UNTIL OBF = 1
        IN      DBBOUT  ; READ DBBOUT
        RET             ; RETURN WITH DATA IN A
```

Figure 8B. 8085A Single Input Port Code

```
;
:UPI INPUT/OUTPUT EXAMPLE - PORT 1 OUTPUT, PORT 2 INPUT
;
RESET:  JNIBF   OUT1    ; IF IBF = 0, DO OUTPUT
        IN      A, DBB  ; IF IBF = 1, READ DBBIN
        OUTL    P1, A   ; TRANSFER DATA TO PORT 1
OUT1:   JOBF    RESET   ; IF OBF = 1, GO TEST IBF
        IN      A, P2   ; IF OBF = 0, READ PORT 2
        OUT     DBB, A  ; TRANSFER PORT DATA TO DBBOUT
        JMP     RESET   ; GO CHECK FOR INPUT
```

Figure 9. Combination Output/Input Port Example

The master software is identical to the separate input/output examples; the master must test IBF and OBF before writing output port data into DBBIN or before reading input port data from DBBOUT respectively.

In all of the three examples above, the UPI treats information from the master solely as data. There has been no need to check if DBBIN information is a command rather than data since the applications do not require commands. But what if both Port 1 and 2 were used as output ports? The UPI needs to know into which port to put the data. Let's use a command to select which port.

Recall that both commands and data pass through DBBIN. The state of the A0 pin at the time of the write to DBBIN is used to distinguish commands from data. By convention, DBBIN writes with A0 = 0 are for data, and those with A0 = 1 are commands. When DBBIN is written into, F1 (Flag 1) is set to the state of A0. The UPI tests F1 to determine if the information in the DBBIN register is data or a command.

For the case of two output ports, let's assume that the master selects the desired port with a command prior to writing the data. (We could just use F1 as a port select but that would not illustrate the subtle differences between commands and data.) Let's define the port select commands such that bit 1 = 1 if the next data is for Port 1 (Write Port 1 = 0000 0010) and bit 2 = 1 if the next data is for Port 2 (Write Port 2 = 0000 0100). (The number of the set bit selects the port.) Any other bits are ignored. This assignment is completely arbitrary; we could use any command structure, but this one has the advantage of being simple.

Note that the UPI must "remember" from DBBIN write to write which port has been selected. Let's use F0 (Flag 0) for this purpose. If a Write Port 1 command is received, F0 is reset. If the command is Write Port 2, F0 is set. When the UPI finds data in DBBIN, F0 is interrogated and the data is loaded into the previously selected port. The UPI software is shown in Figure 10A.

```
;
; UPI DUAL OUTPUT PORT EXAMPLE - BOTH PORT 1 AND 2 OUTPUTS
;         COMMAND SELECTS DESIRED PORT
;         WRITE PORT 1 - 0000 0010 (02H)
;         WRITE PORT 2 - 0000 0100 (04H)
;
;         FLAG 0 USED TO REMEMBER WHICH PORT WAS SELECTED
;         BY LAST COMMAND.
;
RESET:  JNIBF   RESET   ; WAIT FOR MASTER INPUT
        IN      A, DBB  ; READ INPUT
        JF1     CMD     ; IF F1 = 1, COMMAND INPUT
        JF0     PORT2   ; INPUT IS DATA, TEST F0
        OUTL    P1, A   ; F0 = 0, SO OUTPUT TO PORT 1
        JMP     RESET   ; WAIT FOR NEXT INPUT
PORT2:  OUTL    P2, A   ; F0 = 1, SO OUTPUT TO PORT 2
        JMP     RESET   ; WAIT FOR NEXT INPUT
CMD:    JB1     PT1     ; TEST COMMAND BITS (BIT 1)
        JB2     PT2     ; TEST BIT 2
        JMP     RESET   ; NEITHER BIT SET, WAIT FOR INPUT
PT1:    CLR     F0      ; PORT 1 SELECTED, CLEAR F0
        JMP     RESET   ; WAIT FOR INPUT
PT2:    CLR     F0      ; PORT 2 SELECTED, SET F0
        CPL     F0
        JMP     RESET   ; WAIT FOR INPUT
```

Figure 10A. Dual Output Port Example

Initially, the UPI simply waits until IBF is set indicating the master has written into DBBIN. Once IBF is set, DBBIN is read and F1 is tested for a command. If F1 = 1, the DBBIN byte is a command. Assuming a command, bit 1 is tested to see if the command selected port 1. If so, F0 is cleared and the program returns to wait for the data. If bit 1 = 0, bit 2 is tested. If bit 2 is set, Port 2 is selected so F0 is set. The program then loops back waiting for the next master input. This input is the desired port data. If bit 2 was not set, F0 is not changed and no action is taken.

When IBF = 1 is again detected, the input is again tested for command or data. Since it is necesarily data, DBBIN is read and F0 is tested to determine which port was previously selected. The data is then output to that port, following which the program waits for the next input. Note that since F0 still selects the previous port, the next input could be more data for that port. The port selection command could be thought of as a port select flip-flop control; once a selection is made, data may be repeatedly written to that port until the other port is selected. Master software, Figure 10B, simply must check IBF before writing either a command or data to DBBIN. Otherwise, the master software is straightforward.

For the sake of completeness, UPI software for implementing two input ports is given in Figure 11. This case is simpler than the dual output case since the UPI can assume that all writes to DBBIN are port selection commands so no command/data testing is required. Once the Port Read command is input, the selected port is read and the port data is placed in DBBOUT. Note that in this case F0 is used as a UPI error indicator. If the master happened to issue an invalid command (a command without either bit 1 or 2 set), F0 is set to notify the master that the UPI did not know how to interpret the command. F0 is also set if the master commanded a port read before it had read DBBOUT from the previous command. The UPI simply tests OBF just prior to loading DBBOUT and if OBF = 1, F0 is set to indicate the error.

All of the above examples are, in themselves, rather trivial applications of the UPI although they could easily be incorporated as one of several tasks in a UPI handling multiple small tasks. We have covered them primarily to introduce the UPI concept and to illustrate some master/UPI protocol. Before moving on to more realistic UPI applications, let's discuss two UPI features that do not directly relate to the master/UPI protocol but greatly enhance the UPI's data transfer capability.

In addition to the OBF and IBF bits in the STATUS register, these flags can also be made available directly on two port pins. These port pins can then be used as interrupt sources to the master. By executing an EN FLAGS instruction, Port 2 pin 4 reflects the condition of OBF and Port 2 pin 5 reflects the inverted condition of IBF ($\overline{IBF}$). These dedicated outputs can then be enabled or disabled via their respective port bit values; i.e., P24 reflects OBF as long as an instruction is executed which sets P24 (i.e. ORL P2,#10H). The same action applies to the $\overline{IBF}$ output except P25 is used. Thus P24 may serve as a DATA AVAILABLE interrupt output. Likewise for P25 as a READY-TO-ACCEPT-DATA interrupt. This greatly simplifies interrupt-driven master-slave data transfers.

```
;
; 8085 SOFTWARE FOR DUAL OUTPUT PORT EXAMPLE
;       THIS ROUTINE WRITES DATA IN REG. C TO PORT 1
;       (SAME ROUTINE FOR PORT 2 - JUST CHANGE COMMAND)
;
PORT1:  IN      STATUS      ; READ UPI STATUS
        ANI     IBF         ; LOOK AT IBF
        JNZ     PORT1       ; WAIT UNTIL IBF = 0
        MVI     A, 00000010B    ; LOAD WRITE PORT1 CMD
        OUT     UPICMD      ; OUTPUT TO UPI COMMAND PORT
P1:     IN      STATUS      ; READ UPI STATUS AGAIN
        ANI     IBF         ; LOOK AT IBF
        JNZ     P1          ; WAIT UNTIL COMMAND ACCEPTED
        MOV     A, C        ; GET DATA FROM C
        OUT     DBBIN       ; OUTPUT TO DBBIN
        RET                 ; DONE, RETURN
```

Figure 10B. 8085A Dual Output Port Example Code

```
;
; UPI DUAL INPUT PORT EXAMPLE - BOTH PORT 1 AND 2 INPUTS
;       COMMAND SELECTS WHICH PORT IS TO BE READ
;       FLAG 0 USED AS ERROR FLAG
;
RESET:  JNIBF   RESET       ; WAIT FOR INPUT
        CLR     F0          ; CLEAR ERROR FLAG
        IN      A, DBB      ; READ INPUT (COMMAND)
        JB1     PT1         ; TEST BIT 1 (PORT1)
        JB2     PT2         ; TEST BIT 2 (PORT2)
ERROR:  CPL     F0          ; ERROR - COMPLEMENT F0
        JMP     RESET       ; WAIT FOR INPUT
PT1:    IN      A, P1       ; READ PORT 1
        JOBF    ERROR       ; TEST OBF BEFORE LOADING DBBOUT
        OUT     DBB, A      ; LOAD PORT1 DATA INTO DBBOUT
        JMP     RESET       ; WAIT FOR INPUT
PT2:    IN      A, P2       ; READ PORT 2
        JOBF    ERROR       ; TEST OBF BEFORE LOADING DBBOUT
        OUT     DBB, A      ; LOAD PORT2 DATA INTO DBBOUT
        JMP     RESET       ; WAIT FOR INPUT
```

Figure 11. Dual Input Port Example

The UPI also supports a DMA transfer interface. If an EN DMA instruction is executed, Port 2 pin 6 becomes a DMA Request (DRQ) output and P27 becomes a high impedance DMA Acknowledge ($\overline{DACK}$) input. Any instruction which would normally set P26 now sets DRQ. DRQ is cleared when $\overline{DACK}$ is low and either $\overline{RD}$ or $\overline{WR}$ is low. When $\overline{DACK}$ is low, $\overline{CS}$ and A0 are forced low internally which allows data bus transfers between DBBOUT or DBBIN to occur, depending upon whether $\overline{WR}$ or $\overline{RD}$ is true. Of course, the function requires the use of an external DMA controller.

Now that we have discussed the aspects of the UPI protocol and data transfer interfaces, let's move on to the actual applications.

EXAMPLE APPLICATIONS

Each of the following three sections present the hardware and software details of a UPI application. Each application utilizes one of the protocols mentioned in the last section. The first example is a simple 8-digit LED display controller. This application requires only that the UPI perform input operations from the DBBIN; DBBOUT is not used. The reverse is true for the second

application: a sensor matrix controller. The final application involves both DBBOUT and DBBIN operations: a combination serial/parallel I/O device.

The core master processor system with which these applications were developed is the iSBC 80/30 single board computer. This board provides an especially convenient UPI environment since it contains a dedicated socket specifically interfaced for the UPI-41A. The 80/30 uses the 8085A as the master processor. The I/O and peripheral compliment on the 80/30 include 12 vectored priority interrupts (8 on an 8259 Programmable Interrupt Controller and 4 on the 8085A itself), an 8253 Programmable Interval Timer supplying three 16-bit programmable timers (one is dedicated as a programmable baud rate generator), a high speed serial channel provided by a 8251 Programmable USART, and 24 parallel I/O lines implemented with an 8255A Programmable Parallel Interface. The memory compliment contains 16K bytes of RAM using 2117 16K bit Dynamic RAMs and the 8202 Dynamic RAM Controller, and up to 8K bytes of ROM/EPROM with sockets compatible with 2716, 2758, or 2332 devices. The 80/30's RAM uses a dual port architecture. That is, the memory can be considered a global system resource, accessible from the on-board 8085A as well as from remote CPUs and other devices via the MULTIBUS. The 80/30 contains MULTIBUS control logic which allows up to 16 80/30s or other bus masters to share the same system bus. (More detailed information on the iSBC 80/30 and other iSBC products may be found in the latest Intel Systems Data Catalog.)

A block diagram of the iSBC 80/30 is shown in Figure 12. Details of the UPI interface are shown in Figure 13. This interface decodes the UPI registers in the following format:

Register	Operations
Read STATUS	IN E5H
Write DBBIN (command)	OUT E5H
Read DBBOUT (data)	IN E4H
Write DBBIN (data)	OUT E4H

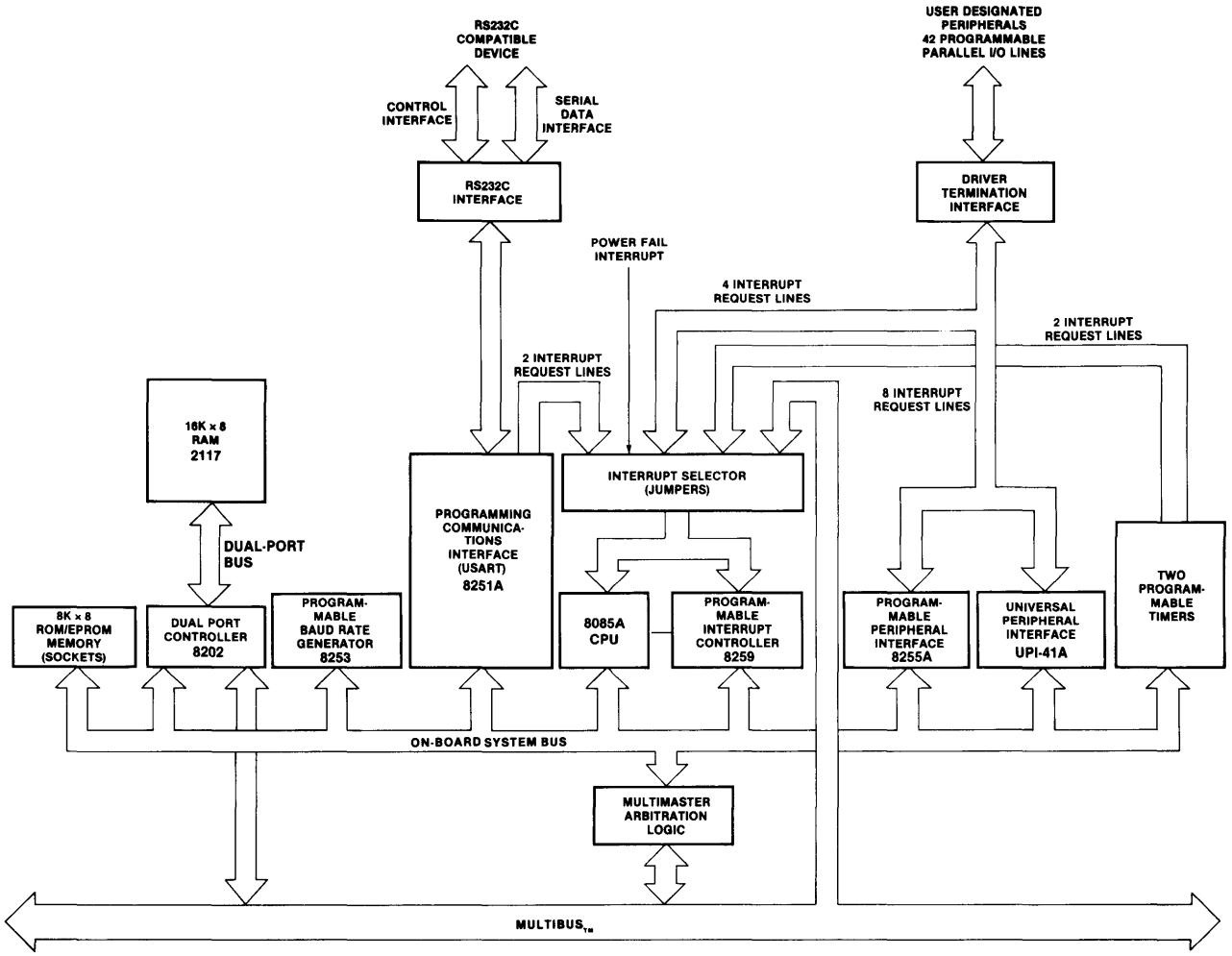


Figure 12. iSBC 80/30 Block Diagram

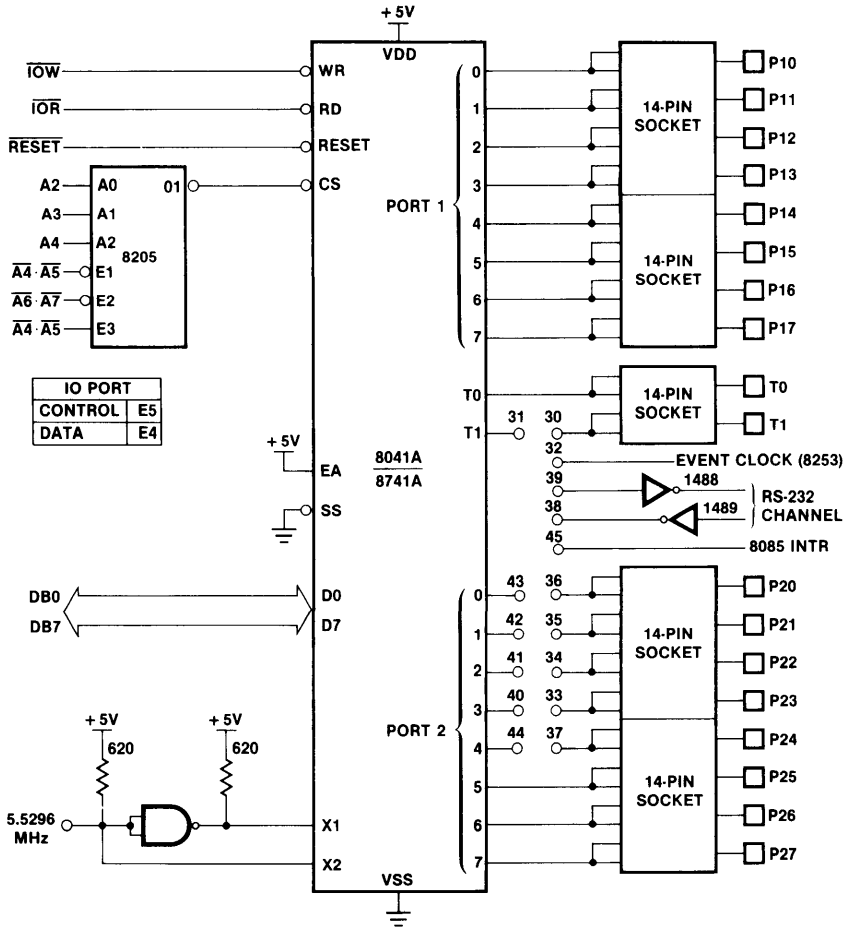


Figure 13. UPI Interface on iSBC 80/30

8-Digit Multiplexed LED Display

The traditional method of interfacing an LED display with a microprocessor is to use a data latch along with a BDC-to-7-segment decoder for each digit of the display. Thus two ICs, seven current limiting resistors, and about 45 connections are required for each digit. These requirements are, of course, multiplied by the total number of digits desired. The obvious disadvantages of this method are high parts count and high power dissipation since each digit is "ON" continuously. Instead, a scheme of time multiplexing the display can be used to decrease both parts count and power dissipation.

Display multiplexing basically involves connecting the same segment (a, b, c, d, e, f, or g) of each digit in parallel and driving the common digit element (anode or cathode) of each digit separately. This is shown schematically in Figure 14. The various digits of the display are not all on at once; rather, only one digit at a time is energized. As each digit is energized, the appropriate segments for that digit are turned on. Each digit is enabled in this way, in sequence, at a rate fast enough to ensure that each digit appears to be "ON" continuously. This implies that the display must be "refreshed" at periodic intervals to keep the digits flicker-free. If the CPU had to handle this task, it would have to suspend normal processing, go update the display, and then return to its normal flow. This extra burden is ideally handled by a UPI. The master CPU could simply give characters to the UPI and let the UPI do the actual segment decoding, display multiplexing, and refreshing.

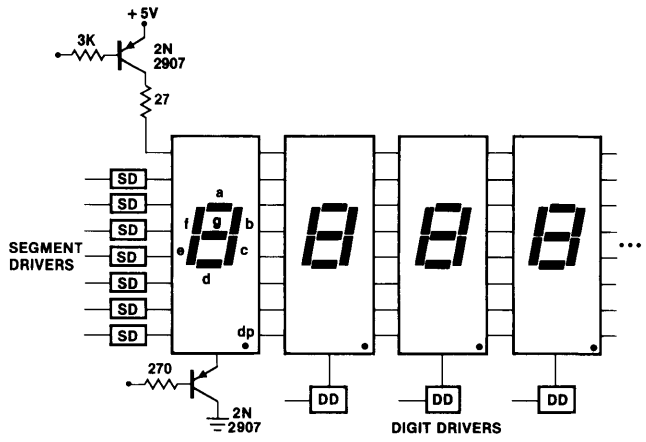


Figure 14. LED Multiplexing

As an example of this technique, Figure 15 shows the UPI controlling an 8-digit LED display. All digit segments are connected in parallel and are driven through segment drivers by the UPI's Port 1. The lower 3 bits of Port 2 are inputs to a 3-to-8 decoder which selects an individual digit through a digit driver. A fourth Port 2 line is used as a decoder enable input. The remaining Port 2 lines plus the T0 and T1 inputs are available for other tasks.

Internally, the UPI uses the counter/timer in the interval timer mode to define the interval between display refreshes. Once the timer is loaded with the desired interval and started, the UPI is free to handle other tasks. It is only when a timer overflow interrupt occurs that the UPI handles the short display multiplexing routine. The display multiplexing can be considered a background task which is entirely interrupt-driven. The amount of time spent multiplexing is such that there is ample time to handle a non-timer task in the UPI foreground. (We'll discuss this timing shortly.)

When a timer interrupt occurs, the UPI turns off all digits via the decoder enable. The next digit's segment contents are retrieved from the internal data memory and output via Port 1 to the segment drivers. Finally, the next digit's location is placed on Port 2 (P20-P22) and the decoder enabled. This displays the digit's segment information until the next interrupt. The timer is then restarted for the next interval. This process continues repeatedly for each digit in sequence.

As a prelude to discussing the UPI software, let's examine the internal data memory structure used in this application, Figure 16. This application requires only 14 of the 64 total data memory locations. The top eight locations are dedicated to the Display Map; one location for each digit. These locations contain the segment and decimal point information for each character. Just how characters are loaded into this section of memory is covered shortly. Register R7 of Register Bank 1 is used as the temporary Accumulator store during the interrupt service routines. Register R3 stores the digit number of the next digit to be displayed. R2 is a temporary storage register for characters during the character input routine. R0 is the offset pointer pointing to the Display Map location of the next digit. That makes 12 locations so far. The remaining two locations are the two stack locations required to store the return address plus status during the timer and input interrupt service routines. The remaining unused locations, all of Register Bank 0, 14 bytes of stack, 4 in Register Bank 1, and 24 general purpose RAM locations, are all available for use by any foreground task.

