μPD70320 / 70322
μPD70330 / 70332
μPD70P322
μPD79011

USER'S MANUAL

μPD70320 / 70322
μPD70330 / 70332
μPD70P322
μPD79011

User`s Manual

Contents:

CONTENTS

Part 1: μPD70320/70322 (V25) User's Manual **Page**

Part 3: μPD70P322 (V25 EPROM) User's Manual

Part 4: μPD79011 Single Chip Microcomputer (V25) with Real Time Operating System: User's Manual

Part 4: μPD79011 Single Chip Microcomputer (V25) with Real Time Operating System; Technical Reference Manual

Part 5: μPD70320/70322, μPD70330/70332, μPD79011
Instruction Set; User's Manual

Tables

Part 1.

μPD 70320/70322

(V25)

User´s Manual

CHAPTER 1 INTRODUCTION

The uPD70322 (V25™) is a single-chip microcomputer which is software compatible with the uPD70108/70116 (V20™/V30™). Although the uPD70322 maintains software compatibility with the uPD70108/70116, its interrupt function, which is essential for control applications, is significantly enhanced. Consequently, powerful high-speed interrupt processing is realized. In addition, an interval timer, serial interface, and DMA controller, etc., are all integrated on a single-chip.

The uPD70322 has 16K bytes of built-in mask ROM for storing programs which require high-speed processing such as the operating system kernal nucleus.

Since the uPD70322 is a CMOS device, it realizes low power consumption. Also, additional power saving features such as a standby mode make it possible for the uPD70322 to operate or retain data with an even greater reduction in power consumption.

The uPD70322 is well suited as the main controller in systems which must process a large volume of data needed for controlling a number of pieces equipment. The uPD70322 is also suited for application to systems which handle a large volume of data such as portable word processors, portable microcomputers, POS terminals, etc.

The uPD70320 is scheduled for limited production and is identical to the uPD70322 except that it contains additional built-in mask ROM.

1.1 Features

- o 16-bit internal architecture, 8-bit external data bus
- o Software compatible with the uPD70108/70116 in the native mode (new instructions are added)
- o Minimum instruction cycle: 400ns (10MHz, 5V)
- o Built-in ROM: 16,384 words x 8 bits (uPD70322)
- o Built-in RAM: 256 words x 8 bits
- o Memory mapped on-chip peripheral hardware (special function register)
- o Input port with comparator (port T): 8 bits
- o I/O line (input port: 4 bits, input/output port: 20 bits)
- o Serial interface (with a built-in baud rate generator): 2 channels (Asynchronous mode, I/O interface mode)
- o Interrupt controller
  - . Programmable priority (8 levels)
  - . Vector interrupt function
  - . Register bank switching function
  - . Macro service function
- o DRAM, pseudo-SRAM refresh function
- o DMA controller: 2 channels
- o Time base counter
- o built-in clock generator
- o Programmable wait function
- o Standby function (STOP/HALT)
- o Variable instruction cycle: 400ns, 800ns, 1.6us(10MHz, 5V)
- o CMOS
- o Single Power Supply

1.2 Ordering Information for V25 family:

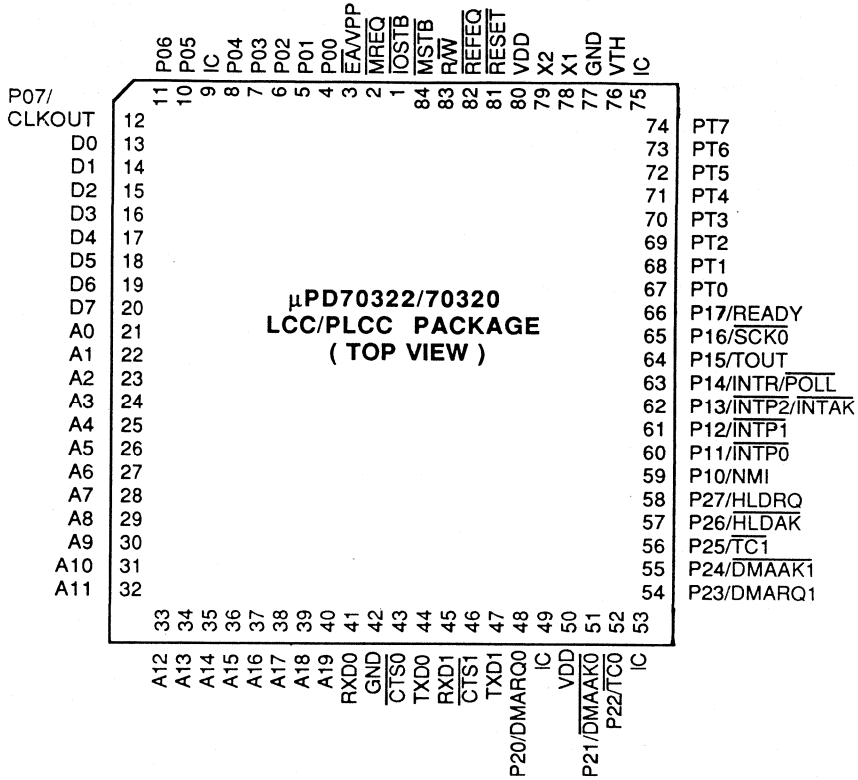
Article	Ext.Data Bus	Package	Ext.Clock	Internal Rom
uPD70322L-xxx	8bit	84pinPLCC	10 MHz	Mask Rom
uPD70320L	8bit	84pinPLCC	10 MHz	Romless
uPD70322L-8-xxx	8bit	84pinPLCC	16 MHz	Mask Rom
uPD70320L-8	8bit	84pinPLCC	16 MHz	Romless
uPD70322GJ-xxx	8bit	94pinPFlat	10 MHz	Mask Rom
uPD70320GJ	8bit	94pinPFlat	10 MHz	Romless
uPD70322GJ-8-xxx	8bit	94pinPFlat	16 MHz	Mask Rom
uPD70320GJ-8	8bit	84pinPFlat	16 MHz	Romless
uPD70P322L	8/16bit	84pinPLCC	10 MHz	OTP
uPD70P322K	8/16bit	84pinCLCC	10 MHz	EPROM

Note:

xxx = Rom Code
PFlat = Plastic Flat
CLCC = Ceramic LCC with Window

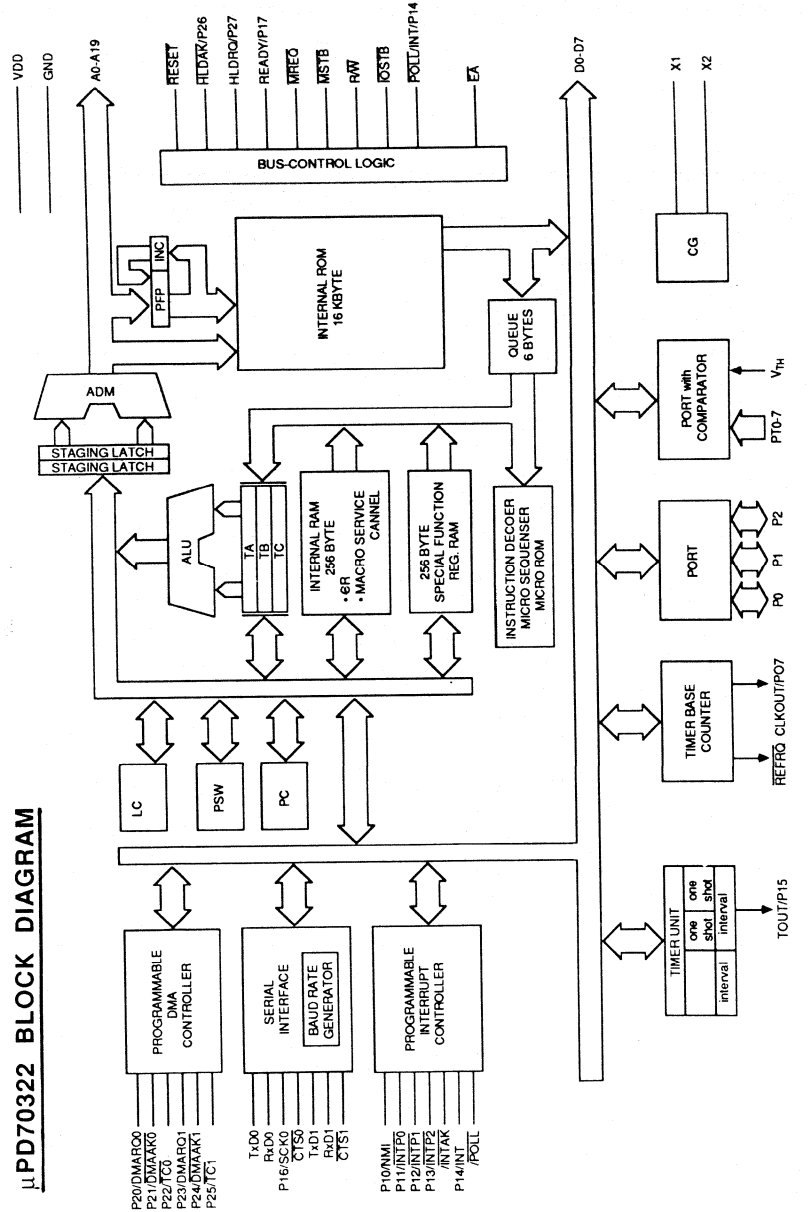
1.3 uPD70322/70320 Pin Configuration

84-pin PLCC (Top View)



IC = Internal Connected
Pull-up (5 to 100kΩ) to each IC-pin

1.4 uPD70322/70320 Block Diagram



CHAPTER 2 PIN FUNCTIONS

2.1 Pin Function Table

2.1.1 Port pins

Pin name	Input/Output	Port function	Control function
P00-P06	I/O	These pins function as a 8-bit I/O port in which input/output can be specified for each bit.	—
P07/CLKOUT	I/O/Output		System clock output
P10/NMI	Input/Input	This pin function as an input port and also functions as a the non-maskable interrupt-request input.	—
P11/$\overline{\text{INTP0}}$		These pins function as an input port, and also function as the external interrupt-request input.	
P12/$\overline{\text{INTP1}}$			
P13/$\overline{\text{INTP2}}$/$\overline{\text{INTAK}}$	I/I/Output		INT acknowledge signal output
P14/$\overline{\text{POLL}}$/INT	I/O/I/Input	This port can be specified for either input or output, and also functions as the $\overline{\text{POLL}}$ input.	External interrupt-request input
P15/TOUT	I/O/Output	These pins function as an input/output port in which input/output can be specified for each bit.	Timer output
P16/$\overline{\text{SCK0}}$			serialclock output
P17/READY	I/O/Input		READY input

Pin name	Input/Output	Port function	Control function
P20/DMARQ0	I/O/Input	These pins function as an 8-bit input/ output port in which input/output can be specified for each bit.	DMA request input (CH0)
P21/$\overline{\text{DMAAK0}}$	I/O/Output		DMA acknowledge output (CH0)
P22/$\overline{\text{TC0}}$			DMA termination output (CH0)
P23/DMARQ1	I/O/Input		DNA request input (CH1)
P24/$\overline{\text{DMAAK1}}$	I/O/Output		DMA acknowledge output (CH1)
P25/$\overline{\text{TC1}}$			DMA termination output (CH1)
P26/$\overline{\text{HLDAK}}$	I/O/Input		HOLD acknowledge output
P27/HLDRQ	I/O/Input		HOLD input
PT0-PT7	Input	These pins function as an input port with an 8-bit comparator.	--

2.1.2 Non-port pins

Pin name	Input/ Output	Function
V_{TH}	Input	Inputs the comparator reference voltage.
TXD0	Output	Outputs serial data.
TXD1		
RXD0	Input	Inputs serial data.
RXD1		
$\overline{CTS0}$	Input/ Output	Inputs the CTS signal in the asynchronous mode, and inputs/outputs receive clock in the I/O interface mode.
$\overline{CTS1}$	Input	Inputs the CTS signal.
$\overline{RESET}$	Input	Inputs the reset signal.
$\overline{EA}$		Sets the ROM-less-mode.
X1		Pins for connecting the system clock crystal. To input the external clock, input the positive and negative phase of the external clock signal to the X1 and X2 pins, respectively.
X2		
D0-D7	Input/ Output	8-bit data bus
A0-A19	Output	Outputs 20-bit address.
$\overline{MREQ}$		This output indicates that the memory bus cycle is initailed.
$\overline{MSTB}$		Outputs the memory read or memory write storbe.

(Cont'd)

Pin name	Input/ Output	Function
R/$\overline{W}$	Output	Outputs the read or write cycle identification signal.
$\overline{REFRQ}$		Outputs the DRAM refreash pulse.
$\overline{IOSTB}$		Outputs the I/O read or I/O write strobe.
V_{DD}		Positive voltage power supply pins. Both V_{DD} pins must be connected.
GND		GND pins. Both GND pin must be connected.
I.C.		Internally connected pin. this pin must be externally set to a high level via pull-up. (5 to 100kΩ)

2.1.3 Pin status in each mode

Pin \ Operating mode		Hold	HALT	STOP	During RESET	Others
Port	During output operation in port mode	Retain	Retain	Retain	Hi-Z	Continue
	During output operation in control mode	Continue	Continue	Retain	Hi-Z	Continue
T_XD0, T_XD1		Continue	Continue	Retain	Hi-Z	Continue
$\overline{CTS0}$		Continue	Continue	Retain	Hi-Z	Continue
D0-D7		Hi-Z	Hi-Z	Hi-Z	Hi-Z	Continue
A0-A19		Hi-Z	Retain (Note)	Retain (Note)	Hi-Z	Continue
$\overline{MREQ}$		Hi-Z	High level	High level	Hi-Z	Continue
$\overline{MSTB}$		Hi-Z	High level	High level	Hi-Z	Continue
R/$\overline{W}$		Hi-Z	High level	High level	Hi-Z	Continue
$\overline{REFRQ}$		Hi-Z	Continue	Retain	Hi-Z	Continue
$\overline{IOSTB}$		Hi-Z	High level	High level	Hi-Z	Continue

Continue: Continues the specified operation.
Retain : Retains the condition of the previous mode.
Hi-Z : High impedance

Note: Although the address data is retained, the address output is undefined.

2.2 Pin Functions

2.2.1 P00-P07 (Port 0) ..... Three-state input/output

These pins function as an 8-bit I/O port (port 0) with an output latch. Pin P07 also functions as the CLKOUT output pin.

Pins P00 to P06 are used only for the port mode. Pin P07 can be selected for either the port or control mode by setting the port 0 mode control register (PMC0) (refer to Table 2-1).

Table 2-1 Port 0 Operation (n=0 to 7)

	PMC0n=1	PMC0n-0	
		PM0n=1	PM0n=0
P00		Input port	Output port
P01		Input port	Output port
P02		Input port	Output port
P03		Input port	Output port
P04		Input port	Output port
P05		Input port	Output port
P06		Input port	Output port
P07	CLKOUT output	Input port	Output port

(1) Port mode

Pins P00 to P07 can be set for either input or output per bit by setting the port 0 mode register (PM0). However, pin P07 must be set to the port mode by the PMC0 register (PMC0 register bit 7=0).

(2) Control mode

When set to the control mode by the PMC0 register (PMC0 register bit 7=1), pin P07 functions as the CLKOUT output pin.

(a) CLKOUT (Clock out) ..... This output shares P07.

This output pin outputs the system clock (CLK) generated in the clock generator. This system clock is used for the CPU and peripheral hardware.

A pin set for the output port retains the data of the previous mode in the hold or standby mode. When specified for CLDKOUT, pin P07 continues to output the clock signal in the hold or HALT mode. In the STOP mode, pin P07 retains the data of the previous mode.

Pins P00 to P07 become an input port $\overline{\text{RESET}}$ during input (high impedance output).

2.2.2 P10-P17 (Port 1) Three-State Input/Output

These pins function as an 8-bit I/O port (Port 1) with an output latch. Each of these pins also functions as a control signal pin.

The operation mode of pins P10 to P17 can be set for each bit to either the port or control mode by setting the port 1 mode control register (PMC1) (refer to Table 2-2). However, pins P10 to P12 can be used only in the port mode and function as interrupt request pins.

Table 2-2 Port 1 Operation (n=0 to 7)

	PMC1n=1	PMC1n=0	
		PM1n=1	PM1n=0
P10		NMI Input	
P11		$\overline{\text{INTP0}}$ input	
P12		$\overline{\text{INTP1}}$ input	
P13	$\overline{\text{INTAK}}$ Output	$\overline{\text{INTP2}}$ input	
P14	INT Input	Input port ($\overline{\text{POLL}}$ input)	Output port
P15	TOUT Output	Input port	Output port
P16	$\overline{\text{SCKO}}$ Output	Input port	Output port
P17	READY Input	Input port	Output port

(1) Port mode

When set to the port mode by setting the PMC1 register (PMC1n=0; n=0 to 7), pins P10 to P17 can be set for each bit to either the input or output port by setting the port 1 mode register (PM1).

Pins P10 to P13 can be used only for an input port, and function as interrupt-request input pins. When set for the input port, pin P14 also functions as the $\overline{\text{POLL}}$ input pin.

(a) NMI (Non-Maskable Interrupt) ..... This input shares pin P10.

This pin inputs the non-maskable interrupt request which cannot be masked by means of software.

The non-maskable interrupt is always accepted by the CPU; therefore, it has priority over any other interrupt.

The NMI input is detected at the effective edge specified by the external interrupt mode register (INTM). This input is sampled in each clock cycle, and the NMI input is

accepted when the active level input lasts for some clock cycles after it has changed from the inactive level to the active level. When the NMI input is accepted, a number 2 vector interrupt is generated after the completion of the instruction currently being executed.

This NMI input is also used to release the standby mode of the CPU.

(b) $\overline{\text{INTP0}}$ to $\overline{\text{INTP2}}$ (INT errupt from Peripheral 0 to 2) .....
These inputs share pins P11 to P13.

These external interrupt-request inputs can be masked by means soft ware.

The $\overline{\text{INTPn}}$ (n=0 to 2) input is detected at the effective edge specified by the external interrupt mode register (INTM). The INTPn (n=0 to 2) input is accepted when the active level input lasts for some clock cycles after it has changed from the inactive level to the active level.

The $\overline{\text{INTPn}}$ input is also used to release the HALT mode.

(c) $\overline{\text{POLL}}$ (poll) ..... This input shares pin P14/INT

The $\overline{\text{POLL}}$ input is checked by the POLL instruction.

If it is low, the execution of the next instruction is initiated. If it is high, the $\overline{\text{POLL}}$ input is checked every five clock cycles until the $\overline{\text{POLL}}$ input becomes low.

These functions are used to synchronize the CPU program and the operation of the external devices.

Note: The $\overline{\text{POLL}}$ functions when P14 is specified for the input mode; otherwise, it is assumed to be at low level when the POLL instruction is executed.

(2) Control mode

Pins P13 to P17 can be set individually as control pins by setting the PMC1 register.

(a) $\overline{\text{INTAK}}$ (Interrupt Acknowledge) ..... This output shares pin P13/$\overline{\text{INTP2}}$.

This is the acknowledge signal output for the software-maskable interrupt request (INT).

The $\overline{\text{INTAK}}$ signal becomes low when the CPU accepts the

INT signal. The external device inputs the interrupt vector to the CPU via the data bus (D0 to D7) in synchronization with this signal.

Together with the INT pin, this pin is used to connect the interrupt controller such as the uPD71059.

(b) INT (INT errupt) ..... This input shares pin P14/$\overline{\text{POLL}}$.

This pin inputs an interrupt request that can be masked by means of software.

This input is active at a high signal level, and is detected in the last clock cycle of an instruction. The external device confirms that the INT interrupt request is accepted by using the $\overline{\text{INTAK}}$ signal output from the CPU. The INT signal must be held high until the first $\overline{\text{INTAK}}$ signal is output.

Together with the $\overline{\text{INTAK}}$ pin, the INT pin is used to connect the interrupt controller such as the uPD71059.

The INT input is also used to release the HALT mode.

(c) TOUT (Timer Output) ..... This output shares pin P15

This is the output pin of the timer unit (channel 0).

(d) $\overline{\text{SCKO}}$ (Serial Clock) ..... This output shares pin 16.

This is the transmit clock output pin of the serial interface (channel 0).

(e) READY (Ready) ..... This input shares pin P17

This input pin is used to externally control the insertion of a wait state during the bus cycle (except during the memory refresh cycle).

The level of the $\overline{\text{READY}}$ pin is sensed, and the wait state is inserted when the $\overline{\text{READY}}$ pin is low.

A pin set for an output port retains the data of the previous mode during the hold or standby mode. A control signal output pin specified for each output port continues the specified operation in the hold or HALT mode, and retains the data of the previous mode in the STOP mode.

Pins P10 to P17 become an input port when a $\overline{\text{RESET}}$ command is input (high impedance output).

2.2.3 P20-P27 (Port 2) Three-State Input/Output

These pins function as an 8-bit I/O port (Port 2) with an output latch. Each of these pins also functions as a control signal pin.

The operation mode of pins P20 to P27 can be set for each bit to either the port or control mode by setting the port 2 mode control register (PMC2) (refer to Table 2-3).

Table 2-2 Port 2 Operation (n=0 to 7)

	PMC2n=1	PMC2n=0	
		PM2n=1	PM2n=0
P20	DMARQ0 input	Input port	Output port
P21	$\overline{\text{DMAAK0}}$ input	Input port	Output port
P22	$\overline{\text{TC0}}$ output	Input port	Output port
P23	DMARQ1 input	Input port	Output port
P24	$\overline{\text{DMAAK1}}$ otuput	Input port	Output port
P25	$\overline{\text{TC1}}$ output	Input port	Output port
P26	$\overline{\text{HLDAK}}$ output	Input port	Output port
P27	HLDRQ input	Input port	Output port

(1) Port mode

When set to the port mode by setting the PMC2 register (PMC2n=0; n=0 to 7), pins P20 to P27 can be set for each bit to either the input or output port by setting the port 2 mode register (PM2).

(2) Control mode

Pins P20 to P27 can be set as control pins by bit by setting the port mode control register (PMC2n=1; n=0 to 7).

(a) DMARQ0, DMARQ1 (DMA Request 0, 1) ..... These pins share pins P20 and P23.

These are the DMA request input pins of the DMA controller channels 0 and 1). These are active high signals.

(b) $\overline{\text{DMAAK0}}$, $\overline{\text{DMAAK1}}$ (DMA Acknowledge 0, 1) ..... These pins share pins P21 and P24.

These are the DMA acknowledge signal output pins of the DMA controller (channels 0 and 1). However, these signals are not output during memory-to-memory DMA transfer operations (burst mode, single step mode).

These pins signal pins are active at a low level.

(c) $\overline{\text{TC0}}$, $\overline{\text{TC1}}$ (Terminal Count 0, 1) ..... These pins share pins P22 and P25.

These pins output the DMA completion signal from the DAM controller (channels 0 and 1).

This signal is output when the corresponding terminal count (TC0, TC1) of the DMA service channel becomes 0.

These pins are active at a low signal level.

(d) $\overline{\text{HLDAK}}$ (Hold Acknowledge) ..... This signal shares pin P26.

This signal outputs the acknowledge signal which indicates that the uPD70322/70320 has accepted the hold request signal (HLDRQ) and that the bus is set to high impedance.

When this signal is active (low level), the address, data, and control bus go to high impedance.

(e) HLDRQ (Hold Request) ..... This signal shares pin P27.

This signal is used by an external device to request that the uPD70322/70320 relinquish the address, data, and control bus.

The HLDRQ input is active at a high signal level.

A pin which has been set for an output port retains the data of the previous mode during the hold or standby mode. A control signal output pin specified for an output port continues the specified operation in the hold or HALT mode, and retains the data of the previous mode in the STOP mode.

Pins P20 to P27 become an input port when $\overline{\text{RESET}}$ is input (high impedance output).

2.2.4 PT0-PT7 (Port with Comparator 0-7) ..... Input

These pins function as a T-port (PT0 to PT7) with a comparator which is capable of changing the threshold voltage (reference voltage) in 16 steps.

2.2.5 V_{TH} (Threshold Voltage) ..... Input

This pin inputs the T-port reference voltage.

2.2.6 TxD0, TxD1 (Transmit Data 0, 1) ..... Output

These pins output serial data from the serial interface (channels 0, 1).

In the asynchronuous mode, the transmit data is transmitted as a frame from the least significant bit (LSB) in the format which includes the start bit, character bits, parity bit, and stop bit. The TxD0 and TxD1 pins become mark state (1) when the transmit operation is disabled or when the serial register has no transmit data.

In the I/O interface mode (for TxD0 pin only), the transmit data length is fixed at 8 bits and transmitted from the most significant bit (MSB).

The TxD0 and TxD1 pins continue the specified operation in the hold or HALT mode. However, in the STOP mode, these pins retain the data of the previous mode.

These pins go to high impedance when $\overline{\text{RESET}}$ is input.

2.2.7 RxD0, RxD1 (Receive Data 0, 1) ..... Input

These are serial data input pins for the serial interface (channel 0, 1).

In the asynchronuous mode, when the receive operation is enabled and a low level is detected on the RxD0 or RxD1 pins, this low level is recognized as the start bit and receive operation is initiated.

In the I/O interface mode (RxD0 pin only), the receive data is input to the serial register at the rising edge of the receive clock.

2.2.8 $\overline{\text{CTS0}}$ (Clear to Send 0) ..... Input/Output

This is the CTS pin of the serial interface (channel 0).

In the asynchronous mode, the $\overline{\text{CTS0}}$ pin is active at a low level, thus enabling transmit operation. In the I/O interface mode, this pin functions as the receive clock input/output pin.

In the hold or HALT mode, the $\overline{\text{CTS0}}$ pin continues the specified operation; however, in the STOP mode, the $\overline{\text{CTS0}}$ pin retains the data of the previous mode.

When $\overline{\text{RESET}}$ is input, this pin functions as the input pin of the I/O interface (high impedance output).

2.2.9 $\overline{\text{CTS1}}$ (Clear to Send 1)

This is the CTS pin of the serial interface (channel 1).

The $\overline{\text{CTS1}}$ pin is a low-level input pin and enables transmit operation when the signal level on this pin is low (in the asynchronous mode).

2.2.10 $\overline{\text{RESET}}$ (Reset) ..... Input

This is a reset input pin which is active at a low signal level.

The $\overline{\text{RESET}}$ input is asynchronous. When this input is at a low level for a certain duration, the system reset is initiated regardless of the clock operation. This operation has priority over all other operations.

This reset input is used for normal initialization/start-up of the uPD70322/70320. In addition, this reset input is also used for releasing the standby mode (STOP/HALT).

2.2.11 $\overline{EA}$ (External Access) ..... Input

For the uPD70322, set this pin to a high level. For the uPD70320, set it to a low level.

2.2.12 X1, X2 (Crystal) ..... Input

The system clock oscillator crystal is connected across these pins.

To input the external clock, input the positive and negative phase of the external clock to the X1 and X2 pins.

2.2.13 D0-D7 (Data Bus 0-7) ..... Input/Output

These pins function as a 8-bit data bus input/output.

All of these pins go to high impedance in the hold or standby mode or when/$\overline{RESET}$ is input.

2.2.14 A0-A19 (Address Bus 0-19) ..... Output

These pins function as a 20-bit address bus output.

In the standby mode, the address data is retained; however, the address data output is undefined.

These pins go to high impedance in the hold mode or when /$\overline{RESET}$ is input.

2.2.15 $\overline{MREQ}$ (Memory Request) ..... Output

This pin outputs a low-level active signal which indicates that the memory bus cycle is started and the memory address on the address bus is valid.

In the standby mode, this pin is set to high level. In the hold mode or when /$\overline{RESET}$ is input, this pin goes to high impedance.

2.2.16 $\overline{MSTB}$ (Memory Strobe) ..... Output

Together with the $\overline{MREQ}$ and R/$\overline{W}$ signals, the $\overline{MSTB}$ pin is used to control memory accessing operations.

During the memory write cycle, this signal indicates that the data output to the data bus is valid. The output of the $\overline{MSTB}$ pin is the same as that of the $\overline{MREQ}$ pin except for the timing of the falling edge.

In the standby mode, this pin is set to a high level. In the hold mode or when /$\overline{\text{RESET}}$ is input, this pin goes to high impedance.

2.2.17 R/$\overline{\text{W}}$ (Read/Write Strobe) ..... Output

When the memory bus cycle is initiated, this signal identifies whether the cycle is a memory read or memory write cycle.

The high level on the R/$\overline{\text{W}}$ output indicates a memory read cycle, while a low level indicates a memory write cycle.

In the standby mode, this pin is set to a high level. In the hold mode or when /$\overline{\text{RESET}}$ is input, this pin goes to high impedance.

2.2.18 $\overline{\text{REFRQ}}$ (Refresh Request) ..... Output

This pin outputs the refresh pulse.

The output of the $\overline{\text{REFRQ}}$ pin is controlled by the contents of the refresh mode register (RFM).

In the HALT mode, the specified operation is continued.

However, in the STOP mode, the data of the previous mode is retained. In the hold mode or when /$\overline{\text{RESET}}$ is input, this pin goes to high impedance.

2.2.19 $\overline{\text{IOSTB}}$ (I/O Strobe) ..... Output

This pin outputs a low level active signal which indicates that the I/O bus cycle has been initiated.

A low level on the $\overline{\text{IOSTB}}$ pin indicates that the I/O address output to pins A0 to A15 is valid.

In the standby mode, this pin is set to a high level. In the hold mode or when /$\overline{\text{RESET}}$ is input, this pin goes to high impedance.

2.2.20 V_{DD} (Power Supply)

This is the power supply pin. The positive side of the power supply must be connected to this pin.

Vdd must be supplied to both Vdd pins.

2.2.21 GND (Ground)

This is the ground pin.

Both ground pins must connected to the GND potential.

2.2.22 I.C. (Internally Connected)

This pin must be externally pulled up to a high level by using a pull-up resistor. 5 to 100kΩ to each IC-pin.

CHAPTER 3 INTERNAL CPU FUNCTIONS

The CPU of the uPD70322/70320 is software-compatible with the uPD70116/70108 in the native mode.

3.1 Registers

The uPD70322/70320 has a general register set which is compatible with that of the uPD70116/70108. In addition, the uPD70322/70320 has various special function registers for controlling the on-chip peripheral hardware. All of these registers are mapped to the memory space. In particular, the built-in RAM is used for a general purpose register set. Therefore, a maximum of 8 banks of register sets can be provided for in the built-in RAM.

The address of these registers can be allocated every 4K bytes. This address is specified by the internal data area base register (IDB) which is one of the special function registers (refer to 3.4.2).

3.1.1 Register banks

The general purpose register set is mapped to the built-in RAM area. The general purpose register set is arranged in bank configuration, and up to eight banks can be set. Each bank uses 32 bytes. Of these eight banks, banks 0 and 1 are also used for the macro service channel (refer to 3.4.3) and DMA service channel (refer to 6.3.3). Banks 0 and 6 can also be accessed as the data memory (refer to 3.4.4).

Normally, the CPU uses bank 7 for program execution. Switching to other register banks is performed automatically by interrupt operation. After a bank is switched by an interrupt operation, returning to the original bank can be performed only by executing the RETRBI instruction (return from interrupt instruction). (This instruction is an addition to the uPD70322/70320) (refer to 4.4.2).

The register bank is configured as shown in Fig. 3-1. (+00H) and (+01H) in the register bank become reserve area when using the register bank. The general purpose register set is mapped to the area from (+08H) to (+1FH) with the offset of the start address of each register bank. Additionally, the area from (+02H) to (+07H) is used for switching register banks; therefore, this area cannot be used for general purposes.

(+02H) sets the value loaded to PC when the register bank is switched (that is, when the offset of the start address of the interrupt handling routine).

(+04H) is the area to which the contents of the PSW is stored when the register bank is switched.

(+06H) is the area to which the contents of the PC is stored when the register bank is switched (refer to 4.4.2). Upon reset, register bank 7 is automatically selected as the current register bank. Additionally, only the registers (refer to 3.1.4) of register bank 7 are initialized upon reset.

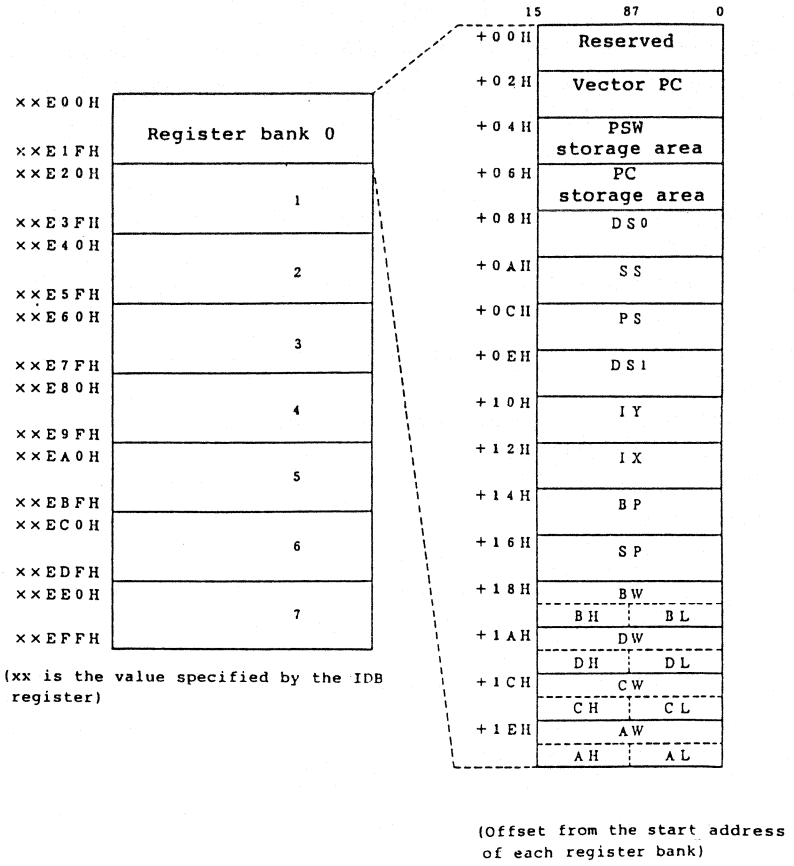


Fig. 3-1 Register Bank Organization

3.1.2 General purpose registers (AW, BW, CW, DW)

The uPD70322/70320 has four 16-bit general purpose registers. Each of these registers can be split into two 8-bit registers, that is, into upper and lower 8-bit registers (AH, AL, BH, BL, CH, CL, DH, DL).

These registers are used as 8-bit or 16-bit registers for various instructions such as transfer, arithmetic, and logical operation instructions.

The following registers are used as default registers for certain instructions.

- AW: Word multiplication/division, word input/output, data conversion
- AL: Byte multiplication/division, byte input/output, interpretation, BCD rotation, data conversion
- AH: Byte multiplication/division
- BW: Interpretation
- CW: Loop control branch, repeat prefix
- CL: Shift instruction, rotation instruction, BCD operation
- DW: Word multiplication/division, indirect addressing input/output

These registers are mapped to the built-in RAM. The address of each of these registers is given as the IDB register* value X 4096) + (0E00H) + (Register bank number X 32) plus the offset of each register.

Note: Refer to 3.4.2 for the IDB register.

Table 3-1 Register Bank Organization

Register	Offset value	Register	Offset value
AW	1EH	AL	1EH
		AH	1FH
BW	18H	BL	18H
		BH	19H
CW	1CH	CL	1CH
		CH	1DH
DW	1AH	DL	1AH
		DH	1BH

3.1.3 Pointers (SP, BP) and index registers (IX, IY)

These are used as the base pointers or index registers when accessing the memory for based addressing (BP), indexed addressing (IX, IY), or based-indexed addressing (BP, IX, IY). The SP register is also used for manipulating the stack. In the same way as the general purpose registers, these registers are used for transfer, arithmetic, and logical operation instructions; however, in this case, these registers cannot be used as 8-bit registers. Additionally, these registers are used as default registers for certain types of processing as follows:

SP: Stack manipulation

IX: Block transfer, source side in BCD string operation

IY: Block transfer, destination side in BCD string operation

These registers are mapped to the built-in RAM. The address of each of these registers is given as (the IDB register* value X 4096) + (0E00H) + (Register bank number x 32) plus the offset of each register. Table 3-2 shows the offset value for each register.

Note: Refer to 3.4.2 for the IDB register.