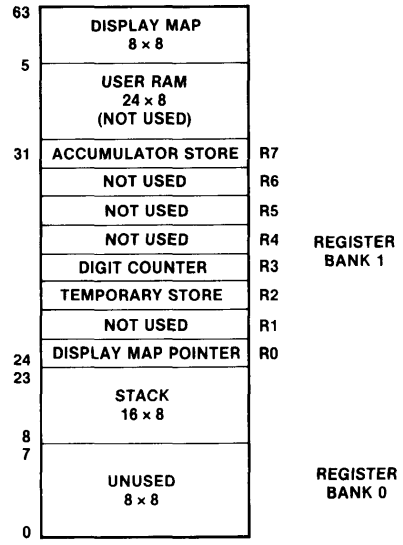


Figure 16. LED Display Controller Data Memory Allocation

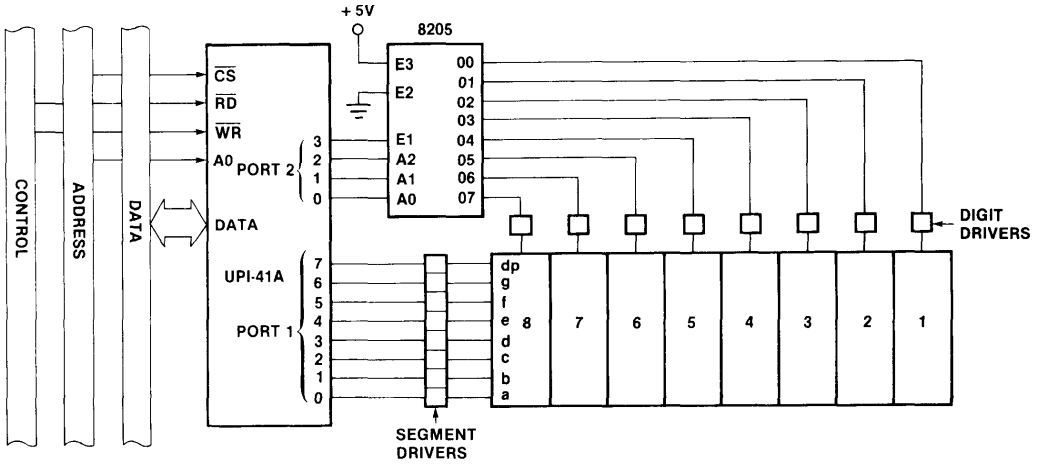


Figure 15. UPI Controlled 8-Digit LED Display

The UPI software consists of only three short routines. One, INIT, is used strictly during initialization. DISPLA is the multiplexing routine called at a timer interrupt. INPUT is the character input handler called at an IBF interrupt. The flow charts for these routines are shown in Figures 17A thru 17C.

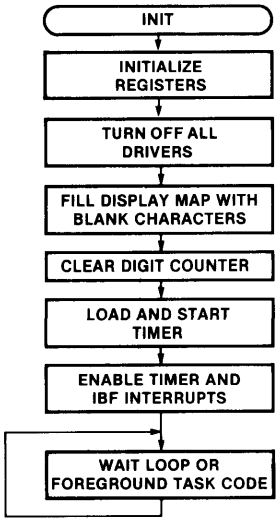


Figure 17A. INIT Routine Flow

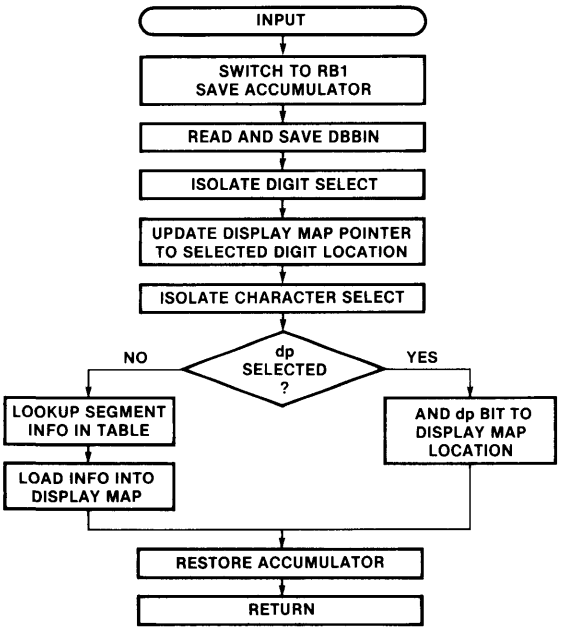


Figure 17B. INPUT Routine Flow

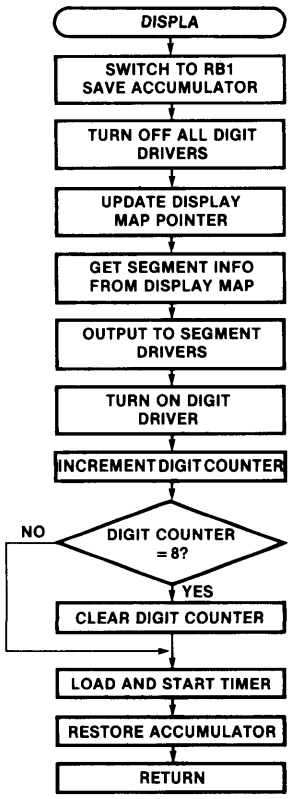


Figure 17C. DISPLA Routine Flow

INIT initializes the UPI by simply turning off all segment and digit drivers, filling the Display Map with blank characters, loading and starting the timer, and enabling both timer and IBF interrupts. Although the flow chart shows the program looping at this point, it is here that the code for any foreground task is inserted. The only restrictions on this foreground task are that it not use I/O lines dedicated to the display and that it not require dedicated use of the timer. It could share the timer if precautions are taken to ensure that the display will still be refreshed at the required interval.

The INPUT routine handles the character input. It is called when an IBF interrupt occurs. After the usual swapping of register banks and saving of the accumulator, DBBIN is read and stored in register R2. DBBIN contains the Display Data Word. The format for this word, Figure 18, has two fields: Digit Select and Character Select. The Digit Select field selects the digit number into which the character from the Character Select field is placed. Notice that the character set is not limited strictly to numerics, some alphanumeric capability is provided. Once DBBIN is read, the offset for the selected digit is computed and placed in the Display Map Pointer R0. Next the segment information for the selected character is found through a look-up table starting in page 3 of the program memory. This segment information is then stored at the location pointed at by the Display Map Pointer. If the Character Select field specified a decimal point, the segment correponding the decimal point is ANDed into the present segment information for that digit. After the accumulator is restored, execution is returned to the main program.

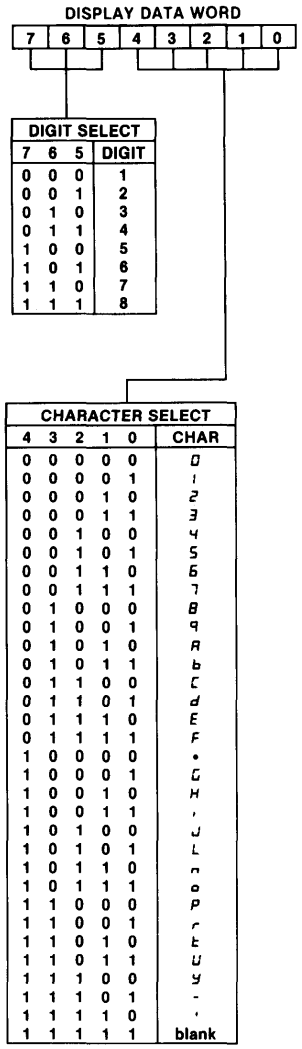


Figure 18. LED Display Controller Display Data Word Format

The DISPLA routine simply implements the multiplexing actions described earlier. It is called whenever a timer interrupt occurs. After saving pre-interrupt status by switching register banks and storing the Accumulator, all digit drivers are turned off. The Display Map Pointer is then updated using the Current Digit Register to point at that digit's segment information in the Display Map. This information is output to Port 1; the segment drivers. The number of the current digit, R3, is then sent to the digit select decoder and the decoder is enabled. This turns on the current digit. The digit counter is incremented and tested to see if all eight digits have been refreshed. If so, the digit counter is reset to zero. If not, nothing is done. Finally, the timer is loaded and restarted, the Accumulator is restored, and the routine returns execution to the main program. Thus DISPLA refreshes one digit each time it is CALLed by the timer interrupt. The digit remains on until the next time DISPLA is executed.

The UPI software listing is included as Appendix A1. Appendix A2 shows the 8085A test routine used to display the contents of a display buffer on the display. The 8085A software takes care of the display digit numbering. Since the application is input-only for the UPI, the only protocol required is that the master must test IBF before writing a Display Data Word into DBBIN.

On the iSBC 80/30, the UPI frequency is at 5.5296 MHz. To obtain a flicker-free display, the whole display must be refreshed at a rate of 50 Hz or greater. If we assume a 50 Hz refresh rate and an 8-digit display, this means the DISPLA routine must be CALLed 50×8 or 400 times/sec. This translates, using the timer interval of 87 μs at 5.5296 MHz, to a timer count of 227. (Recall from the UPI-41 User's Manual that the timer is an "8-bit up-counter".) Hence the TIME equate of 227D in the UPI listing. Obviously, different frequency sources or display lengths would require that this equate be modified.

With the UPI running at 5.5296 MHz, the instruction cycle time is 2.713 μs. The DISPLA routine requires 28 instruction cycles, therefore, the routine executes in 76 μs. Since DISPLA is CALLed 400 times/sec, the total time spent refreshing the display during one second is then 30 ms or 3% of the total UPI time. This leaves 97.0% for any foreground tasks that could be added.

While the basic UPI software is useful just as it stands, there are several enhancements that could be incorporated depending on the application. Auto-incrementing of the digit location could be added to the input routine to alleviate the need for the master to keep track of digit numbers. This could be (optionally) either right-handed or left-handed entry a la TI or HP calculators. The character set could be easily modified by simply changing the lookup table. The display could be expanded to 16 digits at the expense of one additional Port 2 digit select line, the replacement of the 3-to-8 decoder with a 4-to-16 decoder, and 8 more Display Map locations.

Now let's move on to a slightly more complex application that is UPI output-only — a sensor matrix controller.

Sensor Matrix Controller

Quite often a microprocessor system is called upon to read the status of a large number of simple SPST switches or sensors. This is especially true in a process or industrial control environment. Alarm systems are also good examples of systems with a large sensor population. If the number of sensors is small, it might be reasonable to dedicate a single input port pin for each sensor. However, as the number of sensors increase, this technique becomes very wasteful. A better arrangement is to configure the sensors in a matrix organization like that shown in Figure 19. This arrangement of 16 sensors requires only 4 input and 4 output lines; half the number needed if dedicated inputs were used. The line saving becomes even more substantial as the number of sensors increases.

In Figure 19, the basic operation of the matrix involves scanning individual row select lines in sequence while reading the column return lines. The state of any particular sensor can then be determined by decoding the row and column information. The typical configuration

pulls up the column return lines and the selected row is held low. Deselected rows are held high. Thus a return line remains high for an open sensor on the selected row and is pulled low for a closed sensor. Diode isolation is used to prevent a phantom closure which would occur when a sensor is closed on a selected row and there are two or more closures on a deselected row. Germanium diodes are used to provide greater noise margin at the return line input.

If the main processor was required to control such a matrix it would periodically have to output at the row port and then read the column return port. The processor would need to maintain in memory a map of the previous state of the matrix. A comparison of the new return information to the old information would then be made to determine whether a sensor change had occurred. Any changes would be processed as needed. A row counter and matrix map pointer also require maintenance each scan. Since in most applications sensors change very slowly compared to most processing actions, the processor probably would scan the rows only periodically with other tasks being processed between scans.

Rather than require the processor to handle the rather mundane tasks of scanning, comparing, and decoding the matrix, why not use a dedicated processor? The UPI is perfect.

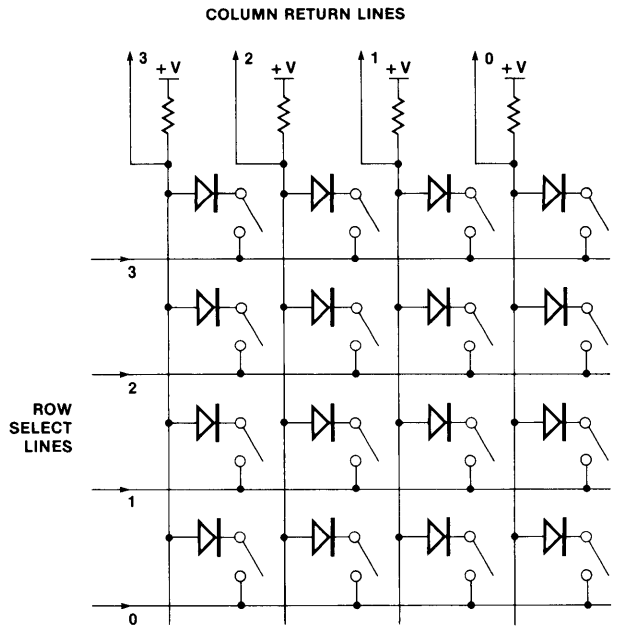


Figure 19. 4 x 4 Sensor Matrix

Figure 20 shows a UPI configuration for controlling up to 128 sensors arranged in a 16 x 8 matrix. The 4-to-16 line decoder is used as the row selector to save port pins and provides the expansion to 128 sensors over the maximum of 64 sensors if the port had been used directly. It also helps increase the port drive capability. The column return lines go directly into Port 1. Features of this design include complete matrix management. As the UPI scans the matrix it compares its present status to the previous scan. If any change is detected, the location of the change is decoded and loaded, along with the sensor's present state, into DBBOUT. This byte is called a Change Word. The Master processor has only to read one byte to determine the status and coordinate of a changed sensor. If the master had not read a previous Change Word in DBBOUT (OBF = 1) before a new sensor change is detected, the new Change Word is loaded into an internal FIFO. This FIFO buffers up to 40 changes before it fills. The status of the FIFO and OBF is made available to the master either by polling the UPI STATUS register, Figure 21A, or as interrupt sources on port pins P24 and P25 respectively, Figure 20. The FIFO NOT EMPTY pin and bit are true as long as there are changes not yet read in the FIFO. As long as the FIFO is not empty, the UPI monitors OBF and loads new Change Words from the FIFO into DBBOUT. Thus, the UPI provides complete FIFO management.

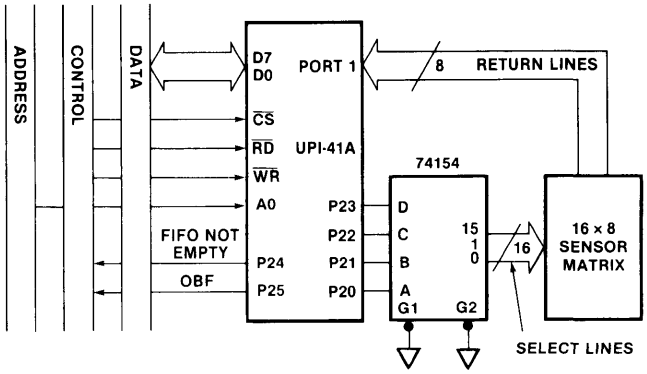


Figure 20. 128 Sensor Matrix Controller

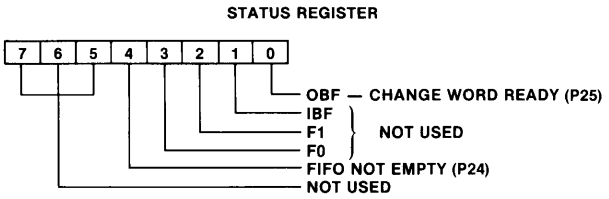


Figure 21A. Sensor Matrix Status Register Format

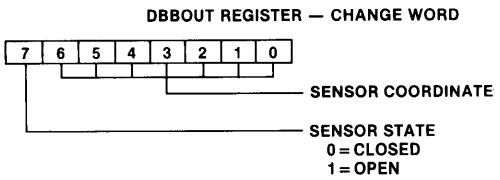


Figure 21B. Sensor Matrix Change Word Format

Internally, the matrix scanning software is programmed to run as a foreground task. This allows the timer/counter to be used by any background task although the hardware configuration leaves only 2 inputs (T0 and T1) plus 2 I/O port pins available. Also, to add a background task, the FIFO would have to be made smaller to accommodate the needed register and data memory space. (It would be possible however to turn the table here and make the scanning software timer/counter interrupt-driven where the timer times the scan interval.)

The data memory organization for this application is shown in Figure 22. The upper 16 bytes form the Matrix Map and store the sensor states from the previous scan; one bit for each sensor. The Change Word FIFO occupies the next 40 locations. (The top and bottom addresses of this FIFO are treated as equate variables in the program so that the FIFO size may easily be changed to accommodate the register needs of other tasks.) Register R0 serves as a pointer into the matrix map area for comparisons and updates of the sensor status. R1 is a general FIFO pointer. The FIFO is implemented as a circular buffer with In and Out pointer registers which are stored in R4 and R5 respectively. These registers are moved into FIFO pointer R1 for actual transfers into or out of the FIFO. R2 is the Row Select Counter. It stores the number of the row being scanned.

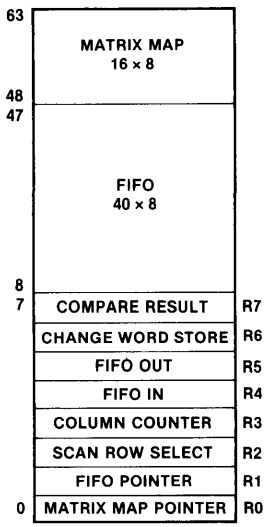


Figure 22. Sensor Matrix Data Memory Map

Register R3 is the Column Counter. This counter is normally set to 00H; however, when a change is detected somewhere in a particular row, it is used to inspect each sensor status bit individually for a change. When a changed sensor bit is found, the Row Select Counter and Column Counter are combined to give the sensor's matrix coordinate. This coordinate is temporarily stored in the Change Word Store, register R6. Register R7 is the Compare Result. As each row is scanned, the return information is Exclusive-OR'd with the return information from the previous scan of that row. The result of this operation is stored in R7. If R7 is zero, there have been no changes on that row. A non-zero result indicates at least one changed sensor.

The basic program operation is shown in the flow chart of Figure 23. At RESET, the software initializes the working registers, the ports, and clears the STATUS register. To get a starting point from which to perform the sensor comparisons, the current status of the matrix is read and stored in the Matrix Map. At this point, the UPI begins looking for changed sensors starting with the first row.

Before delving further into the flow, let's pause to describe the general format of the operation. The UPI scans the matrix one row at a time. If no changes are detected on a particular row, the UPI simply moves to the next row after checking the status of DBBOUT and the FIFO. If a change is detected, the UPI must check each bit (sensor) within the row to determine the actual sensor location. (More than one sensor on the scanned row could have changed.) Rather than test all 8 bits of the row before checking the DBBOUT and FIFO status again, the UPI performs the status check in between each of the bit tests. This ensures the fastest response to the master reading previous Change Words from DBBOUT and the FIFO.

With this general overview in mind, let's go first thru the flow chart assuming we are scanning a row where no changes have occurred. Starting at the Scan-and-Compare section, the UPI first checks if the entire matrix has been scanned. If it has, the various pointers are reset. If not, the address of the next row is placed on Port 20 thru 23. This selects the desired row. The state of the row is then read on Port 1; the column return lines. This present state is compared to the previous state by retrieving the previous state from the matrix map and performing an Exclusive-OR with the present state. Since we are assuming that no change has occurred, the result is zero. No coordinate decoding is needed and the flow branches to the FIFO-DBBOUT Management section.

The FIFO-DBBOUT Management section simply maintains the FIFO and loads DBBOUT whenever Change Words are present in the FIFO and DBBOUT is clear (OBF = 0). The section first tests if the FIFO is full. (If we assume our "no-change" row is the first row scanned, the FIFO obviously would not be full.) If it is, the UPI waits until OBF = 0, at which point the next Change Word is retrieved from the FIFO and placed in DBBOUT. This "unfills" the FIFO making room for more Change Words. At this point, the Column Counter, R3, is checked. For rows with no changes, the Column

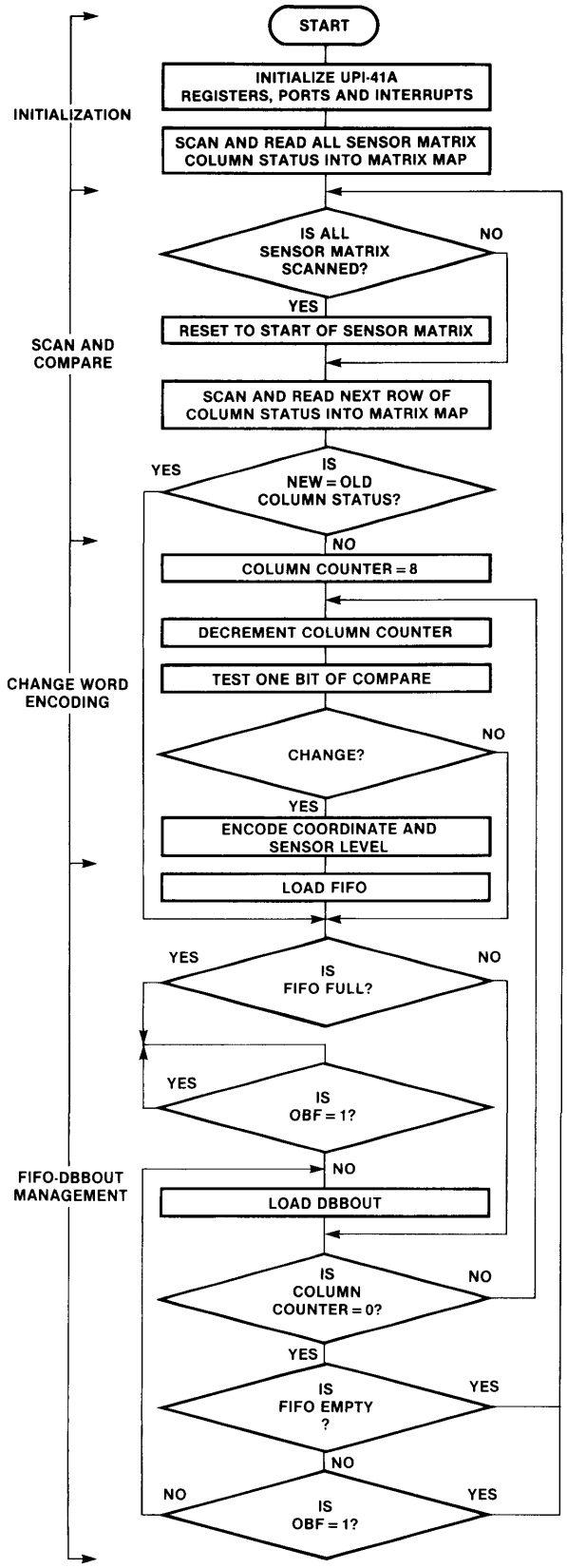


Figure 23. Sensor Matrix Controller Flow Chart

Counter is always zero so the test simply falls through. (We cover the case for changes shortly.) Now the FIFO is tested for being empty. If it is, there is no sense in any further tests so the flow simply goes back up to scan the next row. If the FIFO is not empty, DBBOUT is tested again through OBF. If a Change Word is in DBBOUT waiting for the master to read it, nothing can be done and the flow likewise branches up for the next row. However, if the DBBOUT is free and remembering that the previous test showed that the FIFO was not empty, DBBOUT is loaded with the next Change Word and the last two conditional tests repeat.

Now let's assume the next row contains several changed sensors. Like before, the row is selected, the return lines read, and the sensor status compared to the previous scan. Since changes have occurred, the Exclusive-OR result is now non-zero. Any 1s in the result reflect the positions of the changed sensors. This non-zero result is stored in the Compare Result register, R7. At this point, the Column Counter is preset to 8. To determine the changed sensors' locations, the Compare Result register is shifted bit-by-bit to the left while decrementing the Column Counter. After each shift, bit 7 of the result is tested. If it is a one, a changed sensor has been found. The Column Counter then reflected the sensor's matrix column position while the Scan Row Select register holds it row position. These registers are then combined in R6, the Change Word Store, to form the sensor's matrix coordinate section of the Change Word. The 8th bit of the Change Word Store is coded with the sensor's present state (Figure 21). This byte forms the complete Change Word. It is loaded into the next available FIFO position. If bit 7 of the Compare Result had been a zero, that particular sensor had not changed and the coordinate decoding is not performed.

In between each shift, test, and coordinate encode (if necessary), the FIFO-DBBOUT Management is performed. It is the Column Counter test within this section that routes the flow back up to the Change Word Encoding section if the entire Compare Result (row) has not been shifted and tested.

The FIFO is implemented as a circular buffer with IN and OUT pointers (R4 and R5 respectively). The operations of the FIFO is best understood using an example, Figure 24. This series of figures show how the FIFO, DBBOUT, and OBF interact as changes are detected and Change Words are read by the master. The letters correspond to sequential Change Words being loaded into the FIFO. Note that the figures show only a 4 x 8 FIFO however, the principles are the same in the 40 x 8 FIFO.

Figure 24A shows the condition where no Change Words have been loaded into the FIFO or DBBOUT. In Figure 24B a change, "A", has been detected, decoded, and loaded into the FIFO at the location equal to the value of the FIFO-IN pointer. The FIFO-IN pointer is then incremented and the FIFO-OUT pointer is reset to the bottom of the FIFO since it had reached the FIFO top. Now that a Change Word is in the FIFO, OBF is checked to see if DBBOUT is empty. Because OBF = 0, DBBOUT is empty and the Change Word is loaded from the FIFO location pointed at by the FIFO-OUT pointer. This is shown in Figure 24C. Loading DBBOUT automatically

sets OBF. OBF remains set until the master reads DBBOUT. Figures 24D and 24E show two more Change Words loaded into the FIFO. In Figure 24F the first Change Word is finally read by the master resetting OBF. This allows the next Change Word to be loaded into DBBOUT. Note that each time the FIFO is loaded, the FIFO-IN pointer increments. Each time DBBOUT is read the FIFO-OUT pointer increments unless there are no more Change Words in the FIFO. Both pointers wrap-around to the bottom once they reach the FIFO top. The remaining figures show more Change Words being loaded into the FIFO. When the entire FIFO fills and DBBOUT can not be loaded (OBF = 1), scanning stops until the master reads DBBOUT making room for more Change Words.

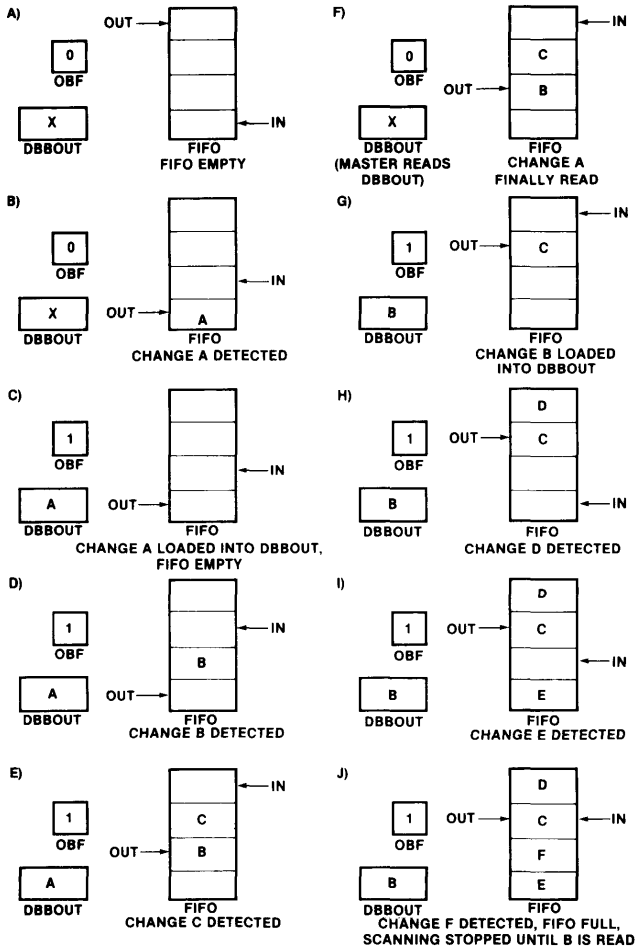


Figure 24A-J. FIFO Operation Example

As was mentioned earlier, two interrrupt outputs to the master are available: Change Word Ready (P25, OBF) and FIFO NOT EMPTY (P24). The Change Word Ready interrupt simply reflects OBF and is handled automatically by the UPI since an EN FLAGS instruction is executed during initialization. The FIFO NOT EMPTY interrupt is generated and cleared as appropriate, each pass through the FIFO management code.

No debouncing is provided although it could be added. Rather, the scan time is left as an equate variable so that it could be varied to account for both debounce time and expected sensor change rates. The minimum scan time for this application is 2 msec when using a 6 MHz clock. Since the matrix controller is coded as a foreground task, scan time simply uses a software delay loop.

The UPI software is included as Appendix B1. Appendix B2 is 8085A test software which builds a Change Word buffer starting at BUFSRT. This software simply polls the STATUS register looking for Change Word Ready to go true. DBBOUT is then read and loaded into the buffer. Now let's move on to an application which combines both the foreground and background concepts.

Combination I/O Device

The final UPI application was designed especially to add additional serial and parallel I/O ports to the iSBC 80/30. This UPI simulates a full-duplex UART (Universal Asynchronous Receiver/Transmitter) combined with an 8-bit parallel I/O port. Features of the UART include: software selectable baud rates (110, 300, 600, or 1200 baud), double buffering for both the transmitter and receiver, and receiver testing for false state bit, framing, and overrun errors. For parallel I/O, one 8-bit port is programmable for either input or output. The output port is statically latched and the input port is sampled.

Figure 25 shows the interface of this combination I/O device to the dedicated UPI socket on the iSBC 80/30. The only external requirement is a 76.8 kHz source which serves as the baud rate standard. The internal baud rates are generated as multiples of this external clock. This clock is obtained from one of the 8253 counters. Otherwise, an RS-232 driver and receiver already available for UPI use in serial I/O applications. Sockets are also provided for termination of the parallel port.

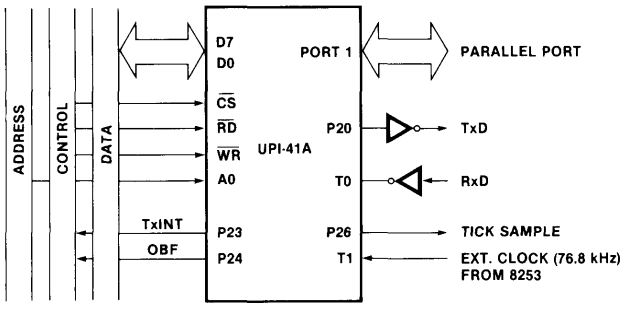


Figure 25. Combination I/O Device

There are three commands for this application. Their format is shown in Figure 26. The CONFIGURE command specifies the serial baud rate and the parallel I/O direction. Normally this command is issued once during system initialization. The I/O command causes a parallel I/O operation to be performed. If the parallel port direction is output, the UPI expects the data byte immediately following an I/O command to be data for the output port. If the port is in the input direction, an I/O command causes the port to be read and the data placed in DBBOUT. The RESET ERROR command resets the serial receiver error bits in the STATUS register.

The STATUS register format is shown in Figure 27. Looking at each bit, bit 0 (OBF) is the DATA AVAILABLE flag. It is set whenever the UPI places data into DBBOUT. Since the data may come from either the receiver or the parallel input port, the F0 and F1 flags (bits 2 and 3) code the source. Thus, when the master finds OBF set, it must decode F0 and F1 to determine the source.

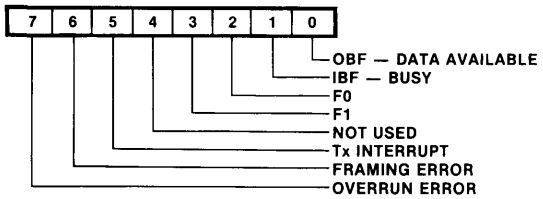
Bit 1 (IBF) functions as a busy bit. When IBF is set, no writes to DBBIN are allowed. Bit 5 is the TxINT (Transmitter Interrupt) bit. It is asserted whenever the transmitter buffer register is empty. The master uses this bit to determine when the transmitter is ready to accept a data character.

COMMAND FORMAT

7	6	5	4	3	2	1	0	
								CONFIGURE COMMAND
0	0	0	A	B	C	D	P	A — 1200 BAUD SELECT B — 600 BAUD SELECT C — 300 BAUD SELECT D — 110 BAUD SELECT P — PARALLEL I/O DIRECTION 0—INPUT 1—OUTPUT
1	0	0	0	0	0	0	0	I/O COMMAND
1	1	0	0	0	0	0	0	RESET ERROR COMMAND

Figure 26. Combination I/O Command Format

STATUS FORMAT



F0	F1	OPERATION (IBF = 1)
0	0	NO OPERATION
0	1	PARALLEL I/O DATA
1	0	SERIAL I/O DATA
1	1	COMMAND ERROR

Figure 27. STATUS Register Format

Bits 6 and 7 are receiver error flags. The framing error flag, bit 6, is set whenever a character is received with an invalid stop bit. Bit 7, overrun error, is set if a character is received before the master has read a previous character. If an overrun occurs, the previous character is overwritten and lost. Once an error occurs, the error flag remains set until reset by a RESET ERROR command. A set error flag does not inhibit receiver operation however.

Figure 28 shows the port pin definition for this application. Port 1 is the parallel I/O port. The UART uses Port 2 and the Test inputs. P20 is the transmitter data out pin. It is set for a mark and reset for a space. P23 is a transmitter interrupt output. This pin has the same timing as the TxINT bit in the STATUS register. It is normally used in interrupt-driven systems to interrupt the master processor when the transmitter is ready to accept a new data character.

The OBF flag is brought out on P24 as a master interrupt when data is available in DBBOUT. P26 is a diagnostic pin which pulses at four times the selected baud rate. (More about this pin later.) The receiver data input uses the T0 input. One of the Port 2 pins could have been used, however, the software can test the T0 in one instruction without first reading a port.

The T1 input is the baud rate external source. The UART divides this input to determine the timing needed for the selected baud rate. The input is a non-synchronous 76.8 kHz source.

Internally, when the CONFIGURE command is received and the selected baud rate is determined, the internal timer/counter is loaded with a baud rate constant and started in the event counter mode. Timer/counter interrupts are then enabled. The baud rate constant is selected to provide a counter interrupt at four times the desired baud rate. At each interrupt, both the transmitter and receiver are handled. Between interrupts, any new commands and data are recognized and executed.

PORT PIN DEFINITION

PORT	BIT	FUNCTION
1	0-7	PARALLEL I/O
2	0	Tx DATA
	1	NOT USED
	2	NOT USED
	3	Tx INTERRUPT
	4	OBF INTERRUPT
	5	NOT USED
	6	NOT USED (TICK SAMPLE)
	7	NOT USED
T0		Rx DATA
T1		EXTERNAL CLOCK (76.8 kHz)

Figure 28. Combination I/O Port Definition

As a prelude to discussing the flow charts, Figure 29 shows the register definition. Register Bank 0 serves the UART receiver and parallel I/O while Register Bank 1 handles the UART transmitter and commands. Looking at RB0 first, R3 is the receiver status register, RxSTS. Reflected in the bits of this register is the current receiver status in sequential order. Figure 30 shows this bit definition. Bit 0 is the Rx flag. It is set whenever a possible start bit is received. Bit 1 signifies that the start bit is good and character construction should begin with the next received bit. Bit 1 is the Good Start flag. Bit 2 is the Byte Finished flag. When all data bits of a character are received, this flag is set. When all the bits, data and stop bits are received, the assembled character is loaded into the holding register (R4 in Figure 29) bit 3, the Data Ready flag, is set. The foreground routine which looks for commands and data continuously, looks at this bit to determine when the receiver has received a character. Bits 4 and 5 signify any error conditions for a particular character.

The parallel I/O port software uses bits 6 and 7. Bit 6 codes the I/O direction specified by the last CONFIGURE command. Bit 7 is set whenever an I/O command is received. The foreground routine tests this bit to determine when an I/O operation has been requested by the master.

As was mentioned, R4 is the receiver holding register. Assembled characters are held in this register until the foreground routine finds DBBOUT free, at which time the data is transferred from R4 to DBBOUT. R5 is the receiver tick counter. Recall that counter interrupts occur at four times the baud rate. Therefore, once a start bit is found, the receiver only needs to look at the data every four interrupts or tick counts. R5 holds the current tick count.

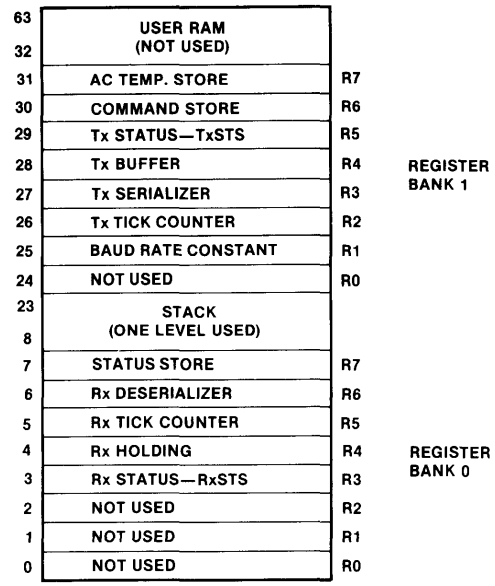


Figure 29. Combination I/O Register Map

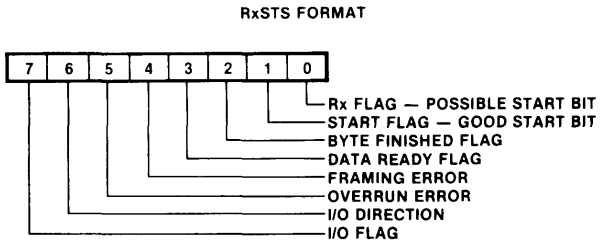


Figure 30. RxSTS Register

R6 is the receiver de-serializing register. Data characters are assembled in this register. R6 is preset to 80H when a good start bit is received. As each bit is sampled every four timer ticks, they are rotated into the leftmost bit of R6. The software knows the character assembly is complete when the original preset bit rotates into the carry.

An image of the upper 4 bits of the STATUS register is stored in R7. These bits are the TxINT, Framing and Overrun bits. This image is needed since the UPI may load the upper 4 STATUS register bits from its accumulator; however, it cannot read STATUS directly.

In Register Bank 1 (Figure 29), R1 holds the baud rate constant which is found from decoding the baud rate select bits of the CONFIGURE command. The counter is reloaded with this constant every timer tick. Like the receiver, the transmitter only needs to update the transmitter output every four ticks. R2 holds the transmitter tick count. The value of R2 determines which portion of the data is being transmitted; start bit, data bits, or stop bit. The transmit serializer is R3. R3 holds the data character as each character bit is transmitted.

R4 is the transmitter holding register. It provides the double buffering for the transmitter. While transmitting one character, it is possible to load the next character into R4 via DBBIN. The TxINT bit in STATUS and pin on Port 2 reflect the "fullness" of R4. If the holding register is empty, the interrupt bit and pin are set. They are reset when the master writes a new data byte for the transmitter into DBBIN. The transmitter Status register (TxSTS) is R5. Like RxSTS, TxSTS contains flag bits which indicate the current state of the transmitter. This flag bit format is shown in Figure 31.

TxSTS bit 0 is the Tx flag. It is set whenever the transmitter is transmitting a character. It is set from the beginning of the start bit until the end of the stop bit. Bit 1 is the Tx Request flag. This bit is set by the foreground routine when it transfers a new character from DBBIN to the Tx Holding register, R4. The transmitter software uses this flag to tell if new data is available. It is reset when the transmitter transfers the character from the holding register to the serializer.

Bit 2 is the Pipelined Tx Data Bit. The transmitter uses a pipelining technique which sets up the next output level in bit 2 after processing the current timer tick. The output level is always changed at the same point after a timer tick interrupt. This technique ensures that no bit timing distortion results from different length processing paths through the receiver and transmitter routines.

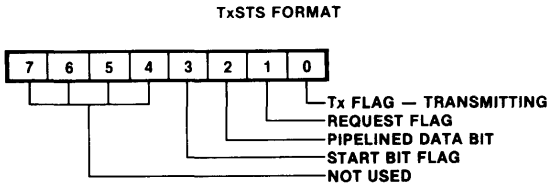


Figure 31. TxSTS Register

Bit 3 of TxSTS is the Start Bit flag. It is set by the transmitter when the start bit space is set up in the Pipelined Data Bit. This allows the transmitter to differentiate between the start bit and data bits on following timer ticks.

The flow charts for this application are shown in Figures 32A-F. At reset, the INIT routine is executed which initializes the registers and port pins. After initialization, IBF and OBF are tested in MNLOOP. These flags are tested continually in this loop. If IBF is set, F1 is tested for command or data and execution is transferred to the appropriate routine (CMD or DATA). If IBF = 0, OBF is checked. If OBF = 0 (DBBOUT is free), the Rx Data Ready and I/O flags in RxSTS are tested. If Rx Data Ready is set, the received data is retrieved from the Rx Holding register and transferred to DBBOUT. Any error flags associated with that data are also transferred to STATUS. If the I/O flag is set and the I/O direction is input, Port 1 is read and the data transferred to DBBOUT. In either case, F0 and F1 are set to indicate the data source.

If IBF is set by a command write to DBBIN, CMD reads the command and decodes the desired operation. If an

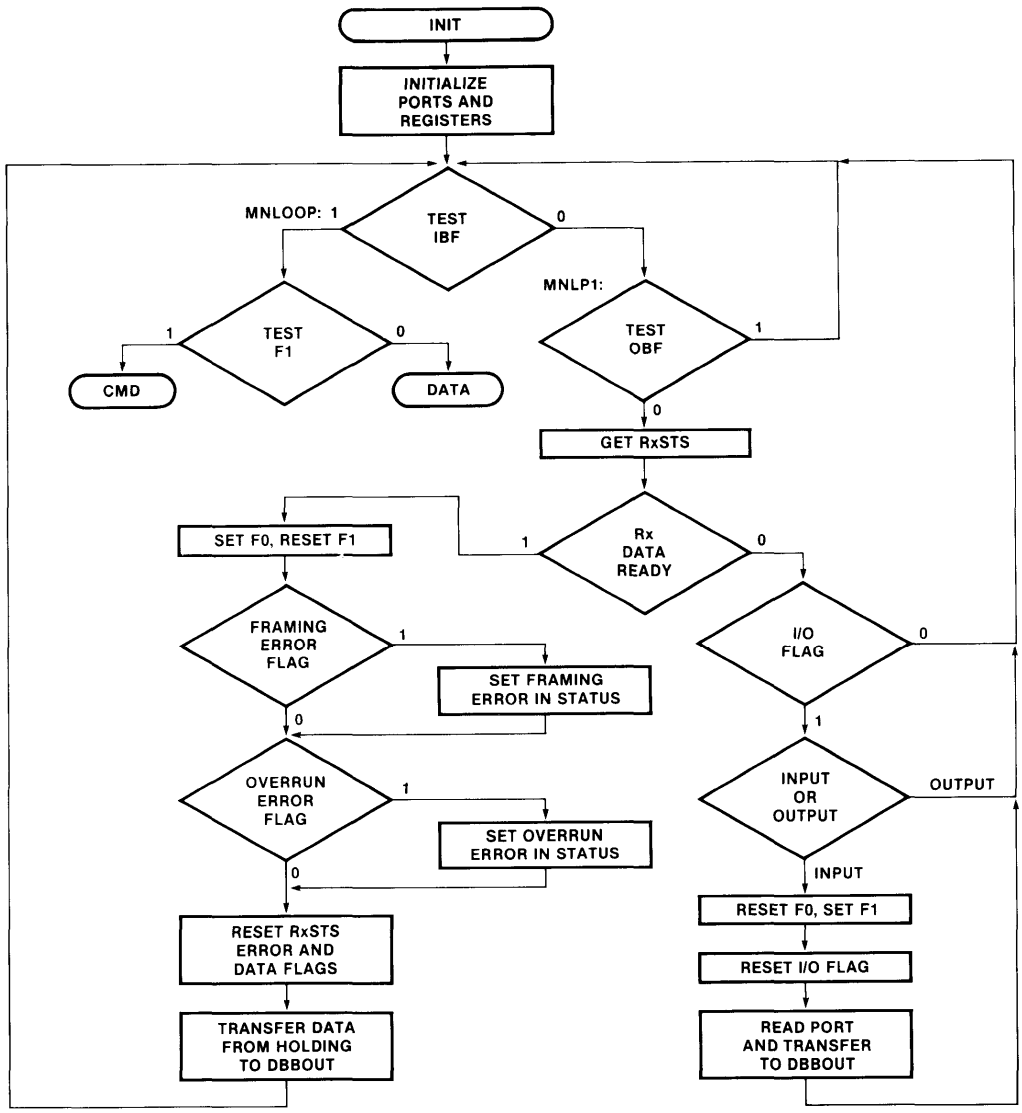


Figure 32A. INIT Flow Chart

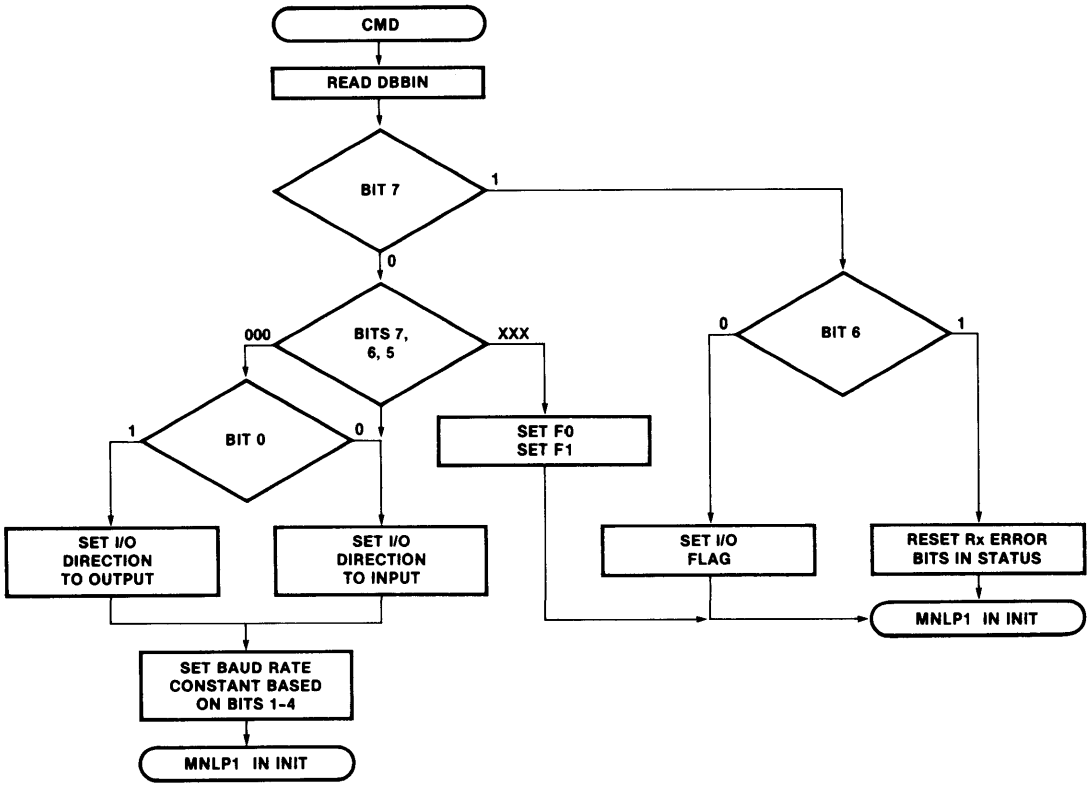


Figure 32B. CMD Flow Chart

I/O operation is specified, the I/O flag is set to indicate to the MNLOOP and DATA routines that an I/O operation is to be performed. If the command is a CONFIGURE command, the constant for the selected baud rate is loaded into both Baud Rate Constant register and the timer/counter. The timer/counter is started in the event counter mode and timer/counter interrupts are enabled. In addition, the I/O port is initialized to all 1's if the I/O direction bit specifies an input port. If the command is a RESET ERROR command, the two error flags in STATUS are cleared.