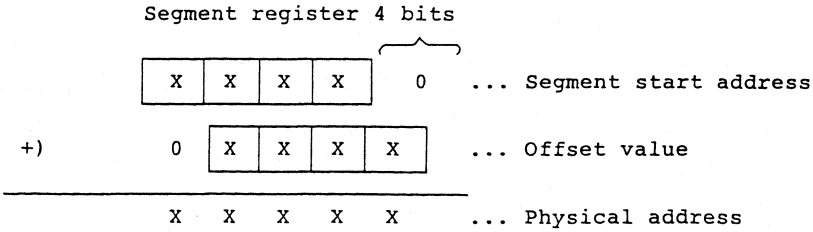
Table 3-2 Offset Value of the Pointers and Indexed Register

Register	Offset Value
SP	16H
BP	14H
IX	12H
IY	10H

3.1.4 Segment registers (PS, SS, DS0, DS1)

The CPU divides the memory space into 64K-byte logical segments. The start address of each segment is specified by the segment register. The offset from the start address is specified by other registers or by the effective address.

Therefore, the physical address is created as follows:



Four segment registers, PS (Program Segment), SS (Stack Segment), DS0 (Data Segment 0), and DS1 (Data Segment 1), are provided. Each of these segments is used as follows:

PS: Program fetch

SS: Stack manipulation instruction, addressed using BP as the base register.

DS0: Accessing a general variable, accessing the source block data for the block transfer instruction.

DS1: Accessing the destination block data for the block transfer instructions.

However, a different segment can be used instead of the DS0 register by using the segment override prefix. Also, when a segment is addressed using the BP register as the base register, a different segment can be used instead of the SS register.

When a reset is executed, the PS register of register bank 7 is initialized to FFFFH, and the SS, DS0, and DS1 registers are initialized to 0000H.

These registers are mapped into the built-in RAM. The address of each of these registers is (the IDB register* value X 4096) + (0E00H) + (Register bank number X 32) plus the offset value of each register. Table 3-3 shows the offset value of each register.

* Note: Refer to 3.4.2 for the IDB register.

Table 3-3 Offset Value of the Segment Registers

Register	Offset value
DS0	08H
DS1	0EH
SS	0AH
PS	0CH

3.1.5 Internal data area base register (IDB)

The IDB register is an 8-bit register which determines the address of the internal data area (refer to 3.4.1) in which the special function registers (refer to 3.4.3) are mapped. The special function registers which can also be used as controlling the built-in RAM and on-chip peripheral hardware. The IDB register can be referenced by two addresses, FFFFFH and the IDB (register value X 4096) + FFFH (refer to 3.4.2).

3.1.6 Special function registers

The uPD70322/70320 contains registers having special functions for setting the mode of and controlling the on-chip peripheral hardware. These registers are mapped in the special function register area in the internal data area, and can be read or written in the same way as conventionally used memory (refer to 3.4.3).

Additionally, the BTCLR instruction (refer to 15.1), which is an added function in the uPD70322/70320, is effective only for these registers.

3.2 Program Counter (PC)

The program counter (PC) is a 16-bit binary counter which retains the offset information of the program memory address whose contents is to be executed by the CPU.

The contents of the PC is incremented each time an instruction byte is fetched from the instruction queue. When executing branch, call, return, break instructions, etc., a new location is loaded to the PC.

When a reset is executed, 0000H is loaded to the PC. Since FFFFH is loaded to the PS when a reset is executed, the CPU starts program execution from address FFFF0H after the reset execution.

3.3 Program Status Word (PSW)

The program status word (PSW) consists of six different status flags, five different control flags, and two user flags.

Status flags

- V (Overflow)
- S (Sign)
- Z (Zero)
- AC (Auxiliary Carry)
- P (Parity)
- CY (Carry)

Control flags

- RB0 to RB2 (Register Bank 0 to 2)
- DIR (Direction)
- IE (interrupt Enable)
- BRK (Break)
- $\overline{\text{IBRK}}$ (I/O Break)

User flags

- F0 (user flag 0)
- F1 (user flag 1)

Status flags are set to 1 or reset to 0 depending on the results of execution (data value) of various instructions. The CY flag can be directly set to 1, reset to 0, or inverted by the instruction.

The control flags are set or reset by the instruction and control the operation of the CPU. The IE and BRK flags are always reset to 0 whenever interrupt processing is initiated.

The user flags can be set, reset, or tested by bit instructions. Users are free to set these flags in any manner desired.

When processing is performed per byte or word, the PSW is imaged as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	RB2	RB1	RB0	V	DIR	IE	BRK	S	Z	F1	AC	F0	P	$\overline{\text{IBRK}}$	CY

The contents of the lower 8 bits of the PSW can be stored in or restored from the AH register by using the MOV instruction.

The contents of each flag can be individually stored or restored from the stack by using the PUSH PSW or POP PSW instruction. However, bits 12 or 14 (RB0-2) of the PSW are not affectecd by the POP PSW instruction execution, and these bits are retained in the stack. In addtion, in the event an interrupt is generated, the contents of the control flags are automatically stored before the flags are changed. Bit 15 of PSW must not be set to 1, and must not be manipulated.

When a $\overline{\text{RESET}}$ is input, the contents of the PSW are initialized to F002H in word image. The $\overline{\text{IBRK}}$ and RB0 to RB2 are set to 1 while all other flags are reset to 0.

3.3.1 CY (Carry Flag)

(1) Binary addition/subtraction

For byte operation, this flag is set to 1 when a carry is made after the operation from bit 7 or a borrow is made to bit 7; otherwise, the flag is reset to 0.

For word operation, this flag is set to 1 when a carry is made after an operation from bit 15 or a borrow is made to bit 15; otherwise, it is reset to 0.

This flag is not affected by increments or decrements instruction execution.

(2) Logical operation

This flag is reset to 0 regardless of the operation result.

(3) Binary multiplication

After an unsigned byte operation, this flag is reset to 0 if the contents of the AH register is 0; otherwise, it is set to 1.

After an signed byte operation, this flag is reset to 0 if the contents of the AH register is a signed expansion of the contents of the AL register; otherwise, it is set to 1.

After an unsigned word operation, this flag is reset to 0 if the contents of the DW register is 0; otherwise, it is set to 1.

After an signed word operation, this flag is reset to 0 if the contents of the DW register is a signed expansion of the contents of the aw register; otherwise, it is set to 1.

For 8-bit immediate operation, this flag is reset to 0 if the product does not exceed 16 bits; otherwise, it is set to 1.

(4) Binary division

Undefined

(5) Shift/rotate

For the shift or rotate operation which involves the CY flag, the CY flag is set to 1. If the bit shifted to the CY flag is 1, and the CY flag is reset to 0 if 0 is shifted to the CY flag.

3.3.2 P (Parity Flag)

(1) Binary addition/subtraction, logical operation, shift of the lower 8 bits of the operation result bits, if the number of 1 bits is even, this flag is set to 1. If it is odd, this flag is reset to 0.

If all bits are 0, this flag is set to 1.

(2) Binary multiplication/division

Undefined

3.3.3 AC (Auxiliary Flag)

(1) Binary addition/subtraction

For byte operation, this flag is set to 1 when a carry is made after an operation from the lower 4 bits to the upper 4 bits or a borrow is made from the lower 4 bits to the upper 4 bits; otherwise, it is reset to 0.

For word operation, this flag is set to 1 when a carry is made after an operation from the lower byte to the upper byte or a borrow is made from the lower byte to the upper byte; otherwise, it is reset to 0.

(2) Logical operation, binary multiplication/division, shift/rotate

Undefined

3.3.4 Z (Zero Flag)

(1) Binary addition/subtraction, logical operation, shift/rotate

This flag is set to 1 when all 8 bits of the operation result are 0s for byte operation and when all 16 bits of the operation result are 0s for word operation; otherwise it is reset to 0.

(2) Binary multiplication/division

Undefined

3.3.5 S (Sign Flag)

(1) Binary addition/subtration, logical operation, shift/rotate

For byte operation, this flag is set to 1 when bit 7 of the result is 1, or reset to 0 when bit 7 is 0.

For word operation, this flag is set to 1 when bit 15 of the result is 1, or reset to 1 when bit 15 is 0.

(2) Binary multiplication/division

Undefined

3.3.6 V (Overflow Flag)

(1) Binary addition/subtraction

For byte operation, this flag is set to 1 when the carries from bits 7 and 6 are not the same, or reset to 0 if both carries are the same.

For word operation, this flag is set to 1 when the carries from bits 15 and 14 are not the same, or reset to 0 if both carries are the same.

(2) Binary multiplication

After an unsigned byte operation, this flag is reset to 0 if the contents of the AH register is 0; otherwise, it is set to 1.

After an signed byte operation, this flag is reset to 0 if the contents of the AH register is a signed expansion of the contents of the AL register; otherwise, it is set to 1.

After an unsigned word operation, this flag is reset to 0 if the contents of the DW register is 0; otherwise, it is set to 1.

After an signed word operation, this flag is reset to 0 if the contents of the DW register is a signed expansion of the contents of the AW register; otherwise, it is set to 1.

For an 8-bit immediate operation, this flag is reset to 0 if the product does not exceed 16 bits; otherwise, it is set to 1.

(3) Binary division

This flag is reset to 0.

(4) Logical operation

This flag is reset to 0.

(5) Shift/rotate

For shift/rotate 1-bit-left operation, this flag is reset to 0 when the contents of the CY flag is the most significant bit (MSB) or set to 1 when the contents of the CY flag is not the most significant bit (MSB).

For shift/rotate 1-bit-right operation, this flag is reset to 0 when the contents of the most significant bit (MSB) is equal to the second highest bit or set to 1 when the

contents of the most significant bit (MSB) is not equal to the second highest bit.

For multi-bit shift/rotate operation, the content of this flag is undefined.

3.3.7 $\overline{\text{IBRK}}$ (I/O) Break Flag)

This flag controls the generation of the software interrupt when an input/output instruction is executed.

If an input/output instruction execution is attempted when the contents of the $\overline{\text{IBRK}}$ flag is 0, a software interrupt (interrupt vector 19) is automatically generated so that the emulation of the input/output instruction can be performed.

When the contents of the $\overline{\text{IBRK}}$ flag is 1, an input/output instruction is normally executed, and no software interrupt is generated.

3.3.8 BRK (Break Flag)

This flag can be set to 1 only when it is stored in the stack as a portion of a PSW, and becomes effective when recovered from the PSW after it has been set.

When the BRK flag is set, a software interrupt (interrupt vector 1) is automatically generated each time an instruction is executed so that tracing can be performed for each instruction.

3.3.9 IE (Interrupt Enable Flag)

An interrupt can be enabled by setting this flag to 1 by using the EI instruction, or an interrupt can be disabled by resetting this flag to 0 by using the DI instruction.

3.3.10 DIR (Direction Flag)

This flag is set by the SET1 DIR instruction or reset by the CLR1 DIR instruction.

When this flag is set to 1, processing is performed from the upper address towards the lower address for the block transfer and input/output instructions. When this flag is reset to 0,

processing is performed from the lower address towards the higher address.

3.3.11 RB0-RB2 (Register Bank 0-2 Flag)

These flags specify the current register bank from the eight register banks located in the built-in RAM.

The contents of these flags is not affected by the POP PSW instruction execution.

Note: The value of RB0 to RB2 stored in the PSW store register must not be modified in the interrupt service routine.

3.3.12 F0, F1 (user Flag 0, Flag 1)

These flags can be used freely by the user.

These flags can be set or reset by an instruction for the PSW. In addition, these flags can also be set, reset, or tested by using the FLAG register which is a special function register.

When manipulated by using the FLAG register, user flags F0 and F1 are manipulated in the following manner.

Symbol	7	6	5	4	3	2	1	0	Address
FLAG	-	-	F1	-	F0	-	-	-	xxFEAH

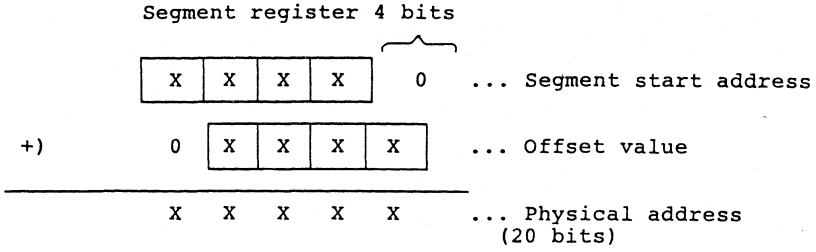
Fig. 3-2 User Flag Register (FLAG) Format

3.4 Memory Space

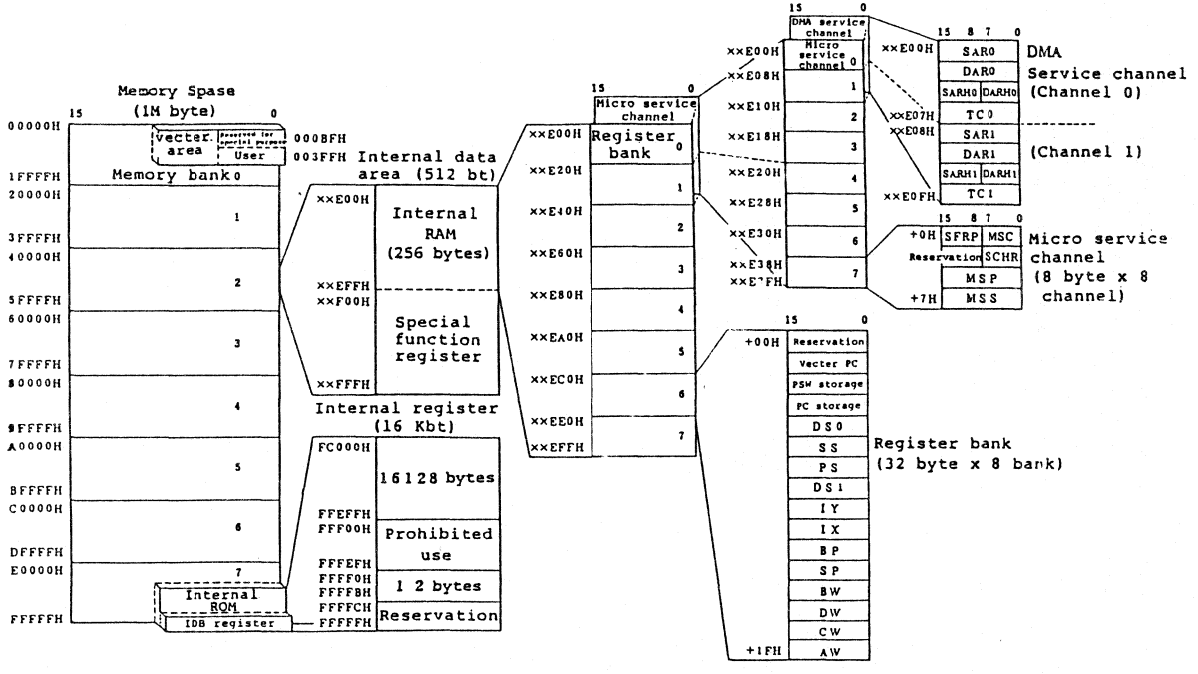
The uPD70322/70320 has 1M byte of memory. Fig. 3-3 shows the memory map of the uPD70322/70320. Addresses 00000 to 003FFH are used as a vector area. However, when this area is not used for vectors, it can be used for other purposes. Addresses xxE00 to xxFFFH (xx is the value specified by the IDB register) are used as an internal data area which can be moved in blocks of 4K bytes. The 4 bytes from FFFFCH to FFFFFH are reserved. The IDB register is allocated to address FFFFFH. Wait cycles can be programmbly inserted in the memory cycle every 128K bytes.

1M byte of physical address space is specified by the segment start address indicated by the segment register and the offset

value from the segment start address indicated by another register or the effective address.



Note: Word referencing at segment boundary
If address FFFFH is referenced in word units, when the offset address value of the segment is set to FFFFFH the reference address of the second byte will not be 0000H in the same segment but FFFFH + 1.
However, in the uPD70108/70116 (V20™/V30™), address 0000H will be referenced.



Note 1: xx is the value specified by the IDB register.

2: + □□H is the address offset value. the actual adress is this offset value plus the plus the starting address of the register bank or the macro aervice channel.

3: Built-in ROm is provided only in the μPD70322.

4: The macro service channels 0 to 3 and 4 to 7 are allocated to register banks 0 and 1, respectively, and the DMA service channels 0 and 1 are allocated to macro service channels 0 and 1, respectively.

Fig. 3-3 Memory Map

3.4.1 Internal data area

The internal data area is a 512-byte area consisting of the built-in RAM and special function register areas. This area can be relocated within the 1M byte memory space in 4K-byte block. The base address of the internal data area is specified by the IDB register (internal data area base register). Of the 20 bits in the internal data area base address, the upper 8 bits are specified by the IDB register, and the lower 12 bits are set to E00H.

The internal data area is manipulated by the memory manipulation instruction.

As shown in Fig. 3-4, the internal data area overlaps the external memory space or the built-in ROM area (uPD70322 only). For memory accessing other than program fetching, the internal data area is accessed. For program fetching, areas other than the internal data area are accessed. The internal data area is not accessed for program fetching.

The lower 256 bytes (xxE00 to xxEFFH, where xx is the value specified by the IDB register) of the internal data area are the built-in RAM area. The built-in RAM is used as a conventional RAM. In addition, register banks, macro service channels, and DMA service channels are functionally allocated to the built-in RAM area. Normal RAM accessing of the built-in RAM can be disabled by resetting (to 0) bit 6 (RAMEN) of the processor control register (PRC) which is a special function register.

The upper 256 bytes (xxF00 to xxFFFH, where xx is the value specified by the IDB register) of the internal data area are the special function register area. Registers having special functions for setting the mode and controlling the on-chip peripheral hardware are mapped onto this special function register area.

When $\overline{\text{RESET}}$ is input, the internal data area is located to FFE00 to FFFFFH. As a result, the IDB register is initialized to FFH upon reset.

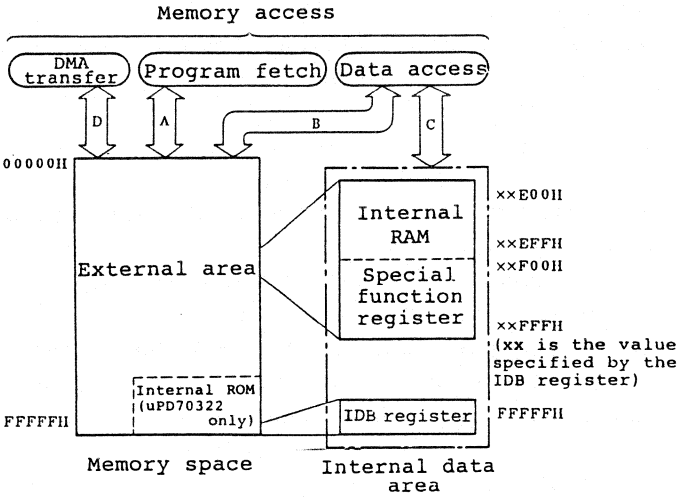


Fig. 3-4 Condition of Accessing Memory Space

A. Used for program fetching, when other than the internal data area is accessed.

B. The ability to access data from an address equivalent to the built-in RAM area when data is accessed from other than the internal data area or when accessing the built-in RAM is disabled.

C. If the conditions for data access do not fall into B above, the priority is placed on accessing the internal data area.

D. DMA transfer is used to access the external area (not for accessing the internal ROM area).

3.4.2 Internal data area base register (IDB)

This register determines the physical address of the internal data area (built-in RAM and special function register area)

which can be relocated in 4K-byte blocks. The IDB register specifies the upper 8 bits of the internal data area base address. The lower 12 bits of the internal data area base address is set to E00H.

Two addresses of the special function register area, xxFFFH (where xx is the value of the IDB register) and FFFFFH (fixed address), are assigned to the IDB register. The contents of the IDB register can be modified or referenced by accessing either of these addresses as memory. There is no difference in effectiveness between these two address modes.

When a reset is executed, FFH is set to the IDB register.

Therefore, the base address value of the internal data area becomes FFE00H when a reset is executed.

3.4.3 Special function register area

Registers which have special functions for setting the mode and controlling the on-chip peripheral hardware are mapped onto xxF00 to xxFFFH (where xx is the value specified by the IDB register). The IDB register is mapped onto two addresses, xxFFFH (xx is the value specified by the IDB register) and FFFFFH (fixed address). Program fetching cannot be performed from these areas.

The special function registers are manipulated by memory accessing. The BTCLR instruction which is added to the uPD70322/70320 instruction set (not provided in the uPD70108/70116) is used to specify the bits in this area. The BTCLR instruction is specifically provided for this area; therefore, wherever this area is relocated, the bits in this area are manipulated by the BTCLR instruction.

Table 3-4 lists the special function registers. The meaning of each item in the table is defined as follows:

- Symbol ..... Indicates built-in special register address. This can be written in the operand field of the instruction.

. R/W ..... Indicates whether the corresponding special function register can be read or written to.
R/W: Can be read or written to
R : Read only
W : Write only

. Manipulation ... Indicates that 16-bit, 8-bit, or 1-bit manipulation is possible.

. When RESET ..... Indicates the status upon $\overline{\text{RESET}}$ input.

xx of the upper 8 bits of the address is specified by the IDB register.

The un-indicated portion of address is reserved. In a read operation, the contents are undefined. An write operation, the operation has no meaning.

Table 3-4

SPECIAL FUNCTION REGISTER LIST

ADRESS	SPECIAL FUNCTION REGISTER	SYMBOL	R/W	MANIPULATION (bit)	RESET CONDITION
XXF00H	Port 0	P 0	R/W	8/1	undefined
XXF01H	Port 0 mode register	PM 0	W	8	FFH
XXF02H	Port 0 mode control register	PMC 0	W	8	00 H
XXF08H	Port 1	P 1	R/W	8/1	undefined
XXF09H	Port 1 mode register	PM 1	W	8	FFH
XXF0AH	Port 1 mode control register	PMC 1	W	8	00 H
XXF10H	Port 2	P 2	R/W	8/1	undefined
XXF11H	Port 2 mode register	PM 2	W	8	FFH
XXF12H	Port 2 mode control register	PMC 2	W	8	00 H
XXF38H	Port T	PT	R	8	undefined
XXF3BH	Port T mode register	PMT	R/W	8/1	00 H
XXF40H	External interrupt mode register	INTM	R/W	8	00 H
XXF44H	External interrupt macro service control register 0	EMS 0	R/W	8/1	undefined
XXF45H	External interrupt macro service control register 1	EMS 1	R/W	8/1	undefined
XXF46H	External interrupt macro service control register 2	EMS 2	R/W	8/1	undefined
XXF4CH	External interrupt request control register 0	EXIC0	R/W	8/1	47 H
XXF4DH	External interrupt request control register 1	EXIC 1	R/W	8/1	47 H
XXF4EH	External interrupt request control register 2	EXIC 2	R/W	8/1	47 H
XXF60H	Receive buffer register 0	RxB0	R	8	undefined
XXF62H	Transmit buffer register 0	TxB0	W	8	undefined
XXF65H	Serial receive macro service register 0	SRMS 0	R/W	8/1	undefined
XXF66H	Serial transmit macro service register 0	STMS 0	R/W	8/1	undefined

SPECIAL FUNCTION REGISTER LIST

ADRESS	SPECIAL FUNCTION REGISTER	SYMBOL	R/W	MANIPULATION (bit)	RESET CONDITION
XXF68H	Serial mode register 0	SCM 0	R/W	8/1	00 H
XXF69H	Serial control register	SCC 0	R/W	8/1	00 H
XXF6AH	Baud rate generator	BRG 0	R/W	8/1	00 H
XXF6BH	Serial error register 0	SCE 0	R	8	00 H
XXF6CH	Serial error interrupt request control register 0	SEIC 0	R/W	8/1	47 H
XXF6DH	Serial receive interrupt request control register 0	SRIC 0	R/W	8/1	47 H
XXF6EH	Serial transmit interrupt request control register 0	STIC 0	R/W	8/1	47 H
XXF70H	Receive buffer register 1	RxB1	R	8	undefined
XXF72H	Transmit buffer register 1	TxB1	W	8	undefined
XXF75H	Serial receive macro service control register 1	SRMS 1	R/W	8/1	undefined
XXF76H	Serial transmit macro service control register 1	STMS 1	R/W	8/1	undefined
XXF78H	Serial mode register 1	SCM 1	R/W	8/1	00 H
XXF79H	Serial control register 1	SCC 1	R/W	8/1	00 H
XXF7AH	Baud rate generator register 1	BRG 1	R/W	8/1	00 H
XXF7BH	Serial error register 1	SCE 1	R	8	00 H
XXF7CH	Serial error interrupt request control register 1	SEIC 1	R/W	8/1	47 H
XXF7DH	Serial receive interrupt request control register 1	SRIC 1	R/W	8/1	47 H
XXF7EH	Serial transmit interrupt request control register 1	STIC 1	R/W	8/1	47 H
XXF80H	Timer register 0	TM0	R/W	16	undefined
XXF82H	Modulo/Timer register 1	MD 0	R/W	16	undefined
XXF88H	Timer register 1	TM 1	R/W	16	undefined
XXF8AH	Modulo/Timer register 1	MD 1	R/W	16	undefined
XXF90H	Timer controler register 0	TMC 0	R/W	8/1	00 H
XXF91H	Timer control register 1	TMC 1	R/W	8/1	00 H
XXF94H	Timer unit macro service control register 0	TMMS 0	R/W	8/1	undefined
XXF95H	Timer unit macro service control register 1	TMMS 1	R/W	8/1	undefined
XXF96H	Timer unit macro service control register 2	TMMS 2	R/W	8/1	undefined

SPECIAL FUNCTION REGISTER LIST

ADRESS	SPECIAL FUNCTION REGISTER	SYMBOL	R/W	MANIPULATION (bit)	RESET CONDITION	
XXF9CH	Timer unit interrupt request control register 0	TMIC 0	R/W	8/1	47 H	
XXF9DH	Timer unit interrupt request control register 1	TMIC 1	R/W	8/1	47 H	
XXF9EH	Timer unit interrupt request control register 2	TMIC 2	R/W	8/1	47 H	
XXFA0H	DMA control register 0	DMAC 0	R/W	8/1	undefined	
XXFA1H	DMA mode register 0	DMAM 0	R/W	8/1	00 H	
XXFA2H	DMA control register 1	DMAC 1	R/W	8/1	undefined	
XXFA3H	DMA mode register 1	DMAM 1	R/W	8/1	00 H	
XXFACH	DMA interrupt request control register 0	DIC 0	R/W	8/1	47 H	
XXFADH	DMA interrupt request control register 1	DIC 1	R/W	8/1	47 H	
XXFEH	Standby control register	STBC	R/W	8/1	undefined	(Note 1)
XXFE1H	Refresh mode register	RFM	R/W	8/1	FC H	
XXFE8H	Wait control register	WTC	R/W	16/8	FFFFH	
XXFEAH	User flag register	FLAG	R/W	8/1	00 H	(Note 2)
XXFEBH	Processor control register	PRC	R/W	8/1	4EH	
XXFECH	Time base interrupt request control register	TBIC	R/W	8/1	47 H	
XXFFFH	Internal data area base register	IDB	R/W	8/1	FFH	

Note 1: Each bit of the standby control register can be set to 1 by an instruction; however, once set, bits can not be reset to 0 by an instruction (only 1 can be written to this register).
Upon power on reset: 00H
Other : No change

2: For the user flag register (FLAG), manipulating bits other than bits 3 and 5 is meaningless. The contents of user flags 0 and 1 (F0 and F1) of the FLAG register are affected by manipulating F0 and F1 of the PSW (refer to 3.3.12).

3.4.4 Built-in RAM area

256 bytes of RAM is mapped onto xxE00 to xxEFFH (xx is the value specified by the IDB register).

Eight register banks are allocated to the built-in RAM.

Additionally, the macro service channel and DMA service channel registers are also allocated to the built-in RAM.

Memory accessing of the built-in RAM can be disabled by resetting bit 6 (RAMEN) of the processor control register (PRC) to 0. Program fetching cannot be executed from the built-in RAM. When memory accessing of the built-in RAM is disabled, the built-in RAM can be accessed only as register.

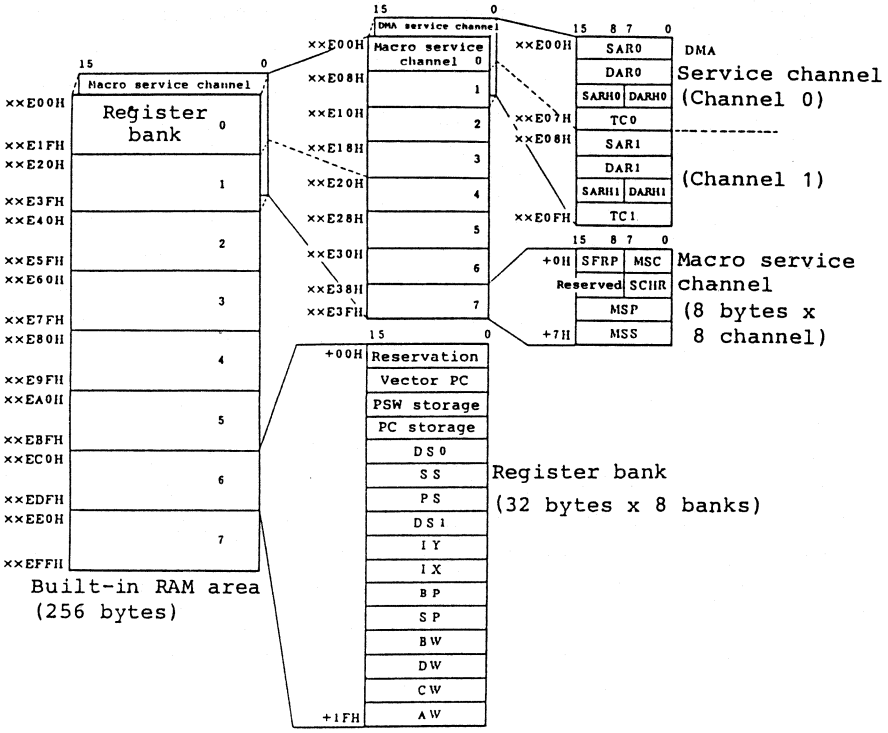


Fig. 3-5 Built-In RAM Area Mapping

3.4.5 Vector table area

A 1K-byte area from 00000 to 003FFH contains the interrupt routine start addresses for 256 vectors (4 bytes per vector). These interrupt routines are initialized by an interrupt request or break instruction.

```
Vector 0     (00000H) : Divide error
Vector 1     (00004H) : Single step
Vector 2     (00008H) : NMI input
Vector 3     (0000CH) : BRK 3 instruction
Vector 4     (00010H) : BRKV instruction
Vector 5     (00014H) : CHKIND instruction
Vector 6     (00018H) : Reserved
Vector 7     (0001CH) : FPO instruction
Vector 8     (00020H) : Reserved
   |            |           |
Vector 11    (0002CH) : Reserved
Vector 12    (00030H) : INTSER0
Vector 13    (00034H) : INTSR0
Vector 14    (00038H) : INTST0
Vector 15    (0003CH) : Reserved
Vector 16    (00040H) : INTSER1
Vector 17    (00044H) : INTSR1
Vector 18    (00048H) : INTST1
Vector 19    (0004CH) : Input/output instruction
Vector 20    (00050H) : INTD0
Vector 21    (00054H) : INTD1
Vector 22    (00058H) : Reserved
Vector 23    (0005CH) : Reserved
Vector 24    (00060H) : INTP0
Vector 25    (00064H) : INTP1
Vector 26    (00068H) : INTP2
Vector 27    (0006CH) : Reserved
Vector 28    (00070H) : INTTU0
Vector 29    (00074H) : INTTU1
Vector 30    (00078H) : INTTU2
Vector 31    (0007CH) : INTTB
Vector 32    (00080H) :\ User area
                       } . BRK imm8 instruction
Vector 255   (003FCH) :/  . INT input
```

For vectors 0 to 31, applications are specified (some are reserved). Therefore, vectors 0 to 31 cannot be used for general applications.

Vectors 32 to 255 can be used in two ways, by 2-byte break instructions and by the INT input. Unused areas can be used for other than vectoring.

Each vector address consists of 4 bytes. The upper 2 bytes of the vector address are loaded to the program segment (PS) and the lower 2 bytes to the program counter (PC).

Example: Vector 0

000H	001H
002H	003H

PC ← (001H, 000H)

PS ← (003H, 002H)

3.4.6 External memory area

In the uPD70322, the external memory (ROM, RAM, etc.) can be allocated to addresses 00000 to FBFFFH.

In the uPD70320, the external memory (ROM, RAM, etc.) can be allocated to addresses 00000 to FFFFEH; however, addresses FFF00 to FFFEFH and FFFFC to FFFFEH are reserved.

The external memory is accessed by using the address bus (A0 to A19), data bus (D0 to D7), $\overline{\text{MREQ}}$, $\overline{\text{MSTB}}$, and R/$\overline{\text{W}}$ signals.

The refresh pulse output pin ($\overline{\text{REFRQ}}$) is provided to refresh the pseudo-static memory so that the pseudo-static memory can be easily connected. A function to output the refresh address automatically is provided to refresh the dynamic memory so that the dynamic memory can also be easily connected (refer to 5.3). Additionally, a wait cycle can be inserted in the memory cycle every 128K bytes by means of a software interrupt (refer to 5.1).

3.4.7 Built-in ROM area

The uPD70322 has built-in mask ROM in the area starting from FC000 to FFFFFH. However, addresses FFF00 to FFFEFH are used for testing by NEC at the factory; therefore, use of this area is prohibited. Therefore, a total of 16,140 bytes are available to the user.

A dedicated bus is provided between the built-in ROM and the instruction queue so that an instruction can be prefetched at high speed independently of the external memory space available. This enables high-speed instruction execution (an instruction can be prefetched in one clock cycle. At least two clock cycles are required for external).

The internal ROM area is not accessed during DMA transfer operation. However, the external memory of the same address is accessed during DMA transfer operation.

3.5 I/O Space

In addition to 1M byte of memory, the uPD70322/70320 has 64K bytes of I/O space. Fig. 3-6 shows the I/O space map.

The I/O space is accessed by using the address bus (A0 to A15), data bus (D0 to D7), $\overline{IOSTB}$, R/$\overline{W}$, $\overline{DMAAK0}$, and $\overline{DMAAK1}$ signals. The output of the upper 4 bits (A16 to A19) of the address bus which are not used are 0s. A wait cycle can be inserted into the I/O cycle by software specification (refer to 5.1).

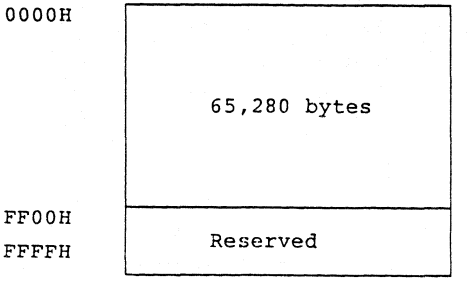


Fig. 3-6 I/O Map (64K bytes)

CHAPTER 4 INTERRUPT FUNCTION

4.1 Interrupt Controller

The uPD70322/70320 has a built-in high-performance interrupt controller which can control multiple processing of 17 interrupt sources. This interrupt controller manages and divides 17 interrupt sources, five external and 12 internal interrupt sources into groups, so that multiple interrupts can be programmably controlled per group. In addition, responses to an interrupt can be selected from three different methods: vector interrupt, register switching, and macro service, depending on the characteristic of the interrupt source.

The number of external interrupt sources can be easily expanded by connecting an interrupt controller such as the uPD71059.

The interrupt controller is controlled by the interrupt control register and the macro service control register provided for each interrupt source. Additionally, EI and DI instructions are provided to control the entire interrupt operation. The RETI and RETRBI instructions are used to return from interrupt processing, and the FINT instruction is used to inform the internal interrupt controller that interrupt processing has been terminated (refer to 15.1).

4.2 Interrupt Sources

The uPD70322/70320 has a total of 17 interrupt sources. Five of these are external and 12 are internal. These 17 interrupt sources are divided into eight groups and are controlled by the interrupt controller. The organization of these eight groups is determined by hardware. Of the eight groups (excluding NMI, INT, and INTTB), for five groups, the priority order can be arbitrarily set from 0 to 7 (0 is the highest) by means of software. Different functions are supported by the interrupt controller depending on the interrupt source. Table 4-1 lists the interrupt sources.