If the IBF flag is set by a data write, the DATA routine reads DBBIN and places the data in the appropriate place. If the I/O flag is set, the data is for the output port so the port is loaded. If the I/O flag is reset, the data is for the UART transmitter. Data for the transmitter resets the TxINT bit and pin plus sets the Tx Request flag in TxSTS. The data is transferred to the Tx Holding register, R4.

Once a CONFIGURE command is received and the counter started, timer/counter interrupts start occurring at four times the selected baud rate. These interrupts cause a vector to the TIMINT routine, Figure 32D. A 76.8 kHz counter input provides a 13.02 μs counter resolution. Since it requires several UPI instruction cycles to reload the counter, the counter is set to two counts less than the desired baud rate and the counter is reloaded in TIMINT synchronous with the second low-going transition after the interrupt. Once the counter is reloaded, an output port (P26) is toggled to give an external indication of internal counter interval. This is a helpful diagnostic feature. After the tick sample output, the pipelined transmitter data in TxSTS is output to the TxD pin. Although this occurs every timer tick, the pipelined data is changed only every fourth tick.

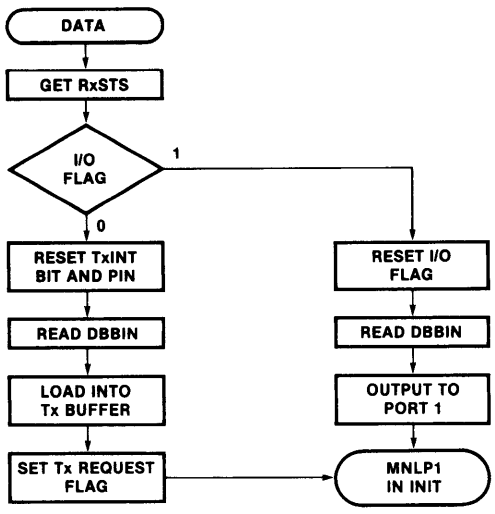


Figure 32C. Data Flow Chart

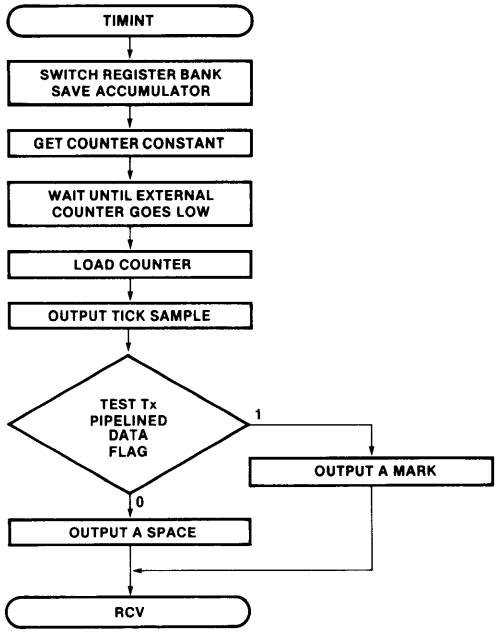


Figure 32D. TIMINT Flow Chart

The receiver is now handled, Figure 32E. The RX flag in RxSTS is examined to see if the receiver is currently in the process of receiving a character. If it is not, the RxD input is tested for a space condition which might indicate a possible start bit. If the input is a mark, no start bit is possible and execution branches to the transmitter flow, XMIT. If the input is a space, the Rx flag is set before proceeding with XMIT.

If the Rx flag is found set when entering RCV, the receiver is in the process of receiving a character. If so, the Start Bit flag is then tested to determine if a good start bit so the Start Bit flag is set, the Rx Tick Counter is initialized to four, and the Rx De-serializer initialized to 80H. A mark indicates a bad start bit so the Rx flag is reset to abort the reception.
start bit so the Start Bit flag is set, the Rx tick counter is initialized to four, and the Rx deserializer initialized to 80H. A mark indicates a bad start bit so the Rx flag is reset to abort the reception.

If the Start Bit flag is set, the program is somewhere in the middle of the received character. Since the data should be sampled every fourth timer tick, the Tick Counter is decremented and tested for zero. If non-zero no sample is needed and execution continues with

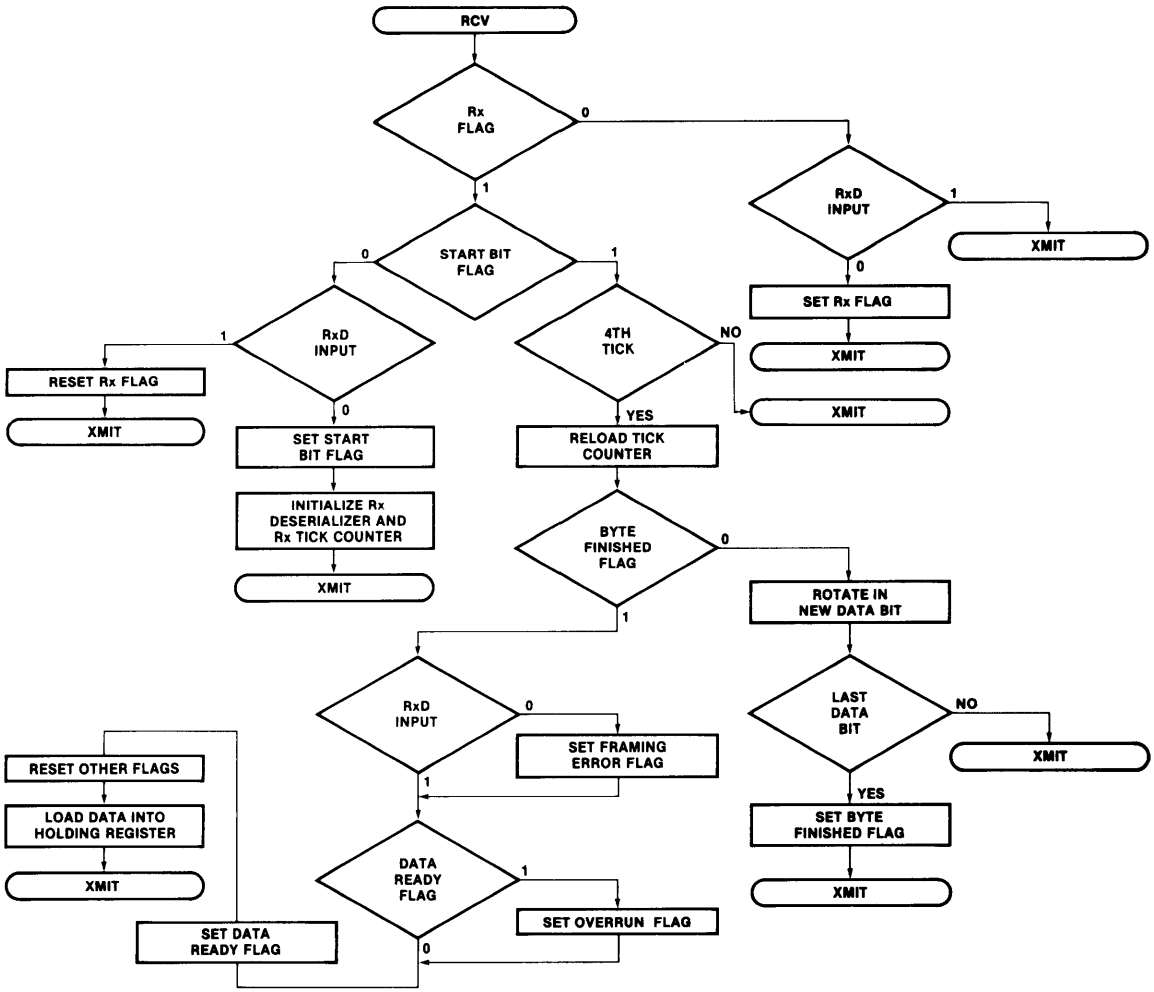


Figure 32E. RCV Flow Chart

XMIT. If zero, the tick counter is reset to four. Now the Byte Finished flag is tested to determine if the data sample is a data or stop bit. If reset, the sample is a data bit. The sample is done and the new bit rotated into the Rx deserializer. If this rotate sets the carry, that data bit was the last so the Byte Finished flag is set. If the carry is reset, the data bit is not the last so execution simply continues with XMIT.

Had the Byte Finished flag been set, this sample is for the stop bit. The RxD input is tested and if a space, the Framing Error flag is set. Otherwise, it is reset. Next, the Rx Data Ready flag is tested. If it is set, the master has not read the previous character so the Overrun Error flag is set. Then the Rx Data Ready flag is set and the received data character is transferred into the Rx Holding register. The Rx, Start Bit, and Byte Finished flags are reset to get ready for the next character.

Execution of the transmitter routine, XMIT, follows the receiver, Figure 32F. The transmitter starts by checking the Start Bit flag in TxSTS. Recall that the actual transmit data is output at the beginning of the timer routine. The Start Bit flag indicates whether the current timer tick interrupt started the start bit. If it is set, the pipelined data output earlier in the routine was the start of the start bit so the flag is reset and the Tx tick counter is initialized. Nothing else is done this timer tick so the routine returns to the foreground.

If the Start Bit flag is reset, the Tx tick counter is incremented and tested. The test is performed module 4. If the counter mod 4 is not zero, it has not been four ticks since the transmitter was handled last so the routine simply returns. If the counter mod 4 is zero, it is time to handle the transmitter and the Tx flag is tested.

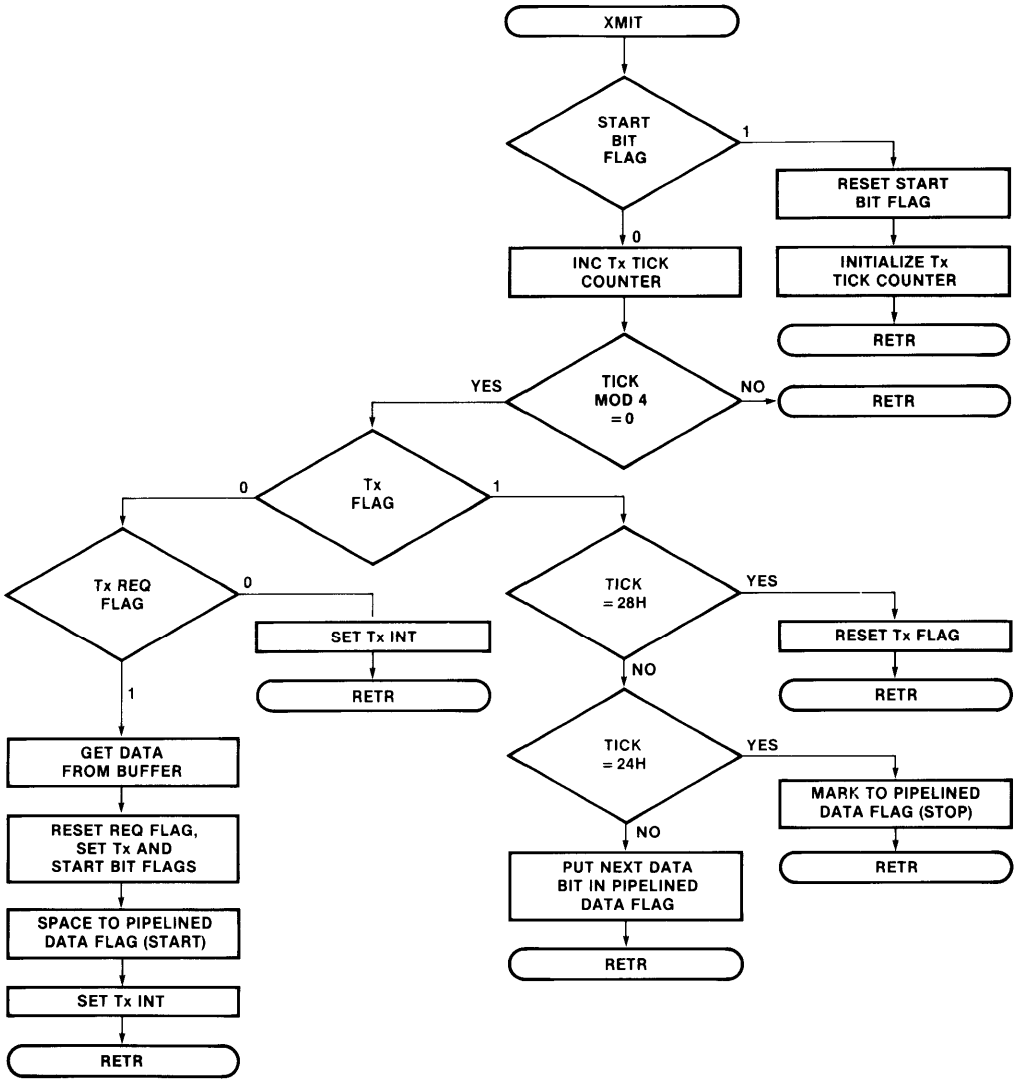


Figure 32F. XMIT Flow Chart

The Tx flag indicates whether the transmitter is active. If the transmitter is inactive, no character is currently being transmitted so the Tx Request flag is tested to see if a new character is waiting in the Tx buffer. If no character is waiting (Tx Request Flag = 0), the Tx interrupt pin and bit are set before returning to the foreground. If there is a character waiting, it is retrieved from the buffer and placed in the Tx serializer. The Tx Request flag is reset while the Tx and Start Bit flags are set. A space is placed in the Tx Pipelined Data bit so a start bit will be output on the next tick. Since the Tx buffer is now empty, the Tx interrupt bit and pin are set to indicate the availability of the buffer to the master. The routine then returns to the foreground.

If the tick counter mod 4 is zero and the Tx flag indicates the transmitter is in the middle of a character, the tick counter is checked to see what transmitter operation is needed. If the counter is 28H (40D), all data bits plus the stop bits are complete. The character is therefore done and the Tx flag is reset. If the counter is 24H (36D), the data bits are complete and the next output should be a mark for the stop bit so a mark is loaded into the Tx Pipelined Data bit.

If neither of the above conditions are met for the counter, the transmitter is some place in the data field, so the next data bit is rotated out of the Tx serializer into the Pipelined Data bit. The next tick outputs this bit.

At this point the program execution is returned to the foreground.

That completes the discussion of the combination I/O device flow charts. The UPI software listing is shown in Appendix C1. Appendix C2 is example 8085A driver software.

Several observations concerning the drivers are appropriate. Notice that since the receiver and input port of the UPI use the OBF flag and interrupt output, the interrupt and flag are cleared when the master reads DBBOUT. This is not true for the transmitter. There is always some time after a master write of new transmitter data before the transmitter interrupt bit and pin are cleared. Thus in an interrupt-driven system, edge-sensitive interrupts should be used. For polled-systems, the software must wait after writing new data for IBF = 0 before re-examining the Tx Interrupt flag in STATUS.

Notice that this application uses none of the user Data Memory above Register Bank 1 and only 361 bytes of Program Memory. This leaves the door open for many improvements. Improvements that come to mind are increased buffering of the transmit or received data, modem control pins, and parallel port handshaking inputs.

This completes our discussion of specific UPI applications. Before concluding, let's look briefly at two debug techniques used during the development of these applications that you might find useful in your own designs.

DEBUG TECHNIQUES

Since the UPI is essentially a single-chip microcomputer, the classical data, address, and control buses are not available to the outside world during normal operation. This fact normally makes debugging a UPI design difficult; however, certain "tricks" can be included in the UPI software to ease this task.

If a UPI is handling multiple tasks, it is usually easier to code and debug each task individually. This is fairly standard procedure. Since each task usually utilizes only a subset of the total number of I/O pins, coding only one task leaves some I/O pins free. Port output instructions can then be added in the task code being debugged which toggle these unused pins to determine which section of task code is being executed at any particular time. The task can also be made to "wait" at various points by using an extra pin as an input and adding code to loop until a particular input condition is met.

One example of using an extra pin as an output is included in the combination serial/parallel device code. During initial development the receiver was not receiving characters correctly. Since this could be caused by incorrect sampling, three lines of code were added to toggle bit 6 of Port 2 at each tick of the sample clock. This code is at lines 184 and 185 of the listing. Thus by looking at the location of the tick sample pulse with respect to the received bit, the UPI sampling interval can be observed. The tick sample time was incorrect and the code was modified accordingly. Similar techniques could be applied at other locations in the program.

The EPROM version of the UPI (8741A) also contains another feature to aid in debug: the capability to single step thru a program. The user may step thru the program instruction-by-instruction. The address of the next instruction to be fetched is available on Port 1 and the lower 2 bits of Port 2. Figure 33 shows the timing used in the discussion below. When the Single Step input, $\overline{SS}$, is brought low, the internal processor responds by stopping during the fetch portion of the next instruction. This action is acknowledged by the processor raising the SYNC output. The address of the instruction to be fetched is then placed on the port pins. This state may be held indefinitely. To step to the next instruction, $\overline{SS}$ is raised high, which causes SYNC to go low, which is then used to return $\overline{SS}$ low. This allows the processor to advance to the next instruction. If $\overline{SS}$ is left high, the processor continues to execute at normal speed until $\overline{SS}$ goes low.

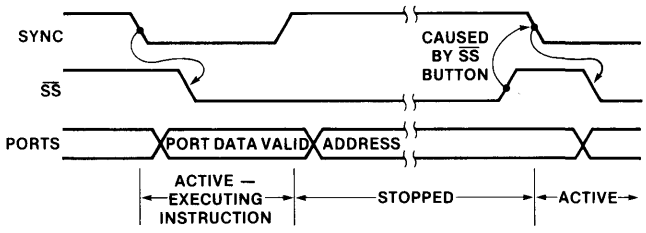


Figure 33. Single Step Timing

To preserve port functionality, port data is valid while SYNC is low. Figure 34 shows the external circuitry required to implement single step while preserving port functionality. S1 is the RUN/STOP switch. When in the RUN position, the 7474 is held preset so $\overline{SS}$ is high and the UPI executes normally. When switched to STOP, the preset is removed and the next low-going transition of SYNC causes the 7474 to clear, lowering $\overline{SS}$. While SYNC is low, the port data is valid and the current instruction is executing. Low SYNC is also used to enable the tri-state buffers when the ports are used as inputs. When execution is complete, SYNC goes high. This transition latches the valid port data in the 74LS374s. SYNC going high also signifies that the address of the next instruction will appear on the port pins. This state can be held indefinitely with the address data displayed on the LEDs.

When the S2 is depressed, the 7474 is set which causes $\overline{SS}$ to go high. This allows the processor to fetch and execute the instruction whose address was displayed. SYNC going low during execution, clears the 7474 lowering $\overline{SS}$. Thus the processor again stops when execution is complete and the next fetch is started.

All UPI functions continue to operate while single stepping (the processor is actuallly executing NOPs internally while stopped). Both IBF and timer/counter interrupts can be serviced. The only change is that the interval timer is prescaled on single stepped instructions and, of course, will not indicate the correct intervals in real time. The total number of instructions which would have been executed during a given interval is the same however.

The single step circuitry can be used to step through a complete program; however, this might be a time-consuming job if the program is long or if only a portion is to be examined. The circuitry could easily be modified to incorporate the output toggling technique to determine when to run and stop. If you would like to step thru a particular section of code, an extra port pin could replace switch S1. Extra instructions would then be added to lower the port when entering the code section and raise the port when exiting the section. The program would then stop when that section of code is reached allowing it to be stepped through. At the end of the section, the program would execute at normal speed.

CONCLUSION

Well, that's it. Machine readable (floppy disk or paper tape) source listings of UPI software for these applications are available in Insite, the Intel library of user-donated programs. Also available in Insite are the source listings for some of Intel's pre-programmed UPI products. These products are:

- 8278 Keyboard Display Controller
- 8295 Dot Matrix Printer Controller.

Other pre-programmed UPIs are the 8294 Data Encryption Unit and the 8292 GPIB (IEEE-488) Controller.

For information about Insite, write to:

Insite
Intel Corp.
3065 Bowers Ave.
Santa Clara, Ca 95051

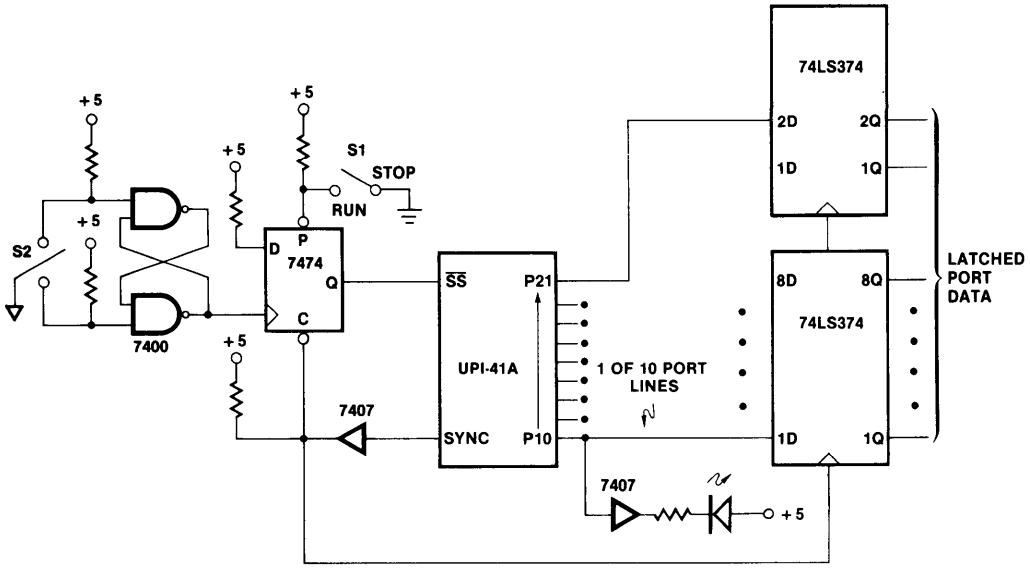


Figure 34. Single Step External Circuitry

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

 LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 ;       ********************************************
                    2 ;       *    UPI-41 8-DIGIT LED DISPLAY CONTROLLER    *
                    3 ;       ********************************************
                    4 ;
                    5 ;
                    6 ;
                    7 ;THIS PROGRAM USES THE UPI-41 AS A LED DISPLAY CONTROLLER
                    8 ;WHICH SCANS AND REFRESHES EIGHT SEVEN-SEGMENT LED DISPLAYS.
                    9 ;THE CHARACTERS ARE DEFINED BY INPUT FROM A MASTER CPU IN THE
                   10 ;FORM OF ONE EIGHT BIT WORD PER DIGIT-CHARACTER SELECTION.
                   11 ;
                   12 ;
                   13 ;
                   14 ;*************************************************************************
                   15 ;
                   16 ;REGISTER DEFINITIONS:
                   17 ;       REGISTER            RB1                     RB0
                   18 ;       --------            ---                     ---
                   19 ;       R0          DISPLAY MAP POINTER             NOT USED
                   20 ;       R1          NOT USED                        NOT USED
                   21 ;       R2          DATA WORD AND CHARACTER STORAGE NOT USED
                   22 ;       R3          DIGIT COUNTER                   NOT USED
                   23 ;       R4          NOT USED                        NOT USED
                   24 ;       R5          NOT USED                        NOT USED
                   25 ;       R6          NOT USED                        NOT USED
                   26 ;       R7          ACCUMULATOR STORAGE             NOT USED
                   27 ;*************************************************************************
                   28 ;
                   29 ;PORT PIN DEFINITIONS:
                   30 ;       PIN         PORT 1 FUNCTION         PORT 2 FUNCTION
                   31 ;       ---         ---------------         ---------------
                   32 ;       P0-7        SEGMENT DRIVER CONTROL  DIGIT DRIVER CONTROL
                   33 ;
                   34 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                    35 ;*********************************************************************
                    36 ;DISPLAY DATA WORD BIT DEFINITION:
                    37 ;       BIT             FUNCTION
                    38 ;       ---             --------
                    39 ;       0-4             CHARACTER SELECT
                    40 ;       5-7             DIGIT SELECT
                    41 ;
                    42 ;CHARACTER SELECT:
                    43 ;               D4  D3  D2  D1  D0  CHARACTER
                    44 ;               0   0   0   0   0      0
                    45 ;               0   0   0   0   1      1
                    46 ;               0   0   0   1   0      2
                    47 ;               0   0   0   1   1      3
                    48 ;               0   0   1   0   0      4
                    49 ;               0   0   1   0   1      5
                    50 ;               0   0   1   1   0      6
                    51 ;               0   0   1   1   1      7
                    52 ;               0   1   0   0   0      8
                    53 ;               0   1   0   0   1      9
                    54 ;               0   1   0   1   0      A
                    55 ;               0   1   0   1   1      B
                    56 ;               0   1   1   0   0      C
                    57 ;               0   1   1   0   1      D
                    58 ;               0   1   1   1   0      E
                    59 ;               0   1   1   1   1      F
                    60 ;               1   0   0   0   0      .
                    61 ;               1   0   0   0   1      G
                    62 ;               1   0   0   1   0      H
                    63 ;               1   0   0   1   1      I
                    64 ;               1   0   1   0   0      J
                    65 ;               1   0   1   0   1      L
                    66 ;               1   0   1   1   0      N
                    67 ;               1   0   1   1   1      O
                    68 ;               1   1   0   0   0      P
                    69 ;               1   1   0   0   1      R
                    70 ;               1   1   0   1   0      T
                    71 ;               1   1   0   1   1      U
                    72 ;               1   1   1   0   0      Y
                    73 ;               1   1   1   0   1      -
                    74 ;               1   1   1   1   0      '
                    75 ;               1   1   1   1   1      "BLANK"
                    76
                    77 ;
                    78 ;DIGIT SELECT:
                    79 ;               D7  D6  D5   DIGIT NUMBER
                    80 ;               0   0   0        1
                    81 ;               0   0   1        2
                    82 ;               0   1   0        3
                    83 ;               0   1   1        4
                    84 ;               1   0   0        5
                    85 ;               1   0   1        6
                    86 ;               1   1   0        7
                    87 ;               1   1   1        8
                    88 ;*********************************************************************
                    89 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                 90 ;*****************************************************************
                 91 ;                        EQUATES
                 92 ;THE FOLLOWING CODE DESIGNATES "TIME" AS A VARIABLE. THIS
                 93 ;ADJUSTS THE AMOUNT OF CYCLES THE TIMER COUNTS BEFORE
                 94 ;A TIMER INTERRUPT OCCURS AND REFRESHES THE DISPLAY. APPROXIMATELY
                 95 ;50 TIMES PER SECOND.
FFF1             96 TIME    EQU     -0FH    ;TIMER VALUE 2.5MSEC
                 97 ;*****************************************************************
                 98 ;                        INTERRUPT BRANCHING
                 99 ;THIS PORTION OF MEMORY IS DEDICATED FOR USE OF RESET AND
                100 ;INTERRUPT BRANCHING. WHEN THE INTERRUPTS ARE ENABLED THE
                101 ;CODE AT THE FOLLOWING DESIGNATED SPOTS ARE EXECUTED WHEN A
                102 ;RESET OR A INTERRUPT OCCURS.
0000            103        ORG     0               ;
0000 0409       104        JMP     START           ;RESET
0002 00         105        NOP                     ;
0003 0438       106        JMP     INPUT           ;IBF INTERRUPT
0005 00         107        NOP                     ;
0006 00         108        NOP                     ;
0007 041F       109        JMP     DISPLA          ;TIMER INTERRUPT
                110 ;*****************************************************************
                111 ;                        INITIALIZATION
                112 ;THE FOLLOWING CODE SETS UP THE UPI-41 AND DISPLAY HARDWARE
                113 ;INTO OPERATIONAL FORMAT.  THE DISPLAY IS TURNED OFF, THE DISPLAY
                114 ;MAP IS FILLED WITH "BLANK" CHARACTERS, THE TIMER SET AND THE
                115 ;INTERRUPTS ARE ENABLED.
                116 ;
0009 D5         117 START:  SEL     RB1             ;
000A 8A08       118        ORL     P2,#08H         ;TURN DIGIT DRIVERS OFF
000C B838       119        MOV     R0,#38H         ;DISPLAY MAP POINTER,BOTTOM OF DISPLAY MAP
000E 23FF       120 BLKMAP: MOV     A,#0FFH         ;FF="BLANK"
0010 A0         121        MOV     @R0,A           ;BLANK TO DISPLAY MAP
0011 18         122        INC     R0              ;INCREMENT DISPLAY MAP POINTER
0012 F8         123        MOV     A,R0            ;DISPLAY MAP POINTER TO ACCUMULATOR
0013 B20E       124        JB5     BLKMAP          ;BLANK DISPLAY MAP TILL FILLED
0015 BB00       125        MOV     R3,#00H         ;SET DIGIT COUNTER TO 0
0017 23F1       126        MOV     A,#TIME         ;TIMER VALUE
0019 62         127        MOV     T,A             ;LOAD TIMER
001A 55         128        STRT    T               ;START TIMER
001B 25         129        EN      TCNTI           ;ENABLE TIMER INTERRUPT
001C 05         130        EN      I               ;ENABLE IBF INTERRUPT
                131 ;*****************************************************************
                132 ;                        USER PROGRAM
                133 ;A USERS PROGRAM WOULD INITIALIZE AT THIS POINT.  THE FOLLOWING
                134 ;CODE IS USED TO TAKE THE PLACE OF A POSSIBLE USER PROGRAM.
                135 ;
                136 ;
001D 041D       137 LOOP:   JMP     LOOP            ;WAIT FOR INTERRUPT
                138 ;*****************************************************************
                139 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                140 ;*************************************************************************
                141 ;                          DISPLAY ROUTINE
                142 ; THIS PORTION OF THIS PROGRAM IS AN INTERRUPT ROUTINE WHICH IS
                143 ;ACTED UPON WHEN THE TIMER COUNT IS COMPLETED.  THE ROUTINE UPDATES
                144 ;ONE DISPLAY DIGIT FROM THE DISPLAY MAP PER INTERRUPT SEQUENTIALLY,
                145 ;THUS EIGHT TIMER INTERRUPTS WILL HAVE REFRESHED THE ENTIRE DISPLAY.
                146 ;REGISTER BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON
                147 ;ENTERING THE ROUTINE.  ONCE THE DISPLAY HAS BEEN REFRESHED THE TIMER
                148 ;IS RESET AND THE ACCUMULATOR AND PRE-INTERRUPT REGISTER BANK IS RESTORED.
                149 ;
001F D5         150 DISPLA: SEL     RB1             ;REGISTER BANK 1
0020 AF         151         MOV     R7,A            ;SAVE ACCUMULATOR
0021 8A08       152         ORL     P2,#08H         ;TURN DIGIT DRIVERS OFF
0023 FB         153         MOV     A,R3            ;DIGIT COUNTER TO ACCUMULATOR
0024 4338       154         ORL     A,#38H          ;"OR" TO GET DISPLAY MAP ADDRESS
0026 A8         155         MOV     R0,A            ;DISPLAY MAP POINTER
0027 F0         156         MOV     A,@R0           ;GET CHARACTER FROM DISPLAY MAP
0028 39         157         OUTL    P1,A            ;OUTPUT CHARACTER TO SEGMENT DRIVERS
0029 FB         158         MOV     A,R3            ;DIGIT COUNTER VALUE TO ACCUMULATOR
002A 3A         159         OUTL    P2,A            ;OUTPUT TO DIGIT DRIVERS
002B 1B         160         INC     R3              ;INCREMENT DIGIT COUNTER
002C D307       161         XRL     A,#07H          ;CHECK IF AT LAST DIGIT
002E 9632       162         JNZ     SETIME          ;RESET TIMER IN NOT LAST DIGIT
0030 BB00       163         MOV     R3,#00H         ;RESET DIGIT COUNTER
0032 23F1       164 SETIME: MOV     A,#TIME         ;TIMER VALUE
0034 62         165         MOV     T,A             ;LOAD TIMER
0035 55         166         STRT    T               ;START TIMER
0036 FF         167         MOV     A,R7            ;RESTORE ACCUMULATOR
0037 93         168         RETR                    ;RETURN
                169 ;*************************************************************************
                170 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                171 ;
                172 ;*****************************************************************
                173 ;                INPUT CHARACTER AND DIGIT ROUTINE
                174 ; THIS PORTION OF THE PROGRAM IS AN INTERRUPT ROUTINE WHICH
                175 ;IS ACTED UPON WHEN THE IBF BIT IS SET.  THE ROUTINE GETS THE
                176 ;DISPLAY DATA WORD FROM THE DBB AND DEFINES BOTH THE DIGIT AND
                177 ;THE CHARACTER TO BE DISPLAYED.  THIS IS DONE BY MEANS OF A
                178 ;CHARACTER LOOP-UP TABLE AND A DISPLAY MAP FOR DIGIT AND CHARACTER
                179 ;LOCATION.  SPECIAL CONSIDERATION IS TAKEN FOR A DECIMAL POINT WHICH IS
                180 ;SIMPLY ADDED TO THE EXISTING CHARACTER IN THE DISPLAY MAP.  REGISTER
                181 ;BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON ENTERING
                182 ;THE ROUTINE.  ONCE THE DATA WORD HAS BEEN FULLY DEFINED THE ACCUMULATOR
                183 ;AND THE PRE-INTERRUPT REGISTER BANK IS RESTORED.
                184 ;
0038 D5         185 INPUT:  SEL     RB1             ;REGISTER BANK 1
0039 AF         186         MOV     R7,A            ;SAVE ACCUMULATOR
003A 22         187         IN      A,DBB           ;GET DATA
003B AA         188         MOV     R2,A            ;SAVE DATA WORD
003C 47         189         SWAP    A               ;DEFINE DIGIT LOCATION
003D 77         190         RR      A               ;
003E 5307       191         ANL     A,#07H          ;
0040 4338       192         ORL     A,#38H          ;
0042 A8         193         MOV     R0,A            ;DIGIT LOCATION IN DIGIT POINTER
0043 FA         194         MOV     A,R2            ;SAVED DATA WORD TO ACCUMULATOR
0044 531F       195         ANL     A,#1FH          ;DEFINE CHARACTER LOOK-UP-TABLE LOC.
0046 E3         196         MOVP3   A,@A            ;GET CHARACTER
0047 AA         197         MOV     R2,A            ;SAVE CHARACTER
0048 D37F       198         XRL     A,#7FH          ;IS CHARACTER DECIMAL POINT
004A C650       199         JZ      DPOINT          ;
004C FA         200         MOV     A,R2            ;SAVED CHARACTER TO ACCUMULATOR
004D A0         201         MOV     @R0,A           ;CHARACTER TO DISPLAY MAP
004E 0453       202         JMP     RETURN          ;
0050 FA         203 DPOINT: MOV     A,R2            ;SAVED CHARACTER TO ACCUMULATOR
0051 50         204         ANL     A,@R0           ;"AND" WITH OLD CHARACTER
0052 A0         205         MOV     @R0,A           ;BACK TO DISPLAY MAP
0053 FF         206 RETURN: MOV     A,R7            ;RESTORE ACCUMULATOR
0054 93         207         RETR
                208 ;*****************************************************************
                209 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                210 ;*************************************************************************
                211 ;                          LOOK-UP TABLE
                212 ; THIS LOOK-UP TABLE ORIGINATES IN PAGE 3 OF THE UPI-41 PROGRAM
                213 ;MEMORY.  IT IS USED TO DEFINE THE CORRECT LEVEL OF EACH SEGMENT
                214 ;AND DECIMAL POINT FOR A SELECTED CHARACTER FROM THE INPUT ROUTINE.
                215 ;INVERSE LOGIC IS USED BECAUSE OF THE SPECIFIC DRIVER CIRCUITRY, THUS
                216 ;A 1 ON A GIVEN SEGMENT MEANS IT IS OFF AND A 0 MEANS IT IS ON.
                217 ;
                218                                  ;*******SEGMENTS********
0300            219          ORG     300H            ;DP  G  F  E  D  C  B  A
0300 C0         220 CH0:     DB      0C0H            ;1   1  0  0  0  0  0  0
0301 F9         221 CH1:     DB      0F9H            ;1   1  1  1  1  0  0  1
0302 A4         222 CH2:     DB      0A4H            ;1   0  1  0  0  1  0  0
0303 B0         223 CH3:     DB      0B0H            ;1   0  1  1  0  0  0  0
0304 99         224 CH4:     DB      99H             ;1   0  0  1  1  0  0  1
0305 92         225 CH5:     DB      92H             ;1   0  0  1  0  0  1  0
0306 82         226 CH6:     DB      82H             ;1   0  0  0  0  0  1  0
0307 F8         227 CH7:     DB      0F8H            ;1   1  1  1  1  0  0  0
0308 80         228 CH8:     DB      80H             ;1   0  0  0  0  0  0  0
0309 98         229 CH9:     DB      98H             ;1   0  0  1  1  0  0  0
030A 88         230 CHA:     DB      88H             ;1   0  0  0  1  0  0  0
030B 83         231 CHB:     DB      83H             ;1   0  0  0  0  0  1  1
030C C6         232 CHC:     DB      0C6H            ;1   1  0  0  0  1  1  0
030D A1         233 CHD:     DB      0A1H            ;1   0  1  0  0  0  0  1
030E 86         234 CHE:     DB      86H             ;1   0  0  0  0  1  1  0
030F 8E         235 CHF:     DB      8EH             ;1   0  0  0  1  1  1  0
0310 7F         236 CHDP:    DB      7FH             ;0   1  1  1  1  1  1  1
0311 C2         237 CHG:     DB      0C2H            ;1   1  0  0  0  0  1  0
0312 89         238 CHH:     DB      89H             ;1   0  0  0  1  0  0  1
0313 FB         239 CHI:     DB      0FBH            ;1   1  1  1  1  0  1  1
0314 E1         240 CHJ:     DB      0E1H            ;1   1  1  0  0  0  0  1
0315 C7         241 CHL:     DB      0C7H            ;1   1  0  0  0  1  1  1
0316 AB         242 CHN:     DB      0ABH            ;1   0  1  0  1  0  1  1
0317 A3         243 CHO:     DB      0A3H            ;1   0  1  0  0  0  1  1
0318 8C         244 CHP:     DB      8CH             ;1   0  0  0  1  1  0  0
0319 AF         245 CHR:     DB      0AFH            ;1   0  1  0  1  1  1  1
031A 87         246 CHT:     DB      87H             ;1   0  0  0  0  1  1  1
031B C1         247 CHU:     DB      0C1H            ;1   1  0  0  0  0  0  1
031C 91         248 CHY:     DB      91H             ;1   0  0  1  0  0  0  1
031D BF         249 CHDASH:  DB      0BFH            ;1   0  1  1  1  1  1  1
031E FD         250 CHAPOS:  DB      0FDH            ;1   1  1  1  1  1  0  1
031F FF         251 BLANK:   DB      0FFH            ;1   1  1  1  1  1  1  1
                252 ;*************************************************************************
                253          END

USER SYMBOLS
BLANK  031F   BLKMAP 000E   CH0    0300   CH1    0301   CH2    0302   CH3    0303   CH4    0304   CH5    0305
CH6    0306   CH7    0307   CH8    0308   CH9    0309   CHA    030A   CHAPOS 031E   CHB    030B   CHC    030C
CHD    030D   CHDASH 031D   CHDP   0310   CHE    030E   CHF    030F   CHG    0311   CHH    0312   CHI    0313
CHJ    0314   CHL    0315   CHN    0316   CHO    0317   CHP    0318   CHR    0319   CHT    031A   CHU    031B
CHY    031C   DISPLA 001F   DPOINT 0050   INPUT  0038   LOOP   001D   RETURN 0053   SETIME 0032   START  0009
TIME   FFF1

ASSEMBLY COMPLETE,  NO ERRORS
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 ;
                    2 ;8085A SUBROUTINE TO DISPLAY THE 8-DIGIT BUFFER STARTING
                    3 ;AT THE LOCATION POINTED AT BY MSGSRT ON THE UPI-CONTROLLED
                    4 ;LED DISPLAY.
                    5 ;
                    6 ;INPUTS:MSGSRT - MESSAGE START LOCATION POINTER
                    7 ;DESTROYS: A, F/F'S
                    8 ;CALLS: OUTCHR
                    9 ;
4000                10          ORG     4000H
00E5                11 STATUS   EQU     0E5H            ;UPI STATUS PORT
0002                12 IBF      EQU     02H             ;UPI IBF FLAG MASK
00E4                13 DBBIN    EQU     0E4H            ;UPI DBBIN PORT
                    14 ;
4000 E5             15 DSPLAY:  PUSH    H               ;SAVE HL
4001 C5             16          PUSH    B               ;SAVE BC
4002 2A2840         17          LHLD    MSGSRT          ;LOAD HL WITH MESSAGE START ADR
4005 0600           18          MVI     B,00H           ;INITIALIZE DIGIT COUNTER
4007 7E             19 S1:      MOV     A,M             ;GET CHR FROM BUFFER
4008 E61F           20          ANI     1FH             ;MAKE IT 5 BITS
400A 80             21          ADD     B               ;ADD IN DIGIT COUNTER
400B 4F             22          MOV     C,A             ;SAVE TOTAL IN C
400C CD1D40         23          CALL    OUTCHR          ;OUTPUT CHR PLUS LOCATION TO UPI
400F 78             24          MOV     A,B             ;GET DIGIT COUNTER
4010 C620           25          ADI     20H             ;INC FOR NEXT DIGIT
4012 DA1A40         26          JC      EXIT            ;DONE IF CARRY SET
4015 47             27          MOV     B,A             ;RESTORE DIGIT COUNTER
4016 23             28          INX     H               ;INC MESSAGE POINTER
4017 C30740         29          JMP     S1              ;GO GET NEXT CHR
                    30 ;
401A C1             31 EXIT:    POP     B               ;RESTORE BC
401B E1             32          POP     H               ;RESTORE HL
401C C9             33          RET                     ;RETURN
                    34 ;
                    35 ;SUBROUTINE TO OUTPUT CHR TO UPI
                    36 ;
401D DBE5           37 OUTCHR:  IN      STATUS          ;READ UPI STATUS
401F E602           38          ANI     IBF             ;LOOK AT IBF
4021 C21D40         39          JNZ     OUTCHR          ;WAIT UNTIL IBF=0
4024 79             40          MOV     A,C             ;GET CHR
4025 D3E4           41          OUT     DBBIN           ;OUTPUT CHR TO UPI DBBIN
4027 C9             42          RET                     ;RETURN
                    43 ;
0002                44 MSGSRT:  DS      02H             ;LOCATION OF MESSAGE START POINTER
                    45 ;
                    46 END
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0


LOC  OBJ        SEQ        SOURCE STATEMENT

                  1 ;       ***********************************************
                  2 ;       *      UPI-41A SENSOR MATRIX CONTROLLER       *
                  3 ;       ***********************************************
                  4 ;
                  5 ;       THIS PROGRAM USES THE UPI-41A AS A SENSOR MATRIX CONTROLLER.
                  6 ;IT HAS MONITORING CAPABILITIES OF UP TO 128 SENSORS.  THE COORDINATE
                  7 ;AND SENSOR STATUS OF EACH DETECTED CHANGE IS AVAILABLE TO THE MASTER
                  8 ;MICROPROCESSOR IN A SINGLE BYTE.  A 40X8 FIFO QUEUE IS PROVIDED FOR
                  9 ;DATA BUFFERING.  BOTH HARDWARE OR POLLED INTERRUPT METHODS CAN BE USED
                 10 ;TO NOTIFY THE MASTER OF A DETECTED SENSOR CHANGE.