Table 4-1 Interrupt Sources

Interrupt source	External /internal	Vector	Macro service	Bank switching	Priority order			Multiple processing control
					Setting	Between groups	Within groups	
NMI (Non Maskable Interrupt)	External	2	Not Provided	Not provided	Im-possible	0	--	Not accepted
INT (INTerrupt)	External	External input	Not provided	Not provided	Im-possible	7	--	Not accepted
INTTU 0 (INTerrupt from Timer Unit 0)	Internal	28	Provided	Provided	Possible	1	1	Accepted
INTTU 1 (INTerrupt from Timer Unit 1)		29					2	
INTTU 2 (INTerrupt from Timer Unit 2)		30					3	
INTD 0 (INTerrupt from DMA channel 0)	Internal	20	Not provided	Provided	Possible	2	1	Accepted
INTD 1 (INTerrupt from DMA channel 1)		21					2	
INTP 0 (INTerrupt from Peripheral#0)	External	24	Provided	Provided	Possible	3	1	Accepted
INTP 1 (INTerrupt from Peripheral#1)		25					2	
INTTU 2 (INTerrupt from Timer Unit#2)		26					3	

Interrupt source	External /internal	Vector	Macro service	Bank switching	Priority order			Multiple processing control
					Setting	Between groups	Within groups	
INTSER 0 (INTerrupt from Serial ERror or channel 0)	Internal	12	Not Provided	Provided	Possible	4	1	Accepted
INTSR 0 (INTerrupt from Serial Receiver of channel 0)		13	Provided				2	
INTST 0 (INTerrupt from Serial Transmitter of channel 0)		14	Provided				3	
INTSER 1 (INTerrupt from Serial ERror or channel 1)	Internal	16	Not Provided	Provided	Possible	5	1	Accepted
INTSR 1 (INTerrupt from Serial Receiver of channel 1)		17	Provided				2	
INTST 1 (INTerrupt from Serial Transmitter of channel 1)		18	Provided				3	
INTTB (INTerrupt from Time Base counter)	Internal	31	Not provided	Not provided	Impossible (Preset 7)	6	--	Accepted

4.3 Interrupt Controller Functions

The interrupt controller controls the priority for executing multiple interrupts simultaneously generated interrupts.

4.3.1 Priority control for multiple interrupts

Priority in multiple interrupt processing (except for NMI and INT) is controlled in group units.

Multiple interrupt processing is controlled when an interrupt is enabled (EI condition). Therefore, when executing multiple processing, an interrupt must be enabled in the interrupt processing routine. However, macro service interrupt are accepted even if the DI condition is in effect. In multiple processing control, if an interrupt is generated which has a higher priority than the one currently being processed, the interrupt having the higher priority is accepted, and the processing of the interrupt currently being processed is interrupted so that the interrupt having the higher priority can be processed. If an interrupt is generated which has a lower priority than the one currently being processed, the interrupt having the lower priority is retained. If the interrupt mask bit of the interrupt control register (provided for each interrupt source) is not set in the interrupt processing routine currently being executed, and the interrupt request flag is not reset, the retained interrupt is accepted when the currently executed interrupt is terminated.

Except for the NMI, INT and Software interrupts, the FINT instruction must be executed at the end of the interrupt controller in order to inform the interrupt controller that the interrupt processing routine has been terminated. An interrupt must have the higher priority than the interrupt just processed; otherwise, it will not be accepted. The NMI and INT interrupt requests are not handled under multiple processing control. Therefore, these interrupts are accepted whenever interrupt operation is enabled (NMI is always accepted).

Eight priority levels, 0 to 7 (0 is the highest), can be arbitrarily assigned to each interrupt group. When using the register bank selection function (described later), this priority order also indicates the register bank number to be selected. The execution priority is set using the PR0 to PR2 bits of the interrupt control register for each interrupt source. However, this setting is effective only in the interrupt control register of the interrupt source having the highest priority among the interrupt groups. Settings in other interrupt control registers are ignored, and the value is fixed to 7 (the lowest priority) when read out.

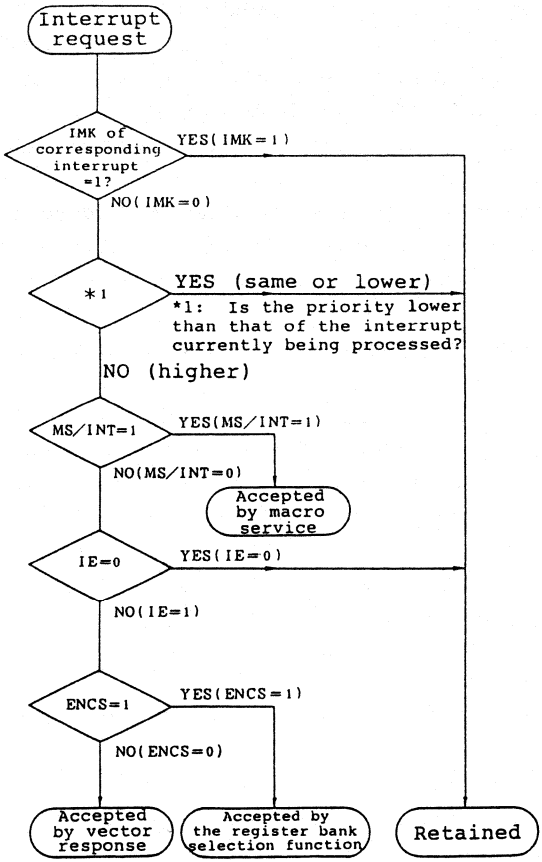


Fig. 4-1 Interrupt In Multiple Processing Control

4.3.2 Priority control when interrupts are generated simultaneously

When interrupts are generated simultaneously, the NMI has the highest priority and the INT has the lowest priority. The priorities of other interrupts are the same as those of the priority of multiple interrupts. If two or more groups are set to the same priority, the priority is controlled according to the priority order set by the hardware. Within the group, the priority is controlled according to the priority order set within the group.

The following shows some examples on how the priority is controlled.

Example 1: When INTSR0 whose priority level is set to 3 and INTTU2 whose priority level is set to 6 are simultaneously generated, INTSER0 is accepted first.

2: When INTP0 and INTP1 are generated simultaneously, INTP0 is accepted first.

3: When NMI and INTD1 are generated simultaneously, NMI is accepted first.

4: When INTTB and INT are generated simultaneously, INTTB is accepted first.

5: When INTTU1, whose priority level is set to 4, and INTSER1 are generated simultaneously, INTTU1 is accepted first.

4.4 Interrupt Response Method

The uPD70322/70320 has three different ways of responding to an interrupt: These include a vector interrupt function, a register bank switching function, and a macro service function. These functions can be selected depending on the purpose of each interrupt. The interrupt controller responds in the method specified by the interrupt control register associated with each interrupt source.

When an interrupt is accepted using the vector interrupt function or register bank switching function, the contents of the PC, PS, and PSW are stored using a method suitable for the selected function. The IE and BRK flags are reset to 0 after the contents of the PSW are stored. For interrupts except the NMI, interrupts in response by macro service function, and single-step interrupts are disabled (all software interrupts except single-step interrupts are generated) (refer to 4.9).

4.4.1 Vector interrupt

In the case of vector interrupts, when an interrupt is generated, the current contents of the PSW, PC, and PS are stored to the stack, and a vector is selected from the vector table. The program is then executed from the address indicated by the vector. All vectors except for the INT are fixed. For the INT interrupt, an interrupt acknowledge cycle is generated, and an interrupt vector is clocked in from the data bus (refer to 4.6 INT). Interrupt vectors for other than the INT interrupt are as shown in Table 4-1.

Recovery from interrupt processing is performed by using the RETI instruction. However, for recovery from interrupt except for the NMI and INT, the FINT instruction must be executed. When recovering from interrupt, the contents of the PC, PS, and PSW are recovered from the stack.

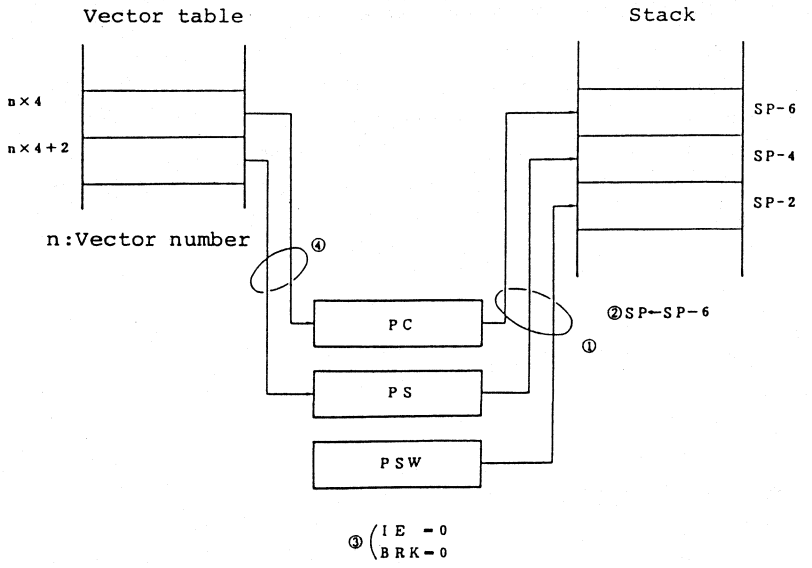


Fig. 4-2 Operation to Accept Interrupt
(performed in order from 1 to 4)

4.4.2 Register bank switching function

For the uPD70322/70320, the general purpose register set is mapped onto the built-in RAM. Therefore, the uPD70322/70320 can have up to eight register banks. By automatically switching these register banks when responding to an interrupt, the need for a software command to store the contents of the registers to the stack is eliminated, so that interrupts can be responded to in a shorter time.

This register bank switching function can be enabled by setting the ENCS bit of the interrupt control register of the corresponding interrupt source to 1. A register bank can be

specified for each interrupt group, and each register bank number coincides with the priority order of the multiple interrupt, and is specified by the PR0 to PR2 bits of the interrupt control register.

The following is the register bank switching sequence (Fig. 4-3):

① The contents of the PSW are stored to the temporary register.

② The register bank is switched.

③ The IE and BRK are both set to 0.

④ The contents of the PC and PSW stored to the temporary register are stored to the respective storage areas in the register bank.

⑤ The offset value of the start address of the interrupt processing routine is loaded from the vector PC area in the register bank to the PC.

The register bank is switched in this sequence and the interrupt processing routine is executed.

Recovery from the register bank switching interrupt can be made by executing the RETRBI instruction after executing the FINT instruction. Interrupts which can use the register bank switching function are limited to those which receive multiple interrupt control. When the RETRBI instruction is executed, the contents of the PC and PSW are recovered from the storage area in the register bank as shown in Fig. 4-4 (since the RETI instruction does not restore the register bank, normal return to the main routine cannot be made by the RETI instruction).

Before using the register bank switching function, a minimum of the PS, vector PC, SS, and SP of the register bank to be selected must be initialized. Other registers can be initialized as necessary. However, the contents of PS must not be modified during the interrupt processing routine.

The register bank switching function can be used only for one interrupt within the interrupt group set to the same priority (refer to 4.7.1).

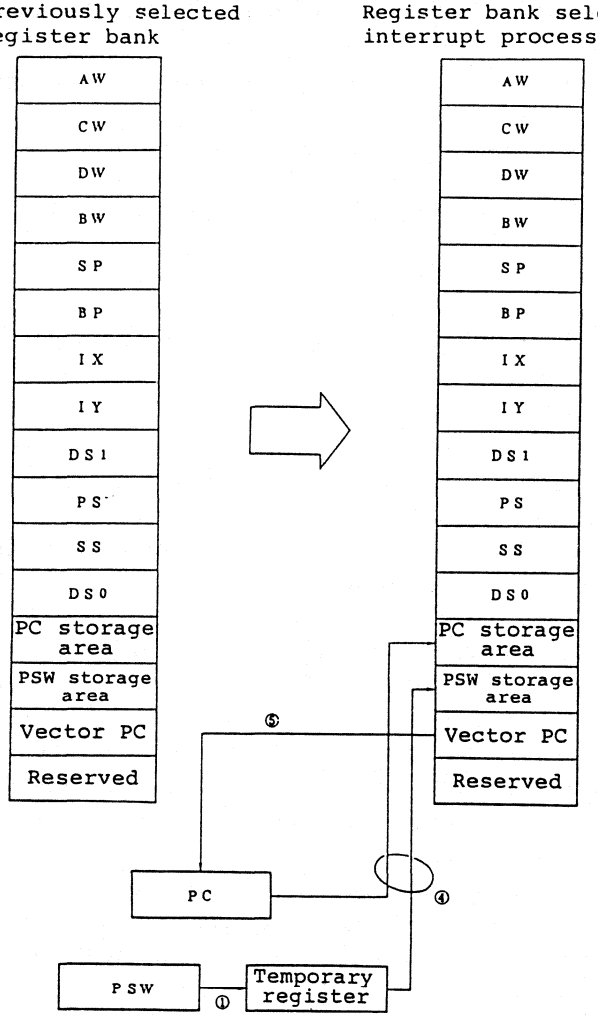


Fig. 4-3 Register Bank Switching Sequence

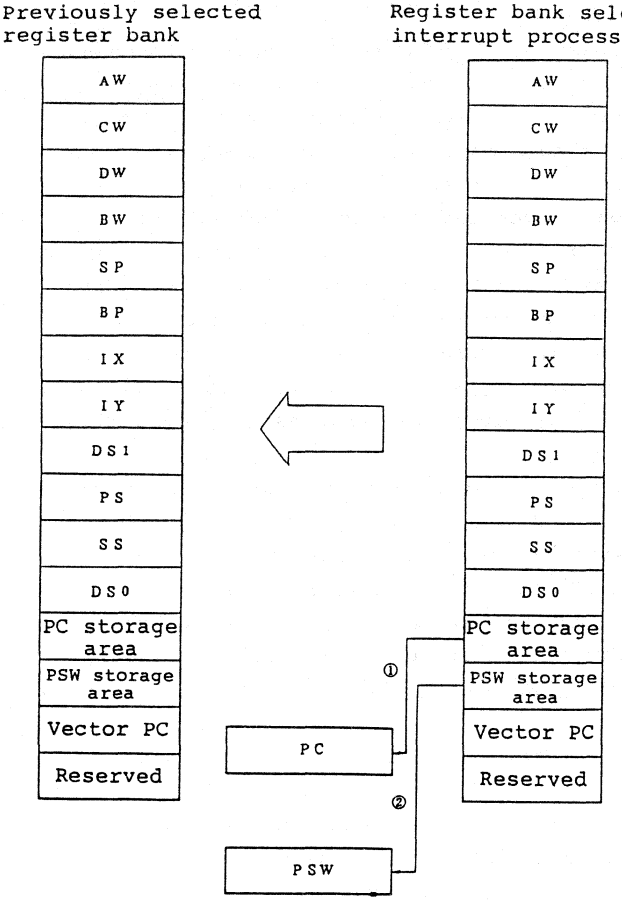


Fig. 4-4 Register Bank Restoration Sequence

4.4.3 Macro service function

The macro service function is used to transfer data between the special function register area and the memory space in response to an interrupt request. Using this function, unsophisticated processing such as simple data transfers can be performed without having to use software interrupt processing. Therefore, the interrupt processing overhead (operations such as storing the contents of the register, initialization, and restoration of the register) can be eliminated. The processing functions performed by the macro service function need not be regarded as software. Therefore, data which has been conventionally processed per byte by software can now be processed in a block, resulting in increased programming effeciency.

The macro service function is different from other interrupt response methods. If the IMK bit (interrupt mask bit) of the interrupt control register provided for each interrupt source is reset to 0 and the MS/$\overline{\text{INT}}$ bit of the same register is set to 1, the macro service function will operate regardless of whether or not the interrupt is enabled (regardless of EI or DI status) (refer to 4.7). However, the priority order is controlled.

The macro service function features the following two modes.

(1) Normal mode

A specified number of data transfers are performed per byte or word each time an interrupt is generated.

Fig. 4-5 shows the operation in the normal mode.

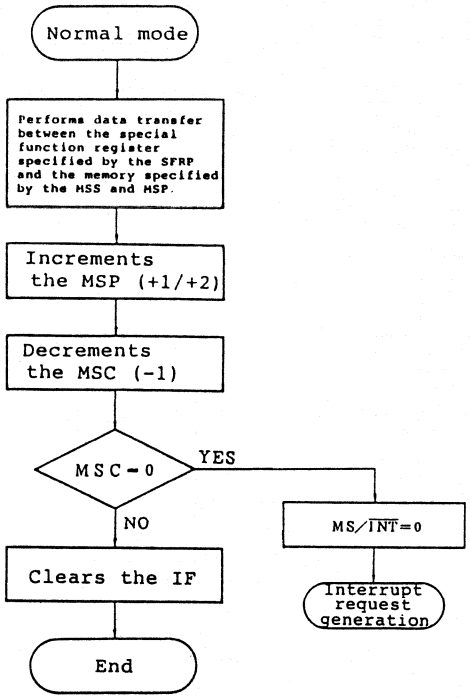


Fig. 4-5 Normal Mode Operation Flow

Note: Refer to 4.4.4 for the SFRP, MSS, MSP, and MSC.
Refer to 4.7.1 for the MS/$\overline{\text{INT}}$.

(2) Character search mode

1-byte data transfer is repeated until the specified number of bytes are transferred or until the transferred data coincides with the specified 8-bit data.

Fig. 4-6 shows the operation flow in the character search mode.

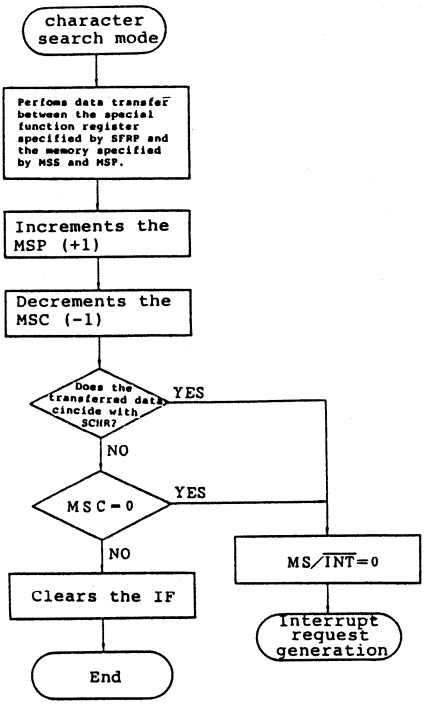


Fig. 4-6 Character Search Mode Operation Flow

Note: Refer to 4.4.4 for SFRP, MSS, MSP, and MSC.
Refer to 4.7.1 for MS/$\overline{\text{INT}}$.

The macro service function is controlled by the macro service control register, which is provided for each interrupt source to which macro service is possible, and the macro channel specified by the macro service control register.

4.4.4 Macro service control register

This is an 8-bit register which controls the macro service function. The macro service control register is organized as shown in the figure below. The following describes the function of each bit in the register.

7	6	5	4	3	2	1	0
MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0

CH0 to CH2 These bits specify the macro service channel. Channels 0 to 7 can be specified.

DIR This bit specifies the data transfer direction. When this bit is 0, data is transferred from the memory to the special function register. When this bit is 1, data is transferred from the special function register to the memory.

MSM0 to MSM2 These bits specify the macro service mode. A combination of these bits can be used to specify either normal or character search mode, and the number of transfer bits (8-bit/16-bit transfer) for the normal mode.

MSM2	MSM1	MSM0	Operation mode
0	0	0	Normal mode (8-bit transfer)
0	0	1	Normal mode (16-bit transfer)
1	0	0	Character search mode (8-bit transfer)
Other combinations			Not allowed

Macro service control registers are allocated in the special function register area and can be read/written to in blocks of 8-bits or 1 bit by accessing the memory.

A macro service control register is provided for each interrupt source to which macro service can be performed. Interrupt sources to which macro service can be performed are timer interrupts (INTTU0-2), external interrupts (INTP0-2), and serial receive/transmit interrupts (INTSR0, 1/INTST0, 1). Refer to the corresponding item for the location of the macro service control register for each interrupt source.

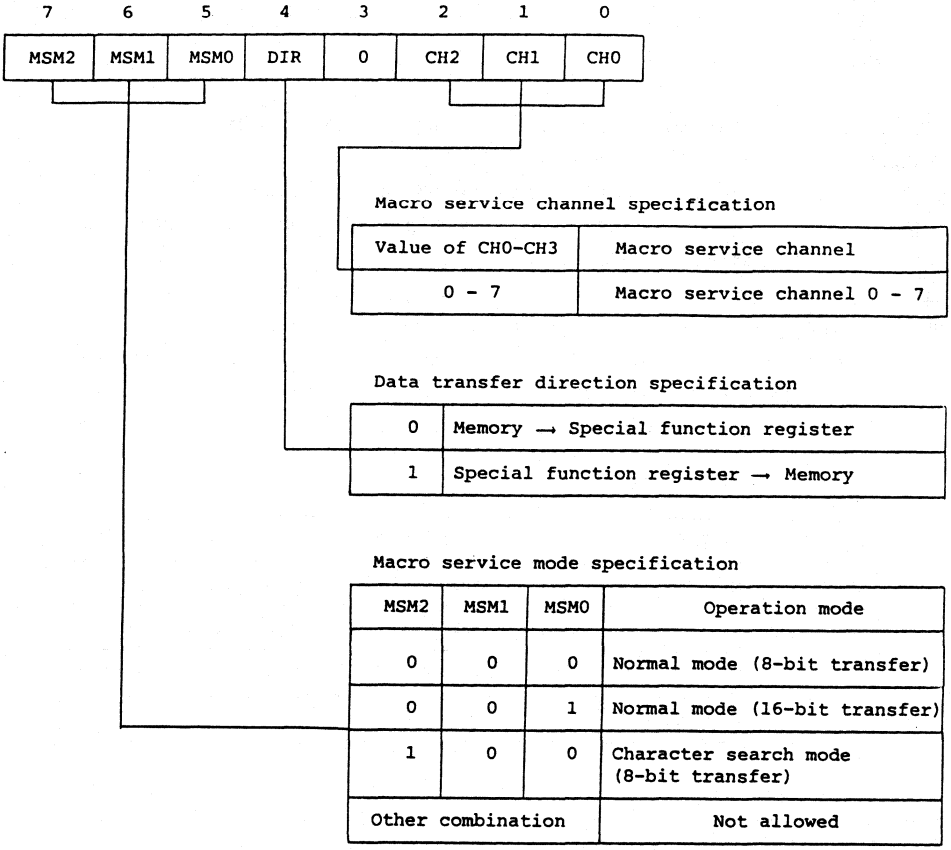


Fig. 4-7 Macro Service Control Register Format

The macro service channels are allocated from xxE00 to xxE3FH (xx is the value specified by the IDB register). The data transfer source, destination, number of transfers, and check character for the macro service can be specified by the macro service channels. Up to eight channels can be used.

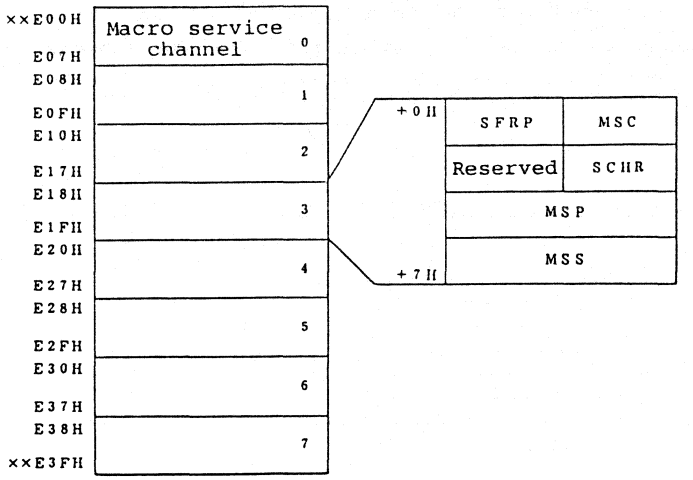


Fig. 4-8 Macro Service Channel Organization

MSC (+0H) : Number of transfers performed in the macro service.

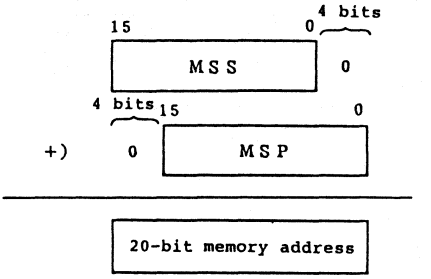
SFRP (+1H) : The offset value of the special function register address. xxF00H + SFRP (xx is the value specified by the IDB register) is the special function register address.

SCHR (+2H) : 8-bit data compared in the character search mode.

MSP (+4H) : The offset value of the memory address used for the data transfer in the macro service.

MSS (+6H) : The segment value of the memory address used for the data transfer in the macro service. The memory address for the data transfer will be MSS x 16 + MSP.

The memory address used for the data transfer in the macro service is indicated by the segment value given in the MSS and the offset value from the segment given in the MSP.



For each data transfer (8-bit/16-bit), the MSC of the macro service channel is decremented (-1) and the MSP is incremented (+1/+2). Afterwards, the inerrupt request flag is cleared. However, when the MSC becomes 0 (if 0 is written to MSC, transfer operation is performed 256 times.) or when the transferred data coincides with the check data (character search mode only), the interrupt request flag is not cleared, and an interrupt request is generated.

4.5 NMI (Non-Maskable Interrupt)

The NMI is an interrupt that has the highest priority and cannot be disabled. This interrupt is detected by the edge specified by bit 0 (ESNMI bit) of the INTM register (special function register). When the ESNMI bit is reset to 0, this interrupt is generated at the falling edge. When the ESNMI bit is set to 1, the interrupt is generated at the rising edge. This interrupt can be responded to only by the vector method, and the vector type is preset to 2. This NMI input shares P10 pin so that the level can be checked by reading P10 pin. When the NMI is accepted, the interrupt disabled status (IE=0) is initiated,

and other interrupts are disabled (however, macro service interrupt requests can still be accepted).

4.6 INT (Interrupt)

The INT is a maskable interrupt. This interrupt is detected by its level (active high). The INT does not receive multiple processing control from the interrupt controller so that this interrupt is accepted whenever the interrupt is enabled (IE=1). However, when two or more interrupts are generated simultaneously, this interrupt assumes the lowest priority. This interrupt can be responded to only by the vector method. The vector type is clocked in from the data bus in the interrupt acknowledge cycle. The interrupt acknowledge cycle can be confirmed by the $\overline{\text{INTAK}}$ output. The INT pin shares the P14/$\overline{\text{POLL}}$ pin and is selected by bit 4 of the port 1 mode control register (PMC1) (special function register). Therefore, when the INT function is not selected, this interrupt is not generated even if the interrupt is enabled (IE=1). $\overline{\text{INTAK}}$ shares the P13/$\overline{\text{INTP2}}$ pin and is selected by bit 3 of the PMC1 (If the $\overline{\text{INTAK}}$ function is not selected, the interrupt acknowledge cycle cannot be generated in the external system).

When an interrupt controller such as the uPD71059 is connected, the number of external interrupts can be expanded up to 64.

When this interrupt is accepted, the interrupt is disabled (IE=0).

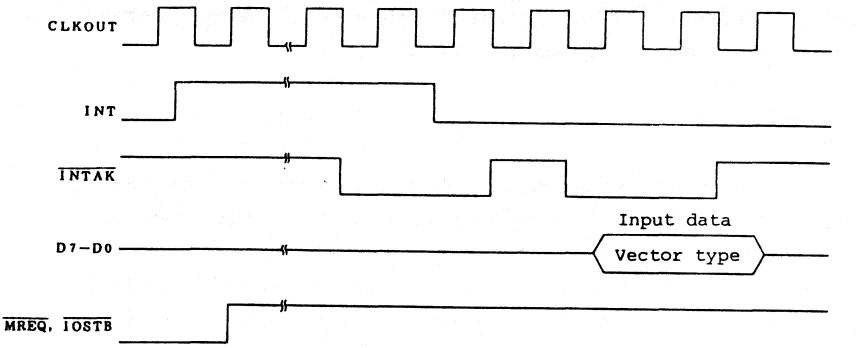


Fig. 4-9 Interrupt Acknowledge Timing

4.7 Interrupts Other Than NMI, INT

All interrupts except the NMI and INT receive multiple interrupt control from the interrupt controller. When an interrupt is accepted, the interrupt function is automatically disabled (IE=0). However, by enabling an interrupt during the interrupt processing routine, an interrupt having higher priority than the interrupt currently being processed can be accepted if generated. If an interrupt having a lower priority is generated, the current interrupt is retained.

For priority order, 15 interrupt sources are divided into six groups to which a priority order from 0 to 7 (8 level; 0 is the highest) can be assigned. However, the priority order of the INTTB (Time Base Counter) is fixed to 7 by means of hardware. When bank switching functions, these priority order levels also indicate the bank number to be selected. When a reset is executed, priority levels are all initialized to 7.

When two or more interrupts are simultaneously generated, the interrupt of the group having the highest priority is accepted. If two or more interrupts having the same priority are simultaneously generated, the one to which the higher priority is set by means of hardware is accepted. In the same way, within the same group, the one to which the higher priority is set by means of hardware is accepted.

4.7.1 Interrupt request control register

This is an 8-bit register which controls interrupts other than the INT and NMI.

The interrupt request control register is organized as shown in the figure below. The following describes the function of each bit of this register.

7	6	5	4	3	2	1	0
IF	IMK	MS/$\overline{\text{INT}}$	ENCS	0	PR2	PR1	PR0

PR0 to PR2 These bits specify the priority order of the interrupt group.

Priority levels 0 to 7 can be specified. This specification is effective only for the interrupt control register having the highest priority in the group, and is not valid for other interrupt request control registers (for read operation, the priority order level is fixed to 7). The priority order of other interrupt request control registers is determined according to the interrupt request control register having the highest priority in the group.

This specification also specifies the register bank to be selected by the register bank switching function.

ENCS This bit specifies whether or not to use the register switching function.

When this bit is set to 1, the register bank switching function is used. When reset to 0, the vector interrupt is used.

MSM/INT This bit selects the interrupt response method.

When this bit is set to 1, the macro service function is used. When this bit is set to 0, the vector interrupt or register bank switching function is used.

IMK This bit specifies the masking of the interrupt.

Setting this bit to 1 masks the interrupt, and 0 releases the masking.

IF This bit indicates whether or not an interrupt is generated.

1 indicates that an interrupt is generated, and 0 indicates that no interrupt is generated. This bit is set to 1 when an interrupt request is generated and reset to 0 when the interrupt request is accepted or the BTCLR instruction (an instruction added in the uPD70322/70320) is executed.

The interrupt control registers are located in the special function register area and can be read or written to in blocks of 8 bits or 1 bit by accessing the memory.

An interrupt request control register is provided for each interrupt source (except for INT and NMI).

Refer to the corresponding item for the location of the interrupt request control register of each interrupt source.

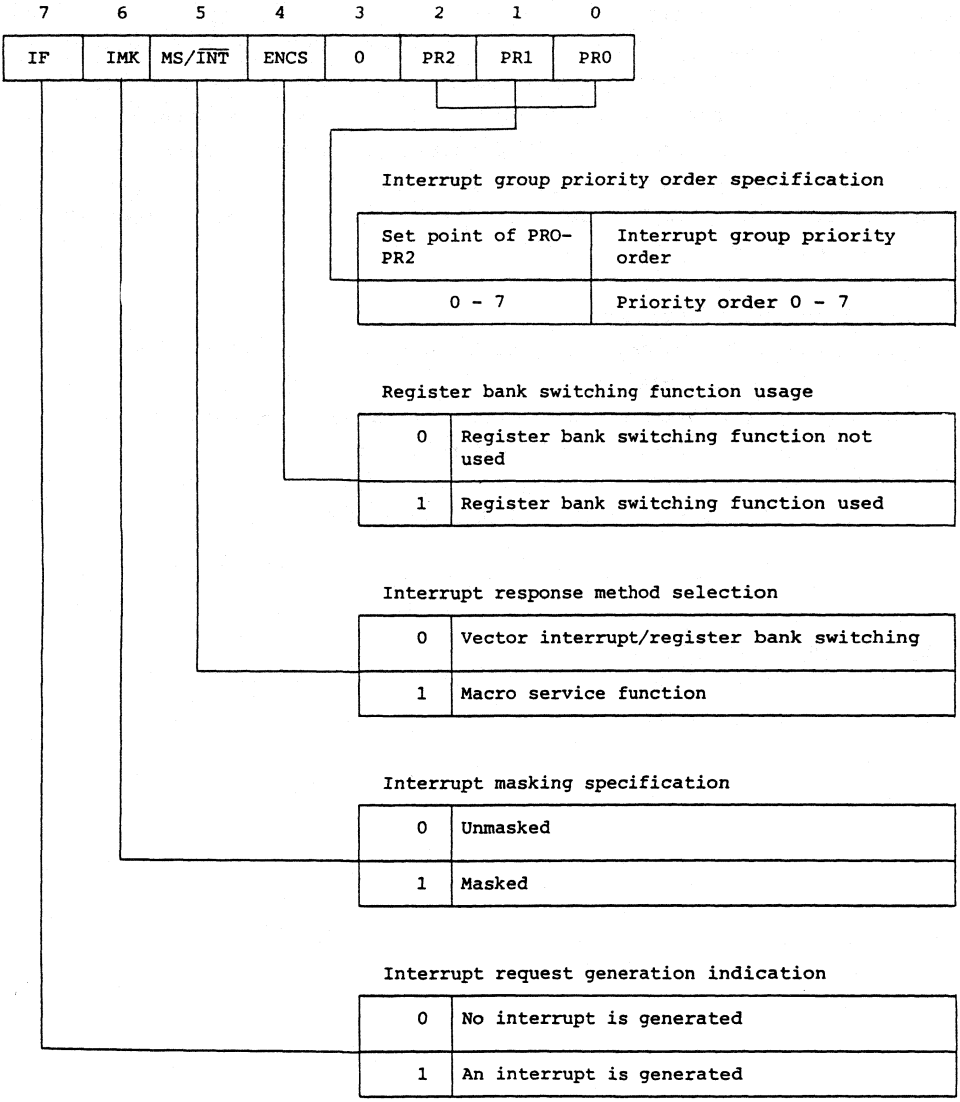


Fig. 4-10 Interrupt Request Control Register Format

4.8 External Interrupt

The uPD70322/70320 has five external interrupt request sources. Of these five, the INT is detected by its level. The others are detected by their edge. For the interrupt requests detected by their edge, the effective edge of each interrupt is specified by the external interrupt mode register (INTM) which is a special function register.

4.8.1 External interrupt mode register (INTM)

This is an 8-bit register which specifies the effective edge of the external interrupt. The NMI and INTP0 to INTP2 are detected by their edge. The effective edge of these interrupt requests is specified by the INTM register.

The INTM register is organized as shown in Fig. 4-9. Fig. 4-11 also describes each bit of the INTM register.

The INTM register is located in the special function register area and can be read or written to in blocks of 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, the INTM register is initialized to 00H.

4.8.2 External Interrupt Request Control Register (EXIC0,EXIC1,EXIC2)

EXICn (n=0-2) register is a 8 bit register which controls interrupt requests generated from the 3 external interrupt request pins (INTP0 - INTP02).

These three interrupt requests consisting as one group for setting priorities. In the group the priority is fixed as below:

EXF0 > EXF1 > EXF2

	7	6	5	4	3	2	1	0
EXIC0	EXF0	EXMK0	MS.INT	ENCS	0	PR2	PR1	PR0
EXIC1	EXF1	EXMK1	MS.INT	ENCS	0	1	1	1
EXIC2	EXF2	EXMK2	MS.INT	ENCS	0	1	1	1

Bit 2 - 0 of EXIC1 and EXIC2 are fixed to '1'.
Priority of interrupt request for EXIC1 and EXIC2 registers are following the setting of PR2 - PR0 in EXIC0 register.

As for each bit of EXICn registor, please refer to chapter 4.7.
EXICn registers can be read/written with 8 bit or 1 bit memory operation command. In this case one wait cycle is automatically inserted.
With RESET input, contents turn to 47H.

4.8.3 External Interrupt Macro Service Control Register (EMS0,EMS1,EMS2)

EMSn (n=0-2) are 8 bit registers, which control macro service initiated by the interrupt requests generated form the external interrupt request pins (INTP0 - INTP2).

EMS0 register controls macro service initiated by the EXF0 flag.

EMS1 register controls macro service initiated by the EXF1 flag.

EMS2 register controls macro service initiated by the EXF2 flag.

EMSn registors can be read/written with 8 bit or 1 bit memory operation command. In this case 1 wait will be inserted.

As for each bit of EMSn register, please refer to chapter 4.4.4.

EMS0, EMS1, EMS2

7	6	5	4	3	2	1	0
MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0

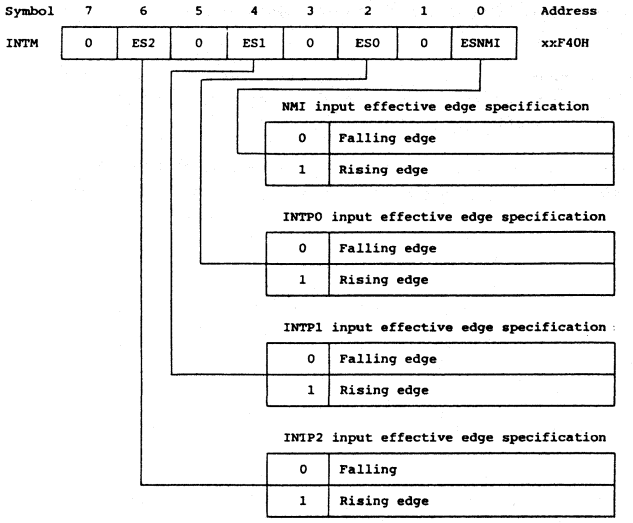


Fig. 4-11 External Interrupt Mode Register (INTM) Format

4.9 Software Interrupt

The uPD70322/70320 has eight different software interrupts (refer to Table 4-2). Of these eight, six are compatible with the software interrupts of the uPD70108/70116 (however, there is no interrupt for the emulation mode), and two are functions specifically provided in the uPD70322/70320.

The vectors for these interrupts are predetermined.

Interrupts other than the BRK interrupt (single-step interrupt) are always accepted whenever the condition in which the interrupt can be generated is satisfied (A priority higher than that for a hardware interrupt will be given).

The BRK flag interrupt is generated when the BRK flag is set to 1 (regardless of hardware or software). However, the BRK flag is automatically reset when the interrupt is accepted. Therefore, the priority of the BRK interrupt becomes lower than other interrupts (regardless of hardware or software) so that the BRK flag interrupt is not accepted when an other interrupt is being processed.

Table 4-2 Software Interrupts

Interrupt source	Vector	Priority order
DIVU divide error	0	1
DIV divide error		
CHKIND boundary over	5	
BRKV	4	
BRK3	3	
BRK imm8	32-255	
BRK flag (single-step)	1	2
Input/output instruction ($\overline{\text{IBRK}}$ flag)	19	1
FPO instruction	7	

4.9.1 General software interrupts

All software interrupts except the interrupts generated by the input/output instruction and the FPO instruction are accepted in the same acceptance sequence as vector inerrupts. That is, the address data for the next instruction and the contents of the PSW are stored to the stack and the IE and BRK flags are set 0, and the contents of the vector are then loaded to the PS and PC.

The following describes each software interrupt.

(1) DIVU divide error, DIV divide error

These interrupts are generated when the quotient overflows as a result of executing a division instruction.

(2) CHKIND boundary over

When the instruction (CHKIND) which checks whether or not the index value has exceeded the boundary of the defined array is executed, this interrupt is generated if the index value has exceeded the boundary.

(3) BRKV

This interrupt is generated if the V flag (Overflow flag) has been set when the BRKV instruction is executed.

(4) BRK 3

This interrupt is generated by executing the BRK 3 instruction.

(5) BRK imm8

This interrupt is generated by executing the BRK imm8 instruction.

(6) BRK flag (single-step)

When the BRK flag is set to 1, this interrupt is generated each time an instruction is executed.

4.9.2 Input/output instruction interrupt

When the $\overline{\text{IBRK}}$ flag is set to 0, an interrupt is generated if an input/output instruction execution is attempted. The address data stored to the stack when this interrupt is accepted is different from general software interrupts (refer to 4.9.1). The address at which the input/output instruction is placed will be the address of this address data. If a prefix is attached to this input/output instruction, the address of the address data will be the address at which the prefix is placed. Other operations of this interrupt are the same as those of general software interrupts. Additionally, when recovering from the input/output instruction interrupt, the value on the stack must be adjusted in order to make a normal recovery.

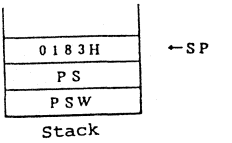
Since the start address of the instruction is stored to the stack as the address data, the instruction execution which actually caused the interrupt generation can be determined in the software. With this function, a program used for the uPD70108/70116 can be easily transplanted to the uPD70322/70320.

When the input/output instruction interrupt occurs, the contents of the PSW are immediately stored to the stack,

afterwards, the BRK and IBRK flags are set to 0 and 1, respectively. By this setting of the $\overline{\text{IBRK}}$ flag to 1, the input/output instruction being handled in the interrupt processing is executed as an input/output instruction, and the $\overline{\text{IBRK}}$ flag returns to the original state ($\overline{\text{IBRK}}$=0) upon recovering from the interrupt.

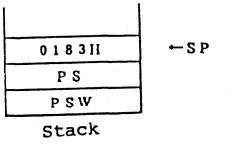
Example 1) Without prefix

Address	Instruction	
	:	
0183H	IN	AW, DW
	:	



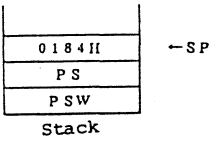
Example 2) With prefix

Address	Instruction
	:
0183H	REP
0184H	INM
	:



Reference) General software interrupt

Address	Instruction
	:
0183H	BRK 3
	:



4.9.3 FPO instruction interrupt

The construction of the external bus of uPD70322/70320 is different from that of the uPD70108/70116, so that the floating point operation co-processor for the uPD70108/70116 cannot be connected to the uPD70322/70320. Because of this, if the execution of the FPO instruction is attempted, which is an instruction for co-pocessors, an interrupt is generated to emulate the operation of the instruction. In the same way as the input/output instruction interrupt, the start address (or the start address of the prefix if a prefix is attached) of

this instruction is stored to the stack as the value of the PC (refer to 4.9.2 Input/output instruction interrupt). Therefore, the FPO instruction can be decoded and emulated by software. In the same way as the input/output instruction interrupt, the value on the stack must be adjusted when recovering from the FPO instruction interrupt.

4.10 Timing At Which No Interrupt Is Accepted

No interrupt is accepted between each of the following instructions and the next instruction.

(i) Manipulation instructions for sreg
MOV sreg, reg16; MOV sreg, mem16; POP sreg

(ii) Prefix instructions
PS:, SS:, DS:, DS1:, REPC, REPNC, REP, REPE, REPZ, REPNE, PEPNZ, BUSLOCK

(iii) EI, RETI, DI

(iv) FINT

Except for the INT, an interrupt generated in the timing at which no interrupt is accepted will be accepted after the next instruction is executed if conditions allows.

4.11 Interrupt Processing During Block Process Instruction Execution

An interrupt is accepted while a block processing instruction (transfer, verify, search, input/output instructions) is being executed. The interrupt is accepted after one process is completed. In this case, the start address of the instruction including the prefix is automatically stored to the PC storage area in the register bank. Block processing that has been interrupted can be resumed by restarting the execution from the prefix when recovering from interrupt processing.

Including the repeat prefix, up to three different prefixes can be logically attached to a block processing instruction.

For the uPD70322/70320, the number of prefixes attached to the block processing instruction which was being executed when an interrupt was accepted can be determined, and the PC value can be automatically decremented according to the condition of the prefix attachment.

Example: REP

```
MOVB SS: SRC BLK, DES BLOCK
```

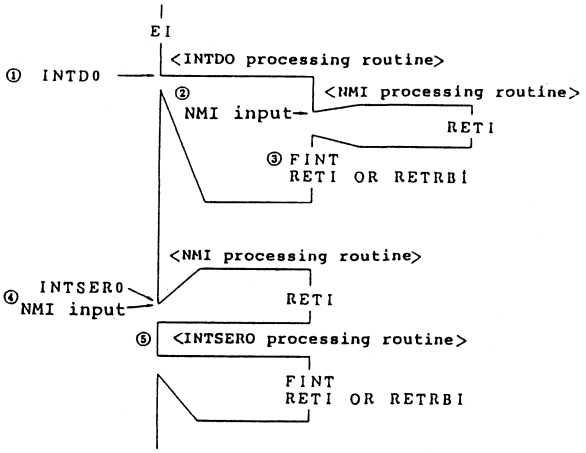
4.12 Acceptance of Interrupt

(1) NMI

o Cannot be masked by software

o Has the highest priority over all other interrupts

<Main routine>



① INTD0, whose priority is set to 2, is generated in the interrupt enabled condition (EI) and the processing of INTD0 is started.

② Interrupt is disabled (DI) while INTD0 is processed. However, when the NMI is generated, the processing of the INTD0 is interrupted, and the processing of the NMI is then started.

③ Upon the completion of NMI processing, the interrupted INTD0 is then processed. These then recovery from the interrupt processing is made.

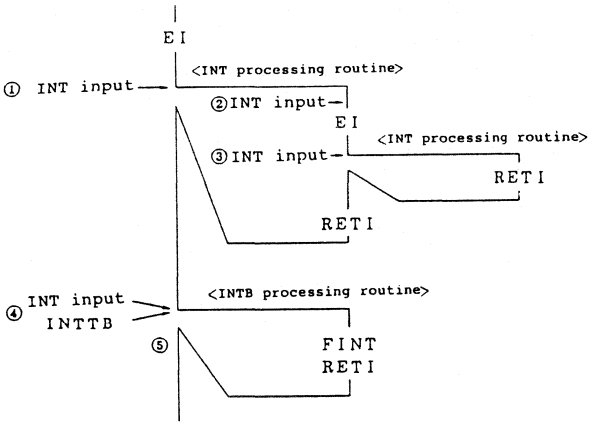
④ When the INTSER0 whose priority is set to 4 and NMI are generated simultaneously, the NMI which has the highest priority over all other interrupts is accepted and processed.

⑤ After the processing of the NMI is completed, the INTSER0 which has been retained is processed.

(2) INT

- o Accepted whenever interrupt is enabled (EI=1)
- o When two or more interrupts are generated simultaneously, this interrupt will have the lowest priority
- o Not controlled by the multiple processing controller

<Main routine>



① When the INT is generated in the interrupt-enabled condition (EI) and the processing of the INT is started.

② When the INT is accepted, the interrupt is automatically disabled (DI) so that another INT is not accepted if generated.

③ If the interrupt is enabled (EI) while the INT is being processed, multiple processing of INT is performed if a subsequent INT is generated.

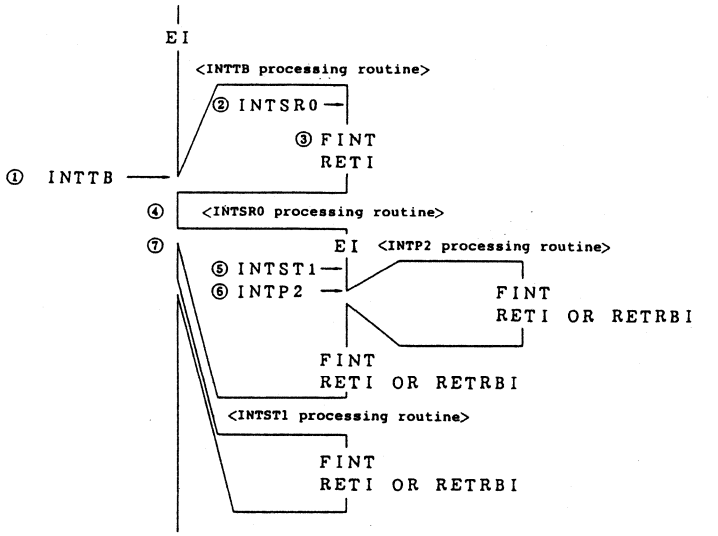
④ When the INT and the INTTB, whose priority is set to level 6, are simultaneously generated, the INT is not accepted (this is because the INT has the lowest priority) and the processing of the INTTB is started.

⑤ If the INT is inactive after the processing of the INTTB is completed, the INT interrupt is not accepted.

(3) Interrupts controlled by multiple processing

o Multiple processing is performed according to the priority order

<Main routine>



① The INTTB whose priority is set to level 6 is generated in the interrupt enabled condition (EI) and the processing of the INTTB is started.

② When the INTTB is accepted, the interrupt is disabled (DI), so that the INTSR0 having the higher priority than the INTTB is not accepted.

③ When recovering from an interrupt which receives a priority order control, the FINT instruction must be executed.

④ When the interrupt is enabled (EI) after the processing of the INTTB is completed, the INTSR0 which has been retained is processed

⑤ The INTST1 whose priority is set to level 5 is generated the interrupt-enabled condition (EI). However, when the INTSR0 is being processed and the priority of the INTST1 is lower than that of the INTSR0, it is not accepted.

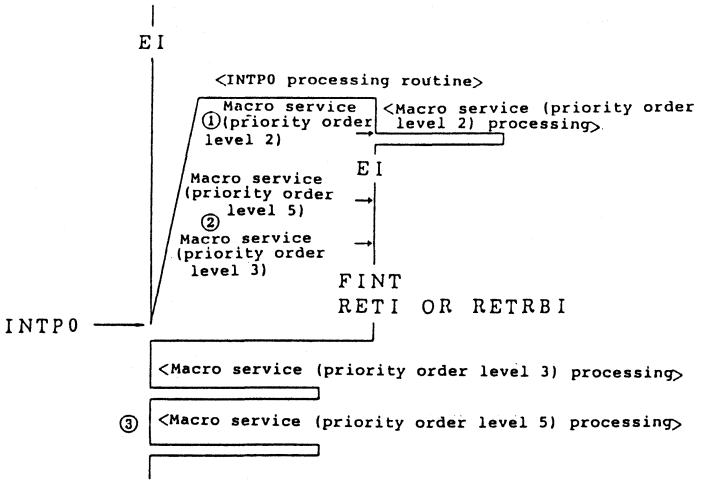
⑥ The INTP2 whose priority is set to level 3 is generated in succession and has a higher priority than the INTSR0, so that it is accepted and processed.

⑦ After the processing of the INTP2 is completed and the INTSR0 processing which has been interrupted is completed, the INTST1 is accepted.

(4) Macro service interrupt ..... A

- o Accepted regardless of EI or DI
- o In the same way as interrupts which receive multiple processing, multiple processing is performed according to the priority order

<Main routine>



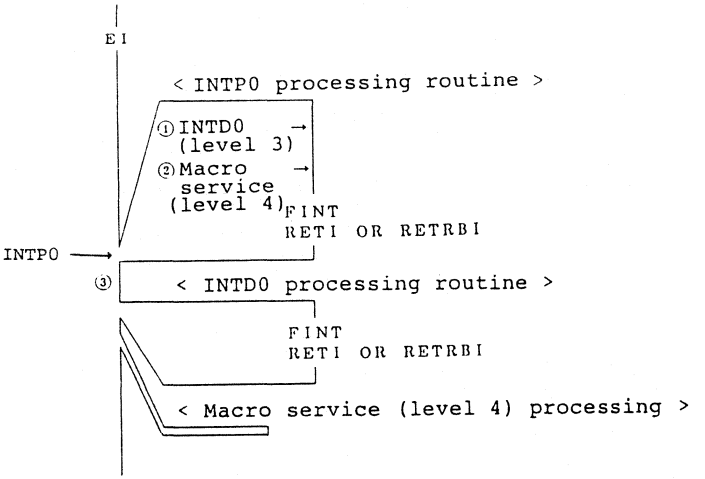
① When a macro service interrupt whose priority is set to level 2 is generated while the INTP0 whose priority is set to level 3 is processed, the macro service processing is performed, even if the interrupt is disabled

② Macro service interrupts whose priorities are set to levels 5 and 3 are generated while the INTP0 interrupt is enabled (EI). However, those priorities are lower than that of the INTP0, so that they are retained.

③ After the completion of INTP0 processing, the macro service interrupts that have been retained are accepted in order of higher priority.

(5) Macro service interrupt ..... B

<Main routine>



① When an INTD0 interrupt request (priority level 3) is generated while an INTP0 interrupt request (priority level 5) is being processed, the INTD0 interrupt request will be retained due to the effect of the interrupt disable state (DI).

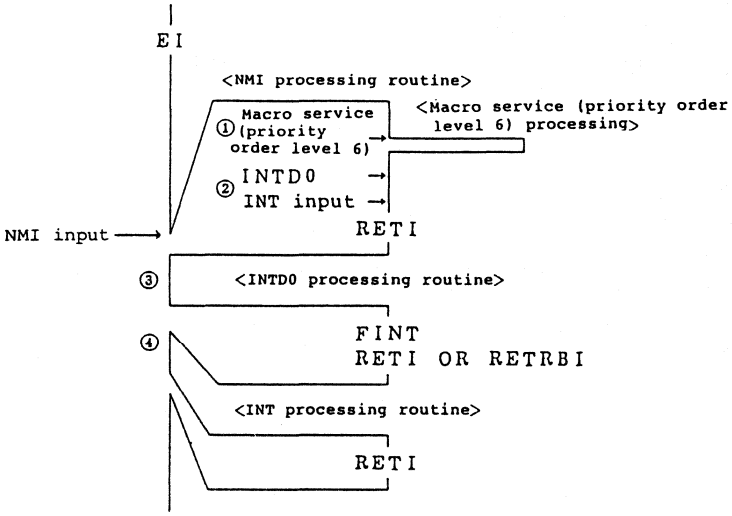
② When a macro service interrupt request with a level 4 priority is generated, this macro service interrupt request is also retained because the interrupt request with a level 3 priority has been retained.

③ The previously retained INTD0 is accepted upon completion of INTP0 processing. Afterwards, the macro service whose priority level is set to 3 is accepted.

(6) Priority order of macro service interrupts compared to other interrupts

o Macro service interrupts are accepted even while the NMI is being processed

<Main routine>



① When a macro service interrupt is generated, macro service processing is performed even while the NMI is being processed and the interrupt is disabled (DI).

② An interrupt which receives multiple processing control (INTD0) or INT is not accepted even if the same NMI is being processed and interrupt is disabled (DI).

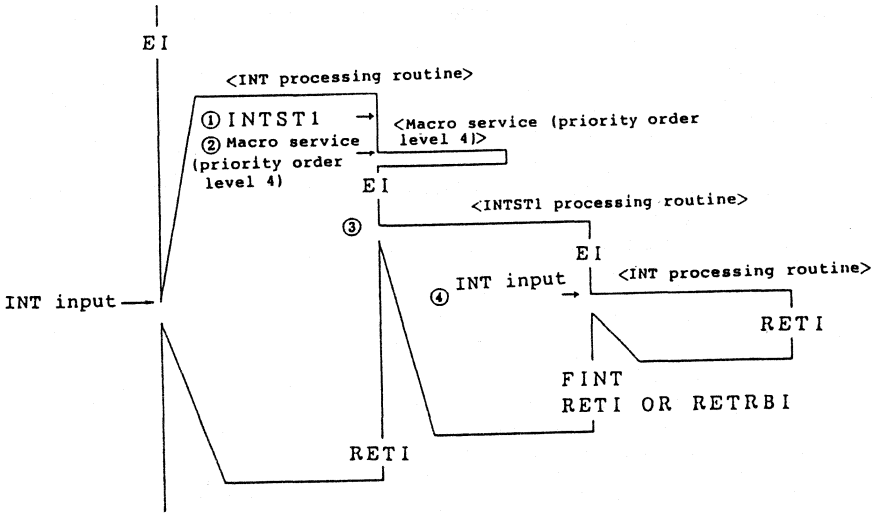
③ Upon the completion of the NMI processing, the INTD0 which has been retained, is accepted.

④ If the INT is active when the processing of the INTD0 is completed, the INT is accepted.

(7) Multiple processing of each interrupt

o INT, interrupts which receive multiple processing control, macro service interrupts

<Main routine>



① The INT is accepted and the interrupt is disabled (DI), so that an INTST1 whose priority is set to level 5 is not accepted.

② When the macro service interrupt whose priority is set to level 4 is generated, the macro service processing is started.

③ When an interrupt is enabled (EI), the INTIST1 which has been retained is processed.

④ If the interrupt is enabled (EI) while the INTIST1 is processed, the processing of the INT is started when the INT is generated.

CHAPTER 5 BUS CONTROL

The uPD70322/70320 has pins for controlling the buses. These pins are listed in Table 5-1. When using a pin shared by two or more functions, the function to be used must be selected by the port mode control register (PMCn).

Table 5-1 Pins for Bus Control

Pin	Input /output	Function	Remarks
A0 - A19	Output	Address bus	
D0 - D8	Input /output	Data bus	
R/$\overline{W}$	Output	Read/write identification	
$\overline{MREQ}$	Output	Indicates memory cycle	
$\overline{MSTB}$	Output	Memory read/memory write strobe signal	
$\overline{IOSTB}$	Output	I/O cycle strobe signal	
$\overline{REFRQ}$	Output	Indicates memory refresh cycle	
HLDRQ	Input	Bus hold request signal	Shares P27
$\overline{HLDAK}$	Output	Bus hold acknowledge	Shares P26
$\overline{DMAAK0}$	Output	Indicates DMA acknowledge cycle	Shares P21
$\overline{DMAAK1}$	Output	Indicates DMA acknowledge cycle	Shares P24
READY	Input	Externally inserts wait state into bus cycle	Shares P17
$\overline{INTAK}$	Output	Indicates interrupt acknowledge cycle	Shares P13 $\overline{INTP2}$

5.1 Programmable Wait Function

For the uPD70322/70320, a wait state can be inserted into the bus cycle (except the memory refresh cycle) by means of a software specification. The 1M-byte memory space is divided into eight blocks, each of which consists of 128K bytes. The insertion of the wait state is specified to each of these blocks and to the I/O space by using the wait control register (WTC) as shown in Fig. 5-1. However, block 6 (C0000 to DFFFFH) and block 7 (E0000 to FFFFFH) of the memory space will have the same specification.

The number of wait states can be selected from 0, 1, 2, and two or more (controlled from the READY pin) as shown in Table 5-2. Number of wait states can be independently and arbitrarily specified for each block. The READY pin is shared with pin P17. Therefore, when controlling the wait state by the READY pin, bit 7 of the port 1 mode control register (PMC1) must be set to 1. When bit 7 of the PMC1 register is reset to 0, it is always in the READY state, that is, the number of wait states is 2. When the control by the READY pin is selected, two wait states are inserted regardless of the status of the READY pin. The READY pin is sensed by its level, and the wait state is inserted while the pin is at low level.

Access operation to the built-in ROM (uPD70322 only) and the internal data area is not affected by the programmable wait function. This setting applies to all accessing except in the refresh cycle.

When $\overline{\text{RESET}}$ is input, the WTC register is initialized to FFFFH.

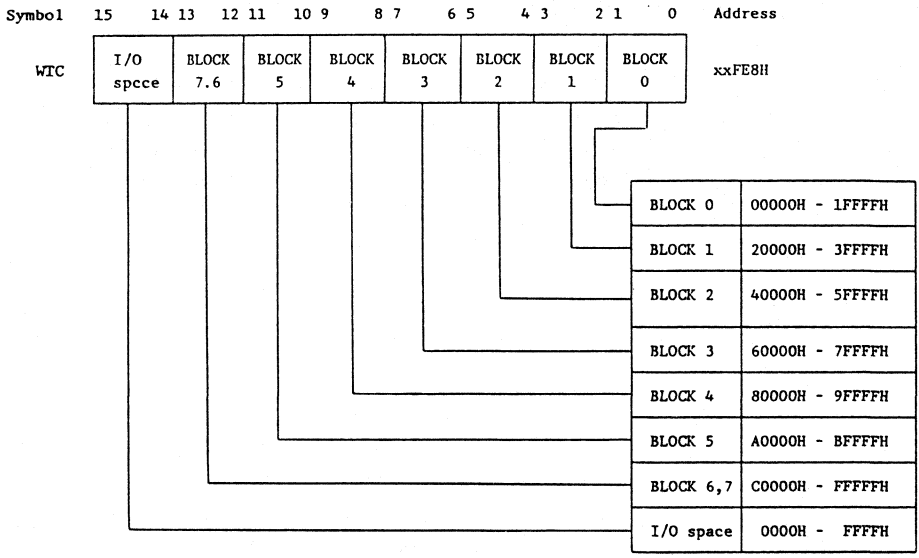


Fig. 5-1 Wait Control Register (WTC) Format

Table 5-2 Wait State Setting

BLOCKn/I/O space	Number of wait states
00	0 state
01	1 state
10	2 state
11	2 states + READY pin

5.2 Bus Hold Function

The uPD70322/70320 has the bus hold function. The external device can drive the HLDRQ pin high to request the bus control to the uPD70322/70320. When detecting the high level of the HLDRQ pin, the uPD70322/70320 sets each output of A0-19, D0-D7, $\overline{REFRQ}$, $\overline{MREQ}$, $\overline{MSTB}$, $\overline{IOSTB}$, and R/$\overline{W}$ to high impedance, and sets the $\overline{HLDAK}$ pin to low indicating that the bus is released to the external device. The uPD70322/70320 then enters the hold mode. During the hold mode, the uPD70322/70320 executes no instruction or accepts no prefetch interrupt, and only on-chip peripheral hardware which does not use the bus is operative. During the hold mode, the HLDRQ pin level is checked, and if a low level is detected, the $\overline{HLDAK}$ signal is set to high indicating that the bus is not relinguished to the external device, and the uPD70322/70320 resumes the instruction execution after one clock cycle.

The bus hold request can also be accepted during the HALT mode (refer to 12.2). In this case, the same hold mode is not different from the normal hold mode. The HALT mode is resumed when the hold mode is released (when the HLDRQ signal is low).

While an instruction which follows the BUSLOCK* prefix is executed or an operation which accepts an interrupt is executed, the bus hold request is not accepted.

For the uPD70322/70320, the memory refresh cycle can be inserted during the hold mode. This can be performed by setting the HLDRF (bit 6) of the refresh mode (RFM) register to 1. The refresh cycle is inserted after checking that the HLDRQ signal is

set to low by setting the $\overline{\text{HLDAK}}$ signal to high at every refresh timing. Afterwards, the hold mode is resumed if the HLDRQ signal has been set to high. In this case, if the HLDRQ signal remains set to low, the hold mode is released and the instruction execution is resumed.

The HLDRQ pin is shared with pin P27 and the $\overline{\text{HLDAK}}$ pin is shared with pin P26. Therefore, to used the bus hold function, bits 6 and 7 of the port 2 mode control register (PMC2) must be set to 1s.

Note: BUSLOCK
REP
MOVBK

When a program is written in this manner, the bus hold request cannot be accepted while a block processing instruction is being executed.

5.3 Refresh Function

The uPD70322/70320 has various functions to refresh the DRAM and pseudo-SRAM. The following functions are provided: a function that periodically inserts a refresh cycle into a serise of bus cycles, a function that outputs a refresh address for refreshing the pseudo-SRAM, a function that outputs a refresh pulse, a function that supports the pseudo-SRAM power down self-refresh mode, a function that generates the refresh cycle during the hold mode and HALT mode, and a function that inserts a wait state into the refresh cycle.

5.3.1 Refresh mode register (RFM)

The RFM register is an 8-bit register that controls the refresh functions. The RFM register can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, the RFM register is initialized to FCH.

The RFM register is organized as shown in the figure below.

The following describes each bit of this register.

Symbol	7	6	5	4	3	2	1	0	Address
RFM	RFLV	HLDRF	HLTRF	RFEN	RFW1	RFW0	RFT1	RFT0	xxFE1H

RFT0 , RFT1 These bits specify the refresh synchronization.

The refresh synchronization can be selected from output taps 3 to 6 of the time base counter (refer to CHAPTER 10). The refresh cycle is generated with the intervals as indicated in Table 5-3.

Table 5-3 Refresh Intervals

f_{CLK}=5MHz (=1/2f_x:f_x=10MHz)

RFT1	RFT0	Refresh interval
0	0	$2^4/f_{CLK}$ (3.2us)
0	1	$2^5/f_{CLK}$ (6.4us)
1	0	$2^6/f_{CLK}$ (12.8us)
1	1	$2^7/f_{CLK}$ (25.6us)

RFW0 , FRW1 These bits specify the number of wait states inserted into the refresh cycle.

The number of wait states inserted into the refresh cycle is not specified by the previously mentioned program wait functions (refer to 5.1), but is specified by the RFW0 and RFW1 bits as indicated in Table 5-4.

Table 5-4 Number of Wait States Inserted Into Refresh Cycle

RFW1	RFW0	Number of wait states
0	0	0
0	1	1
1	0	2
1	1	2

RFEN This bit enables the automatic insertion of the refresh cycle.

When this bit is set to 1, the automatic insertion of the refresh cycle is enabled. When this bit is reset to 0, the automatic insertion of the refresh cycle is disabled. The output of the $\overline{\text{REFRQ}}$ pin controlled by the content of the RFLV bit (refer to the explanation of the RFLV bit for detail).

HLTRF This bit enables the automatic insertion of the refresh cycle in the HALT mode.

When this bit is set to 1, the automatic insertion of the refresh cycle is enabled in the HALT mode. When this bit is reset to 0, the automatic insertion of the refresh cycle is disabled in the HALT mode. When the RFEN bit is reset to 0, the automatic insertion of the refresh cycle is disabled regardless of the contents of the HLTRF bit.

HLDRF This bit enables the automatic insertion of the refresh cycle in the hold mode.

When this bit is set to 1, the automatic insertion of the refresh cycle is enabled in the hold mode. When this bit is reset to 0, the automatic insertion of the refresh cycle is disabled in the hold mode. When the automatic insertion is enabled (when this bit is set to 1), the output of the HLDAK signal is enforceably set high to insert the refresh cycle.

RFLV This bit specifies the output level of the $\overline{\text{REFRQ}}$ signal.

The control circuit of the $\overline{\text{REFRQ}}$ bit is configured as shown in figure 5-2, and its output is determined by the logic indicated in Table 5-5.

When reading, the output of the master RFLV is read. When writing, data is written to the slave RFLV. Additionally, writing to the master RFLV is performed when the refresh timing is generated.

The pseudo-SRAM power down self-refresh code can be supported by using the RFLV bit.

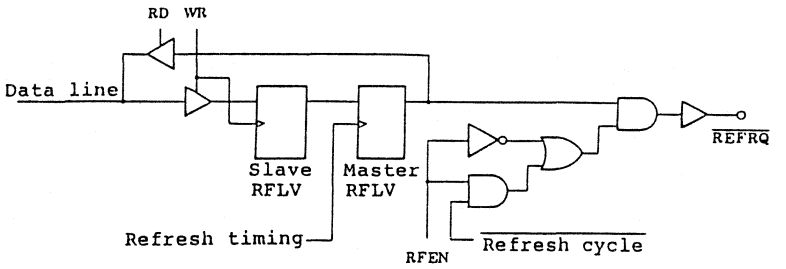


Fig. 5-2 Control Circuit Using the RFLV Bit

Table 5-5 Output Level of the $\overline{\text{REFRQ}}$ Signal

RFLV	RFEN	Status of the $\overline{\text{REFRQ}}$ Signal
0	0	0
0	1	0
1	0	1
1	1	Outputs the refresh pulse

The insertion of the refresh cycle is performed at the refresh timing when the RFEN bit is set to 1. In this case, the $\overline{\text{MREQ}}$, $\overline{\text{MSTB}}$, and $\overline{\text{IOSTB}}$ signals are set high, the refresh

address is output to A0-A8, the high level is output to A9-A19, and the refresh pulse is output from the $\overline{\text{REFRQ}}$ pin.

However, the data written to the RFLV bit is not read until the next refresh timing. This should be noted when using a bit manipulation instruction.

5.3.2 Configuration with pseudo-SRAM

Fig. 5-3 shows a circuit example of how to connect the pseudo-SRAM, equivalent to the uPD42832, to the uPD70322/uPD70320.

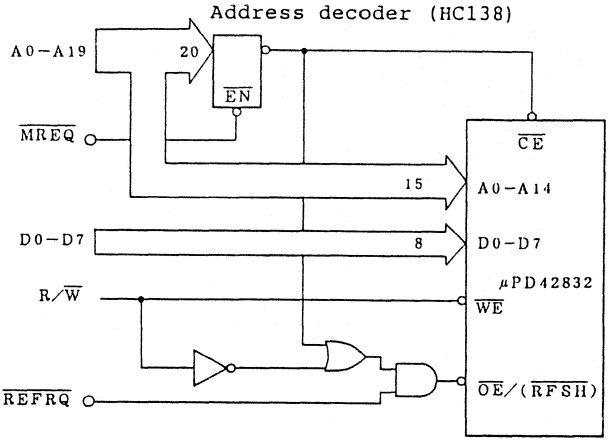


Fig. 5-3 Connection with the uPD42832

This connection can be used for the pulse refresh mode when a pulse is applied from the REFRQ pin to the $\overline{OE}$/($\overline{RFSH}$) pin when the $\overline{CE}$ pin is at high level, and for the power-down refresh mode when the $\overline{REFRQ}$ pin is set to low by resetting bit 7 (RFLV) of the refresh mode register (RFM) to 0 by means of software.

The power down refresh mode is used when the pulse refresh mode cannot be used because the CPU is in the standby mode (STOP). Therefore, the RFLV bit must be reset to 0 before initiating the standby mode, and set to 1 after recovering from the standby mode to enable pulse refresh operation.

Note: The $\overline{REFREQ}$ pin becomes high impedance during reset ($\overline{RESET}$ pin = 0). Therefore, care must be paid when using a $\overline{RESET}$ input to release the STOP mode.

5.3.3 Connection with DRAM

Fig. 5-4 shows a circuit example on how to connect the uPD41256 (256K x 1-bit) to the uPD70322/70320.

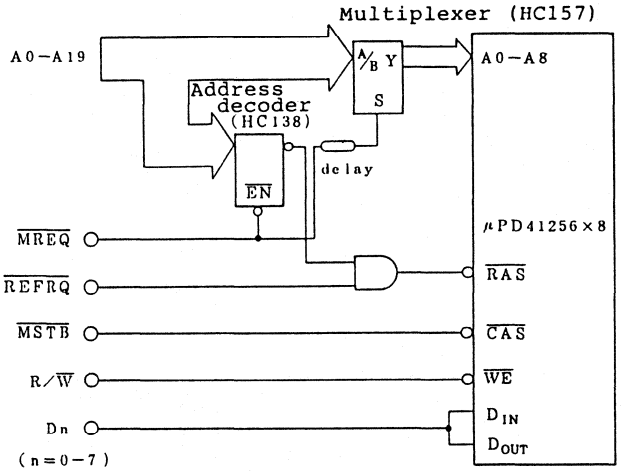


Fig. 5-4 Typical Connection of uPD41256 to the uPD70322/70320

In this connection, refresh operation is performed by the $\overline{RAS}$-only refresh method in which the 9-bit refresh address is output to the address bus in synchronization with the pulse output from the REFRQ pin.

5.4 Bus Control

The following describes the order of the bus control priority in the uPD70322/70320.

(1) Refresh cycle (refer to 5.3)

The refresh cycle is generated whenever the insertion of the refresh cycle is enabled. However, during the hold mode, if the insertion of the refresh cycle in the hold mode is enabled, the refresh cycle is executed when the HLDRQ signal is set to low after the $\overline{\text{HLDAK}}$ signal is enforcebly set to high.

(2) Hold mode (refer to 5.2)

The uPD70322/70320 enters the hold mode except when executing an instruction which follows the BUSLOCK prefix and during the interrupt acknowledge cycle.

(3) DMA cycle (refer to CHAPTER 6)

(4) Other bus cycles

However, when the interrupt acknowledge cycle is being executed, the refresh cycle, hold mode, and DMA cycle are temporarily suspended.

In addition, the bus does not operate in the stop mode (refer to Table 12-13 for bus status).

5.5 Bus Timings

Figs. 5-5 to 5-15 show major bus timings (except DMA).

When the bus is not being accessed, the control pins become inactive and the output of the data bus and address bus become undefined.