                 11 ;
                 12 ;*************************************************************************
                 13 ;
                 14 ;REGISTER DEFINITIONS:
                 15 ;       REGISTER            RBQ                 RBI
                 16 ;       --------            ---                 ---
                 17 ;         R0          MATRIX MAP POINTER      NOT USED
                 18 ;         R1          FIFO POINTER            NOT USED
                 19 ;         R2          SCAN ROW SELECT         NOT USED
                 20 ;         R3          COLUMN COUNTER          NOT USED
                 21 ;         R4          FIFO-IN                 NOT USED
                 22 ;         R5          FIFO-OUT                NOT USED
                 23 ;         R6          CHANGE WORD             NOT USED
                 24 ;         R7          COMPARE                 NOT USED
                 25 ;
                 26 ;*************************************************************************
                 27 ;
                 28 ;PORT PIN DEFINITIONS:
                 29 ;
                 30 ;PIN      PORT 1 FUNCTION     PIN       PORT 2 FUNCTION
                 31 ;---      ---------------     ---       ---------------
                 32 ;P0-7     COLUMN LINE INPUTS  P0-3      ROW SELECT OUTPUTS
                 33 ;                             P4        FIFO NOT EMPTY INTERRUPT
                 34 ;                             P5        OBF INTERRUPT
                 35 ;                             P6-7      NOT USED
                 36 ;
                 37 ;*************************************************************************
                 38 ;
                 39 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                 40 ;*****************************************************************
                 41 ;
                 42 ;CHANGE WORD BIT DEFINITION:
                 43 ;
                 44 ;                BIT              FUNCTION
                 45 ;                ---              --------
                 46 ;                D0-6             SENSOR COORDINATE
                 47 ;                D7               SENSOR STATUS
                 48 ;
                 49 ;*****************************************************************
                 50 ;
                 51 ;STATUS REGISTER BIT DEFINITION:
                 52 ;
                 53 ;                BIT              FUNCTION
                 54 ;                ---              --------
                 55 ;                D0               OBF
                 56 ;                D1-3             IBF, F0, F1 (NOT USED)
                 57 ;                D4               FIFO NOT EMPTY
                 58 ;                D5-7             USED DEFINED (NOT USED)
                 59 ;
                 60 ;*****************************************************************
                 61 ;
                 62 ;                        EQUATES
                 63 ;
                 64 ;THE FOLLOWING CODE DESIGNATES THREE VARIABLES; SCANTM,FIFOBA
                 65 ;AND FIFOTA.  SCANTM ADJUSTS THE LENGTH OF A DELAY BETWEEN
                 66 ;SCANNING SWITCH.  THIS SIMULATES DEBOUNCE FUNCTIONS.  FIFOBA
                 67 ;IS THE BOTTOM ADDRESS OF THE FIFO.  FIFOTA IS THE TOP ADDRESS
                 68 ;OF THE FIFO.  THIS MAKES IT POSSIBLE TO HAVE A FIFO 3 TO 40
                 69 ;BYTES IN LENGTH.
                 70 ;
                 71 ;*****************************************************************
                 72 ;
000F             73 SCANTM  EQU     0FH             ;SCAN TIME ADJUST
0008             74 FIFOBA  EQU     08H             ;FIFO BOTTOM ADDRESS
002F             75 FIFOTA  EQU     2FH             ;FIFO TOP ADDRESS
                 76 ;
                 77 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  78 ;*****************************************************************************
                  79 ;
                  80 ;                              INITIALIZATION
                  81 ;
                  82 ;THE PROGRAM STARTS AT THE FOLLOWING CODE UPON RESET.  WITHIN
                  83 ;THIS INITIALIZATION SECTION THE REGISTERS THAT MAINTAIN THE MATRIX
                  84 ;MAP,FIFO AND ROW SCANNING ARE SET UP.  PORT 1 IS SET HIGH FOR USE
                  85 ;AS AN INPUT PORT FOR THE COLUMN STATUS.  BIT 4 OF STATUS REGISTER IS
                  86 ;WRITTEN TO CONVEY A FIFO EMPTY CONDITION.  THE INITIAL COLUMN STATUS
                  87 ;OF ALL THE ROWS IN THE SENSOR MATRIX IS THEN READ INTO THE MATRIX
                  88 ;MAP.  ONCE THE MATRIX MAP IS FILLED THE OBF INTERRUPT (PORT 2-4) IS
                  89 ;ENABLED.
                  90 ;
                  91 ;*****************************************************************************
                  92 ;
0000              93         ORG     0
0000 B83F         94 INITMX: MOV     R0,#3FH          ;MATRIX MAP POINTER REGISTER,TOP ADDRESS
0002 BA0F         95         MOV     R2,#0FH          ;SCAN ROW SELECT REGISTER,TOP ROW
0004 BC08         96         MOV     R4,#FIFOBA       ;FIFO INPUT ADDRESS REGISTER,BOTTOM OF FIFO
0006 BD2F         97         MOV     R5,#FIFOTA       ;FIFO OUTPUT ADDRESS REGISTER,TOP OF FIFO
0008 89FF         98         ORL     P1,#0FFH         ;INITIALIZE PORT 1 HIGH FOR INPUTS
000A 2300         99         MOV     A,#00H           ;INITIALIZE STATUS REGISTER, FIFO EMPTY
000C 90          100         MOV     STS,A            ;WRITE TO STATUS REGISTER, BITS 4-7
000D FA          101 FILLMX: MOV     A,R2             ;SCAN ROW SELECT TO ACCUMULATOR
000E 3A          102         OUTL    P2,A             ;OUTPUT SCAN ROW SELECT TO PORT 2
000F 09          103         IN      A,P1             ;INPUT COLUMN STATUS PORT 1
0010 A0          104         MOV     @R0,A            ;LOAD MATRIX MAP WITH COLUMN STATUS
0011 FA          105         MOV     A,R2             ;CHECK SCAN ROW SELECT REGISTER VALUE FOR 0
0012 C618        106         JZ      OBFINT           ;IF 0 ENABLE OBF INTERRUPT
0014 C8          107         DEC     R0               ;DECREMENT TO NEXT MATRIX MAP ADDRESS
0015 CA          108         DEC     R2               ;DECREMENT TO SCAN NEXT ROW
0016 040D        109         JMP     FILLMX           ;FILL NEXT MATRIX MAP ADDRESS
0018 BA10        110 OBFINT: MOV     R2,#10H          ;BIT 4 HIGH IN ROW SCAN SELECT REGISTER
001A FA          111         MOV     A,R2             ;ROW SCAN SELECT VALUE TO ACCUMULATOR
001B 3A          112         OUTL    P2,A             ;INITIALIZE PORT 2, BIT 4 FOR "EN FLAGS"
001C F5          113         EN      FLAGS            ;ENABLE OBF INTERRUPT PORT 2, BIT 4
                 114 ;
                 115 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                116 ;*****************************************************************
                117 ;
                118 ;                       SCAN AND COMPARE
                119 ;
                120 ;THE FOLLOWING CODE IS THE SCAN AND COMPARE SECTION OF THE PROGRAM.
                121 ;UPON ENTERING THIS SECTION A CHECK IS MADE TO SEE IF THE ENTIRE MATRIX
                122 ;HAS BEEN SCANNED.  IF SO THE REGISTERS THAT MAINTAIN THE MATRIX MAP AND ROW
                123 ;SCANNING ARE RESET TO THE BEGINNING OF THE SENSOR MATRIX.  IF THE ENTIRE
                124 ;MATRIX HASNT BEEN SCANNED THE REGISTERS INCREMENT TO SCAN THE NEXT ROW.
                125 ;FROM THIS POINT ON THE ROW SCAN SELECT REGISTER IS USED FOR TWO FUNCTIONS.
                126 ;BITS 0-3 FOR SCANNING AND BITS 4 AND 5 FOR THE EXTERNAL INTERRUPTS.  THUSLY
                127 ;ALL USAGE OF THE REGISTERS IS DONE BY LOGICALLY MASKING IT SO AS TO ONLY
                128 ;AFFECT THE FUNCTION DESIRED.  ONCE THE REGISTERS ARE RESET, ONE ROW OF THE
                129 ;SENSOR MATRIX IS SCANNED.  A DELAY IS EXECUTED TO ADJUST FOR SCAN TIME
                130 ;(DEBOUNCE).  A BYTE OF COLUMN STATUS IS THEN READ INTO THE MATRIX MAP.
                131 ;AT THE TIME THE NEW COLUMN STATUS IS COMPARED TO THE OLD.  THE RESULT IS
                132 ;STORED IN THE COMPARE REGISTER.  THE PROGRAM IS THEN ROUTED ACCORDING TO
                133 ;WHETHER OR NOT A CHANGE WAS DETECTED.
                134 ;
                135 ;*****************************************************************
                136 ;
001D FA         137 ADJREG: MOV     A,R2            ;SCAN ROW SELECT TO ACCUMULATOR
001E 530F       138         ANL     A,#0FH          ;CHECK FOR 0 SCAN VALUE ONLY,NOT INTERRUPT
0020 C626       139         JZ      RSETRG          ;IF 0 RESET REGISTERS
0022 C8         140         DEC     R0              ;DECREMENT MATRIX MAP POINTER
0023 CA         141         DEC     R2              ;DECREMENT SCAN ROW SELECT
0024 042C       142         JMP     SCANMX          ;SCAN MATRIX
0026 B83F       143 RSETRG: MOV     R0,#3FH         ;RESET MATRIX MAP POINTER REGISTER,TOP ADDRESS
0028 FA         144         MOV     A,R2            ;SCAN ROW SELECT TO ACCUMULATOR
0029 430F       145         ORL     A,#0FH          ;RESET SCAN ROW SELECT,NO INTERRUPT CHANGE
002B AA         146         MOV     R2,A            ;SCAN ROW SELECT REGISTER
002C FA         147 SCANMX: MOV     A,R2            ;SCAN ROW SELECT TO ACCUMULATOR
002D 3A         148         OUTL    P2,A            ;OUTPUT SCAN ROW SELECT TO PORT 2
002E B80F       149         MOV     R3,#SCANTM      ;SET DELAY FOR OUTPUT SCAN TIME
0030 EB30       150 DELAY2: DJNZ    R3,DELAY2       ;DELAY
0032 09         151         IN      A,P1            ;INPUT COLUMN STATUS FROM PORT 1 TO ACCUMULATOR
0033 20         152         XCH     A,@R0           ;STORE NEW COLUMN STATUS SAVE OLD IN ACCUMULATOR
0034 D0         153         XRL     A,@R0           ;COMPARE OLD WITH NEW COLUMN STATUS
0035 AF         154         MOV     R7,A            ;SAVE COMPARE RESULT IN COMPARE REGISTER
0036 C669       155         JZ      CHFFUL          ;IF THE SAME, CHECK IF FIFO IS FULL
                156 ;
                157 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                158 ;*************************************************************************
                159 ;
                160 ;                       CHANGE WORD ENCODING
                161 ;
                162 ;THE FOLLOWING CODE IS THE CHANGE WORD ENCODING SECTION.  THIS
                163 ;SECTION IS ONLY EXECUTED IF A CHANGE WAS DETECTED.  THE COLUMN COUNTER
                164 ;IS SET AND DECREMENTED TO DESIGNATE EACH OF THE 8 COLUMNS.  THE COMPARE
                165 ;REGISTER IS LOOKED AT ONE BIT AT A TIME TO FIND THE EXACT LOCATION OF
                166 ;THE CHANGE(S).  WHEN A CHANGE IS FOUND IT IS ENCODED BY GIVING IT A
                167 ;COORDINATE FOR ITS LOCATION.  THIS IS DONE BY COMBINING THE PRESENT VALUE
                168 ;IN THE ROW SCAN SELECT REGISTER AND THE COLUMN COUNTER.  THE ACTUAL STATUS
                169 ;OF THAT SENSOR IS ESTABLISHED BY LOOKING AT THE CORRESPONDING BYTE IN
                170 ;THE MATRIX MAP.  THIS STATUS IS COMBINED WITH THE COORDINATE TO ESTABLISH
                171 ;THE CHANGE WORD.  THE CHANGE WORD IS THEN STORED IN THE CHANGE WORD REGISTER.
                172 ;
                173 ;*************************************************************************
                174 ;
0038 B808       175         MOV     R3,#08H         ;SET COLUMN COUNTER REGISTER TO 8
003A CB         176 RRLOOK: DEC     R3              ;DECREMENT COLUMN COUNTER
003B F0         177         MOV     A,@R0           ;COLUMN STATUS TO ACCUMULATOR
003C 77         178         RR      A               ;ROTATE COLUMN STATUS RIGHT
003D A0         179         MOV     @R0,A           ;ROTATED COLUMN STATUS BACK TO MATRIX MAP
003E FF         180         MOV     A,R7            ;COMPARE REGISTER VALUE TO ACCUMULATOR
003F 77         181         RR      A               ;ROTATE COMPARE VALUE RIGHT
0040 AF         182         MOV     R7,A            ;ROTATED COMPARE VALUE TO COMPARE REGISTER
0041 F245       183         JB7     ENCODE          ;TEST BIT 7 IF CHANGE DETECTED ENCODE CHANGE WORD
0043 0469       184         JMP     CHFFUL          ;IF NO CHANGE IS DETECTED CHECK FOR FIFO FULL
0045 FA         185 ENCODE: MOV     A,R2            ;SCAN ROW SELECT TO ACCUMULATOR  0000XXXX
0046 530F       186         ANL     A,#0FH          ;ROTATE ONLY SCAN VALUE
0048 E7         187         RL      A               ;ROTATE LEFT                     000XXXX0
0049 E7         188         RL      A               ;ROTATE LEFT                     00XXXX00
004A E7         189         RL      A               ;ROTATE LEFT                     0XXXX000
004B 4B         190         ORL     A,R3            ;ESTABLISH MATRIX COORDINANT     0XXXXXXX
                191                                 ;(OR) COLUMN COUNTER VALUE WITH ACCUMULATOR
004C AE         192         MOV     R6,A            ;SAVE COORDINANT IN CHANGE WORD REGISTER
004D F0         193         MOV     A,@R0           ;COLUMN STATUS FROM MATRIX MAP TO ACCUMULATOR
004E 5380       194         ANL     A,#80H          ;0 ALL BITS BUT BIT 7
0050 4E         195         ORL     A,R6            ;(OR) SENSOR STATUS WITH COORDINATE FOR COMPLETED CHANGE WORD
0051 AE         196         MOV     R6,A            ;SAVE CHANGE WORD                XXXXXXXX
                197 ;
                198 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                199 ;*****************************************************************
                200 ;
                201 ;                   FIFO-DBBOUT MANAGEMENT
                202 ;
                203 ;THE FOLLOWING CODE IS THE FIFO-DBBOUT MANAGEMENT SECTION OF THE
                204 ;PROGRAM.  THIS SECTION TAKES AN ENCODED CHANGE WORD AND LOADS IT INTO
                205 ;THE FIFO.  THE FIFO NOT EMPTY INTERRUPT IS THEN SET AND THE FIFO-IN
                206 ;POINTER GETS UPDATED.  A FIFO FULL CONDITION IS THEN CHECKED FOR AND
                207 ;ROUTED ACCORDINGLY.  IF BOTH THE FIFO AND OBF HAVE CHANGE WORDS THE
                208 ;PROGRAM LOCKS UP UNTIL THIS HAS CHANGED.  IF THE FIFO ISNT FULL COLUMN
                209 ;COUNTER= 0, FIFO EMPTY AND OBF CONDITIONS ARE CHECKED.  THE FIFO-OUT
                210 ;POINTER IS SET AND DBBOUT IS LOADED IF THE FIFO ISNT EMPTY AND OBF ISNT
                211 ;SET.  IF THIS ISNT THE SITUATION, PROGRAM FLOW IS ROUTED BACK TO THE
                212 ;THE SCAN AND COMPARE SECTION TO SCAN THE NEXT ROW.
                213 ;
                214 ;*****************************************************************
                215 ;
0052 FC         216 LOADFF: MOV     A,R4          ;FIFO INPUT ADDRESS TO ACCUMULATOR
0053 A9         217         MOV     R1,A          ;FIFO POINTER USED FOR INPUT
0054 FE         218         MOV     A,R6          ;CHANGE WORD TO ACCUMULATOR
0055 A1         219         MOV     @R1,A         ;LOAD FIFO AT FIFO INPUT ADDRESS
0056 2310       220 STATNE: MOV     A,#10H        ;BIT 4 FOR FIFO NOT EMPTY
0058 90         221         MOV     STS,A         ;WRITE TO STATUS REGISTER, FIFO NOT EMPTY
0059 8A20       222 INTRH1: ORL     P2,#20H       ;FIFO NOT EMPTY INTERRUPT PORT 2-5 HIGH
005B FA         223         MOV     A,R2          ;ROW SCAN SELECT TO ACCUMULATOR
005C 4320       224         ORL     A,#20H        ;SAVE INTERRUPT, NO CHANGE TO SCAN VALUE
005E AA         225         MOV     R2,A          ;ROW SCAN SELECT REGISTER
005F 232F       226 ADJFIN: MOV     A,#FIFOTA     ;FIFO TOP ADDRESS TO ACCUMULATOR
0061 DC         227         XRL     A,R4          ;COMPARE WITH CURRENT FIFO INPUT ADDRESS
0062 C667       228         JZ      RSFFIN        ;IF THE SAME RESET FIFO INPUT REGISTER
0064 1C         229         INC     R4            ;NEXT FIFO INPUT ADDRESS
0065 0469       230         JMP     CHFFUL        ;CHECK FIFO FULL
0067 BC08       231 RSFFIN: MOV     R4,#FIFOBA    ;RESET FIFO INPUT REGISTER,BOTTOM OF FIFO
0069 FC         232 CHFFUL: MOV     A,R4          ;FIFO INPUT ADDRESS TO ACCUMULATOR
006A DD         233         XRL     A,R5          ;COMPARE INPUT WITH OUTPUT FIFO ADDRESS
006B 967D       234         JNZ     CHCNTR        ;IF NOT SAME CHECK COLUMN COUNTER VALUE
006D 866D       235 CHOBF1: JOBF    CHOBF1        ;IF OBF IS 1 THEN CHECK OBF
006F 232F       236 ADJFOT: MOV     A,#FIFOTA     ;FIFO TOP ADDRESS TO ACCUMULATOR
0071 DD         237         XRL     A,R5          ;COMPARE TOP TO OUTPUT FIFO ADDRESS
0072 C677       238         JZ      RSFFOT        ;IF THE SAME RESET FIFO OUTPUT REGISTER
0074 1D         239         INC     R5            ;NEXT FIFO OUTPUT ADDRESS
0075 0479       240         JMP     LOADDB        ;LOAD DBBOUT
0077 BD08       241 RSFFOT: MOV     R5,#FIFOBA    ;RESET FIFO OUTPUT ADDRESS TO BOTTOM OF FIFO
0079 FD         242 LOADDB: MOV     A,R5          ;OUTPUT FIFO ADDRESS TO ACCUMULATOR
007A A9         243         MOV     R1,A          ;FIFO POINTER USED FOR OUTPUT
007B F1         244         MOV     A,@R1         ;CHANGE WORD TO ACCUMULATOR
007C 02         245         OUT     DBB,A         ;CHANGE WORD TO DBBOUT
007D FB         246 CHCNTR: MOV     A, R3         ;COLUMN COUNTER TO ACCUMULATOR
007E 963A       247         JNZ     RRLOOK        ;IF NOT 0 FINISH CHANGE WORD ENCODING
0080 2308       248 CHFFEM: MOV     A,#FIFOBA     ;FIFO BOTTOM ADDRESS TO ACCUMULATOR
0082 DC         249         XRL     A,R4          ;COMPARE FIFO INPUT ADDRESS WITH FIFO BOTTOM ADDRESS
0083 C68C       250         JZ      ADJFEM        ;IF THE SAME,ADJUST TO CHECK FOR FIFO EMPTY
0085 FC         251         MOV     A,R4          ;FIFO INPUT ADDRESS TO ACCUMULATOR
0086 07         252         DEC     A             ;DECREMENT FIFO INPUT ADDRESS IN ACCUMULATOR
0087 DD         253         XRL     A,R5          ;COMPARE INPUT TO OUTPUT FIFO ADDRESSES
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

0088 C691        254        JZ      STATMT         ;IF SAME,WRITE STATUS REGISTER FOR FIFO EMPTY
008A 049C        255        JMP     CHOBF2         ;CHECK OBF
008C 232F        256 ADJFEM: MOV    A,#FIFOTA      ;FIFO TOP ADDRESS TO ACCUMULATOR
008E D0          257        XRL     A,R5           ;COMPARE TOP TO OUTPUT FIFO ADDRESS
008F 969C        258        JNZ     CHOBF2         ;IF NOT SAME THEN FIFO IS NOT EMPTY, CHECK OBF
0091 2300        259 STATMT: MOV    A,#00H         ;CLEAR BIT 0 FOR FIFO EMPTY
0093 90          260        MOV     STS,A          ;WRITE TO STATUS REGISTER
0094 9ADF        261 INTRLO: ANL    P2,#0DFH       ;FIFO EMPTY, INTERRUPT PORT 2-5 LOW
0096 FA          262        MOV     A,R2           ;SCAN ROW SELECT TO ACCUMULATOR
0097 53DF        263        ANL     A,#0DFH        ;SAVE INTERRUPT, NO CHANGE TO SCAN VALUE
0099 AA          264        MOV     R2,A           ;SCAN ROW SELECT REGISTER
009A 041D        265        JMP     ADJREG         ;ADJUST REGISTERS
009C 861D        266 CHOBF2: JOBF   ADJREG         ;IF OBF=1 THEN ADJUST REGISTERS
009E 046F        267        JMP     ADJFOT         ;ADJUST FIFO OUT ADDRESS TO LOAD DBBOUT
                 268 ;
                 269        END

USER SYMBOLS
ADJFEM 008C   ADJFIN 005F   ADJFOT 006F   ADJREG 001D   CHCNTR 007D   CHFFEM 0080   CHFFUL 0069   CHOBF1 006D
CHOBF2 009C   DELAY2 0030   ENCODE 0045   FIFOBA 0008   FIFOTA 002F   FILLMX 000D   INITMX 0000   INTRH1 0059
INTRLO 0094   LOADDB 0079   LOADFF 0052   OBFINT 0018   RRLOOK 003A   RSETRG 0026   RSFFIN 0067   RSFFOT 0077
SCANMX 002C   SCANTM 000F   STATMT 0091   STATNE 0056

ASSEMBLY COMPLETE,  NO ERRORS
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, X108        MODULE   PAGE    1
8085A/UPI SENSOR MATRIX CONTROLLER

 LOC  OBJ        SEQ        SOURCE STATEMENT

                   1 ;
                   2 ;SUBROUTINE TO READ ALL CHANGES IN THE UPI AND BUILD A BUFFER
                   3 ;STARTING AT BUFSRT.  REG. B CONTAINS THE NUMBER OF CHANGES
                   4 ;UPON EXIT.  THE MAXIMUM NUMBER OF CHANGES IN ANY ONE CALL
                   5 ;IS 255.