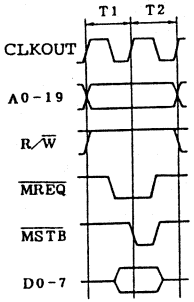


Fig. 5-5 Memory Read Cycle

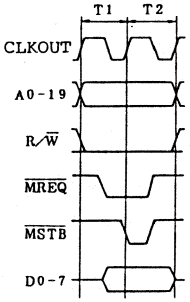


Fig. 5-6 Memory Write Cycle

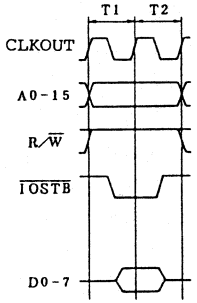


Fig. 5-7 I/O Read Cycle

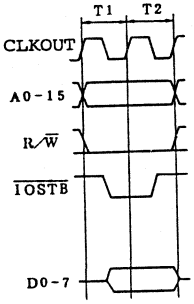


Fig. 5-8 I/O Write Cycle

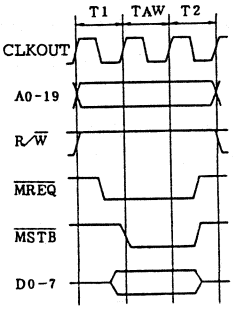


Fig. 5-9 Memory Read Cycle (with one wait state inserted)

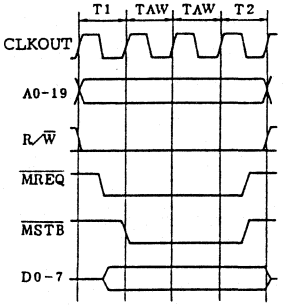


Fig. 5-10 Memory Write Cycle (with two wait states inserted)

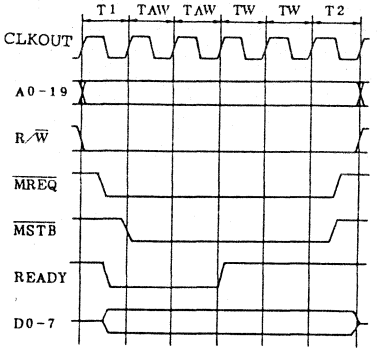


Fig. 5-11 Memory Write Cycle (when controlled by the READY pin)

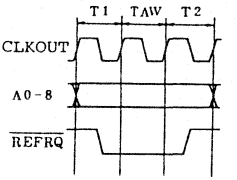


Fig. 5-12 Refresh Cycle (with one wait state inserted)

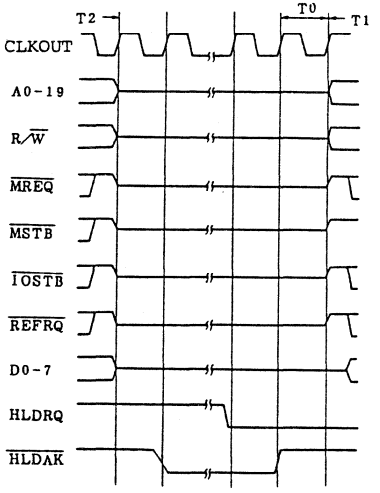


Fig. 5-13 Bus Hold Accept and Release Timings

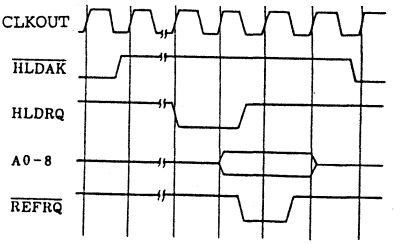


Fig. 5-14 Refresh Cycle in Hold Mode (with no wait state inserted)

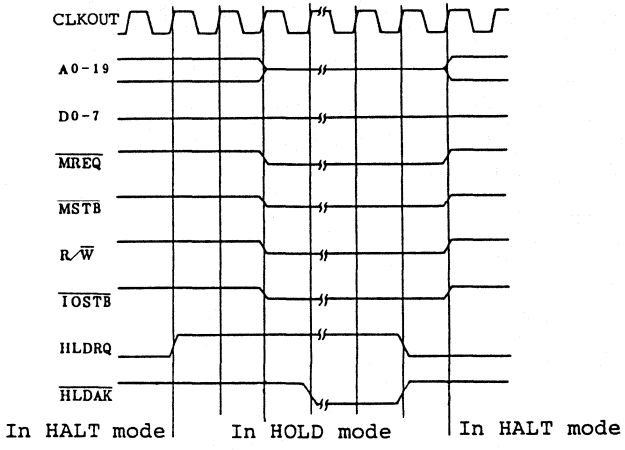


Fig. 5-15 Bus Hold Accept and Release Timing in HALT Mode

CHAPTER 6 DMA CONTROLLER

The uPD70322/70320 has a built-in 2-channel DMA controller which can directly specify 1M byte of memory space.

6.1 Pin Functions

The following pins are provided for the DMA controller. All of these pins are shared with port pins; therefore, when using these pins, the corresponding bit of the port 2 mode control register (PMC2) must be set to 1.

(1) DMARQ0, DMARQ1 (P20, P23)

These are active high DMA request input pins.

(2) $\overline{\text{DMAAK0}}$, $\overline{\text{DMAAK1}}$ (P21, P24)

These are active low DMA acknowledge signal pins. However, these signals are not output for memory-to-memory DMA transfer operation (burst mode, single-step mode).

(3) $\overline{\text{TC0}}$, $\overline{\text{TC1}}$ (P22, P25)

These are active low DMA transfer completion signal output pins. These signals are output when TC0, and TC1 in the DMA service channel become 0, respectively.

6.2 DMA Operation

The uPD70322/70320 has four different DMA transfer modes. Table 6-1 lists the function of each of these transfer modes.

For DMA transfer, the internal ROM area and internal data area cannot be accessed. If an attempt is made to access an address equivalent to the internal ROM area or the internal data area, the external memory of the same address will be accessed.

Table 6-1 Function of Each Transfer Mode

Mode name	Source/ destination	Function	DMA is initialized	Method of stopping	Interrupt	During HALT mode	DMA request during DMA operation
Single -step	Memory -memory	Alternately repeats one instruction execution and one DMA transfer for a specified number of times each time a DMA request is generated.	o At the rising edge of the DMARQ signal. o When the TDMA bit of the DMA control register is set.	By software	All interrupts are accepted.	DMA transfer is succes-sively performed for a spec-ified number of times.	DMA of channel 1 is retained or interrupted and the DMA of channel 0 is performed.
Burst	Memory -memory	Repeats DMA trans-fer for a specified number of times when a DMA request is generated.	o At the rising edge of the DMARQ signal. o When the TDMA bit of the DMA control register is set.	None	No interrupt is accepted during DMA transfer operation.	DMA transfer is succes-sively performed for a spec-ified number of times.	Other DMAs are retained until the current DMA transfer is completed.
1 Trans-mission	Memory - I/O	Performs a DMA transfer each time a DMA request is generated.	o At the rising edge of the DMARQ signal.	By software	All interrupts are accepted.	The same as normal	The requested DMA is performed after one DMA transfer is completed.
Demand release	Memory - I/O	Performs DMA transfer while the DMARQ pin is set to high.	o When the DMARQ signal is at high level.	o During DMA transfer, DMA is terminated when the DMARQ signal is set low. o In other cases, DMA is terminated by software.	o During DMA transfer, no interrupt is accepted. o In other cases, all interrupts are accepted.	The same as normal	Other DMAs are retained during DMARQ is high level.

For memory-to-memory DMA transfer, the $\overline{DMAAK}$ signal is not output. For memory -- I/O DMA transfer, the $\overline{DMAAK}$ signal is output every DMA cycle.

The programmable wait function (refer to 5.1) is effective also during the DMA transfer. For memory-to-memory DMA transfer, a specified number of wait states is inserted individually to the transfer source and destination. For memory -- I/O DMA transfer, in order to complete one transfer in one bus cycle, a wait state is inserted to the memory or I/O, whichever is slower.

The bus hold function and refresh function remain effective during the DMA transfer so that the DMA transfer can be temporarily interrupted by these functions.

DMA transfer is performed whenever requested, even if a block processing (transfer, comparison, retrieve input/output) instruction with a repeat prefix is being executed. In this case, the block processing instruction is temporarily interrupted. In addition, DMA transfer is also performed in the same way when the BUSLOCK prefix is attached.

During the DMA transfer, no interrupt is accepted and any interrupt generated is retained.

In the HALT mode, the DMA transfer is performed if requested. In this case, the HALT mode is resumed when the DMA transfer is completed. If the DMA transfer completion interrupt has been generated when returned to the HALT mode, the HALT mode is released.

When the DMA requests are simultaneously generated, the priority is given to channel 0.

An interrupt can be generated when the DMA transfer is completed (when the specified number of DMA transfers are completed).

6.3 Registers for Controlling DMA

The DMA mode register and DMA control register are provided for specifying the DMA transfer mode, etc. Additionally, the DMA service channel is mapped in the built-in RAM to specify the transfer destination, transfer source, and number of transfers.

A register to control interrupt request is also provided. Each of these registers provided for each channel.

6.3.1 DMA mode register (DMAM0, DMAM1)

This is an 8-bit register that specifies the DMA transfer mode, etc. The DMAMn register (n=0, 1) can be read/written to per 8 bits or 1 bit cy accessing the memory.

When $\overline{\text{RESET}}$ is input, the DMAMn register is initialized to 00H.

	Symbol	7	6	5	4	3	2	1	0	Address
(Channel 0)	DMAM0	MD2	MD1	MD0	W	EDMA	TDMA	0	0	xxFA1H
(Channel 1)	DMAM1									xxFA3H

MD2 , MD1 , MD0 These bits specify the transfer mode.

MD2	MD1	MD0	Transfer mode
0	0	0	Single-step mode
0	0	1	Demand release mode (I/O → memory)
0	1	0	Demand release mode (memory → I/O)
0	1	1	Use of this combination inhibited
1	0	0	Burst mode
1	0	1	1 transfer mode (I/O → memory)
1	1	0	1 transfer mode (memory → I/O)
1	1	1	Use of this combination inhibited

W This bit specifies whether the transfer is performed per byte or per word.

When this bit is 0, the transfer is performed per byte. When this bit is 1, the transfer is performed per word.

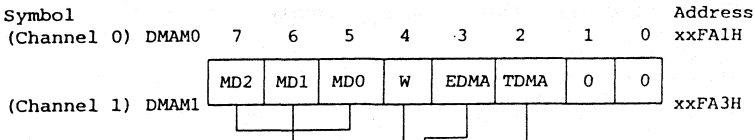
EDMA This bit enables or diasables DMA transfer.

When this bit is 1, DMA transfer is enabled. When this bit is 0, DMA transfer is disabled. This bit is

automatically cleared to 0 when the terminal counter (TC) of the DMA service channel becomes 0.

TDMA Transfer start bit

This specification is effective only in the single-step mode or burst mode. The DMA is started by writing 1 to this bit (however, when the EDMA bit is set to 1). The read out level of this bit is always 0. This bit has no meaning in the demand release mode and 1-transfer mode.



EDMA	TDMA	Transfer mode
0	0	DMA disabled state
1	0	Maintain condition
1	1	Start DMA transfer

EDMA	DMA transfer enable/disable
0	DMA disabled state
1	DMA enabled state

W	Transfer processing method
0	Byte transfer
1	Word transfer

MD2	MD1	MD0	Transfer mode
0	0	0	Single-step mode
0	0	1	Demand release mode (I/O → memory)
0	1	0	Demand release mode (memory → I/O)
0	1	1	Use to this combination inhibited
1	0	0	Busrt mode
1	0	1	1 transfer mode (memory → I/O)
1	1	0	1 transfer mode (memory → I/O)
1	1	1	Use to this combination inhibited

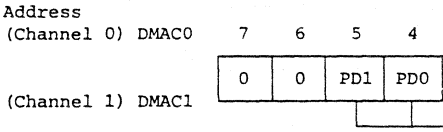
Fig. 6-1 DMA Mode Register (DMAM0, DMAM1) Format

6.3.2 DMA control register (DMAC0, DMAC1)

This is an 8-bit register which specifies the modification method of the source address and destination address for DMA transfer. The DMACn register (n=0, 1) can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, the contents of the DMACn register are retained, and are undefined.

As shown in Fig. 6-2, bits 1 and 0 (PS1 and PS0) of the DMACn register specify the modification mode of the offset value of the source address. Bits 5 and 4 (PD1, PD0) specify the offset value of the destination address.



PS1/PD1	PS0/PD0	Address offset value modification mode
0	1	Increments the offset value of address
1	0	Decrements the offset value of address
0	0	No modification
1	1	

Fig. 6-2 DMA Control Register (DMAC0, DMAC1) Format

6.3.3 DMA service channel

The DMA service channel specifies the transfer source, destination, and number of transfers for DMA transfer, and is mapped in the built-in RAM. Channel 0 is mapped into xxD00 to xxE07H on the built-in RAM address, and channel 1 is mapped into xxE08 to xxE0FH on the built-in RAM address (xx is the value specified by the IDB register). It should be noted that these areas are allocated to the same area as macro service channels 0 and 1 and register bank 0.

The source and destination addresses for DMA are specified in the same way as when normally accessing the memory. That is, these addresses are specified by the segment and the offset value from the segment. However, on the segment, only the upper 8 bits can be specified, and the lower 8 bits are fixed to 0s. When the address is modified by DMA transfer, only this offset value is modified.

Fig. 6-3 shows the organization of this DMA service channel. Fig. 6-4 shows how the DMA address is generated.

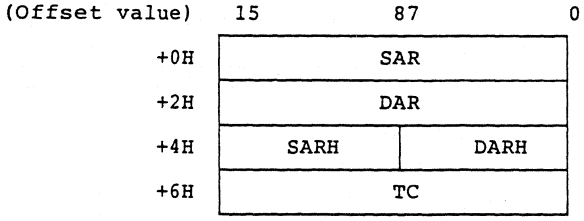


Fig. 6-3 DMA Service Channel Format

- SAR (+0H) : Specifies the offset value (lower 16 bits) of the source address of the DMA transfer.
- DAR (+2H) : Specifies the offset value (lower 16 bits) of the destination address of the DMA transfer.
- DARH (+4H) : Specifies the upper 8 bits of the segment value of the destination address of the DMA transfer. The lower 8 bits are fixed to 0s.

. SARH (+5H) : Specifies the upper 8 bits of the segment value of the source address of the DMA transfer. The lower 8 bits are fixed to 0s.

. TC (+6H) : Specifies the number of DMA transfers.

Parentheses indicate the offset value from the DMA service channel start address.

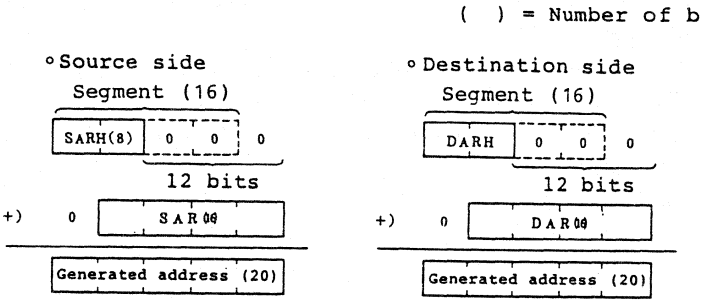


Fig. 6-4 DMA Address Generation Method

DMA service channel 0 is allocated to xxE00H, and DMA service channel 1 is allocated to xxE08H (xx is the value specified by the IDB register).

The DMA service channel is automatically modified by the DMA operation. TC is decremented (-1) each time one DMA transfer is performed (regardless of whether the transfer is performed per byte or word). The offset value of the address is modified according to the mode specified by the DMA control register (DMACn).

For byte data transfer, the offset value is incremented or decremented (+/-1). For word data transfer, the offset value is incremented/decremented (+/-2) or remains unchanged.

6.3.4 DMA interrupt request control register (DIC0, DIC1)

This is an 8-bit register that controls an interrupt which is generated upon the completion of the DMA transfer. The interrupt is generated when the terminal count (TC) becomes 0.

The DICn (n=0, 1) can be read/written to per 8 bits or 1 bit by accessing the memory.

This register is initialized to 47H when $\overline{\text{RESET}}$ is input.

The macro service function is not supported for this interrupt. The DMA transfer completion interrupt of channel 0 (INTD0) and channel 1 (INTD1) forms groups with the DIC0 and DIC1 registers, respectively, and channel 0 has the higher priority. INTD0 is controlled by the DIC0 register and its vector is 36, and INTD1 is controlled by the DIC1 register and its vector is 37 (refer to 3.4.5).

(Interrupt priority order) DF0 > DF1

Symbol	7	6	5	4	3	2	1	0	Address
DIC0	DF0	DMK0	0	ENCS	0	PR2	PR1	PR0	xxFACH
DIC1	DF1	DMK1	0	ENCS	0	1	1	1	xxFADH

Fig. 6-5 DMA Interrupt Request Control Register (DIC0, DIC1) Format

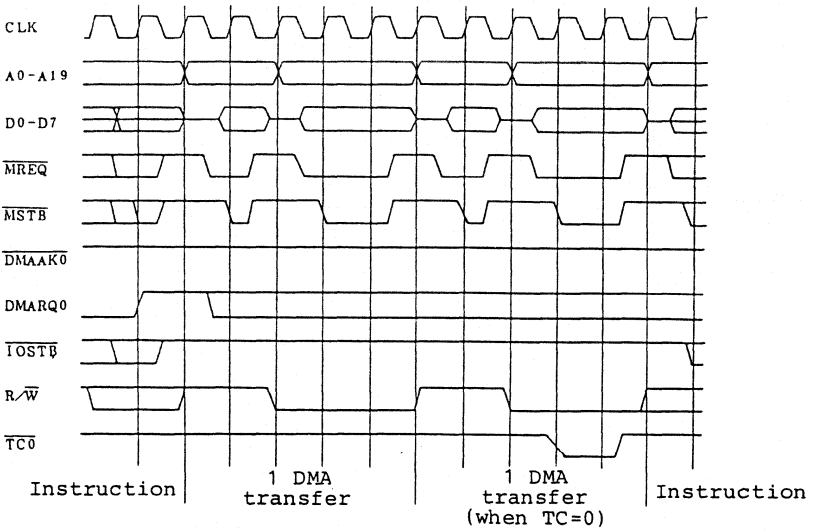
Note: Bits 2 to 0 of the DIC1 register are fixed to 1 by means of hardware. However, bits 2 to 0 of the DIC0 register are the bit field (PR2 to PR0) which specifies the priority order of the interrupt request per group and form a group with the DIC1 register.
The interrupt priority order of the DIC1 register is determined by the PR2 to PR0 bits of the DIC0 register.

The DF0/DF1 bit is the DMA transfer completion interrupt request flag. The DMK0/DMK1 bit is the DMA transfer completion mask bit.

The function of other bit fields are explained in 4.7.1.

6.4 DMA Transfer Timing

Figs. 6-6 to 6-9 show major DMA transfer timings.



Figs. 6-6 In Burst Mode (When DMA is initialized when TC is 2, and no wait state is inserted into the transfer source and one wait state is inserted into the transfer destination)

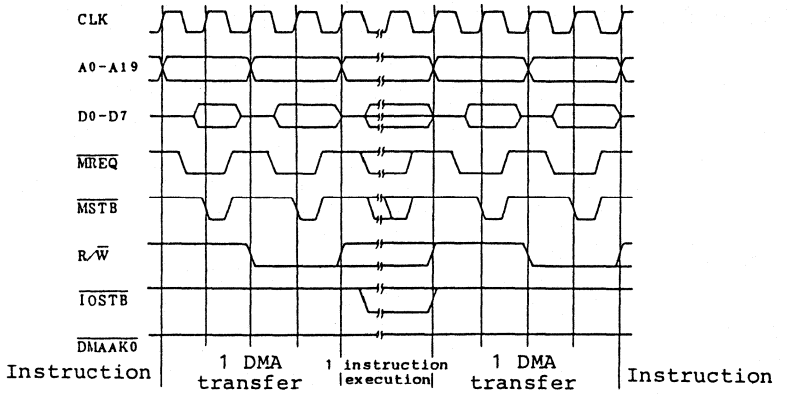


Fig. 6-7 Single-Step Mode

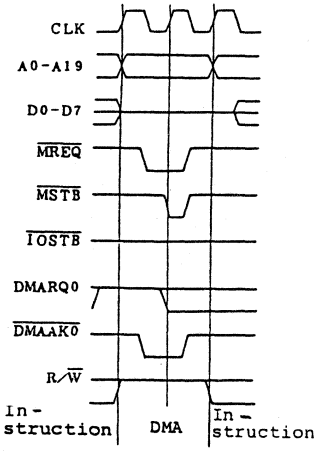


Fig. 6-8 1-Transfer Mode (Memory → I/O, with no wait state inserted)

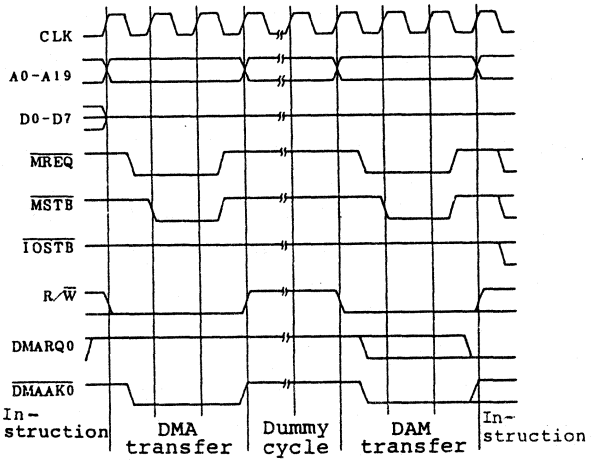


Fig. 6-9 Demand Release Mode (I/O → memory, I/O: one wait state inserted, memory: no wait state inserted)

For I/O-to-memory or memory-to-I/O DMA transfer, the IOSTB signal (high level) is not output. The DMAAK signal (low level) is output.

Therefore, whether or not the I/O device is accessed is determined by using the DMAAK signal. The DMAAK signal is not output for memory-to-memory DMA transfer.

Fig. 6-10 shows how to generate the IORD and IOWR signals from the R/W, IOSTB, and DMAAK signals.

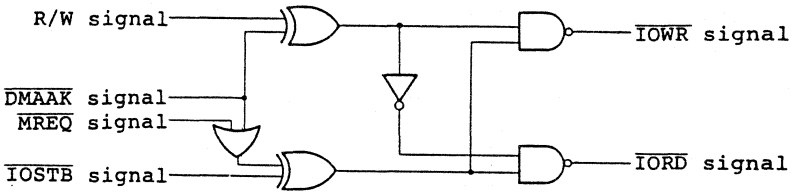


Fig. 6-10 Typical Circuit for Generating IORD and IOWR Signals

CHAPTER 7 PORT FUNCTIONS

7.1 Ports 0-2

7.1.1 Hardware organization

Ports 0 to 2 of the uPD70322/70320 are basically organized as a three-state bidirectional port as shown in Fig. 7-1.

When $\overline{\text{RESET}}$ is input, all bits of each port mode register are set to 1 and specified for the input port and all port pins go to the high impedance state. The contents of the output latch is not affected by $\overline{\text{RESET}}$ input.

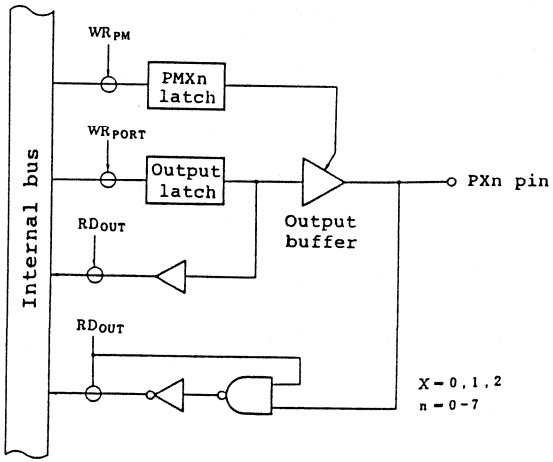


Fig. 7-1 Port (0-2) Organization

Note: PMXn latch:
Bit n of port mode register PMX.

(1) When specified for output port (PMXn=0)

The output latch becomes effective, and data can be transferred between the output latch and the accumulator by the transfer instruction.

The contents of the output latch can be freely set by a logical operation instruction. Data once written to the latch is retained until the next port manipulation instruction is executed.

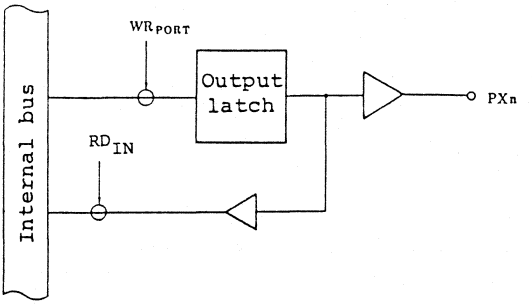


Fig. 7-2 Port Specified for Output Port

(2) When specified for input port (PMXn=1)

The port pin level can be loaded to the accumulator. Also in this case, writing to the output latch is possible so that the data transferred from the accumulator by the transfer instruction can be stored to all output latches regardless of whether the port is specified for input or output. However, a bit specified for the input port is at high impedance so that the contents of the buffer is not output to the port pin (the contents of the output latch is output when the bit specified for the input port is switched to the output port). Additionally, the contents of the output latch of the bit specified for the input port cannot be loaded to the accumulator (refer to Fig. 7-3).

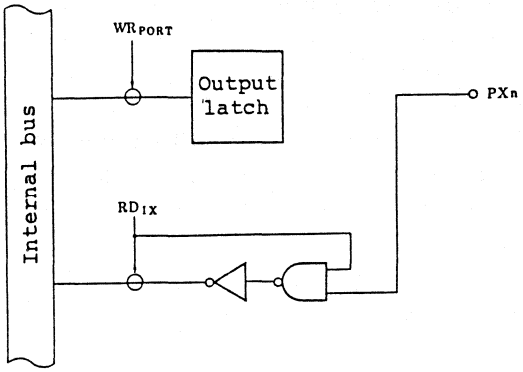


Fig. 7-3 Port Specified for Input Port

(3) When specified for control signal pin (PMCXn=1)

Pins of ports 0 to 2 can be specified for control signal input or output per bit by setting the corresponding bit of the port mode control register (PMCX) to 1 regardless of the setting of the port mode register (PMX). The status of the control signal of the pin used as a control signal pin can be checked by executing the port access instruction.

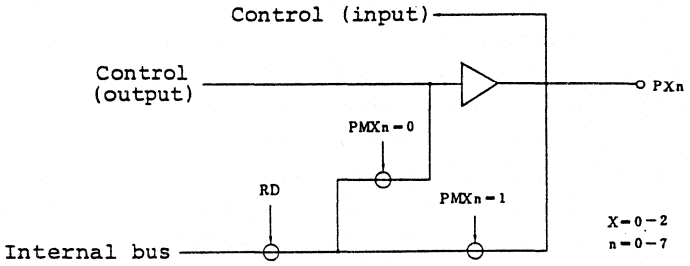


Fig. 7-4 When Specified for Control Signal Pin

(a) For control signal output

The status of each control signal pin can be read by executing the port read instruction when the port mode register (PMXn) is set to 1.

If the port read instruction is executed when the port mode register is reset to 0, the status of the internal signal is read.

(b) For control signal input

The status of each control signal pin can be read by executing the port read instruction only when the port mode register is set to 1.

7.1.2 Function of each port

(1) P00-07 (port 0) ... Three-state input/output

This is an 8-bit special input/output port. This port functions as a general purpose input/output port which can be specified for input/output per bit. In addition to that, a pin of this port can function as the system clock output pin (shared with P07). Function of each pin can be individually selected by the port 0 mode register (PM0) and the port 0 mode control register (PMC0).

Table 7-1 Port 0 Operation (n=0-7)

	PMC0n=1	PMC0n=0	
		PM0n=1	PM0n=0
P00		Input port	Output port
P01		Input port	Output port
P02		Input port	Output port
P03		Input port	Ourput port
P04		Input port	Output port
P05		Input port	Output port
P06		Input port	Output port
P07	CLKOUT output	Input port	Output port

(a) Port 0 mode control register (PMC0)

This is an 8-bit register that specifies port 0 for port/system clock output per bit. Therefore, the PMC0 register can be written only in 8-bit units by executing memory access operation. If the corresponding bit of the PMC0 register is set to 1, the port functions in the system clock output mode (P07). If reset to 0, the port functions in the port mode. When $\overline{\text{RESET}}$ is input, all bits of the PMC0 register are reset to 0, and the port enters the port mode.

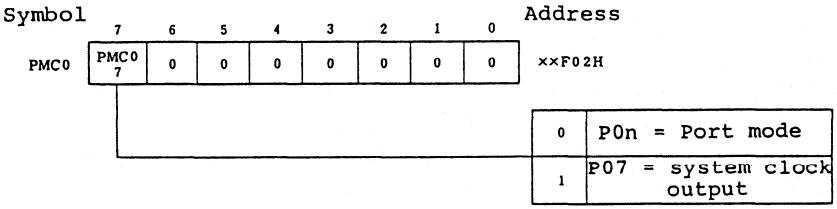


Fig. 7-5 Port 0 Mode Control Register (PMC0) Format

(b) Port 0 mode register (PM0)

PM0 is an 8-bit register which can specify port 0 for input/output per bit. The PM0 register can be written only in 8-bit units by executing memory access operation.

The PM0 register becomes effective when the bit corresponding to the PM0 register of the PMC0 register is 0.

When $\overline{\text{RESET}}$ is input, all bits of this register are set to 1.

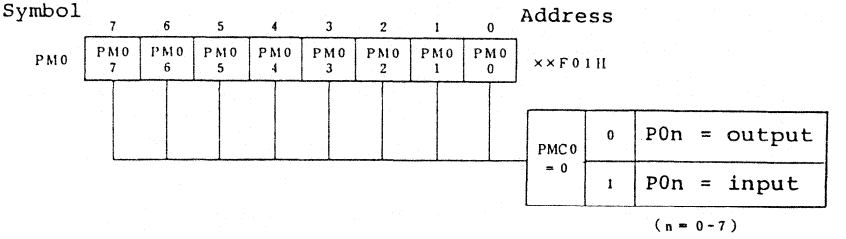


Fig. 7-6 Port 0 Mode Register (PM0) Format

(2) P10-17 (port 1) Three-state input/output

This is an 8-bit special input/output port. In the same way as port 0, this port functions as a general purpose input/output port which can be specified for input/output per bit. In addition, each pin of this port can function as a control pin. The function of each pin can be individually selected by the port 1 mode register (PM1) and the port 1 mode control register (PMC1).

The pin levels of the P10 to P13 pins can be read by directly reading port 1 (P1).

Table 7-2 Port 1 Operation (n=0-7)

	PMC1n=1	PMC1n=0	
		PM1n=1	PM1n=0
P10		MN1 input	
P11		$\overline{\text{INTP0}}$ input	
P12		$\overline{\text{INTP1}}$ input	
P13	$\overline{\text{INTAK}}$ output	$\overline{\text{INTP2}}$ input	
P14	INT input	Input port ($\overline{\text{POLL}}$ input)	Output port
P15	TOUT output	Input port	Output port
P16	$\overline{\text{SCK0}}$ output	Input port	Output port
P17	READY output	Input port	Output port

(a) Port a mode control register (PMC1)

This is an 8-bit register that specifies port 1 for port/control signal input/output per bit. Therefore, the PMC1 register can be written only in 8-bit units by executing memory access operation. If the corresponding bit of the PMC1 register is set to 1, the bit functions in the control signal input/output mode. If reset to 0, the port functions in the port mode. When $\overline{\text{RESET}}$ is input, all bits of the PMC2 register are reset to 0, and the port enters the port mode.

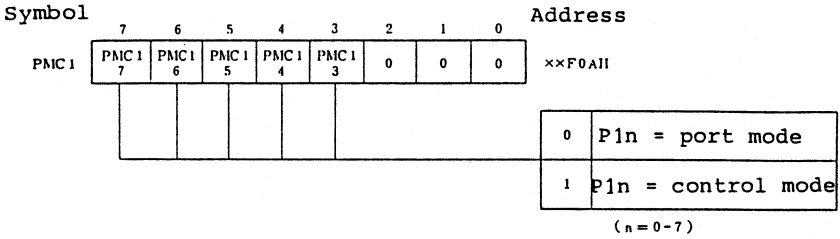


Fig. 7-7 Port 1 Mode Control Register (PMC1) Format

(b) Port 1 mode register (PM1)

PM1 is an 8-bit register which can specify port 1 for input/output per bit. The PM1 can be written only in 8-bit units by executing memory access operation.

The PM1 register becomes effective when the bit corresponding to the PM1 register of the PMC1 is 0.

When $\overline{\text{RESET}}$ is input, all bits of this register are set to 1.

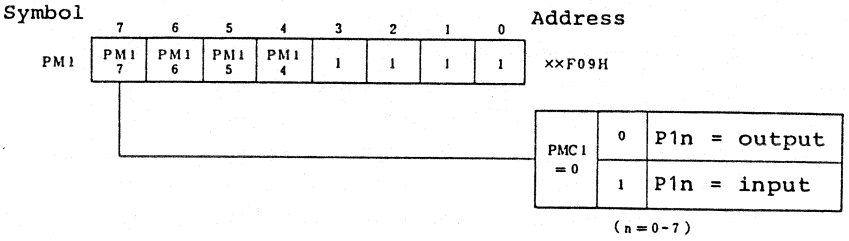


Fig. 7-8 Port 1 Mode Register (PM1) Format

(3) P20-P27 (port 2) Three-state input/output

This is an 8-bit special input/output port. In the same way as port 0, this port functions as a general purpose input/output port which can be specified for input/output per bit. In addition, each pin of this port can function as a control pin. The function of each pin can be individually selected by the port 2 mode register (PM2) and the port 2 mode control register (PMC2).

Table 7-3 Port 2 Operation

	PMC2=1	PMC2=0	
		PM2n=1	PM2n=0
P20	DMARQ0 input	Input port	Output port
P21	$\overline{\text{DMAAK0}}$ input	Input port	Output port
P22	$\overline{\text{TC0}}$ output	Input port	Output port
P23	DMARQ1 input	Input port	Ourput port
P24	$\overline{\text{DMAAK1}}$ output	Input port	Output port
P25	$\overline{\text{TC1}}$ output	Input port	Output port
P26	$\overline{\text{HLDAK}}$ output	Input port	Output port
P27	HLDRQ output	Input port	Output port

(a) Port 2 mode control register (PMC2)

This is an 8-bit register that specifies port 2 for port/control signal input/output per bit.

Therefore, the PMC2 register can be written only in 8-bit units by executing memory access operation.

If the corresponding bit of the PMC2 register is set to 1, the bit functions in the control signal input/output mode mode. If $\overline{RESET}$ to 0, the port functions in the port mode. When $\overline{RESET}$ is input, all bits of the PMC2 register are reset to 0, and the port enters the port mode.

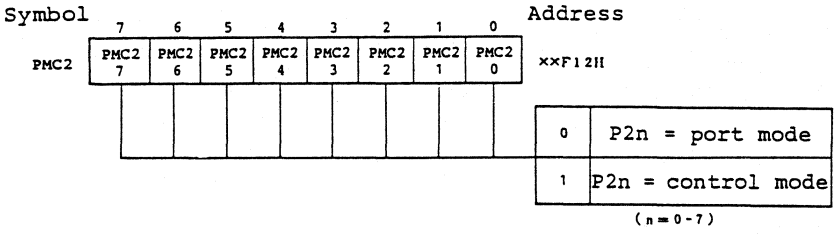


Fig. 7-9 Port 2 Mode Control Register (PMC2) Format

(b) Port 2 mode register (PM2)

PM2 is an 8-bit register which can specify port 2 for input/output per bit. The PM2 can be written only in 8-bit units by executing memory access operation.

The PM2 register becomes effective when the bit corresponding to the PM2 register of the PMC2 is 0.

When $\overline{RESET}$ is input, all bits of this register are set to 1.

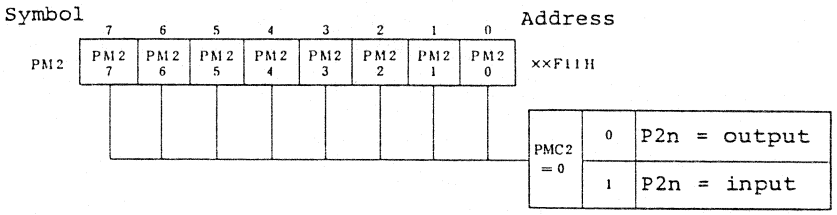


Fig. 7-10 Port 2 Mode Register (PM2) Format

7.2 Port T (PT0-PT7)

Port T is an 8-bit input port whose threshold voltage (reference voltage) can be changed in 16 steps. Comparator operation can be performed by using this port.

7.2.1 Hardware organization

As shown in Fig. 7-11, port T consists of the PT0-PT7 comparator inputs, VTH reference voltage input pin, multiplexing circuit (MPX) which sets the reference voltage (VREF) to one of 16 levels (1/16 x VTH to 16/16 x VTH), port T mode register (PTM) which controls the MPX, and eight latches.

The VREF selected by the PTM setting and the PT0 to PT7 inputs are compared, and the results are latched to the port T input latches.

$$\begin{cases} \text{VREF} > \text{PTn} \rightarrow 0 \\ \text{VREF} < \text{PTn} \rightarrow 1 \end{cases}$$

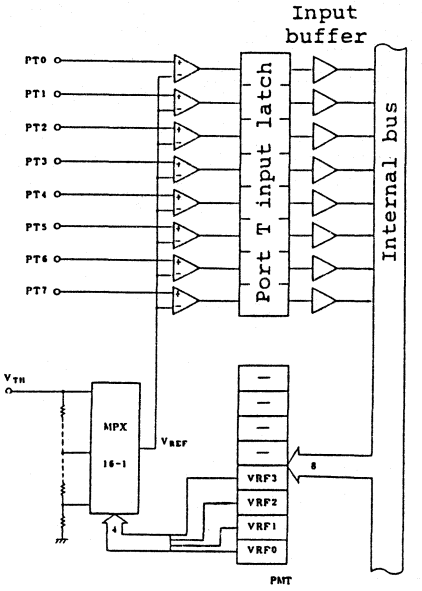


Fig. 7-11 Port T Block Diagram

Note: The V_{TH} pin is connected to the GND pin through a high-resistance resistor. Therefore, the current drawn will be increased if voltage is applied to the V_{TH} pin in the standby mode.

7.2.2 Port T mode register (PMT)

The PMT register selects the comparator reference voltage (VREF) from 16 settings as shown in Fig. 7-12. The PMT register can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, all bits of the PMT register are reset to 0.

Symbol

	7	6	5	4	3	2	1	0	Address
PMT	–	–	–	–	VREF 3	VREF 2	VREF 1	VREF 0	××F3BH

VREF 3	VREF 2	VREF 1	VREF 0	V_{REF}
0	0	0	0	$V_{TH} \times 16 / 16$
0	0	0	1	$V_{TH} \times 1 / 16$
0	0	1	0	$V_{TH} \times 2 / 16$
0	0	1	1	$V_{TH} \times 3 / 16$
0	1	0	0	$V_{TH} \times 4 / 16$
0	1	0	1	$V_{TH} \times 5 / 16$
0	1	1	0	$V_{TH} \times 6 / 16$
0	1	1	1	$V_{TH} \times 7 / 16$
1	0	0	0	$V_{TH} \times 8 / 16$
1	0	0	1	$V_{TH} \times 9 / 16$
1	0	1	0	$V_{TH} \times 10 / 16$
1	0	1	1	$V_{TH} \times 11 / 16$
1	1	0	0	$V_{TH} \times 12 / 16$
1	1	0	1	$V_{TH} \times 13 / 16$
1	1	1	0	$V_{TH} \times 14 / 16$
1	1	1	1	$V_{TH} \times 15 / 16$

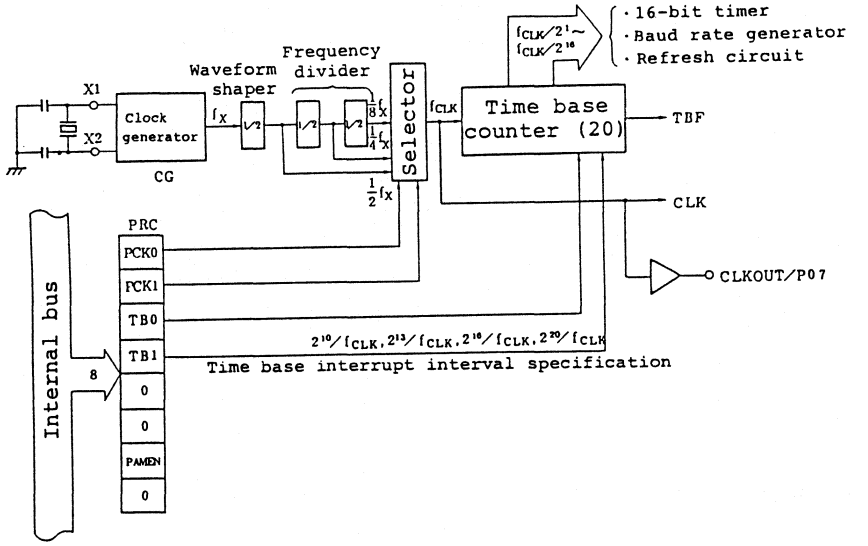
Fig. 7-12 Port T Mode Register (PMT) Format

CHAPTER 8 CLOCK GENERATOR

The clock generator supplies various clocks to the CPU and peripheral hardware and controls the CPU operation mode.

8.1 Clock Generator Organization

The clock generator is organized as shown in Fig. 8-1.



f_X : Oscillation frequency
f_{CLK} : System clock frequency
PRC : Processor control register
TBF : Time base interrupt request flag
CLKOUT : System clock output pin

Fig. 8-1 Clock Generater Block Diagram

The clock generator generates a clock using a crystal oscillator or ceramic oscillator connected across the X1 and X2 pins. The output of the clock generator is shaped by the waveform shaper (divided by 2), and then further divided by the two frequency dividers, each of which halves the input frequency. The system clock (CLK) is selected from these divided frequencies by the selector.

The dividing ratio of CLK can be selected from 1/2, 1/4, and 1/8 the oscillation frequency by the specification of bits 0 and 1 (PCK0, PCK1) in the processor control register (PRC).

When the system voltage drops in the battery operated system, the operating time can be extended by selecting the lower system clock frequency (f_{CLK}).

8.2 Processor Control Register (PRC)

The PRC register is an 8-bit register that specifically controls the items related to the CPU. The internal system controls items such as the CPU operation clock, time base interrupt interval, built-in RAM reference enable operation, etc.

The PRC register can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, this register is initialized to 4EH.

The PCK0 and PCK1 bits specify the division ratio of the system clock. The oscillation frequency of the clock generator is divided by the value specified by the PCK0 and PCK1 bits and used as the system clock (CLK).

The TB0 and TB1 bits specify the time base interrupt interval. Four different long intervals can be selected by the TB0 and TB1 bits.

The RAMEN bit controls whether or not the built-in RAM memory reference operation is enabled. When disabled (when the RAMEN bit is set to 0), the built-in RAM address is not evaluated, and the external memory is always accessed.

When the RAM is referenced as a register, the built-in RAM is always accessed.

Symbol	7	6	5	4	3	2	1	0	Address
PRC	0	RAMEM	0	0	TB1	TB0	PCK1	PCK0	xxFEBH

PCK1	PCK0	System clock (CLK) division ratio specification
0	0	f_{CLK} = Oscillation frequency (f_X) x 1/2
0	1	f_{CLK} = Oscillation frequency (fX) x 1/4
1	0	f_{CLK} = Oscillation frequency (fX) x 1/6
1	1	This combination is not allowed

TB1	TB0	Time base interrupt interval specification
0	0	Generates an interrupt with 2^{10} /f_{CLK} interval
0	1	Generates an interrupt with 2^{13} /f_{CLK} interval
1	0	Generates an interrupt with 2^{16} /f_{CLK} interval
1	1	Generates an interrupt with 2^{20} /f_{CLK} interval

RAMEN	Bilt-in RAM enable specification
0	Disables the built-in RAM
1	Enables the built-in RAM

Fig. 8-2 Processor Control Register (PRC) Format

CHAPTER 9 TIMER UNIT

The timer unit of the uPD70322/70320 can be used as an interval timer, one-shot timer, or a square-wave generator.

9.1 Organization and Operation of Timer Unit

The timer unit consists of two 16-bit timer registers, two 16-bit modulo/timer registers, and an 8-bit timer control register. The following describes the organization and operation for each operating mode of the timer unit.

(1) Interval timer mode

When the timer unit is set to the interval timer mode, two timers, timer 0 and timer 1, can be used as shown in Fig. 9-1.

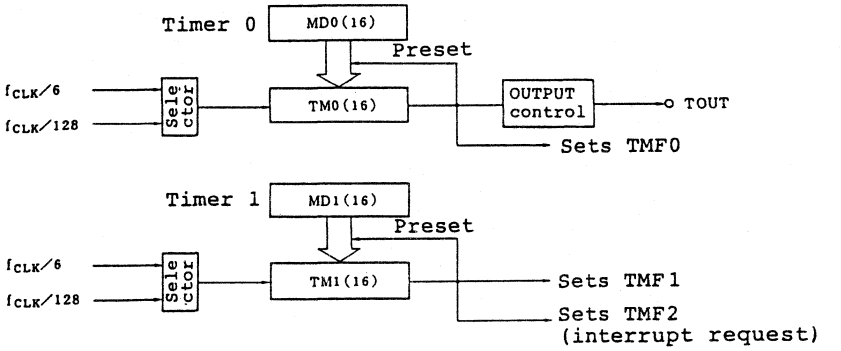


Fig. 9-1 Timer Unit Organization in Interval Timer Mode

When the timer unit is set to the interval timer mode by using the timer control register (TMC0) and the TS0 bit is set to 1, the MD0 register value is set to the TM0 register and the timer unit starts to count down the clock specified by the TCLK0 bit. When an underflow occurs during the count down operation,

the MD0 register value is again set to the TM0 register and the count down operation is repeated.

Count down operation is performed for the timer 1 register in the same way as the timer 0 register.

(2) One-shot timer mode

When the timer unit is set to the one-shot timer mode, timer 0 is used as shown in Fig. 9-2. However, timer 1 can be simultaneously operated as the interval timer.

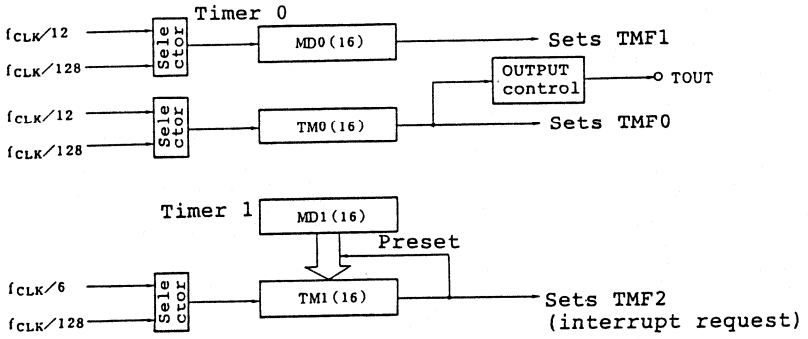


Fig. 9-2 Timer Unit Organization in One-Shot Timer Mode

When the timer unit is set to the one-shot timer mode by using the timer control register (TMC0) and the TS0/MS0 bit is set to 1, the TM0/MD0 register counts down the clock specified by the TCLK/MCLK bit. When an underflow occurs while counting down, the counting operation stops. When it stops, the TM0/MD0 register retains 0000H.

9.2 Timer Control Register (TMC0, TMC1)

The TMC0/TMC1 register is an 8-bit register that controls the operations of the TM0/TM1 and MD0/MD1 registers.

The TMC0/TMC1 register can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, this register is initialized to 00H. The formats of the TMC0 and TMC1 registers differ as shown in the figure below.

(Interval timer mode, one-shot timer mode)

Symbol	7	6	5	4	3	2	1	0	Address
TMC0	TS0	TCLK0	MS0	MCLK0	ENT0	ALV	MOD1	MOD0	xxF90H

(Interval timer mode)

	7	6	5	4	3	2	1	0	
TMC1	TS1	TCLK1	0	0	0	0	0	0	xxF91H

Bits 0 and 1 (MOD0, MOD1) of the TMC0 and TMC1 registers specify the operation of timer 0 consisting of TM0 and MD0 and timer 1 consisting of TM1 and MD1.

MOD0 , MOD1 These bits specify the operation mode of timer 0 and timer 1.

When both the MOD0 and MOD1 bits are set to 1, the timer unit operates in the interval timer operation mode. When the MOD0 bit is set to 1 and the MOD1 bit is reset to 0, the timer unit operates in the one-shot timer mode. In the interval timer operation mode, the TM0 and TM1 function as the timer registers that count down the specified value, and the MD1 and MD2 registers function as the modulo registers that retain the specified interval value. In the one-shot timer operation mode, both the TM0 and MD0 function as the timer registers that count down the specified value. However, for timer 1, bits 0 and 1 of the TMC1 register are fixed to 0s so that it can operate only as the interval timer.

Therefore, timer 0 can operate as a 16-bit interval timer or 16-bit one-shot timer consisting of the TM0 and MD0 registers by the TMC0 register setting. Timer 1 can operate as a 16-bit interval timer consisting of the TM1 and MD1 registers.

In addition, timer 0 can output square-wave signals to the TOUT pin. Square-wavesignal output to the TOUT pin is controlled by the TMC0 register. However, the TOUT pin shares P15 pin; therefore, the port must be set to the port mode by setting bit 5 (PMC15) of the port 1 mode control register to 1.

ALV This bit specifies the active level of the TOUT pin.

When the ENTO bit is reset to 0, the active level of the TOUT pin is low level active if the ALV bit is reset to 0, and high level active if the ALV bit is set to 1.

ENTO This bit specifies the operation of outputting a square-wave to the TOUT pin.

When this bit is reset to 0, the active level of the TOUT pin is determined by the ALV bit specification. When this bit is set to 1, the timing of the TOUT pin level is inverted at which the timer unit interrupt request flag (TMF0) is set to 1.

The following describes the functions of other bits in the TMC0 and TMC1 registers for each mode.

(1) Interval timer mode (MOD0=0, MOD1=0) However, timer 0, 1 (n=0, 1)

TCLKn This bit specifies the count clock of the TMn register.

Table 9-1 indicates the reference values when the system clock frequency (f_{CLK}) is 5MHz.

TSn This bit controls the operation of timer n.

When the TSn bit is set to 1, the MDn register value is set to the TMn register and the TMn register starts counting down. When the TSn bit is cleared to 0, the TMn register stops counting down as it retains the value ofthe TMn and MDn registers.

If an underflow occurs or the TSn bit is set to 1 during the count down operation, the MDn register value is again set to the TMn register and the down count operation is resumed.

Table 9-1 Count Time (n=0, 1) of Timer Registration (TMn) in Interval Timer Mode

f_{CLK}=5MHz (=1/2fx:fx=10MHz)

TCLKn	Count clock	Resolution	Full count
0	$f_{CLK}/6$	1.2us	78.6ms
1	$f_{CLK}/128$	25.6us	1.7s

(2) One-shot timer mode (MOD0=1, MOD1=0) Timer 0 only

TCLK0 This bit specifies the count clock of the TM0 register.

Table 9-2 indicates the reference values when the system clock frequency (f_{CLK}) is 5MHz.

TS0 This bit controls the operation of the TM0 register.

When the TS0 bit is set to 1, count down is started from the value currently retained in the TM0 register. When an underflow occurs, the TS0 bit is cleared to 0 and the count down operation is stopped. When the TS0 bit is cleared, the down count operation is stopped and the value of the TM0 register is retained.

MCLK0 This bit specifies the count clock of the MD0 register.

Table 9-2 indicates the reference values when the system clock (f_{CLK}) is 5MHz. In the interval timer mode, the MCLK0 bit has no effect on the count operation.

MS0 This bit controls the operation of the MD0 register.

When the MS0 bit is set to 1, count down is started from the value currently retained in the MD0 register. When an underflow occurs, the MS0 bit is cleared to 0 and the down count operation is stopped. When the MS0 bit is cleared, the down count operation is stopped and the MD0 register value is retained.

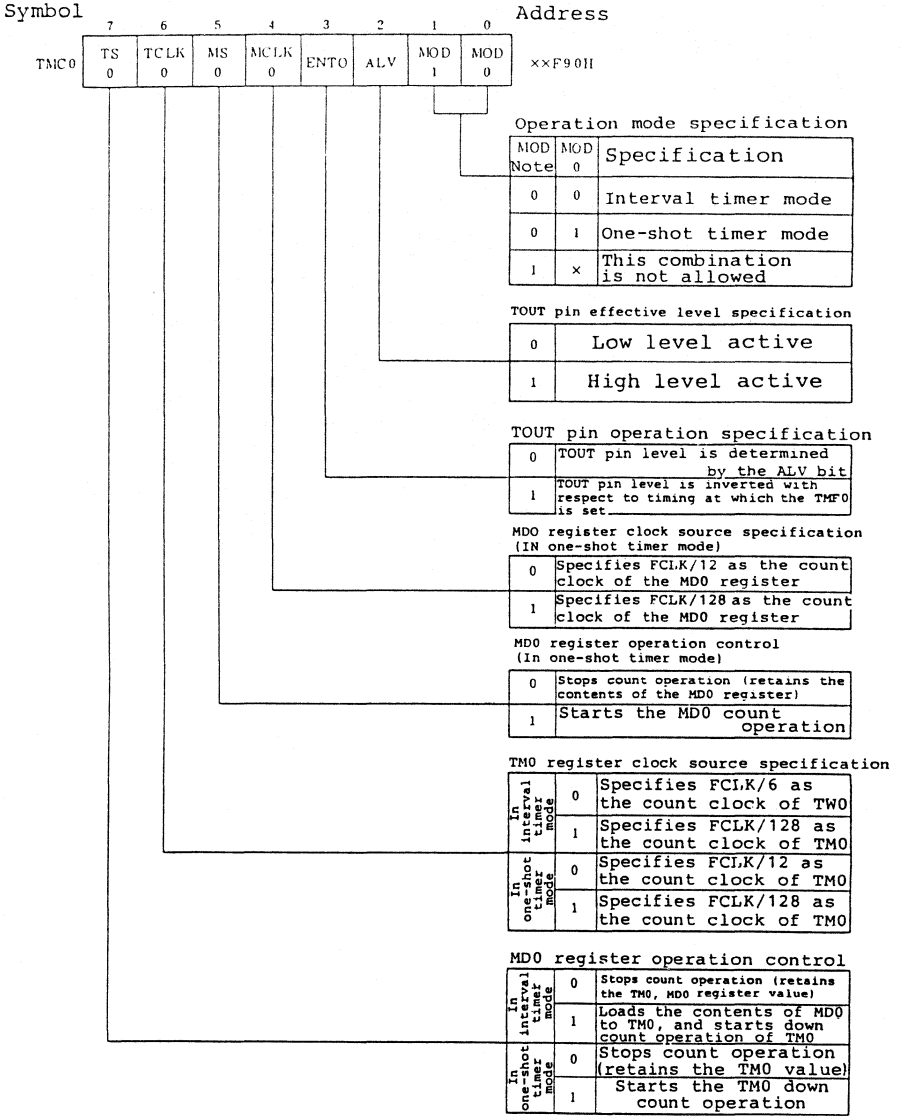
In the interval timer mode, the MS0 bit has no effect on the count operation.

Table 9-2 Count Timer (n=0, 1) of Timer Register 0 (TM0) and Mudulo Timer Register 0 (MD0) in one-Shot Timer Mode

f_{CLK}=5MHz (=1/2f_x:f_x=10MHz)

TCLK0/MCLK0	Count clock	Resolution	Full count
0	f_{CLK}/12	2.4us	157.3ms
1	f_{CLK}/128	25.6us	1.7s

Note: The count clock value of the TM0 register differs depending on the interval timer mode and the one-shot timer mode.



Note: 0 must be written to the MOD1 bit.

Fig. 9-3 Timer Control Register 0 (TMC0) Format

Symbol	7	6	5	4	3	2	1	0	Address
TMC1	TS 1	TCLK 1	0	0	0	0	0	0	xxF91H

TM1 register clock source specification

0	Specifies $f_{CLK}/6$ as the count clock of TM0
1	Specifies $f_{CLK}/128$ as the count clock of TM0

TM1 register operation countrol

0	Stops count operation (retains the TM1 and MD1 register values)
1	Loads the contents of MD1 to TM1, and starts count down operation of TM1

Fig. 9-4 Timer contorl Register 1 (TMC1) Format

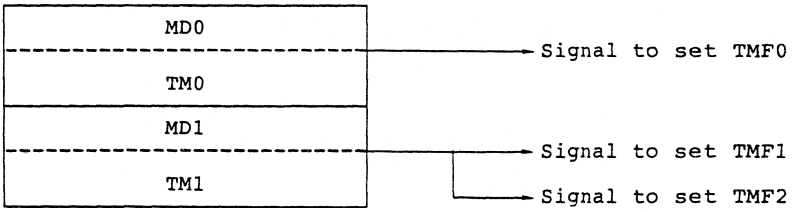
9.3 Timer Unit Interrupt Request

The timer unit can generate three different interrupt requests (TMF0-2). The conditions in which these interrupt requests are generated differ depending of which timer operation mode is specified.

When the timer unit is set to the interval mode, the TMF0 flag is set to 1 when an underflow occurs as the result of the count down operation of the TM0 register. The TMF1 and TMF2 flags are set when an underflow occurs as the result of the count down operation of the TM1 register (refer to Fig. 9-5a).

When the TM0 and TM1 registers are set to the one-shot timer mode, TMF0 is set when an underflow occurs as the result of the count down operation of the TM0 register, and the TMF1 flag is set to 1 when an underflow occurs as the result of the count down operation of the MD0 register. In this case, the TMF2 flag is set to 0 when an underflow occurs as the result of the count down operation of the TM1 register which functions as the interval timer in this mode.

a. When the TM0 and MD0 registers are set to the interval timer mode



b. When the TM0 and MD0 registers are set to the one-shot timer mode

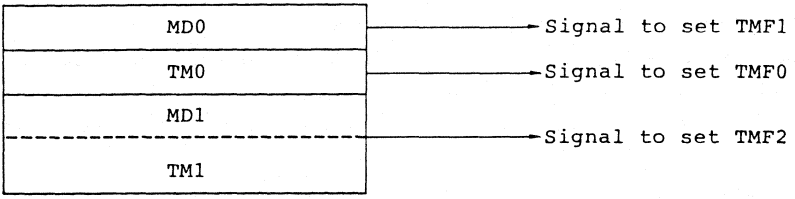


Fig. 9-5 Interrupt Requests From the Timer Unit

TMF0-2: Timer unit interrupt request flags 0 to 2

9.3.1 Timer unit interrupt request register (TMIC0, TMIC1, TMIC2)

The TMICn (n=0-2) register is an 8-bit register that controls the three interrupt requests generated from the timer unit. These interrupt requests form a group of which the priority is programmbly specified as a timer unit interrupt request. Within the group, priorities are fixed by means of hardware as follows:

TMF0 > TMF1 > TMF2

Symbol	7	6	5	4	3	2	1	0	Address
TMIC0	TMF0	TMMK0	MS/$\overline{INT}$	ENCS	0	PR2	PR1	PR0	xxF9CH
TMIC1	TMF1	TMMK1	MS/$\overline{INT}$	ENCS	0	1	1	1	xxF9DH
TMIC2	TMF2	TMMK2	MS/$\overline{INT}$	ENCS	0	1	1	1	xxF9EH

Fig. 9-6 Timer Unit Interrupt Request Control Register (TMIC0, TMIC1, TMIC2) Format

Note: Bits 2 to 0 of the TMIC1 and TMIC2 registers are set to 1 by means of hardware. However, the bit field consisting of bits 2 to 0 (PR2-0) specifies the

priority order of the interrupt request per group thus forming a group which includes the TMIC0.
The interrupt request priority order of the TMIC1 and TMIC2 registers are determined in accordance with the PR2 to PR0 bits of the TMIC0 register.

The function of each bit of the TMICn register is explained in 3.7.

The TMICn register can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, the TMICn register is initialized to 47H.

9.3.2 Timer unit macro service control register (TMMS0, TMMS1, TMMS2)

This is an 8-bit register that controls the macro service started by the three different interrupt requests generated from the timer unit.

The TMMS0 register controls the macro service started by the TMF0 flag.

The TMMS1 and TMMS2 registers control the macro services started by the TMF1 and TMF2 flags, respectively.

The TMMSn (n=0-2) register can be read/written to per 8 bits or 1 bit by accessing the memory.

Symbol	7	6	5	4	3	2	1	0	Address
TMMS0									xxF94H
TMMS1	MSM 2	MSM 1	MSM 0	DIR	0	CH 2	CH 1	CH 0	xxF95H
TMMS2									xxF96H

Fig. 9-7 Timer Unit Macro Service Register (TMMS0, TMMS1, TMMS2) Format

The function of each bit of the TMMSn register is explained in 4.4.3.

CHAPTER 10 TIME BASE COUNTER

The uPD70322/70320 has a function which can be used as a long-interval timer for clock operation.

10.1 Organization of Timer Base Counter

Fig. 10-1 shows the organization of the time base counter.

The time base counter consists of frequency divider which divides the system clock (CLK) into 20 different frequencies. The lower side of the divider output taps are used for the 16-bit timer count clock, input clock for the baud rate generator, refresh timer generation, and refresh address generation. Of the 20 output taps, output taps 10, 13, 16, and 20 are used for the time base interrupt request.

The time base counter is cleared to 00000H only by inputting $\overline{\text{RESET}}$; following this, it is continuously incremented.

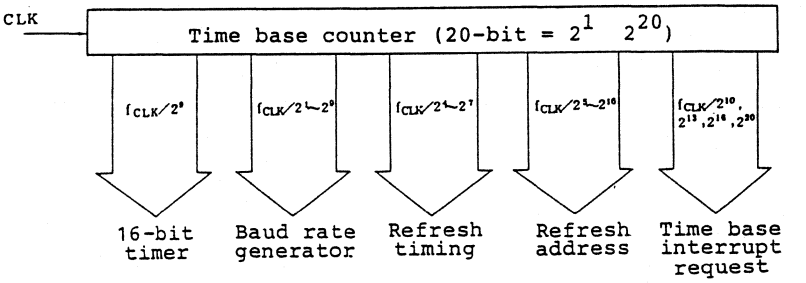


Fig. 10-1 Organization of the Timer Base Counter

10.2 Specification of the Time Base Interval

The interval time for the interupt request generated from the time base counter can be selected from four different intervals by setting bits 2 and 3 (TB0 and TB1) of the processor control register (PRC).

Symbol	7	6	5	4	3	2	1	0	Address
PRC	0	RAMEM	0	0	TB1	TB0	PCK1	PCK0	xxFEBH

f_{CLK} = 5MHz (=1/2 f_X: f_X=10MHz)

TB1	TB0	Interval time
0	0	$2^{10}/f_{CLK}$ (205 us)
0	1	$2^{13}/f_{CLK}$ (1.64 ms)
1	0	$2^{16}/f_{CLK}$ (13.1 ms)
1	1	$2^{20}/f_{CLK}$ (210 ms)

Note: The time from when the TB0 and TB1 bits are set to when the first interrupt request is generated is undefined.

Fig. 10-2 Processor Control Register (PRC) Interval Timer Mode

10.3 Time Base Interrupt Request Control Register (TBIC)

The TBIC register is an 8-bit register that controls the masking of the interrupt generated from the time base counter.

The TBIC register can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, the TBIC register is initialized to 47H.

Symbol	7	6	5	4	3	2	1	0	Address
TBIC	TBF	TBMK	0	0	0	1	1	1	xxFECH

Fig. 10-3 Time Base Interrupt Request Control Register (TBIC) Format

This interrupt request is generated when the interrupt request flag (TBF) is set to 1. The interrupt request flag (TBF) is set when the output tap of the time base counter specified by the processor control register (PRC) becomes high. Bits 4 and 5 of the TBIC register are fixed to 0s and have no register bank switching function or macro service function for the time base counter interrupt operation. Additionally, bits 0 to 2 of the TBIC register are fixed to 1s; therefore, the priority order level of the time base interrupt (INTTB) is fixed to 7. In addition, among interrupt requests whose priority level is 7 the priority order level is fixed to the lowest priority. However, it can receive multiple processing control.

CHAPTER 11 SERIAL INTERFACE

11.1 Organization of Serial Interface

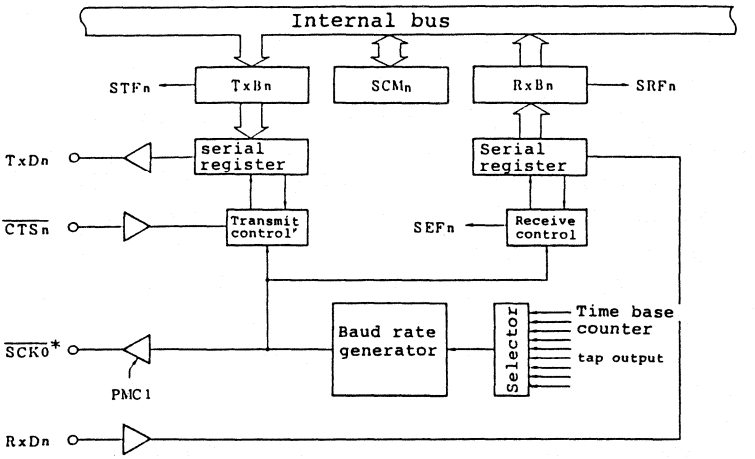
The uPD70322/70320 has two serial interface channels each of which has a built-in baud rate generator.

The serial interface can operate in two different modes, an asynchronous mode and an I/O interface mode. In the asynchronous mode, the start/stop bit transmit/receive method is employed so that the bit synchronization and the character synchronization are obtained by the start bit. In the I/O interface mode, in the same way as the serial data transfer performed in the uCOM-87 family, data is transferred in synchronization with the controlled serial clock.

Fig. 11-1 shows the organization of the serial interface in both the asynchronous mode and I/O interface mode.

The serial interface block consists of the serial data input (RxDn) pin, serial data output (TxDn) pin, serial clock output ($\overline{SCK0}$) pin, clear-to-send signal input ($\overline{CTSn}$) pin, transfer control section, 8-bit transmit serial register, 8-bit receive register, transmit buffer (TxBn), receive buffer (RxBn), and baud rate generator. A serial register and buffer are independently provided for transmit and receive operations so that transmit and receive operations can be performed independently (full-duplexed data communications are possible). Since the $\overline{CTSn}$ pin functions as the receive clock input/output pin in the I/O interface mode, full-duplexed serial data communications are also possible in the I/O interface mode.

(a) In asymchronous mode (n=0, 1)



* For Ch. 0 only, $\overline{SCK}$ outputs high level when set in asynchronous mode.

(b) In I/O interface mode (chaanel 0 only)

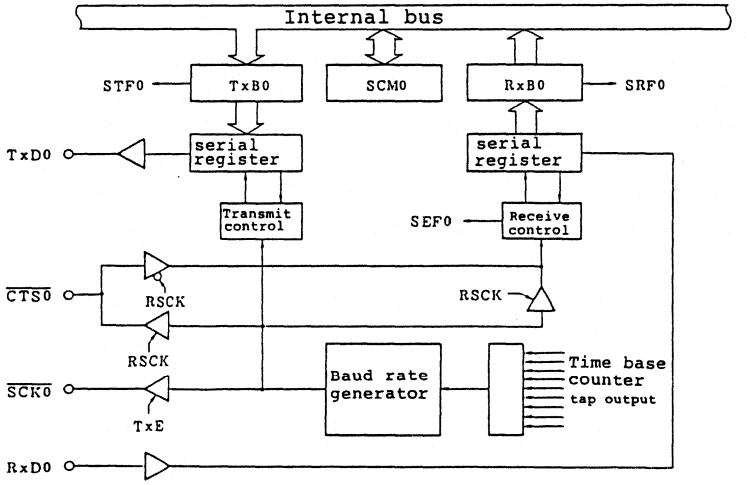


Fig. 11-1 Serial Interface Function

11.2 Asynchronous Mode

In the asynchronous mode, the character length, number of stop bits, parity enable/disable, and even/odd parity can be specified by the serial mode register (SCMn).

(1) Transmit

The transmit operation is enabled (see Note) when bit 7 (TxRDY) of the serial mode register (SCMn) is set and the $\overline{CTSn}$ pin is active (0).

The transmit operation can be started in the following three ways.

(a) When a transmit operation is enabled and the transmit buffer (TxB) is empty, a transmit completion interrupt is generated and the transmit data is written to the transmit buffer during the interrupt processing operation.

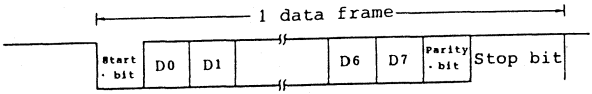
(b) When the transmit data is transferred to the transmit buffer and the transmit operation is enabled, the transmit data is output in succession to the preceding transmit data.

(c) If the transmit data is transferred to the transmit buffer when the transmit operation is disabled, the transmit data stored in the transmit buffer is output when the transmit operation is enabled.

Note: There is no constraint on the procedure to enable the transmit operation. The TxRDY pin can be set to 1 before setting the $\overline{CTSn}$ pin to active, or the $\overline{CTSn}$ pin can be set to active before setting the TxRDY pin to 1.

As shown in the figure below, in the transmit data format, one data frame consists a start bit, character bits, a parity bit, and a stop bit. Transmit data is output from the TxDn pin with the least significant bit (LSB) first.

The TxDn pin is in the mark state (1) when the transmit operation is disabled or when there is no data to be transmited in the serial register.



Stop bit .....1 bit
Character bits ....7/8 bits
Parity bit Odd/even/0/no-parity
Stop bit .....1/2 bits

A transmit completion interrupt request is immediately generated when the transmit buffer (TxBn) becomes empty.

When $\overline{\text{RESET}}$ is input, the transmit buffer (TxBn) is emptied. If a transmit operation is enabled, a transmit completion interrupt request is generated. A transmit completion interrupt request is also generated when the transmit operation is started and the transmit buffer becomes empty after the transmit data is transferred from the transmit buffer to the shift register.

The transmit data can be transferred to the transmit buffer each time the transmit completion interrupt request is generated so that the data can be continuously transmitted without inserting a mark state (1).

If the transmit operation is disabled when the transmit operation is in progress, the transmit operation is continued to the end of the current frame. In this case, the data stored in the transmit buffer is not transferred to the shift register, and the contents of the transmit buffer are retained as is. The contents of the transmit buffer are transferred to the shift register when the transmit operation is enabled. The transmit completion interrupt request is simultaneously generated when the transmit operation is started.

(2) Receive

A receive operation is enabled when bit 6 (RxE) of the serial mode register (SCM2) is set to 1. When receive operation is disabled, the receive hardware continues to standby in the initialized condition.

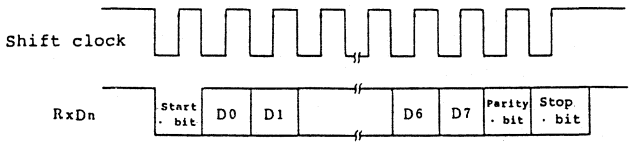
The RxDn pin input is sampled by the clock input to the baud rate generator. When the falling edge of the signal input to the RxDn pin is detected, the receive operation is started and the receive baud rate generator starts counting. When the low level of the RxDn pin is detected at the first timing signal from the receive baud rate generator, this low level is recognized as the start bit so that further receive operation is continued.

If a high level is detected at the first timing signal, it is not recognized as the start bit so that the baud rate generator is initialized and terminates its operation.

As shown in the figure below, the sampling of the receive data is performed in synchronization with the rising edge of the shift clock after the start bit is detected.

Receive Data Sampling Timing

Receive Data Sampling Timing



When the data whose character length is specified by bit 3 (CL) of the serial mode register is received, the data in the shift register is transferred to the receive buffer (RxBn) and the receive completion interrupt request is generated.

The receive error flag is set and the receive error interrupt request is generated when any one of the following

conditions occurs in the receive operation; a parity error occurs when an odd/even parity check is performed (when the PRTY bit (see Note) is set to 1); the stop bit is at a low level (framing error); the next daum is sent to the receive buffer when the receive buffer is full (overrun error) (refer to 11.6).

Note: Bit 5 of the serial mode register is the PRTY bit.

11.3 I/O Interface Mode

The I/O interface mode of the uPD70322/70320 is same as the serial interface mode of the uCOM-87 family. This mode is useful when externally expanding an input/output device or connecting the I/O controller (A/D converter, LCD controller, etc.).

For the I/O interface mode, the character length is fixed to 8 bits, no parity bit is provided, and the data is transferred from the most significant bit (MSB).

The I/O interface mode is available only on channel 0.

(1) Transmit

Transmit operation is enabled when bit 7 (TxE) of the serial mode register (SCMn) is set to 1. In the I/O interface mode, the $\overline{SCK0}$ pin functions as the transmit clock output pin.

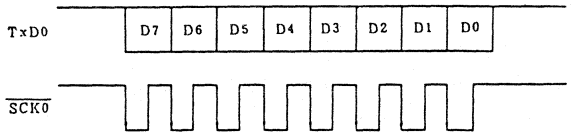
In the same way as the asynchronous mode, transmit operation can be started in the following three ways:

(a) The transmit operation is enabled when the transmit buffer (TxB0) is empty, the transmit completion interrupt is generated and the transmit data is written to the transmit buffer during interrupt processing.