                   6 ;
                   7 ;INPUTS: NOTHING
                   8 ;OUTPUTS: CHANGE WORD BUFFER AT BUFSRT
                   9 ;         CHANGE WORD COUNT IN REG. B
                  10 ;CALLS: NOTHING
                  11 ;
4000               12         ORG     4000H
00E5               13 STATUS  EQU     0E5H            ;UPI STATUS PORT
00E4               14 DBBOUT  EQU     0E4H            ;UPI DBBOUT PORT
0010               15 FIFO    EQU     10H             ;FIFO NOT EMPTY MASK
0001               16 OBF     EQU     01H             ;OBF MASK
4300               17 BUFSRT  EQU     4300H           ;BUFFER START LOCATION
                   18 ;
4000 210043        19 START:  LXI     H,BUFSRT        ;INITIALIZE BUFFER POINTER
4003 0600          20         MVI     B,00H           ;CLEAR CHANGE WORD COUNTER
4005 DBE5          21 POLL1:  IN      STATUS          ;READ UPI STATUS
4007 E611          22         ANI     FIFO OR OBF     ;TEST FIFO NOT EMPTY AND OBF
4009 C8            23         RZ                      ;RETURN IF ZERO
400A DBE5          24         IN      STATUS          ;READ UPI STATUS
400C E601          25         ANI     OBF             ;TEST OBF FLAG
400E CA0540        26         JZ      POLL1           ;WAIT IF NOT READY
4011 DBE4          27         IN      DBBOUT          ;READ CHANGE WORD
4013 77            28         MOV     M,A             ;LOAD BUFFER WITH CHANGE WORD
4014 23            29         INX     H               ;INC BUFFER POINTER
4015 04            30         INR     B               ;INC CHANGE WORD COUNTER
4016 C8            31         RZ                      ;EXIT IF COUNTER = 256
4017 C30540        32         JMP     POLL1           ;CHECK IF MORE CHANGE WORDS
                   33 ;
                   34 END
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0
AP-41 COMBINATION I/O DEVICE

  LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 $MOD42
                    2 ;************************UNIOD*****************************************
                    3 ;
                    4 ;THIS UPI-41 PROGRAM IMPLEMENTS A FULL-DUPLEX UART WITH ON-CHIP
                    5 ;BAUD RATE GENERATION IN COMBINATION WITH AN 8-BIT PARALLEL I/O
                    6 ;PORT.  THE BAUD RATE IS SELECTABLE FROM 110 TO 1200 BAUD.  THE
                    7 ;PARALLEL I/O PORT IS PROGRAMMABLE FOR EITHER INPUT OR OUTPUT.
                    8 ;
                    9 ;INTERRUPT OUTPUTS ARE AVAILABLE FOR DATA AVAILABLE ON THE RECEIVER
                   10 ;AND PARALLEL INPUT.  THE STATUS REGISTER MUST BE READ TO DETERMINE
                   11 ;WHICH SOURCE CAUSED THE INTERRUPT.  THE FLAGS F0 AND F1 CODE THE
                   12 ;INTERRUPT SOURCE.  F0 AND F1 ALSO GIVE AN INDICATION OF COMMAND
                   13 ;ERRORS.
                   14 ;
                   15 ;*************************************************************************
                   16 ;
                   17 ;REGISTER DEFINITION
                   18 ;              RB0                    RB1
                   19 ;              ---                    ---
                   20 ;       0      NOT USED               NOT USED
                   21 ;       1      NOT USED               BAUD RATE CONSTANT
                   22 ;       2      NOT USED               TX TICK COUNTER
                   23 ;       3      RX STATUS (RXSTS)      TX SERIALIZER
                   24 ;       4      RX HOLDING             TX BUFFER
                   25 ;       5      RX TICK COUNTER        TX STATUS (TXSTS)
                   26 ;       6      RX DESERIALIZER        COMMAND STORE
                   27 ;       7      STATUS REG STORE       ACC. INTERRUPT SAVE
                   28 ;
                   29 ;*************************************************************************
                   30 ;
                   31 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                32 ;
                33 ;*************************************************************************
                34 ;
                35 ;COMMANDS
                36 ;
                37 ;       CONFIGURE: 0 0 0 A B C D P
                38 ;                                          A - 1200 BAUD SELECT
                39 ;                                          B - 600 BAUD SELECT
                40 ;                                          C - 300 BAUD SELECT
                41 ;                                          D - 110 BAUD SELECT
                42 ;                                          E - PARALLEL I/O DIRECTION
                43 ;                                                 0 - INPUT
                44 ;                                                 1 - OUTPUT
                45 ;
                46 ;       I/O:        1 0 0 0 0 0 0 0    (PERFORM I/O OPERATION)
                47 ;       RESET ERROR:1 1 0 0 0 0 0 0     (RESET RX ERROR IN STATUS)
                48 ;
                49 ;*************************************************************************
                50 ;
                51 ;STATUS REGISTER DEFINITION
                52 ;
                53 ;       BIT             DEFINITION
                54 ;       ---             ----------
                55 ;        0              OBF - DATA AVAILABLE
                56 ;        1              IBF - BUSY
                57 ;        2              F0
                58 ;        3              F1
                59 ;        4              NOT USED
                60 ;        5              TXINT - TX INTERRUPT
                61 ;        6              FRAMING ERROR
                62 ;        7              OVERRUN ERROR
                63 ;
                64 ;       F0      F1      OPERATION
                65 ;       --      --      ---------
                66 ;       0       0       X
                67 ;       0       1       PARALLEL I/O DATA AVAILABLE
                68 ;       1       0       SERIAL I/O DATA AVAILABLE
                69 ;       1       1       COMMAND ERROR
                70 ;
                71 ;*************************************************************************
                72 ;
                73 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  74 ;
                  75 ;*******************************************************************************
                  76 ;
                  77 ;STATUS REGISTER DEFINITIONS
                  78 ;
                  79 ;                        RXSTS                   TXSTS
                  80 ;                        -----                   -----
                  81 ;              0         RX FLAG - SPACE         TX FLAG - TRANSMITTING CHR
                  82 ;              1         START FLAG - GOOD START REQUEST BYTE - CHR IN BUFFER
                  83 ;              2         BYTE FINISHED           TX PIPELINED DATA BIT
                  84 ;              3         DATA READY              START BIT FLAG
                  85 ;              4         FRAMING ERROR           NOT USED
                  86 ;              5         OVERRUN ERROR           NOT USED
                  87 ;              6         IO DIRECTION            NOT USED
                  88 ;              7         IO FLAG                 NOT USED
                  89 ;
                  90 ;*******************************************************************************
                  91 ;
                  92 ;PORT 2 DEFINITIONS
                  93 ;
                  94 ;       BIT             DEFINITION
                  95 ;       ---             ----------
                  96 ;        0              TX DATA
                  97 ;        1              NOT USED
                  98 ;        2              NOT USED
                  99 ;        3              TX INTERRUPT
                 100 ;        4              OBF INTERRUPT (RX OR I/O DATA AVAILABLE)
                 101 ;        5              NOT USED
                 102 ;        6              NOT USED (TICK SAMPLE)
                 103 ;        7              NOT USED
                 104 ;
                 105 ;*******************************************************************************
                 106 ;
                 107 ;MISC.
                 108 ;
                 109 ;       RX DATA         T0 INPUT
                 110 ;       EXT CLOCK       T1 INPUT 76.8KHZ (1.2288MHZ/16)
                 111 ;
                 112 ;*******************************************************************************
                 113 ;
                 114 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                115 ;
                116 ;*****************************************************************************
                117 ;
                118 ;SYSTEM EQUATES:
                119 ;
0001            120 RXFLG   EQU     01H             ;RECEIVE FLAG IN RXSTS
0002            121 SRTFLG  EQU     02H             ;START BIT FLAG IN RXSTS
0004            122 BFFLG   EQU     04H             ;BYTE FINISHED FLAG IN RXSTS
0008            123 DATRDY  EQU     08H             ;DATA READY FLAG IN RXSTS
0010            124 FRAMER  EQU     10H             ;FRAMING ERROR FLAG IN RXSTS
0020            125 OVRUN   EQU     20H             ;OVERRUN ERROR FLAG IN RXSTS
0040            126 IODIR   EQU     40H             ;I/O DIRECTION FLAG IN RXSTS
0080            127 IOFLG   EQU     80H             ;I/O REQUEST FLAG IN RXSTS
0001            128 TXFLG   EQU     01H             ;TX FLAG IN TXSTS
0002            129 REQFLG  EQU     02H             ;REQUEST BYTE FLAG IN TXSTS
0040            130 TICOUT  EQU     40H             ;TICK SAMPLE BIT IN PORT 2
0080            131 RXINTL  EQU     80H             ;RX DESERIALIZER INITIALIZATION
0004            132 TICSRT  EQU     04H             ;TICK INITIALIZATION
007F            133 ASCMSK  EQU     7FH             ;ASCII MASK
0003            134 TXTIC   EQU     03H             ;TX TICK MOD MASK
0028            135 TXEND   EQU     40D             ;TICK COUNT AT END OF TX CHARACTER
0024            136 STPEND  EQU     36D             ;TICK COUNT AT END OF TX DATA
0004            137 MARK    EQU     04H             ;MARK OUTPUT
00FB            138 SPACE   EQU     0FBH            ;SPACE OUTPUT
0000            139 ZERO    EQU     00H             ;GENERAL CLEAR
0008            140 TXINT   EQU     08H             ;TX INTERRUPT OUTPUT IN PORT 2
0020            141 TXBIT   EQU     20H             ;TX INTERRUPT BIT IN STATUS
0020            142 TIMCON  EQU     32D             ;TIMER CONSTANT RAM LOCATION
003F            143 RSTERR  EQU     3FH             ;RESET ERROR MASK FOR STATUS
0040            144 FESTS   EQU     40H             ;FRAMING ERROR BIT IN STATUS
0080            145 OVSTS   EQU     80H             ;OVERRUN ERROR BIT IN STATUS
0001            146 MKOUT   EQU     01H             ;MARK OUTPUT TO PORT
00FE            147 SPOUT   EQU     0FEH            ;SPACE OUTPUT TO PORT
0008            148 SBIT    EQU     08H             ;TX START BIT FLAG
0003            149 RXSTS   EQU     R3              ;RX STATUS REGISTER
0005            150 TXSTS   EQU     R5              ;TX STATUS REGISTER
                151 ;
                152 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                153 ;*****************************************************************************
                154 ;
                155 ;RESET VECTOR LOCATION
                156 ;
                157 ;*****************************************************************************
                158 ;
0000            159        ORG     0000H
                160 ;
0000 C5         161 RESET: SEL     RB0             ;GET INTO RB0 AT RESET
0001 4400       162        JMP     INIT            ;GO TO INITIALIZATION
                163 ;
                164 ;*****************************************************************************
                165 ;
                166 ;TIMER INTERRUPT LOCATION - TIMER IS SET TO 4 TIMES THE BAUD RATE.  THE
                167 ;RECEIVER AND TRANSMITTER ARE SERVICED EVERY FOUR TIMER TICKS.  SOFTWARE
                168 ;DELAY LOOP IS USED FOR TIMING FINE-TUNING.  RB1 R1 POINTS AT DELAY
                169 ;CONSTANT AT INTERRUPT.  R1-1 POINTS AT TIMER CONSTANT.
                170 ;
                171 ;*****************************************************************************
                172 ;
0007            173        ORG     0007H
                174 ;
0007 D5         175 TIMINT: SEL    RB1             ;INTERRUPT PROCESSING IN RB1
0008 AF         176        MOV     R7,A            ;SAVE ACCUMULATOR IN R7
0009 F9         177        MOV     A,R1            ;GET TIMER CONSTANT
000A 00         178        NOP                     ;DELAY TO GET INTO T1 HIGH
000B 560B       179 INT1:  JT1     INT1            ;WAIT UNTIL T1 IS LOW
000D 62         180        MOV     T,A             ;THEN LOAD COUNTER
                181 ;
                182 ;TICK SAMPLE OUTPUT
                183 ;
000E 9ABF       184        ANL     P2,#NOT T1COUT
0010 8A40       185        ORL     P2,#T1COUT
                186 ;
                187 ;*****************************************************************************
                188 ;
                189 ;TRANSMITTER OUTPUT - TIME CRITICAL TASKS DONE FIRST.  DATA BIT OUTPUT
                190 ;PIPELINED IN TXSTS BIT 2 IS OUTPUT NOW.
                191 ;
                192 ;*****************************************************************************
                193 ;
0012 FD         194 TXOUT: MOV     A,TXSTS         ;GET TX STATUS
0013 5219       195        JB2     MOUT            ;TEST PIPELINED DATA
0015 9AFE       196        ANL     P2,#SPOUT       ;OUTPUT SPACE IF RESET
0017 041B       197        JMP     RCV             ;DO RECEIVER
0019 8A01       198 MOUT:  ORL     P2,#MKOUT       ;OUTPUT MARK IS SET
                199 ;
                200 ;*****************************************************************************
                201 ;
                202 ;START OF RECEIVER FLOW -  RXSTS REGISTER
                203 ;HOLDS RECEIVER STATUS.
                204 ;
                205 ;*****************************************************************************
                206 ;
001B C5         207 RCV:   SEL     RB0             ;SWITCH TO RX BANK
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

001C FB         208        MOV     A,RXSTS         ;GET RXSTS
001D 1226       209        JB0     RCV1            ;TEST RECEIVE FLAG
                210                                ;0 - NO CHR BEING RECEIVED
                211                                ;1 - POSSIBLE START BIT, DO TEST
001F 3668       212        JT0     XMIT            ;TEST RXD INPUT
                213                                ;0 - SPACE, SET RX FLAG
                214                                ;1 - MARK, GO CHECK XMIT
0021 4301       215        ORL     A,#RXFLG        ;SPACE - SET RX FLAG
0023 AB         216        MOV     RXSTS,A         ;RESTORE RXSTS
0024 0468       217        JMP     XMIT            ;GO HANDLE XMTR
                218 ;
                219 ;START BIT TEST
                220 ;
0026 3238       221 RCV1:  JB1     RCV3            ;FIRST TEST START BIT FLAG
0028 3633       222        JT0     RCV2            ;TEST RXD INPUT
                223                                ;0 - SPACE, GOOD START BIT
                224                                ;1 - MARK, BAD START BIT, IGNORE
002A 4302       225        ORL     A,#SRTFLG       ;GOOD START - SET START BIT FLAG
002C AB         226        MOV     RXSTS,A         ;RESTORE RXSTS
002D BE80       227        MOV     R6,#RXINTL      ;SETUP RX DESERIALIZER
002F BD04       228        MOV     R5,#TICSRT      ;LOAD RX TICK COUNTER
0031 0468       229        JMP     XMIT            ;GO HANDLE XMTR
                230 ;
                231 ;BAD START BIT - RESET FLAGS
                232 ;
0033 53FE       233 RCV2:  ANL     A,#NOT RXFLG    ;RESET RECEIVE FLAG
0035 AB         234        MOV     RXSTS,A         ;RESTORE RXSTS
0036 0468       235        JMP     XMIT            ;GO HANDLE XMTR
                236 ;
                237 ;IN MIDDLE OF CHR - SAMPLE EVERY 4 TIMER TICKS
                238 ;
0038 ED68       239 RCV3:  DJNZ    R5,XMIT         ;WAIT UNTIL 4TH TICK
003A BD04       240        MOV     R5,#TICSRT      ;RELOAD RX TICK COUNTER
003C 524D       241        JB2     RCV5            ;TEST BYTE FINISHED FLAG
                242                                ;0 - MIDDLE OF CHR, CONTINUE
                243                                ;1 - DONE WITH STOP BITS
003E 97         244        CLR     C               ;CLEAR CARRY BEFORE ROTATE
003F 2642       245        JNT0    RCV4            ;TEST RXD INPUT
0041 A7         246        CPL     C               ;RXD IS MARK, SET CARRY
0042 FE         247 RCV4:  MOV     A,R6            ;GET DESERIALIZER
0043 67         248        RRC     A               ;ROTATE IN NEW BIT
0044 AE         249        MOV     R6,A            ;RESTORE DESERIALIZER
0045 E668       250        JNC     XMIT            ;TEST CARRY AFTER ROTATE
                251                                ;0 - MIDDLE OF CHR
                252                                ;1 - STOP BIT COMING NEXT
0047 FB         253        MOV     A,RXSTS         ;GET RXSTS
0048 4304       254        ORL     A,#BFFLG        ;SET BYTE FINISHED FLAG
004A AB         255        MOV     RXSTS,A         ;RESTORE RXSTS
004B 0468       256        JMP     XMIT            ;GO HANDLE XMTR
                257 ;
                258 ;BYTE FINISHED - DO STOP BIT TEST
                259 ;
004D 2660       260 RCV5:  JNT0    RCV8            ;TEST RXD INPUT
                261                                ;0 - SPACE, INVALID STOP BIT
                262                                ;1 - MARK, VALID STOP BIT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

004F 53EF        263        ANL     A,#NOT FRAMER   ;NO FRAMING ERROR, RESET FLAG
                 264 ;
                 265 ;OVERRUN TEST - IF RX DATA READY STILL SET, OVERRUN ERROR
                 266 ;
0051 7264        267 RCV6:  JB3     RCV9            ;IF DATA READY STILL SET, ERROR
0053 53DF        268        ANL     A,#NOT OVRUN    ;NO OVERRUN, RESET FLAG
                 269 ;
                 270 ;CLEAN UP RXSTS AT CHR COMPLETE
                 271 ;
0055 4308        272 RCV7:  ORL     A,#DATRDY       ;SET DATA READY
0057 53F8        273        ANL     A,#NOT (RXFLG OR SRTFLG OR BFFLG)       ;RESET OTHER FLAGS
0059 AB          274        MOV     RXSTS,A         ;RESTORE RXSTS
005A FE          275        MOV     A,R6            ;GET DESERIALIZER REG
005B 537F        276        ANL     A,#ASCMSK       ;MAKE IT 7 BITS
005D AC          277        MOV     R4,A            ;PUT DATA INTO HOLDING REG
005E 0468        278        JMP     XMIT            ;GO HANDLE XMTR
                 279 ;
                 280 ;BAD STOP - SET FRAMING ERROR FLAG
                 281 ;
0060 4310        282 RCV8:  ORL     A,#FRAMER       ;SET FRAMING ERROR FLAG
0062 0451        283        JMP     RCV6            ;CONTINUE
                 284 ;
                 285 ;OVERRUN ERROR - SET OVERRUN FLAG
                 286 ;
0064 4320        287 RCV9:  ORL     A,#OVRUN        ;SET OVERRUN FLAG
0066 0455        288        JMP     RCV7            ;CONTINUE
                 289 ;
                 290 ;*************************************************************************
                 291 ;
                 292 ;START OF TRANSMITTER FLOW - TRANSMITTER IS SERVICED EVERY 4 TICKS.
                 293 ;THE TX TICK COUNTER SERVES AS THE TX BIT COUNTER.  TRANSMITTER STATUS
                 294 ;IS HELD IN THE TXSTS REGISTER.
                 295 ;
                 296 ;*************************************************************************
                 297 ;
0068 D5          298 XMIT:  SEL     RB1             ;BE SURE WE'RE IN RB1
0069 FD          299        MOV     A,TXSTS         ;GET TX STATUS
006A 72B3        300        JB3     SRTBIT          ;THIS IS START OF START BIT
006C 1A          301        INC     R2              ;INC TX TICK COUNTER
006D 2303        302        MOV     A,#1XTIC        ;TEST TICK COUNTER MOD 4
006F 5A          303        ANL     A,R2
0070 96B0        304        JNZ     RETURN          ;IF NON-ZERO, MIDDLE OF BIT
0072 FD          305        MOV     A,TXSTS         ;ZERO, GET TXSTS
0073 37          306        CPL     A               ;COMPLEMENT FOR 0 TEST
0074 129C        307        JB0     XMT4            ;TEST TX FLAG
                 308                                ;0 - NOT TXING, CHECK FOR NEW CHR
                 309                                ;1 - CURRENTLY IN CHR
0076 2328        310        MOV     A,#TXEND        ;CHECK FOR END OF DATA AND STOP
0078 DA          311        XRL     A,R2            ;XOR WITH CURRENT TICK COUNT
0079 9681        312        JNZ     XMT1            ;NOT DONE, CONTINUE
007B FD          313        MOV     A,TXSTS         ;DONE, GET TXSTS
007C 53FE        314        ANL     A,#NOT TXFLG    ;RESET TX FLAG
007E AD          315        MOV     TXSTS,A         ;RESTORE TXSTS
007F 04B0        316        JMP     RETURN          ;GO EXIT
                 317 ;
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                318 ;CHECK IF IT'S TIME FOR STOP BIT
                319 ;
0081 2324       320 XMT1:   MOV     A,#STPEND       ;CHECK FOR STOP BIT TIME
0083 DA         321         XRL     A,R2            ;COMPARE WITH TICK COUNTER
0084 968C       322         JNZ     XMT2            ;NOT TIME, DO NEXT BIT
                323 ;
                324 ;TRANSMIT STOP BIT
                325 ;
0086 FD         326         MOV     A,TXSTS         ;GET TX STATUS
0087 4304       327         ORL     A,#MARK         ;SETUP PIPELINED STOP BIT
0089 AD         328         MOV     TXSTS,A         ;RESTORE TX STATUS
008A 04B0       329         JMP     RETURN          ;RETURN
                330 ;
                331 ;IN MIDDLE OF CHR - TRANSMIT NEXT BIT
                332 ;
008C FB         333 XMT2:   MOV     A,R3            ;GET TX SERIALIZER
008D 67         334         RRC     A               ;ROTATE NEXT BIT INTO CARRY
008E AB         335         MOV     R3,A            ;RESTORE SERIALIZER
008F FD         336         MOV     A,TXSTS         ;GET TX STATUS FOR PIPELINED DATA
0090 F697       337         JC      XMT3            ;OUTPUT A MARK IF 1
0092 53FB       338         ANL     A,#SPACE        ;RESET TXDATA BIT
0094 AD         339         MOV     TXSTS,A         ;RESTORE TX STATUS
0095 04B0       340         JMP     RETURN          ;GO EXIT
0097 4304       341 XMT3:   ORL     A,#MARK         ;SET TXDATA BIT
0099 AD         342         MOV     TXSTS,A         ;RESTORE TX STATUS
009A 04B0       343         JMP     RETURN          ;GO EXIT
                344 ;
                345 ;TEST REQUEST FLAG SINCE NOT CURRENTLY TRANSMITTING
                346 ;
009C 32A8       347 XMT4:   JB1     XMT5            ;TEST TX REQUEST FLAG
                348                                 ;0 - NO CHR WAITING IN BUFFER
                349                                 ;1 - CHR WAITING IN BUFFER
009E FC         350         MOV     A,R4            ;CHR WAITING, GET IT FROM HOLDING
009F AB         351         MOV     R3,A            ;PUT IN SERIALIZER
00A0 FD         352         MOV     A,TXSTS         ;GET TXSTS
00A1 53FD       353         ANL     A,#NOT REQFLG   ;RESET REQUEST FLAG
00A3 4309       354         ORL     A,#TXFLG OR SBIT        ;SET TX AND START BIT FLAGS
00A5 53FB       355         ANL     A,#SPACE        ;SETUP TXDATA FOR START BIT
00A7 AD         356         MOV     TXSTS,A         ;RESTORE TXSTS
                357 ;
                358 ;TX BUFFER EMPTY - SET TXINT PIN AND BIT
                359 ;
00A8 8A08       360 XMT5:   ORL     P2,#TXINT       ;SET TXINT PIN
00AA C5         361         SEL     RB0             ;SWITCH FOR STS
00AB FF         362         MOV     A,R7            ;GET STS
00AC 4320       363         ORL     A,#TXBIT        ;SET TXINT BIT
00AE AF         364         MOV     R7,A            ;RESTORE STS
00AF 90         365         MOV     STS,A           ;LOAD STATUS
                366 ;
                367 ;*************************************************************************
                368 ;
                369 ;EXIT FOR TIMER INTERRUPT ROUTINE POINT
                370 ;
                371 ;*************************************************************************
                372 ;
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

0080 D5          373 RETURN: SEL     RB1             ;MAKE SURE WE'RE IN RB1
0081 FF          374         MOV     A,R7            ;RESTORE A
0082 93          375         RETR                    ;RETURN WITH RESTORE
                 376 ;
                 377 ;*************************************************************************
                 378 ;
                 379 ;GET HERE IF INTERRUPT IS FIRST FOR START BIT - CLEAR START BIT FLAG IN
                 380 ;TXSTS AND SETUP TX TICK COUNTER.