(b) The transmit data is transferred to the transmit buffer (TxB0)when the transmit operation is enabled and the transmit data is then output in succession to the preceding transmit data.

(c) If the transmit data is transferred to the transmit buffer (TxB0) when the transmit operation is diabled, the transmit data stored in the transmit buffer (TxB0) is later output when the transmit operation is enabled.

The transmit data format in this mode is as shown in the figure below. The character length is fixed at 8 bits. The data is output from the most significant bit (MSB).



The transmit completion interrupt request is immediately generated when the transmit buffer (TxB0) becomes empty.

When $\overline{\text{RESET}}$ is input, the transmit buffer (TxB0) is emptied. If a transmit operation is enabled at this time, a transmit completion interrupt request is generated. A transmit completion interrupt request is also generated when the transmit operation is started and the transmit buffer (TxB0) becomes empty after the transmit data is transferred from the transmit buffer (TxB0) to the shift register.

(2) Receive

The receive operation is enabled when bit 6 (RxE) of the serial mode register (SCM0) is set to 1. The receive data is input to the serial register at the rising edge of the receive clock. When the serial register has received 8 bits of data, the data is transferred from the serial register to the receive buffer (RxB0), and a receive completion interrupt request is generated.

In the I/O interface mode, either the external receive clock or the internal receive clock can be selected as the

receive clock by specifying bit 2 (RSCK) of the serial mode register (SCM0).

The $\overline{CTS0}$ pin functions as the receive clock input/output pin in the I/O interface mode.

The receive error flag is set and the receive error interrupt request is generated when the next data is sent to the receive buffer when the receive buffer is full (overrun error).

11.4 Serial Mode Register (SCM0, SCM1)

The SCMn (n=0, 1) register is an 8-bit register that specifies the transfer mode of the serial interface. The SCM0 and SCM1 registers are assigned to channel 0 and channel 2, respectively. The functions of bits 2 to 7 of the SCMn register differ depending on the settings of bits 1 and 0 (MD1 and MD0) in the SCMn register.

	Symbol	7	6	5	4	3	2	1	0	Address
(Channel 0)	SCM0							MD 0	MD 0	xxF68H
(Channel 1)	SCM1									xxF78H

MD1, MD0=0,1 (Acynchronous mode)

Symbol	7	6	5	4	3	2	1	0
SCM0								
SCM1	TxRDY	RxE	PRTY 1	PERY 0	CL	SL	0	1

MD1, MD0=0,1 (Acynchronous mode)

Symbol	7	6	5	4	3	2	1	0
SCM0								
SCM1	TxE	RxE	0	0	TSK	RSCK	0	0

The bit field consisting of the MD1 and MD0 bits specifies the transfer mode of the serial interface. When the MD1 is set to 0 and MD0 to 1, the serial interface is set to the asynchronous mode. When the MD1 and MD0 bits are both set to 0, the serial interface is set to the I/O interface mode; however, only the SCM0 can be set for the I/O interface mode.

The SCMn register can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, the SCMn register is cleared to 00H.

(1) In asynchronous mode

RxE This bit enables/disables receive operation.

If receive operation is disabled (RxE=0) during reception, receive processing is interrupted, and a receive completion interrupt is not generated.

SL This bit specifies the number of stop bits.

When this bit is reset to 0, the number of stop bits is 1. When thisbit is set to 1, the number of stop bits is 2.

CL This bit specifies the character length.

When this bit is reset to 0, the character length is 7 bits. When this bit is set to 1, the character length is 8 bits.

PRTY0 , PRITY1 These bits specify whether or not parity is added.

The PRTY0 and PRTY1 bits specify non-parity, odd parity, even parity, or 0 parity. When 0 parity is specified, the parity bit is set to 0 during transmission and is ignored during reception.

TxRDY This bit enables/disables transmit operation.

Transmit operation is enabled when the $\overline{\text{CTSn}}$ pin is at a low level and the TxRDY pin is set to 1.

Symbol	7	6	5	4	3	2	1	0	Address
SCM0	TxRDY	RxE	PRTY1	PRTY0	CL	SL	0	1	xxF68H
SCM1									xxF78H

Stop bit length specification

0	1 stop bit
1	2 stop bits

Character length specification

0	7 bits
1	8 bits

Parity specification

PRTY 1	PRTY 0	Parite specification
		No parity
0	0	No parity
0	1	0 parity Note
1	0	Odd parity
1	1	Even parity

Note: When 0 parity is specified, the parity bit is set to 0 during transmission and is ignored during reception.

Receive operation enable/disable control

0	Disables receive operation
1	Enables receive operation

Transmit enable/disable control

0	Disabled transmit operation
1	Enables transmit when the $\overline{\text{CTSn}}$ pin is at a low level

Fig. 11-2 Serial Mode Register (SCM0, SCM1) Format in Asynchronous Mode

(2) In I/O interface mode

RSCK This bit specifies the serial receive clock source.

When the RSCK bit is reset to 0, the external receive clock is used for receive operation. When the RSCK bit is set to 1, the internal receive clock is used for receive operation. The $\overline{CTS0}$ pin is used to input/output the receive clock.

TSK This bit is the receive the clock output trigger bit.

This bit is effective when the RSCK bit is set to 1. Eight receive clocks are output from the $\overline{CTS0}$ pin when 1 is written to the TSK bit.

When the serial clock is output, this bit is automatically reset to 0.

RxE This bit enables/disables receive operation.

Receive operation is enabled when the RxE bit is set to 1. When this bit is reset to 0, receive operation is disabled. If receive operation is disabled during reception, receive operation is terminated at that time, and no receive completion interrupt request is generated.

TxE This bit enables/disables transmit operation.

Transmit operation is enabled when this bit is set to 1. When this bit is reset to 0, transmit operation is disabled.

If the transmit data is written to the transmit buffer while the transmit operation is enabled (TxE=1), the serial transmit data is output after the preceding transmission has been completed or is output immediately if there is no preceding transmission operation. If the transmit data is written to the transmit buffer while transmit operation is disabled (TxE=0), the data in the transmit buffer is retained as is. Afterwards, the data retained in the transmit buffer is transmited when transmit is enabled.

Even if the TxE bit is reset to 0 (disabling transmit operation), the transmit operation is performed to the end. However, the next data already stored in the transmit buffer at the time transmit is disabled is retained in as is in the transmit buffer for the next transmission opportunity.

Symbol	7	6	5	4	3	2	1	0	Address
SCM0	TxE	RxE	0	0	TSK	RSCK	0	0	xxF68H

Recive clock specification

0	Uses the external clock for operation ($\overline{CTS0}$ pin is in input mode)
1	Uses the internal clock for operation ($\overline{CTS0}$ pin is in output mode)

Serial interface control

0	——
1	Receive operation enable/disable control

Receive operation enable/disable cotrol

0	Disables recive operation
1	Enables receive operation

Transmit enable/disable control

0	Disables transmit operation
1	Enables transmit operation

Fig. 11-3 Serial Mode Register (SCM0) Format in I/O Interface Mode

11.5 Baud Rate Generator

This baud rate generator is an 8-bit timer specifically provided for the serial interface, which generates the transmit/receive shift clock. Each channel has baud rate generators for transmit and receive operations. The same baud rate is used for transmit and receive operations. The baud rate is determined by setting a value to the baud rate generator (BRGn). However, the maximum baud rate is 750kbps. Data transmit/receive operation with 750kbps or higher can be realized by inserting one or more idle states between data.

The input clock of the baud rate generator is specified by selecting the time base counter (refer to 10.1) output by setting the PRS3 to PRS0 bits of the serial control register (SCCn). The output signal of the baud rate generator is used as the shift clock of the serial interface. To select the baud rate, parameters are set in the baud rate in such a manner that the following expression is satisfied.

$$B \times G = 10^6 \times \frac{fCLK}{2^{n+1}}$$

Where, each parameter is defined as follows:

- B: Transfer baud rate bps
  B = 110, 150, ..., 9600, 19200, ...
- G: The value ($2 \leq G \leq 255$) set to the baud rate generator (BRGn)
- n: A value ($0 \leq n \leq 8$) specified by the serial control register to determine the input clock of the baud rate generator.
- CLK: System clock frequency (MHz)

Table 11-1 indicates parameters for setting the baud rate generator for each standard transfer baud rate when a 10MHz crystal is connected.

Table 11-1 Values for Setting Baud Rate Generator (reference)

Transfer baud rate	n	Value set to BRGn register	Error (%)
110	7	178	0.25
150	7	130	0.16
300	6	130	0.16
600	5	130	0.16
1200	4	130	0.16
2400	3	130	0.16
4800	2	130	0.16
9600	1	130	0.16
19200	0	130	0.16
38400	0	65	0.16
1.25M	0	2	0

n: Number to specify the input clock of the baud rate generator.

11.5.1 Serial control register (SCC0, SCC1)

The SCCn register (n=0, 1) is an 8-bit register that controls the transfer rate of the serial interface.

The SCCn register can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{\text{RESET}}$ is input, the SCCn register is initialized to 00H.

The bit field consisting of the PR3 to PR0 bits specifies the time base counter output tap whose output is input to the baud rate generator.

Symbol	7	6	5	4	3	2	1	0	Address
(Channel 0) SCC0	0	0	0	0	PRS 3	PRS 2	PRS 1	PRS 0	xxF69H
(Channel 1) SCC1									xxF79H

PRS 3	PRS 2	PRS 1	PRS 0	Input clcok for the baud rate generator
0	0	0	0	f_{CLK}
0	0	0	1	$f_{CLK}/2$
0	0	1	0	$f_{CLK}/4$
0	0	1	1	$f_{CLK}/8$
0	1	0	1	$f_{CLK}/32$
0	1	0	0	$f_{CLK}/16$
0	1	1	0	$f_{CLK}/64$
0	1	1	1	$f_{CLK}/128$
1	0	0	0	$f_{CLK}/256$

f_{CLK}: System clock frequency

Fig. 11-4 Serial Control Register (SCC0, SCC1) Format

11.6 Serial Error Processing

The following three different errors which occur in serial receive operation can be detected:

(a) Parity error (asynchronous mode)
The transmit parity and receive parity do not coincide.

(b) Framing error (asynchronous mode)
The stop bit is not detected.

(c) Overrun error (asynchronous mode, I/O interface mode)
Before receiving the previous data from the RxB, the next data is received.

11.6.1 Serial error register (SCE0, SCE1)

This is an 8-bit register that indicates the status of the three error flags corresponding to each receive error. This register is provided to each channel (channels 0 and 1).

The SCEn (n=0, 1) is a read only register and can be read per 8 bits by accessing the memory.

When $\overline{\text{RESET}}$ is input, the SCEn register is initialized to 00H.

ERPn Parity error flag

The ERP flag is set to 1 when the transmit parity and the receive parity do not coincide, and is reset to 0 when the receive data is read out from the receive buffer.

ERFn Framing error flag

The ERF flag is set to 1 when the stop bit is not detected, and is reset to 0 when the receive data is read out from the receive buffer.

EROn Overrun error flag

The ERO flag is set to 1 when the next data receive operation is completed before receiving the previous data from the receive buffer (RxB), and is reset to 0 when the receive data is read from the receive buffer.

RxDn This is the RxB bit which checks the input status of the receive pin.

When $\overline{\text{RESET}}$ is input, the serial error register (SCEn) is initialized to 00H.

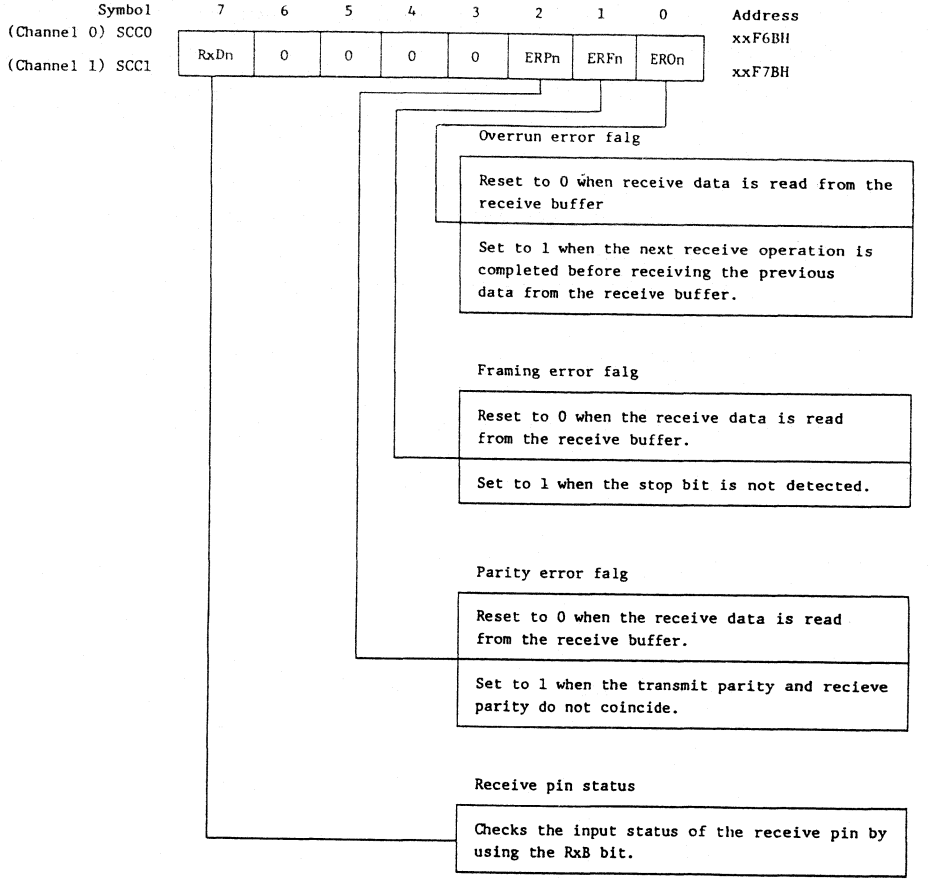


Fig. 11-5 Serial Error Register (SCE0, SCE1) Format ... n=0, 1

11.7 Break Detect Function

In the uPD70322/70320, a break condition in the line can be detected by means of software processing (asynchronous mode only). The following describes the procedure for detecting the break condition.

(1) Generation of the receive error interrupt request at the first framing error

The receive data is checked in the receive error processing routine to determine whether or not it is 00H. At the same time, the receive error flag is checked to determine whether or not the error is a framing error.

(2) Generation of the receive error interrupt request at the second framing error

If a break condition has occurred, a framing error is again generated.

The break condition in the line can be confirmed by checking that the receive data is again 00H, and that the data in 00H is continuously received with a framing error. Also, the pin status can be directly confirmed by using bit 7 (RxDn) of the serial error register (SCEn).

11.8 Serial Interface Interrupt Request

Three different receive error interrupt requests - a transmit complete interrupt request, a receive complete interrupt request, and a receive error interrupt request are generated from each serial interface channel.

11.8.1 Interrupt request control register (SEICn, SRICn, STICn) n=0, 1

Each of these registers controls the three interrupt requests generated from the serial interface, the receive error interrupt request (SEFn), the receive complete interrupt request (SRFn), and the transmit complete interrupt request (STFn).

The three interrupt request control registers form a group in which the interrupt priority can be specified. Within the group, priorities are fixed by means of hardware as follows:

SEFn > SRFn > STFn

Fig. 11-6 Interrupt Control Register
(SEICn, SRICn, STICn) Format ... n= 0, 1

Symbol	7	6	5	4	3	2	1	0	Address
SEIC0									xxF6CH
SEIC1	SEFn	SEMKn	MS/$\overline{INT}$	ENCS	0	PR2	PR1	PR0	xxF7CH
SRIC0									xxF6DH
SRIC1	SRFn	SRMKn	MS/$\overline{INT}$	ENCS	0	1	1	1	xxF7DH
STIC0									xxF6EH
STIC1	STFn	STMKn	MS/$\overline{INT}$	ENCS	0	1	1	1	xxF7EH

Note: Bits 2 to 0 of the SRICn and STICn registers are fixed to 1 by means of hardware. However, the bit field consisting of bits 2 to 0 (PR2-0) specifies the priority order of the interrupt request per group which forms a group with the SEICn.
The order of interrupt request priorities in the SRICn and STICn registers is determined according to the PR2 to PR0 bits of the SEICn register.

The SEFn, SRFn, and STFn bits are interrupt request flags which are set to 1 when a receive error is generated, receive operation is completed, and transmit operation is completed, respectively, and are reset to 0 when the interrupt request is accepted or by means of software.

The function of the other bit fields is explained in 4.7.1.

The SEICn, SRICn, and STICn registers can be read/written to per 8 bits or 1 bit by accessing the memory.

When $\overline{RESET}$ is input, the SEICn, SRICn, and STICn registers are initialized to 47H.

11.8.2 Macro service control register (SRMSn, STMSn) n=0, 1

The SRMSn register is an 8-bit register that specifies the macro service processing mode associated with the completion of the serial interface receive operation and the channel. The SRMSn and STMSn registers are provided for each serial interface channel.

The SRMSn and STMSn registers can be read/written to per 8 bits or 1 bit by accessing the memory.

The function of each bit of the macro service register is explained in 4.4.4.

Fig. 11-7 Macro Service Control Register (SRMSn, STMSn) Format ... n=0, 1

Symbol	7	6	5	4	3	2	1	0	Address
SRMS0	MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0	xxF65H
SRMS1									xxF75H
STMS0	MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0	xxF66H
STMS1									xxF76H

CHAPTER 12 STANDBY FUNCTION

The uPD70322/70320 has two standby modes which control the operation clock.

- HALT mode ... In this mode, the CPU operation clock is stopped. However, the status of the CPU is retained as well as all the data, and the contents of RAM, and the peripheral hardware continues to function.
  When combined with the normal operating mode, total power consumption of the system can be reduced by intermittent operation.
- STOP mode ... In this mode, the operation of the clock oscillator is stopped. In this mode, the data stored in the RAM and the output data on ports can be retained with reduced power consumption.

These modes are set by the HALT and STOP instructions, respectively.

12.1 Standby Control Register (STBC)

The STBC is an 8-bit register that contains the standby flag (SBF). The upper 7 bits of the STBC register are fixed to 0.

The SBF flag is used to determine recovery from the STOP condition.

The SBF can be set to 1 only by executing an instruction. This flag is cleared only when power (V_{DD}) is applied, and cannot be cleared by executing an instruction.

Therefore, by testing the SBF flag, it can be determined whether recovery has been made from a reset (SBF=0) or from the STOP mode (SBF=1).

The STBC register is not initialized when a RESET is input.

Note: The SBF flag must be set to 1 before initiating the STOP mode.

Symbol	7	6	5	4	3	2	1	0	Address
STBC	0	0	0	0	0	0	0	SBF	xxFE0H

Reset to 0 when V_{DD} is applied from 0V.

Set to 1 by STBC manipualtion instruction

Fig. 12-1 Standby Control Register (STBC) Format

12.2 HALT Mode

This mode is used to stop the operation clock of the CPU.

The system's overall power consumption can be reduced by setting the uPD70322/70320 to the HALT mode while the CPU is idle. The HALT mode can be initiated by executing the HALT instruction.

In the HALT mode, the CPU clock is stopped and program execution is halted. However, the contents of all registers and built-in RAM are retained. The status of each piece of hardware in the HALT mode will be as indicated in Table 12-2.

12.2.1 Releasing the HALT mode

The HALT mode can be released by the non-maskable interrupt request (NMI), unmasked maskable interrupt request, and $\overline{\text{RESET}}$ input (refer Fig. 12-2).

Macro service or DMA processing can be initiated from the HALT mode by the macro service request or DMA processing request (refer to Fig. 12-3). After macro service or DMA processing is completed, the HALT mode is resumed. However, the HALT mode is released depending on the condition of the macro service or DMA processing indicated in Table 12-1.

(1) Release by an interrupt request

(a) When the HALT mode is set during interrupt processing routine

The HALT mode is released when a maskable interrupt

request which is not masked having a higher priority than the currently processed interrupt or a nonmaskable interrupt is generated.

(b) In other than (a)

The HALT mode is released when either a non-maskable interrupt request or a maskable interrupt request which is not masked is generated, regardless of the priority order.

(2) Release by $\overline{\text{RESET}}$ input

The same as the normal reset operation.

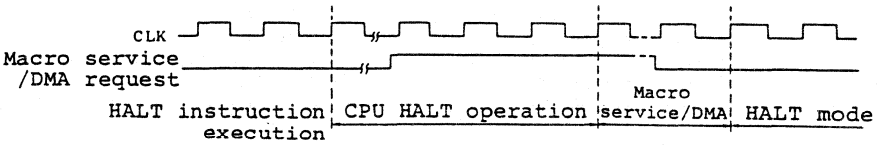


Fig. 12-2 Releasing the HALT Mode by Interrupt Request

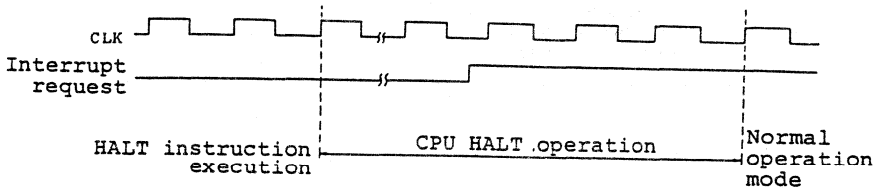


Fig. 12-3 Start of Macro Service/DMA in HAOT Mode

Table 12-1 Operation after the HALT Mode is Released by Interrupt

Release source	EI state	DI state
Non-maskable interrupt request	Branches to the vector address after the HALT mode is released.	Branches to the vector address after the HALT mode is released.
Maskable interrupt request	Branches to the vector address after the HALT mode is released.	Executes the next instruction after the HALT mode is released.
Macro service request	Branches to the vector address when the macro service is started and the macro service counter value becomes 0H. If the macro service counter does not become 0, the HALT mode is resumed.	When the macro service is started and the macro service counter value becomes 0H, the HALT mode is released and the next instructio is executed.
DMA request	Branches to the vector address when the DMA is started and the terminal count becomes 0H. If the terminal count does not become 0H, the HALT mode is resumed.	When the DMA is started and the terminal count value becomes 0H, the HALT mode is released and the next instruction is executed.

12.3 STOP Mode

In this mode, the operation of the clock oscillator is stopped. In this mode, the power consumption can be further reduced when the operation of the entire system is stopped. The STOP mode can be initiated by executing the STOP instruction. In the STOP mode, all clocks stop operating. The program execution is

stopped; however, the contents of all registers and built-in RAM are retained. The status of each piece of hardware will be as indicated in Table 12-2.

12.3.1 Releasing the STOP mode

The STOP mode can be released by the NMI interrupt request or by the RESET input.

(1) Releasing the STOP mode by the NMI interrupt (refer to Fig. 12-4)

When the effective edge is input to the NMI pin, the operation of the clock oscillator is resumed. The time base counter (TBC) also starts operating. The clock is not immediately supplied after the STOP mode is released. The clock is supplied after the oscillation stabilazation time measured by the TBC elapses. One half of the value specified by sign bits 2 and 3 (TB0, TB1) of the processor control register (PRC) will be the value of the oscillation stabilization time (however, the interrupt request generated from TBC is disabled during this period).

(2) Releasing the STOP mode by inputting $\overline{\text{RESET}}$

The same as the normal reset operation.

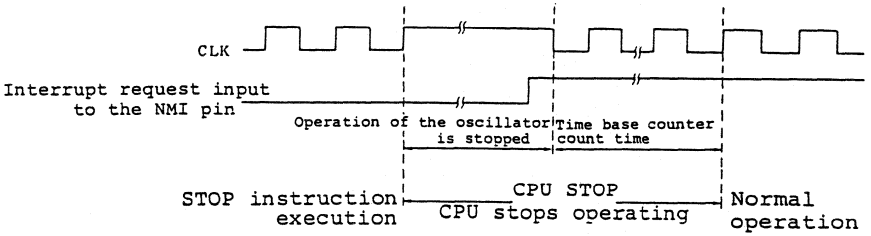


Fig. 12-4 Releasing the STOP Mode by NMI Pin Input

Table 12-2 HALT Mode/STOP Mode

Item		HALT mode	STOP mode
Oscillator		Operates	Stops
Internal system clock		Stops	
16-bit timer		Operates	
Time base counter			
HOLD circuit			
Serial interface			
Interrupt request controller			
DMA controller			
I/O line		Data is retained	Data is retained
Bus line	A0 - A19	Data is retained	Data is retained
	D0 - D7	High impedance	High impedance
R/$\overline{W}$ output		High level	High level
Refresh operation		Operates/stops	Stops
Data retaining operation		All internal data such as the CPU status, the contents of RAM, etc., are retained.	All internal data such as the CPU status, the contents of RAM, etc., are retained.
Release method		o Non-maskable interrupt request o Maskable interrupt request o $\overline{\text{RESET}}$ input o Macro service request (See note) o DMA (See note)	o Non-maskable interrupt request o $\overline{\text{RESET}}$ input

Note: The HALT mode is resumed following macro service or DMA processing.

CHAPTER 13 RESET FUNCTION

The system reset is effected when a low level is input to the $\overline{\text{RESET}}$ input pin and the status of each piece of hardware will be as indicated in Tables 13-1 and 13-2. When the $\overline{\text{RESET}}$ input pin level becomes high, the reset is released and the program execution is resumed. The contents of each register needs to be initialized in the program as necessary.

Table 13-1 Status of Each Piece of Hardware after Reset Operation

Hardware (Symbol)			Address (See note) (Lower 12 bit: xx□□□H)	Status after RESET
Program counter		PC		0000H
Program status word		PSW		F002H
Internal RAM	Data memory			Undefined
	General purpose register	AW, CW, DW, BW, SP, BP, IX, IY	EFEH - EF0H	Undefined
	Segment register	DS1, SS, DS0	EEEH, EEAH, EE8H	0000H
		PS	EECH	FFFFH
Port	Port register	P0, P1, P2	F01H, F09H, F11H	Undefined
		PT	F3BH	
	Port mode register	PM0, PM1, PM2	F01H, F09H, F11H	FFFFH
		PMT	P3BH	00H
	Port mode control register	PMC0, PMC1, PMC2	F02H, F0AH, F12H	00H
Timer unit	Timer register	TM0, TM1	F80H, F88H	Undefined
	Modulo/timer register	MD0, MD1	F82H, F8AH	Undefined
	Timer control register	TMC0, TMC1	F90H, F91H	00H
	Interrupt request control register	TMIC0 - TMIC2	F9CH - F9EH	47H
	Macro service control	TMMS0 - TMMS2	F94H - F96H	Undefined

Hardware (Symbol)			Address (See note) (Lower 12 bit: xx□□□H)	Status after RESET
DMA controller	DMA mode register	DMAM0, DMAM1	FA1H, FA3H	00H
	DMA control register	DMAC0, DMAC1	FA0H, FA2H	undefined
	Interrupt request control register	DIC0, DIC1	FACH, FADH	47H

Note: xx of the upper 8 bits of the address is the value specified by the IDB register.

Table 13-2 Status of Each Piece of Hardware after Reset Operation (continued)

Hardware (symbol)			Address (See Note 1) (lower 12 bits: xx □□□ H)	Condition after reset
Serial interface register	Serial mode register	SCM0, SMC1	F68H, F78H	00H
	Serial control register	SCC0, SCC1	F69H, F79H	00H
	Baud rate generator setting value	BRG0, BRG1	F6AH, F7AH	00H
	Receive buffer register	RxB0, RxB1	F60H, F70H	Undefined
	Transmit buffer register	TxB0, TxB1	F62H, F72H	Undefined
	Serial error register	SCE0, SCE1	F6BH, F7BH	00H
	Interrupt request control register	(error) SEIC0, SEIC1	F6CH, F7CH	47H
		(receive) SRIC0, SRIC1	F6DH, F7DH	
		(transmit) STIC0, STIC1	F6EH, F7EH	
	Macro service control register	(receive) SRMS0, SRMS1	F65H, F75H	Undefined
		(transmit) STMS0, STMS1	F66H, F76H	
Timer base interrupt request control register		TBIC	FECH	47H
User flag register		FLAG	FEAH	00H
Internal data area base register		IDB	FFFH	FFH
Processor control register		PRC	FEBH	4EH
Wait control register		WTC	FE8H	FFFFH
Refresh mode register		RFM	FE1H	FCH
Standby control register		STBC	FE0H	Undefined (See Note2)
External interrupt register	External interrupt mode register	INTM	F40H	00H
	Interrupt request control register	EXIC0-EXIC2	F4CH-F4EH	47H
	Macro service control register	EMS0-EMS2	F44H-F46H	Undefined

Note 1: xx of the upper 8 bits of the address is the value specified by the IDB register.

2: Upon power on reset: 00H

Others : No change

Pay attention for!!!!

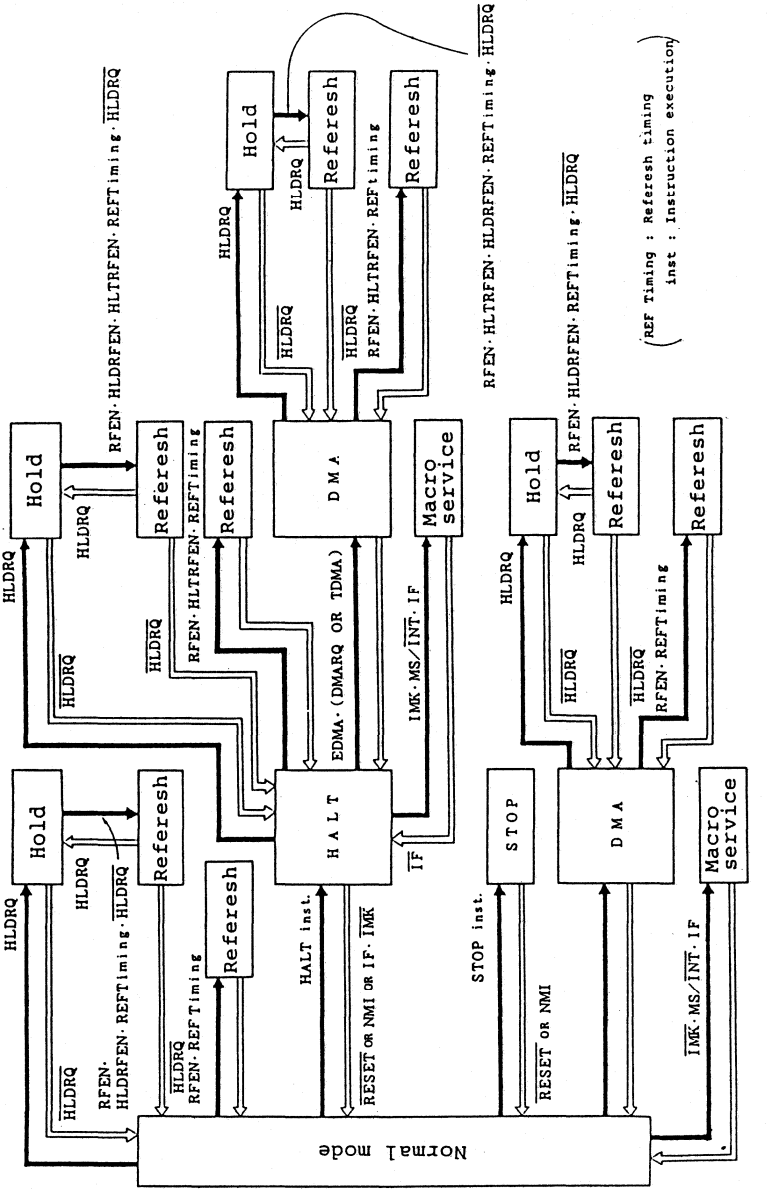
During active RESET (RESET pin is low level) all pins are in high impedance state except the following pins:

X1
X2
GND
VDD
RESET
I.C.

APPENDIX A

FLOW OF OPERATION STATUS CHANGE

The operating mode (macro service, DMA, refresh, hold, HALT, STOP) of the uPD70320/322 and uPD70330/332 changes as shown in the following figure according to the conditions indicated in the figure.



Flow of Operating Status Change

APPENDIX B

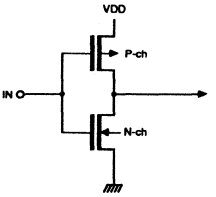
μPD70320/70322

INPUT/OUTPUT CIRCUITS

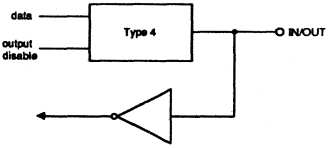
PIN	TYPE	PIN	TYPE
P00-P06	5	P20/DMARQ0	5
P07/CLKOUT	5	P21/DMAAK0	5
NMI	2	P22/TC0	5
P11/INTP0	1	P23/DMARQ1	5
P12/INTP1	1	P24/DMAAK1	5
P13 INTP2/INTAK	5	P25/TC1	5
P14/INT/POLL	5	P26/HLDAK	5
P15/TOUT	5	P27/HLDRQ	5
P16/SCK0	5	PT0 PT7	7
P17/READY	5		

PIN	TYPE	PIN	TYPE
TxD0	4	$\overline{EA}$	1
TxD1	4	D0-D7	5
RxD0	1	A0-A19	4
RxD1	1	$\overline{MREQ}$	4
$\overline{CTS0}$	5	$\overline{MSTB}$	4
$\overline{CTS1}$	1	R/$\overline{W}$	4
$\overline{REFRQ}$	4	IOSTB	4
$\overline{RESET}$	2		

Type 1

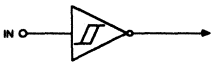


Type 5



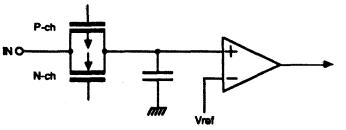
(I/O power circuit consit of Push-Pull output (Type 4) and Input Buffer (Type 1)

Type 2



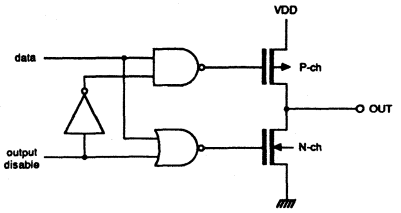
(Schmitt trigger Input which has hysterisys charcateristic)

Type 7



(Threshhold voltage)

Type 4



(Push-Pull output which can make output high impedance both P-ch ,N-ch off)

APPENDIX C

μPD70320/70322

UNUSED PIN CONNECTIONS (1)

pin name	recommendation
p00 - p06 p07 /clkout	input mode : pull up to vdd level through resistor output mode: open
p10 /nmi	pull down to gnd level through resistor
p11 /- intp0 p12 /- intp1	pull up to vdd level through resistor or pull down to gnd level through resistor
p13 /- intp2/- intak p14 / int/- poll p15 /tout p16 /- sck0 p17 /ready p20 /dmarq0 p21 /- dmaak0 p22 /- tc0 p23 /dmarq1 p24 /- dmaak1 p25 /- tc1 p26 /- hldak p27 /hldrq	input mode : pull up to vdd level through resistor output mode: open
pt0 - pt7	pull down to gnd level through resistor
txd0 txd1	open
rxd0 rxd1	pull up to vdd level through resistor or pull down to gnd level through resistor
-cts0	input mode : pull up to vdd level through resistor output mode: open

Unused pin connections (2)

pin name	recommendation
-cts1	pull up to vdd level through resistor or pull down to gnd level through resistor
-refrq	open
vth	pull down to gnd level through resistor
d0 - d7	input mode : pull up to vdd level through resistor output mode: open
a0 - a19 -mreq -mstb r/ -w -iostb	open

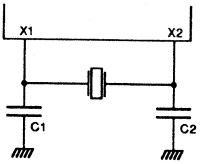
Use for pull - up and pull - down: 5 to 100 kohm

APPENDIX D

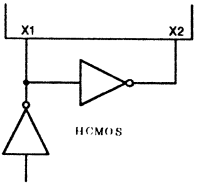
μPD 70320/70322

CLOCK GENERATION CIRCUITS

a. Cristal / Ceramic OSC



b. External Clock input



Note:

1. Set OSC circuit to the nearest position to X1 and X2
2. Do not attach any other connections to X1 and X2
3. When using Cristal OSC, set C1 = C2 = 15 pF
4. Following chart shows recommended OSC and values of C1 and C2 when using Ceramic OSC:

Device	Supplier	Type	C1(pF)	C2(pF)
μPD70322-8,70320-8	Murata	CSA16.00MX040	30	30
μPD70322,70320 μPD70322-8,70320-8 μPD70332,70330 μPD70P322		CSA10.0MT	47	47
μPD70322,70320 μPD70322-8,70320-8 μPD70332,70330 μPD70P322	Kyocera	KBR-10.0M	33	33
μPD70322-8,70320-8	T D K	FCR16.0M2S	15	6

APPENDIX E

μPD70320/70322

WAIT CONTROL WITH READY

Choosing the wait control from READY pin with WTC, the CPU inserts 2 waits between T1 state and T2 state.
CPU starts sampling the READY pin condition at falling edge of CLKOUT in TW (TAW) state.