                 381 ;
                 382 ;*************************************************************************
                 383 ;
0083 53F7        384 SRTBIT: ANL     A,#NOT SBIT     ;RESET START BIT FLAG IN TXSTS
0085 AD          385         MOV     TXSTS,A         ;RESTORE TX STATUS
0086 BA01        386         MOV     R2,#01H         ;INITIALIZE TX TICK COUNTER
0088 0480        387         JMP     RETURN          ;RETURN
                 388 ;
                 389 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                390 ;
                391 ;*****************************************************************************
                392 ;
                393 ;COMMAND RECOGNIZER - GET HERE FROM IBF WRITE WITH F1 SET.  COMMAND
                394 ;IS STORED IN R6.  BAUD RATE SELECTION BITS ARE EVALUATED RIGHT TO LEFT.
                395 ;THE FIRST SET BIT FOUND DETERMINES THE BAUD RATE.  IF AN INVALID COMMAND
                396 ;IS DETECTED, BOTH F1 AND F0 ARE SET AND NO ACTION IS TAKEN.
                397 ;THE TIMER BAUD RATE CONSTANT IS SET TO TWO COUNTS LESS THAN THE DESIRED
                398 ;NUMBER.
                399 ;
                400 ;*****************************************************************************
                401 ;
0100            402        ORG     0100H
                403 ;
0100 D5         404 CMD:   SEL     RB1             ;SELECT RB1
0101 22         405        IN      A,DBB           ;READ COMMAND
0102 AE         406        MOV     R6,A            ;SAVE COMMAND IN R6
0103 F227       407        JB7     IOER            ;IF BIT 7 SET, IO OPERATION
0105 53E0       408        ANL     A,#0E0H         ;TEST TOP 3 BITS
0107 963A       409        JNZ     ERROR           ;IF NON-ZERO, ERROR
0109 C5         410        SEL     RB0             ;IO FLAG IN RB0
010A 1221       411        JB0     CMD2            ;IF BIT 0=1, OUTPUT PORT
010C 89FF       412        ORL     P1,#0FFH        ;INPUT PORT, SET ALL HIGH
010E FB         413        MOV     A,RXSTS         ;GET RXSTS
010F 53BF       414        ANL     A,#NOT IODIR    ;RESET IO DIRECTION FLAG
0111 AB         415        MOV     RXSTS,A         ;RESTORE RXSTS
0112 D5         416 CMD1:  SEL     RB1             ;BAUD RATE CONSTANTS IN RB1
0113 B920       417        MOV     R1,#TIMCON      ;POINT AT TIMER CONSTANT LOCATION
0115 FE         418        MOV     A,R6            ;GET COMMAND
0116 323E       419        JB1     B110            ;110 BAUD SELECTED
0118 5242       420        JB2     B300            ;300 BAUD SELECTED
011A 7246       421        JB3     B600            ;600 BAUD SELECTED
011C 924A       422        JB4     B1200           ;1200 BAUD SELECTED
011E B5         423        CPL     F1              ;RESET F1
011F 4414       424        JMP     MNLP1           ;DONE, JUMP BACK TO MAIN LOOP
                425 ;
                426 ;PORT IS SELECTED AS OUTPUT PORT - SET IO DIRECTION FLAG
                427 ;
0121 FB         428 CMD2:  MOV     A,RXSTS         ;GET RXSTS
0122 4340       429        ORL     A,#IODIR        ;SET IO DIRECTION FLAG
0124 AB         430        MOV     RXSTS,A         ;RESTORE RXSTS
0125 2412       431        JMP     CMD1            ;CONTINUE
                432 ;
                433 ;HERE WITH EITHER IO OR RESET ERROR COMMAND
                434 ;
0127 D231       435 IOER:  JB6     ERRST           ;IF BIT 6 SET, RESET ERROR FLAGS
0129 C5         436        SEL     RB0             ;IO FLAG IN RXSTS
012A FB         437        MOV     A,RXSTS         ;GET RXSTS
012B 4380       438        ORL     A,#IOFLG        ;SET IO FLAG
012D AB         439        MOV     RXSTS,A         ;RESTORE RXSTS
012E B5         440        CPL     F1              ;RESET F1
012F 4414       441        JMP     MNLP1           ;DONE, JUMP BACK TO MAIN LOOP
                442 ;
                443 ;RESET ERROR COMMAND
                444 ;
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

0131 C5         445 ERRST: SEL    RB0           ;STS IN RB0
0132 FF         446        MOV    A,R7          ;GET STS
0133 533F       447        ANL    A,#RSTERR     ;RESET ERROR FLAGS
0135 AF         448        MOV    R7,A          ;RESTORE STS
0136 90         449        MOV    STS,A         ;LOAD STATUS
0137 B5         450        CPL    F1            ;RESET F1
0138 4414       451        JMP    MNLP1         ;DONE, BACK TO MAIN LOOP
                452 ;
                453 ;COMMAND ERROR - SET BOTH F1 AND F0
                454 ;
013A 85         455 ERROR: CLR    F0            ;SET F0
013B 95         456        CPL    F0
013C 4414       457        JMP    MNLP1         ;DONE, BACK TO MAIN LOOP
                458 ;
                459 ;110 BAUD CONSTANTS
                460 ;
013E B954       461 B110:  MOV    R1,#-(174D-2D) ;LOAD 110 BAUD CONSTANT
0140 244C       462        JMP    STTIMR        ;GO START TIMER
                463 ;
                464 ;300 BAUD CONSTANTS
                465 ;
0142 B9C2       466 B300:  MOV    R1,#-(64D-2D)  ;LOAD 300 BAUD CONSTANT
0144 244C       467        JMP    STTIMR        ;GO START COUNTER
                468 ;
                469 ;600 BAUD CONSTANTS
                470 ;
0146 B9E2       471 B600:  MOV    R1,#-(32D-2D)  ;LOAD 600 BAUD CONSTANT
0148 244C       472        JMP    STTIMR        ;GO START COUNTER
                473 ;
                474 ;1200 BAUD CONSTANTS
                475 ;
014A B9F2       476 B1200: MOV    R1,#-(16D-2D)  ;LOAD 1200 BAUD CONSTANT
                477 ;
                478 ;START COUNTER
                479 ;
014C F9         480 STTIMR: MOV   A,R1          ;GET COUNTER CONSTANT
014D 62         481        MOV    T,A           ;LOAD COUNTER
014E 45         482        STRT   CNT           ;START COUNTER
014F 25         483        EN     TCNTI         ;ENABLE TIMER INTERRUPTS
0150 B5         484        CPL    F1            ;RESET F1
0151 4414       485        JMP    MNLP1         ;DONE, BACK TO MAIN LOOP
                486 ;
                487 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                488 ;
                489 ;*****************************************************************************
                490 ;
                491 ;DATA ROUTINE - GET HERE WITH IBF WRITE WITH F1 RESET.  THIS ROUTINE
                492 ;FIRST TESTS IF THE I/O FLAG IS SET IN THE RXSTS REGISTER.  IF SO, THE DATA
                493 ;IS FOR THE OUTPUT PORT.  OTHERWISE, THE DATA IS FOR THE TRANSMITTER AND
                494 ;IS PLACED IN THE TX BUFFER REGISTER.  THE TXINT BIT AND PIN ARE RESET.
                495 ;
                496 ;*****************************************************************************
                497 ;
0153 C5         498 DATA:   SEL     RB0             ;DATA HANDLED MOSTLY IN RB0
0154 FB         499         MOV     A,RXSTS         ;GET RXSTS
0155 F267       500         JB7     IODATA          ;IF IO FLAG SET, DATA IN FOR I/O
0157 FF         501         MOV     A,R7            ;GET STS
0158 53DF       502         ANL     A,#NOT TXBIT    ;RESET TXINT BIT IN STS
015A AF         503         MOV     R7,A            ;RESTORE STS
015B 90         504         MOV     STS,A           ;LOAD STATUS
015C 9AF7       505         ANL     P2,#NOT TXINT   ;RESET TXINT PIN
015E D5         506         SEL     RB1             ;TXSTS IN RB1
015F 22         507         IN      A,DBB           ;READ DATA
0160 AC         508         MOV     R4,A            ;PUT DATA IN TX BUFFER
0161 FD         509         MOV     A,TXSTS         ;GET TXSTS
0162 4302       510         ORL     A,#REQFLG       ;SET REQUEST FLAG IN TXSTS
0164 AD         511         MOV     TXSTS,A         ;RESTORE TXSTS
0165 4414       512         JMP     MNLP1           ;BACK TO MAIN LOOP
                513 ;
                514 ;IO DATA ROUTINE
                515 ;
0167 537F       516 IODATA: ANL     A,#NOT IOFLG    ;RESET IO FLAG
0169 AB         517         MOV     RXSTS,A         ;RESTORE RXSTS
016A 22         518         IN      A,DBB           ;READ IO DATA FROM DBBIN
016B 39         519         OUTL    P1,A            ;OUTPUT TO PORT 1
016C 4414       520         JMP     MNLP1           ;DONE, BACK TO MAIN LOOP
                521 ;
                522 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                523 ;
                524 ;*****************************************************************************
                525 ;
                526 ;INITIALIZATION - GET HERE AT RESET.  THIS ROUTINE RESETS THE INTERRUPT
                527 ;OUTPUTS AND ENABLES THEM, AND CLEARS THE APPROPRIATE STATUS AND DATA
                528 ;REGISTERS.
                529 ;
                530 ;*****************************************************************************
                531 ;
0200            532         ORG     0200H
                533 ;
0200 9AF7       534 INIT:   ANL     P2,#0F7H            ;RESET TXINT PIN
0202 F5         535         EN      FLAGS           ;ENABLE INTERRUPTS OUTPUT
0203 2300       536         MOV     A,#ZERO         ;CLEAR A
0205 AB         537         MOV     RXSTS,A         ;CLEAR RXSTS
0206 AD         538         MOV     R5,A            ;CLEAR RX TICK COUNTER
0207 AF         539         MOV     R7,A            ;CLEAR STS
0208 D5         540         SEL     RB1             ;SWITCH BANKS
0209 AE         541         MOV     R6,A            ;CLEAR CONFIGURE STORE
020A BD04       542         MOV     TXSTS,#MARK     ;SETUP PIPELINED TX DATA
                543 ;
                544 ;*****************************************************************************
                545 ;
                546 ;MAIN LOOP - IBF AND OBF ARE HANDLED IN THIS LOOP.  IF IBF=1, THE
                547 ;APPROPRIATE COMMAND OR DATA ROUTINE IS ACCESSED.  IF IBF=0, THEN OBF
                548 ;IS TESTED.  IF OBF=1, IBF IS TESTED AGAIN.  AS SOON AS OBF=0, RXSTS
                549 ;IS EXAMINED TO SEE IF DATA IS WAITING FOR OUTPUT.  WHEN RX DATA
                550 ;READY IS SET, F0 IS SET AND F1 IS CLEARED, AND THE DATA IS TRANSFERRED
                551 ;FROM THE RX HOLDING REGISTER INTO DBBOUT AFTER TESTING FOR ERROR
                552 ;FLAGS.  ANY ERROR FLAGS SET ARE TRANSFERRED TO THE STATUS REGISTER.
                553 ;IF THE I/O FLAG IS SET, THE PORT IS READ AND THE DATA TRANSFERRED TO
                554 ;DBBOUT.
                555 ;
                556 ;*****************************************************************************
                557 ;
020C D614       558 MNLOOP: JNIBF   MNLP1           ;IF IBF=0, TEST OBF
020E 7612       559         JF1     CMDJ1           ;IBF=1, TEST F1 FOR COMMAND
0210 2453       560         JMP     DATA            ;F1=0, JUMP TO DATA ROUTINE
0212 2400       561 CMDJ1:  JMP     CMD             ;OUT-OF-PAGE COMMAND JUMP
0214 860C       562 MNLP1:  JOBF    MNLOOP          ;WAIT UNTIL DBBOUT IS FREE
0216 C5         563         SEL     RB0             ;RXSTS IN RB0
0217 FB         564         MOV     A,RXSTS         ;GET RXSTS
0218 721E       565         JB3     RXRDY           ;TEST RX DATA READY FLAG
021A F23C       566         JB7     IOFLAG          ;TEST IO FLAG
021C 440C       567         JMP     MNLOOP          ;LOOP
                568 ;
                569 ;RX DATA READY - TRANSFER TO DBBOUT
                570 ;
021E 85         571 RXRDY:  CLR     F0              ;SET F0
021F 95         572         CPL     F0
0220 A5         573         CLR     F1              ;RESET F1
0221 922E       574         JB4     RXF             ;CHECK FRAMING ERROR FLAG
0223 FB         575 RXRDY1: MOV     A,RXSTS         ;GET RXSTS
0224 B235       576         JB5     RXO             ;CHECK FOR OVERRUN ERROR
0226 FB         577 RXRDY2: MOV     A,RXSTS         ;GET RXSTS AGAIN
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

0227 53C7        578        ANL     A,#NOT (DATRDY OR FRAMER OR OVRUN)      ;RESET SOME FLAGS
0229 AB          579        MOV     RXSTS,A          ;RESTORE RXSTS
022A FC          580        MOV     A,R4             ;GET DATA FROM HOLDING REG
022B 02          581        OUT     DBB,A            ;PUT IN DBBOUT
022C 440C        582        JMP     MNLOOP           ;LOOP
                 583 ;
                 584 ;FRAMING ERROR FLAG SET
                 585 ;
022E FF          586 RXF:   MOV     A,R7             ;GET STS
022F 4340        587        ORL     A,#FESTS         ;SET FRAMING ERROR FLAG
0231 AF          588        MOV     R7,A             ;RESTORE STS
0232 90          589        MOV     STS,A            ;LOAD STATUS
0233 4423        590        JMP     RXRDY1           ;CONTINUE
                 591 ;
                 592 ;OVERRUN ERROR FLAG SET
                 593 ;
0235 FF          594 RXO:   MOV     A,R7             ;GET STS
0236 4380        595        ORL     A,#OVSTS         ;SET OVERRUN ERROR FLAG
0238 AF          596        MOV     R7,A             ;RESTORE STS
0239 90          597        MOV     STS,A            ;LOAD STATUS
023A 4426        598        JMP     RXRDY2           ;CONTINUE
                 599 ;
                 600 ;IO FLAG SET - TEST DIRECTION
                 601 ;
023C FB          602 IOFLAG: MOV     A,RXSTS          ;GET RXSTS
023D D20C        603        JB6     MNLOOP           ;PORT IS OUTPUT - NO ACTION
023F 85          604        CLR     F0               ;RESET F0
0240 A5          605        CLR     F1               ;SET F1
0241 B5          606        CPL     F1
0242 537F        607        ANL     A,#NOT IOFLG     ;RESET IO FLAG
0244 AB          608        MOV     RXSTS,A          ;RESTORE RXSTS
0245 09          609        IN      A,P1             ;READ PORT 1
0246 02          610        OUT     DBB,A            ;PUT DATA IN DBBOUT
0247 440C        611        JMP     MNLOOP           ;LOOP
                 612 ;
                 613        END

USER SYMBOLS
ASCMSK 007F   B110   013E   B1200  014A   B300   0142   B600   0146   BFFLG  0004   CMD    0100   CMD1   0112
CMD2   0121   CMDJ1  0212   DATA   0153   DATRDY 0008   ERROR  013A   ERRST  0131   FESTS  0040   FRAMER 0010
INIT   0200   INT1   000B   IODATA 0167   IODIR  0040   IOER   0127   IOFLAG 023C   IOFLG  0080   MARK   0004
MKOUT  0001   MNLOOP 020C   MNLP1  0214   MOUT   0019   OVRUN  0020   OVSTS  0080   RCV    001B   RCV1   0026
RCV2   0033   RCV3   0038   RCV4   0042   RCV5   004D   RCV6   0051   RCV7   0055   RCV8   0060   RCV9   0064
REQFLG 0002   RESET  0000   RETURN 0080   RSTERR 003F   RXF    022E   RXFLG  0001   RXINTL 0080   RXO    0235
RXRDY  021E   RXRDY1 0223   RXRDY2 0226   RXSTS  0003   SBIT   0008   SPACE  00FB   SPOUT  00FE   SRTBIT 0003
SRTFLG 0002   STPEND 0024   STTIMR 014C   TICOUT 0040   TICSRT 0004   TIMCON 0020   TIMINT 0007   TXBIT  0020
TXEND  0028   TXFLG  0001   TXINT  0008   TXOUT  0012   TXSTS  0005   TXTIC  0003   XMIT   0068   XMT1   0081
XMT2   008C   XMT3   0097   XMT4   009C   XMT5   00A8   ZERO   0000

ASSEMBLY COMPLETE,  NO ERRORS
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 ;
                    2 ;TEST ROUTINE WHICH OUTPUTS THE ASCII CHARACTER SET TO THE
                    3 ;UPI TRANSMITTER AND DISPLAYS ON THE 80/30 CONSOLE ANY
                    4 ;CHARACTERS RECEIVED BY THE UPI RECEIVER.
                    5 ;
                    6 ;INPUTS: NOTHING
                    7 ;OUTPUTS: CHARACTERS TO CONSOLE
                    8 ;CALLS: NOTHING
                    9 ;
4000                10          ORG     4000H
00DF                11 MODE53   EQU     0DFH            ;8253 CONTROL PORT
00DC                12 CNT0     EQU     0DCH            ;8253 CNT 0 PORT
00E5                13 CMD      EQU     0E5H            ;UPI COMMAND PORT
00E5                14 STATUS   EQU     0E5H            ;UPI STATUS PORT
00E4                15 DBBIN    EQU     0E4H            ;UPI DBBIN PORT
00E4                16 DBBOUT   EQU     0E4H            ;UPI DBBOUT PORT
0020                17 TXINT    EQU     20H             ;TXINT MASK
0001                18 OBF      EQU     01H             ;OBF MASK
0002                19 IBF      EQU     02H             ;IBF MASK
00ED                20 STAT51   EQU     0EDH            ;8251 STATUS PORT
00EC                21 DATA51   EQU     0ECH            ;8251 DATA PORT
0001                22 TXRDY    EQU     01H             ;8251 TXRDY MASK
                    23 ;
4000 3E36           24 START:   MVI     A,36H           ;8253 CNT0 MODE WORD
4002 D3DF           25          OUT     MODE53          ;8253 CONTROL PORT
4004 3E10           26          MVI     A,10H           ;DIVIDE BY 16D
4006 D3DC           27          OUT     CNT0            ;8253 CNT0 PORT LSB
4008 3E00           28          MVI     A,00H           ;
400A D3DC           29          OUT     CNT0            ;8253 CNT0 PORT MSB
400C 0620           30          MVI     B,20H           ;INITIALIZE OUTPUT CHR
400E 3E10           31          MVI     A,10H           ;CONFIGURE COMMAND - 1200 BAUD
4010 D3E5           32          OUT     CMD             ;UPI COMMAND PORT
4012 DBE5           33 POLL1:   IN      STATUS          ;READ UPI STATUS
4014 E621           34          ANI     TXINT OR OBF    ;TEST TXINT AND OBF
4016 CA1240         35          JZ      POLL1           ;WAIT UNTIL ONE IS SET
4019 DBE5           36          IN      STATUS          ;READ UPI STATUS AGAIN
401B E601           37          ANI     OBF             ;WAS IT OBF?
401D C23840         38          JNZ     RX              ;YES, GO DO RECEIVER
                    39                                  ;NO, MUST BE TRANSMITTER
4020 78             40          MOV     A,B             ;GET NEXT CHR FOR OUTPUT
4021 D3E4           41          OUT     DBBIN           ;OUTPUT TO UPI DBBIN
4023 FE5A           42          CPI     'Z'             ;WAS IT LAST CHR?
4025 CA3340         43          JZ      NEWB            ;YES, RESET REG. B
4028 04             44          INR     B               ;OTHERWISE, INC B
4029 DBE5           45 POLL2:   IN      STATUS          ;TEST IF IBF STILL SET
402B E602           46          ANI     IBF             ;TEST IBF
402D C22940         47          JNZ     POLL2           ;WAIT UNTIL IBF=0
4030 C31240         48          JMP     POLL1           ;BEFORE LOOKING AT STATUS AGAIN
                    49 ;
4033 0620           50 NEWB:    MVI     B,20H           ;RESET REG. B
4035 C32940         51          JMP     POLL2           ;GO BACK
                    52 ;
```

```
 LOC  OBJ        SEQ        SOURCE STATEMENT

 4038 DBE4         53 RX:    IN      DBBOUT        ;READ DBBOUT FOR RECEIVED CHR
 403A 4F           54        MOV     C,A           ;SAVE IT IN C
 403B DBED         55 RX1:   IN      STAT51        ;READ 8251 STATUS
 403D E601         56        ANI     TXRDY         ;TEST TXRDY
 403F CA3B40       57        JZ      RX1           ;WAIT UNTIL READY
 4042 79           58        MOV     A,C           ;GET CHR
 4043 D3EC         59        OUT     DATA51        ;OUTPUT CHR TO CONSOLE
 4045 C31240       60        JMP     POLL1         ;GO TEST UPI AGAIN
                   61 ;
                   62 END

PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
CMD    A 00E5   CNT0   A 00DC   DATA51 A 00EC   DBBIN  A 00E4   DBBOUT A 00E4   IBF    A 0002   MODE53 A 00DF
NEWB   A 4033   OBF    A 0001   POLL1  A 4012   POLL2  A 4029   RX     A 4038   RX1    A 403B   START  A 4000
STAT51 A 00ED   STATUS A 00E5   TXINT  A 0020   TXRDY  A 0001

ASSEMBLY COMPLETE,  NO ERRORS
```

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 • (408) 987-8080
Printed in U.S.A./T-202.1/1278/ 5K BL