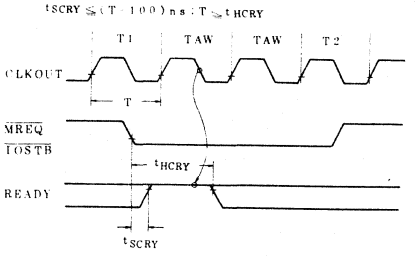
During TW (TAW) sampling, if READY pin is at 'L' level, CPU inserts wait states with same number of times as sampling.

During wait states, there is no insertion of refresh cycle. This would mean, that the DRAM controller would not be able to maintain the refresh interval, if the wait states become too long.

The following figures shows the corresponding timings:

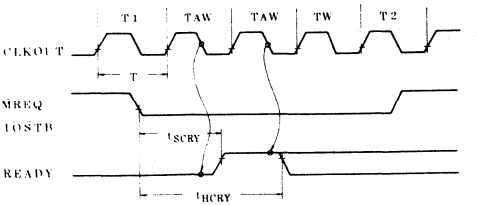
Wait by READY pin

a. without supplement

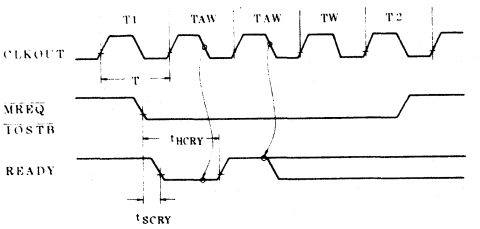


b. with one wait supplement

(i) $t_{SCRY} \geq (2T-100)\,ns : 2T \geq t_{HCRY}$

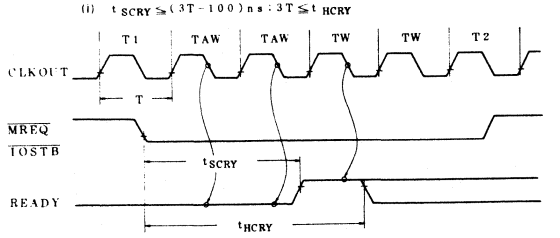


(ii) $t_{SCRY} \geq (T-100)\,ns : T \geq t_{HCRY}$

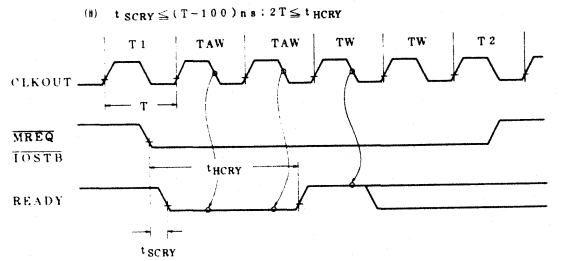


c. with two wait supplement

(i) $t_{SCRY} \geq (3T-100)\,ns : 3T \leq t_{HCRY}$



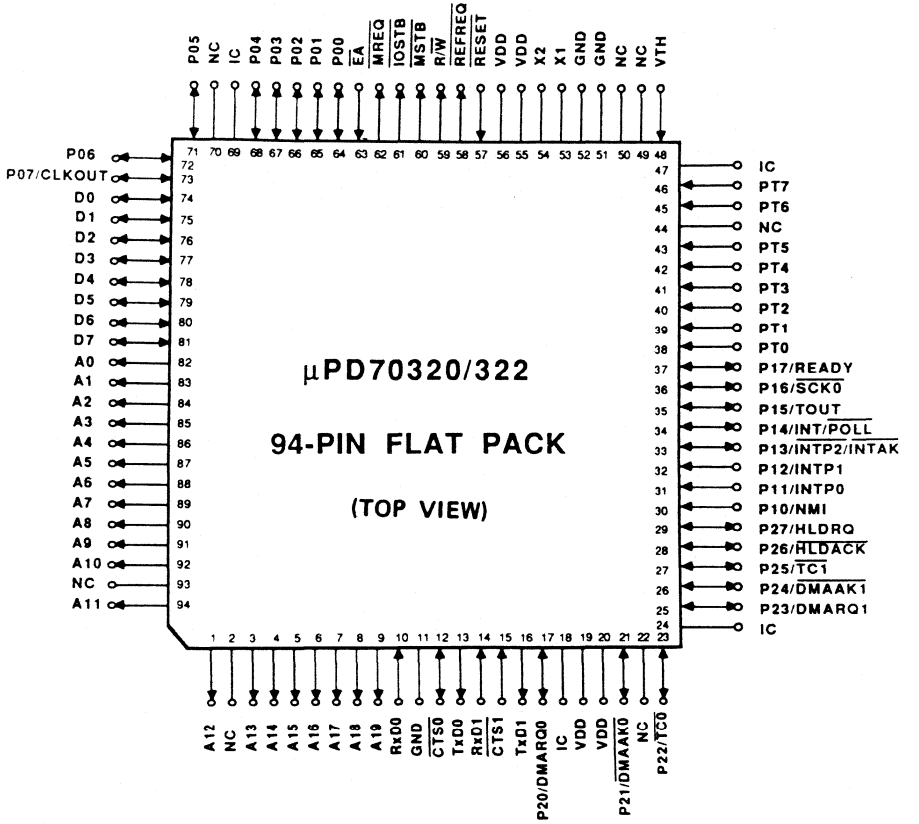
(ii) $t_{SCRY} \leq (T-100)\,ns : 2T \leq t_{HCRY}$



APPENDIX F

μPD70320/70322 PIN CONFIGURATION

94-PIN PLASTIC FLAT

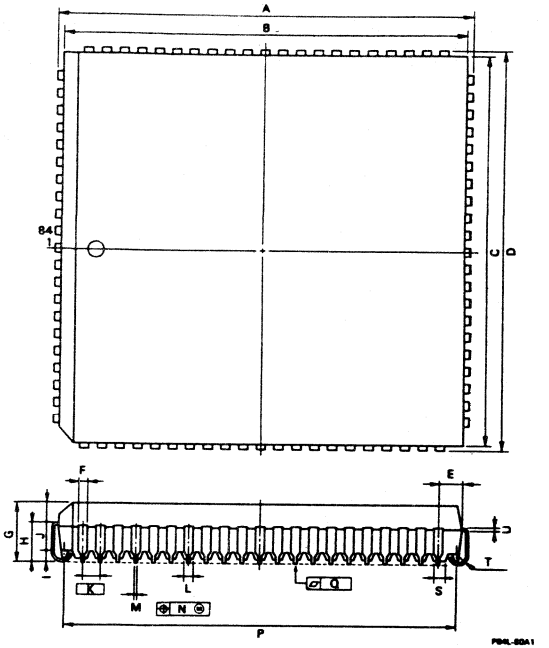


IC = Internal Connected
Pull-up (5 to 100kΩ) to each IC-pin

APPENDIX G

μPD70320/70322

84-PIN PLCC PACKAGE OUTLINE



P84L-80A1

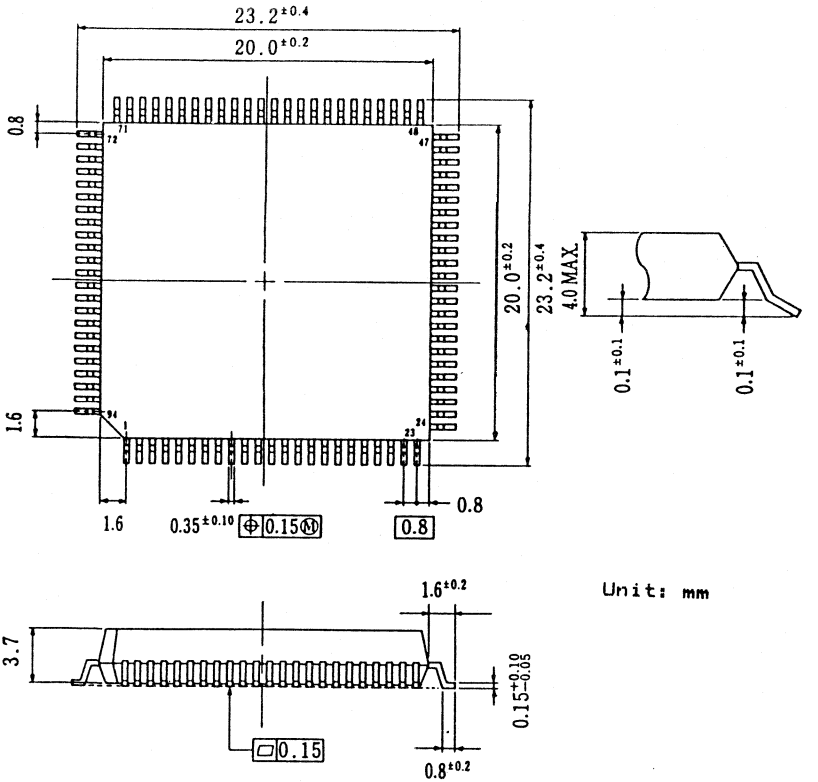
NOTE
Each lead centerline is located within 0.12 mm (0.005 inch) of its true position (T.P.) at maximum material condition.

ITEM	MILLIMETERS	INCHES
A	30.4±0.2	1.197±0.008
B	29.30	1.154
C	29.30	1.154
D	30.4±0.2	1.197±0.008
E	1.95±0.16	0.077±0.006
F	0.6	0.024
G	4.4±0.2	0.173±0.008
H	2.5±0.1	0.098±0.004
I	0.6 MIN.	0.024 MIN.
J	3.7	0.146
K	1.27 (T.P.)	0.050 (T.P.)
L	0.7	0.028
M	0.40±0.10	0.016±0.004
N	0.12	0.005
P	28.51±0.20	1.122±0.008
Q	0.15	0.006
S	1.0	0.040
T	R 0.8	R 0.031
U	0.20	0.008

APPENDIX H

μPD70320/70322

94-PIN PLASTIC FLAT PACKAGE OUTLINE



Unit: mm

APPENDIX I

μPD70320/322

ELECTRICAL SPECIFICATION

Absolute Maximum Ratings (Ta = 25°C)

Parameter	Symbol	Test Condition	Ratings	Unit
Power Supply Voltage	VDD		-0.5 to +7.0	V
	VTH		-0.5 to VDD+0.5 ≦+7.0	V
Input Voltage	VI		-0.5 to VDD+0.5 ≦+7.0	V
Output Voltage	VO		-0.5 to VDD+0.5 ≦+7.0	V
Output Current Low	IOL	All Output Pin	4.0	mA
		All Output Pin Total	50	mA
Output Current High	IOH	All Output Pin	-2.0	mA
		All Output Pin Total	-20	mA
Operating Temperature	Topt		-40 to +85	°C
Storage Temperature	Tstg		-65 to +150	°C

Capacitance (Ta = 25°C, VDD = OV)

Parameter	Symbol	Test Condition	Min.	Typ	Max.	Unit
Input Capacitance	CI	fc = 1MHz			10	PF
Output Capacitance	CO	Unmeasured Pins			20	PF
I/O Capacitance	CIO	Returned to OV			20	PF

DC Charcteristics (Ta = −10°C to +70°C, VDD = +5.0V ± 10%)

Parameter	Symbol	Test Condition	Min.	Typ	Max.	Unit
Input Low Voltage	V_{IL}		0		0.8	V
Input High Voltage	V_{IH1}	All except $\overline{RESET}$ P10/NMI, X1, X2	2.2		V_{DD}	V
	V_{IH2}	$\overline{RESET}$, P10/NMI, X1, X2	0.8 V_{DD}		V_{DD}	V
Output Low Voltage	V_{OL}	I_{OL} = 2.0mA			0.45	V
Output High Voltage	V_{OH}	I_{OH} = −0.4mA	V_{DD}−1.0			V
Input Current	I_I	EA, P10/NMI $0 \leqq V_I \leqq V_{DD}$			±20	μA
Input Leakage Current	I_{LI}	All Except EA, P10/NMI; $0 \leqq V_I \leqq V_{DD}$			±10	μA
Output Leakage Current	I_{LO}	$0 \leqq V_O \leqq V_{DD}$			±10	μA
VTH Supply Current	I_{TH}	$0 \leqq V_{TH} \leqq V_{DD}$		0.5	1.0	mA
VDD Supply Current	I_{DD1}	Operation Mode f_{CLKOUT} = 5MHz/8MHz		50/85	100/120	mA
	I_{DD2}	HALT Mode f_{CLKOUT} = 5MHz/8MHz		20/25	40/50	mA
	I_{DD3}	STOP Mode		10	30	μA

Comparator Characteristics (Ta = −10°C to + 70°C, BDD = +5.0V±10%)

Parameter	Symbol	Test Condition	Min.	Typ	Max.	Unit
Camparison Accuracy	V_{ACOMP}				±100	mV
Threshold Voltage	V_{TH}		0		V_{DD}+0.1	V
Comparison Time	t_{COMP}		64		65	TCYK
PT Input Voltage	V_{IPT}		0		V_{DD}	V

A.C. Characteristics (Ta = − 10°C to +70°C, VDD = +5.0V ± 10%)

Parameter	Symbol	Test Condition	Min.	Max.	Unit
Input Rise Time	t_{IR}	Except X1, X2 RESET, NMI		20	ns
Input Fall Time	t_{IF}			20	ns
Input Rise Time (SCHMITT)	t_{IRS}	RESET, NMI		30	ns
Input Fall Time (SCHMITT)	t_{IFS}			30	ns
Output Rise Time	t_{OR}	Except CLKOUT		20	ns
Output Fall Time	t_{OF}	Except CLKOUT		20	ns
X1 Input Cycle Time	t_{CYX}	f_{CLK}=5MHz/8MHz	98/62	250	ns
X1 Width Low	t_{WXL}	f_{CLK}=5MHz/8MHz	35/20		ns
X1 Width High	t_{WXH}	f_{CLK}=5MHz/8MHz	35/20		ns
X1 Rise Time	t_{XR}	f_{CLK}=5MHz/8MHz		10	ns
X1 Fall Time	t_{XF}	f_{CLK}=5MHz/8MHz		10	ns
CLKOUT Cycle Time	t_{CYK}	f_{OSC}/2	200/125	2000	ns
CLKOUT Width Low	t_{WKL}	f_{CLK}=5MHz/8MHz	0.5T−15		ns
CLKOUT Width High	t_{WKH}	f_{CLK}=5MHz/8MHz	0.5T−15		ns
CLKOUT Rise Time	t_{KR}			15	ns
CLKOUT Fall Time	t_{KF}			15	ns
Address Delay Time	t_{DKA}			90	ns
Address Valid to Input Data Valid	t_{DADR}			(n+1.5)T−120	ns
$\overline{MREQ}$ to Data Delay Time	t_{DMRD}			(n+1)T−90	ns
$\overline{MSTB}$ to Data Delay Time	t_{DMSD}			(n+0.5)T−90	ns
$\overline{MREQ}$ to $\overline{MSTB}$ Delay Time	t_{DMRMS}		0.5T−50	0.5T+50	ns
$\overline{MREQ}$ Width Low	t_{WMRL}		(n+1)T−40		ns
Address Hold Time	t_{HMA}		0.5T−50		ns
Input Data Hold Time	t_{HMDR}		0		ns
Next Control Setup Time	t_{SCC}		T−25		ns
$\overline{MREQ}$ to $\overline{TC}$ Delay Time	t_{DMRTC}			0.5T+50	ns

A. C. Characteristics (Continued)

Parameter	Symbol	Test Condition	Min.	Max.	Unit
Address to Data Output	t_{DADW}			0.5T+50	ns
$\overline{MREQ}$ Delay Time	t_{DAMR}		0.5T−50		ns
$\overline{MSTB}$ Delay Time	t_{DAMS}		T−50		ns
$\overline{MSTB}$ Width Low	t_{WMSL}		(n+0.5)T−40		ns
Data Output Setup time	t_{SDM}		(n+1)T−50		ns
Data Output Hold Time	t_{HMDW}		0.5T−50		ns
$\overline{IOSTB}$ Delay Time	t_{DAIS}		0.5T−50		ns
$\overline{IOSTB}$ to Data Input	t_{DISD}			(n+1)T−120	ns
$\overline{IOSTB}$ Width Low	t_{WISL}		(n+1)T−40		ns
Address Hold Time	t_{HISA}		0.5T−50		ns
Data Input Hold Time	t_{HISDR}		0		ns
Output Data Setup Time	t_{SDIS}		(n+1)T−50		ns
Output Data Hold Time	t_{HISDW}		0.5T−50		ns
Next DMARQ Setup Time	t_{SDADQ}	Demand Mode		1T	ns
DMARQ Hold Time	t_{HDADQ}	Demand Mode	0		ns
$\overline{DMAAK}$ Read Width Low	t_{WDMRL}		(n+1.5)T−40		ns
$\overline{DMAAK}$ to TC Delay Time	t_{DDATC}			0.5T+50	ns
$\overline{TC}$ Width Low	t_{WTCL}		2T−40		ns
$\overline{DMAAK}$ Write Width Low	t_{WDMWL}		(n+1)T−40		ns
$\overline{REFRQ}$ Delay Time	t_{DARF}		0.5T−50		ns
$\overline{REFRQ}$ Width Low	t_{WRFL}		(n+1)T−40		ns
Address Hold Time	t_{HRFA}		0.5T−50		ns

A. C. Characteristics (Continued)

Parameter	Symbol	Test Condition	Min.	Max.	Unit
$\overline{\text{RESET}}$ Width Low (STOP/POR.)	t_{WRSL1}		30		ms
$\overline{\text{RESET}}$ Width Low (System Reset)	t_{WRSL2}		5		μs
MREQ, IOSTB to READY Setup Time	t_{SCRY}	$n \geqq 2$		(n−1) T−100	ns
MREQ, IOSTB to READY Hold Time	t_{HCRY}	$n \geqq 2$	(n−1) T		ns
HLDRQ Setup Time	t_{SHQK}		30		ns
$\overline{\text{HLDAK}}$ Output Delay Time	t_{DKHA}			80	ns
Bus Control Float to $\overline{\text{HLDAK}}\downarrow$	t_{CFHA}		1T−50		ns
$\overline{\text{HLDAK}}\uparrow$ to Control Output Time	t_{HAC}		1T−50		ns
HLDRQ to $\overline{\text{HLDAK}}$ Delay Time	t_{DHQHA}			3T+160	ns
HLDRQ↓ to Control Float	t_{DHQC}		3T+30		ns
HLDRQ Width Low	t_{WHQL}		1,5T		ns
HLDAK Width Low	t_{WHAL}		1T		ns
INTP, DMARQ Setup Time	t_{SIQK}		30		ns
INTP, DMARQ Width High	t_{WIQH}		8T		ns
INTP, DMARQ Width Low	t_{WIQL}		8T		ns
$\overline{\text{POLL}}$ Setup Time	t_{SPLK}		30		ns
NMI Width High	t_{WNIH}		5		μs
NMI width Low	t_{WNIL}		5		μs
$\overline{\text{CTS}}$ Width Low	t_{WCTL}		2T		ns
INTR Setup Time	t_{SIRK}		30		ns
$\overline{\text{INTAK}}$ Delay Time	t_{DKIA}			80	ns
INTR Hold Time	t_{HIAIQ}		0		ns
$\overline{\text{INTAK}}$ Width Low	t_{WIAL}		2T−40		ns
$\overline{\text{INTAK}}$ Width High	t_{WIAH}		1T−40	2T−130	ns
$\overline{\text{INTAK}}$ to DATA Delay Time	t_{DIAD}			0.5T	ns
$\overline{\text{INTAK}}$ to DATA Hold Time	t_{HIAD}		0		ns

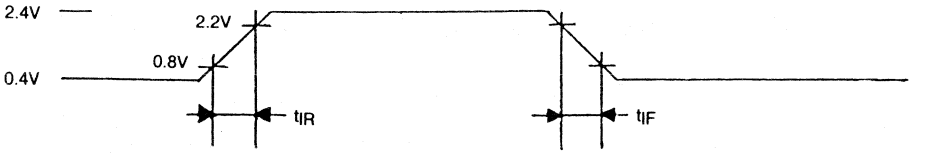
A. C. Characteristics (Continued)

Parameter	Symbol	Test Condition	Min.	Max.	Unit
$\overline{SCKO}$ Cycle Time	t_{CYTK}		1000		ns
$\overline{SCKO}$ (TSCK) Width High	t_{WSTH}		450		ns
$\overline{SCKO}$ (TSCK) Width Low	t_{WSTL}		450		ns
TXD Delay Time	t_{DTKD}			210	ns
$\overline{CTSO}$ (RSCK) Cycle Time	t_{CYRK}		1000		ns
$\overline{CTSO}$ (RSCK) Width High	t_{WSRH}		420		ns
$\overline{CTSO}$ (RSCK) Width Low	t_{WSRL}		420		ns
RXD Setup Time	t_{SRDK}		80		ns
RXD Hold Time	t_{HKRD}		80		ns

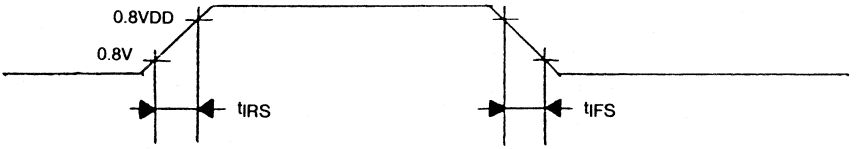
CL = 100 pF

Timing Waveforms

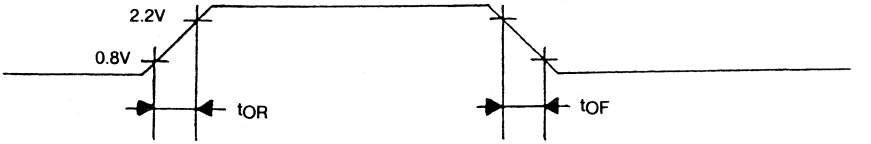
AC Input Waveform (1) (Except X1, X2, $\overline{\text{RESET}}$, $\overline{\text{NMI}}$)



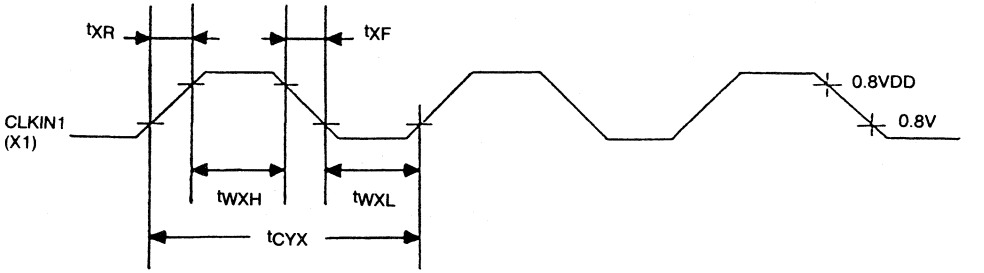
AC Input Waveform (2) ($\overline{\text{RESET}}$, $\overline{\text{NM}}$

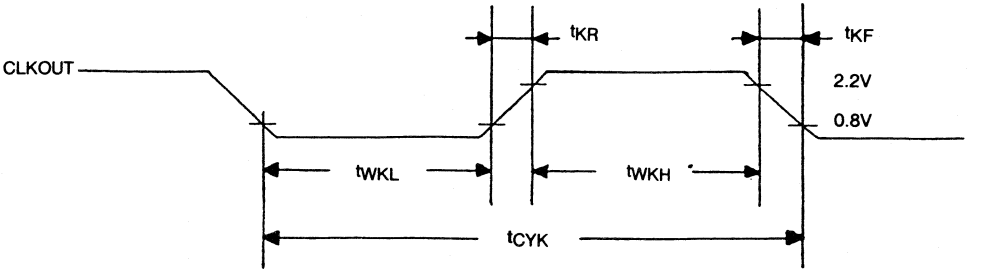


AC Output Test Point (Except CLKOUT)

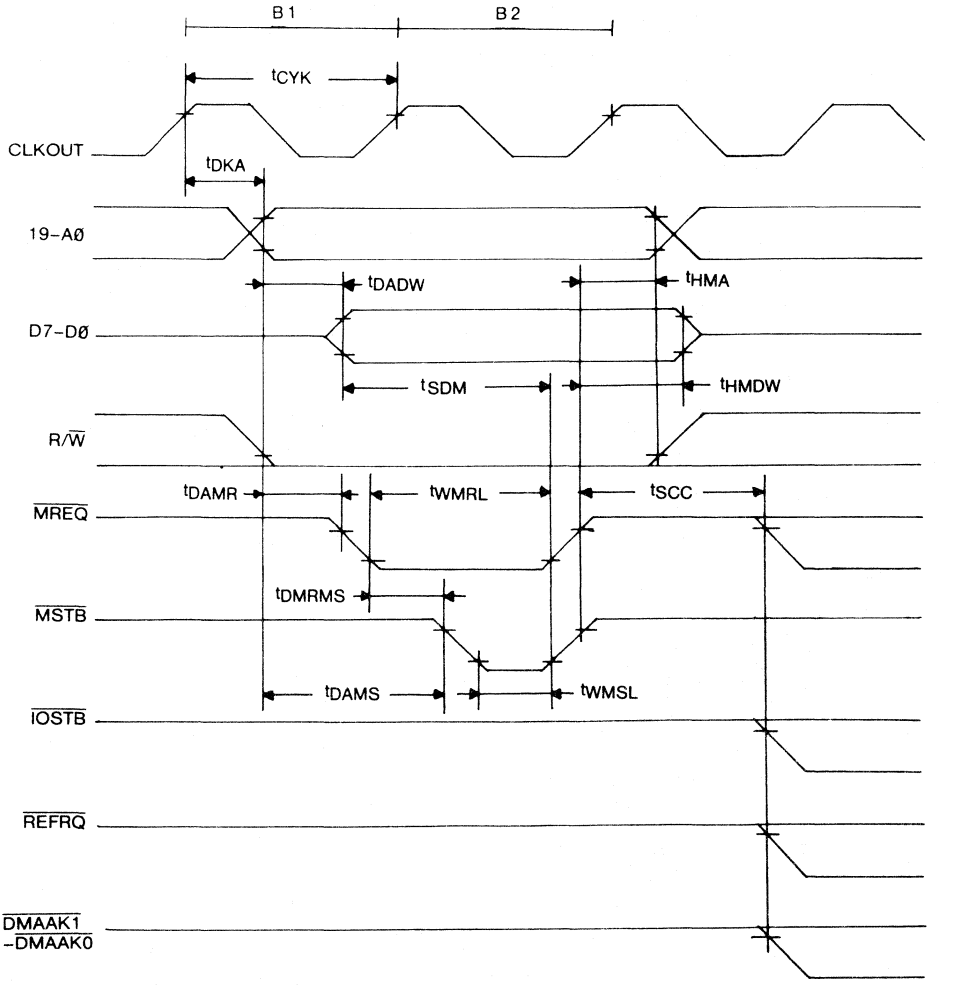


Clock Timing

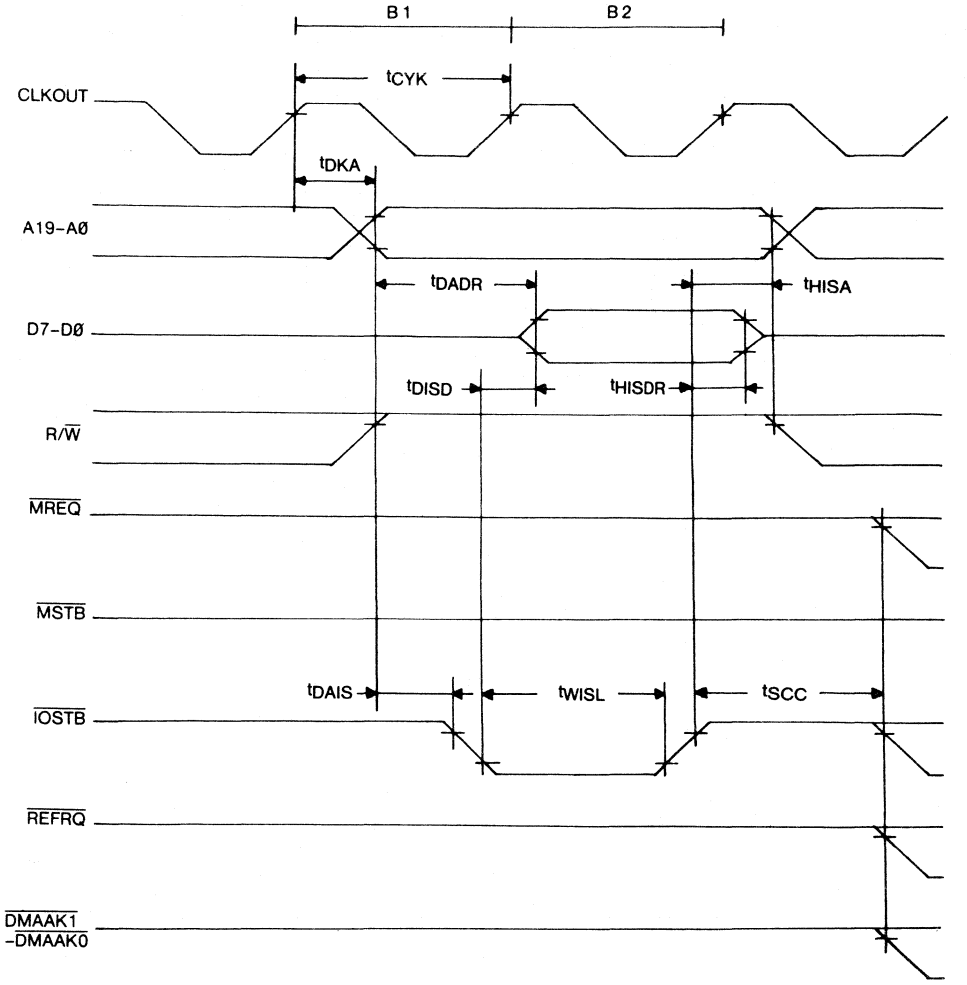




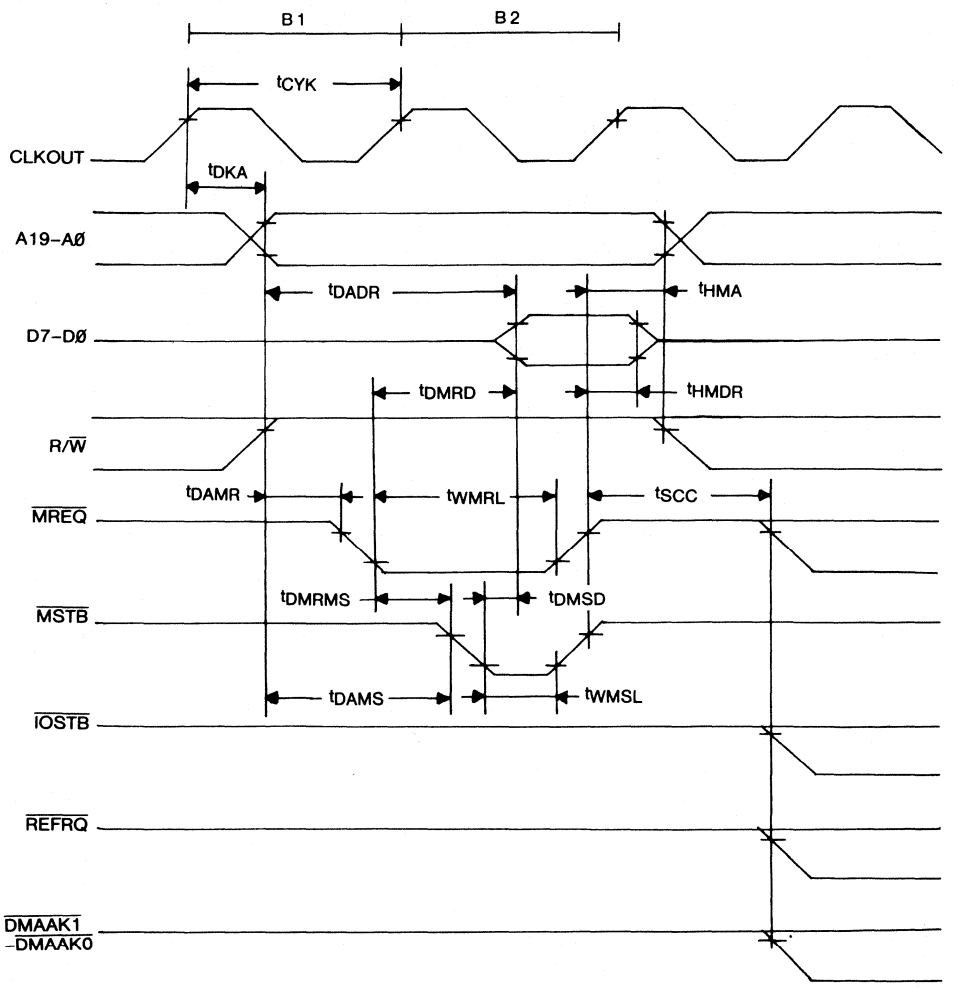
Memory Write Timing



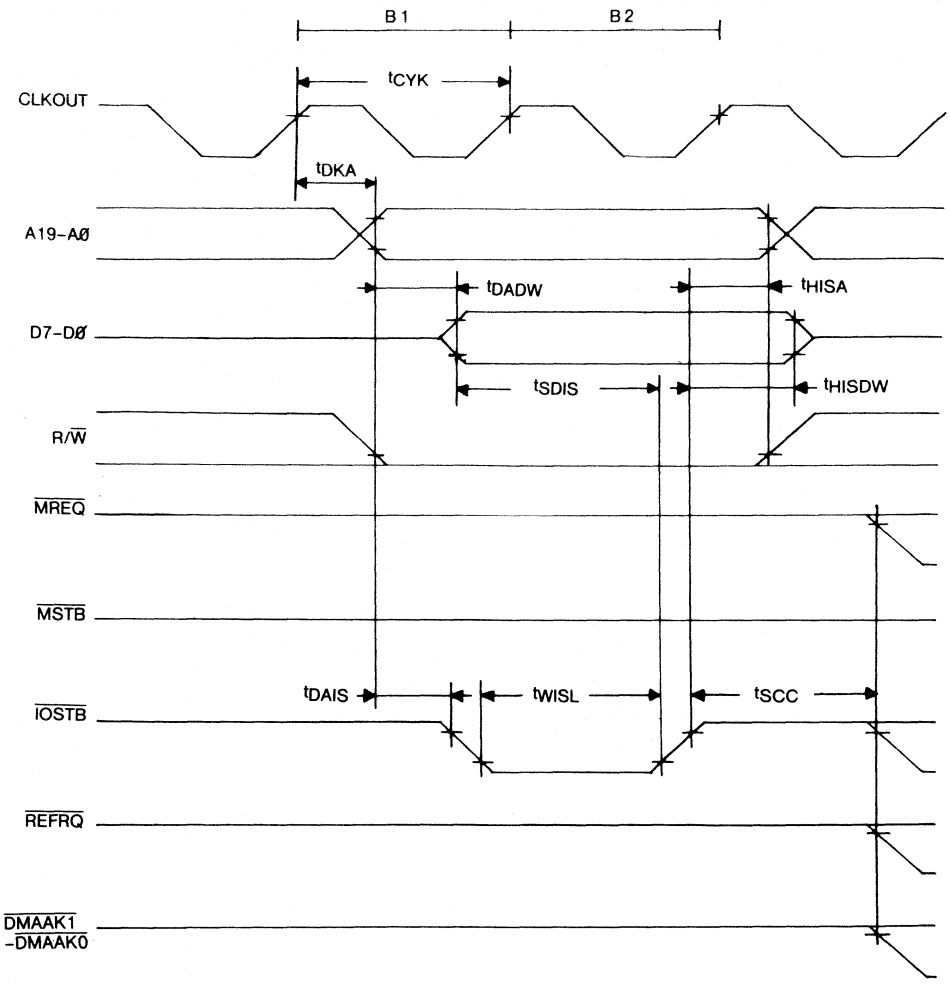
I/O Read Timing



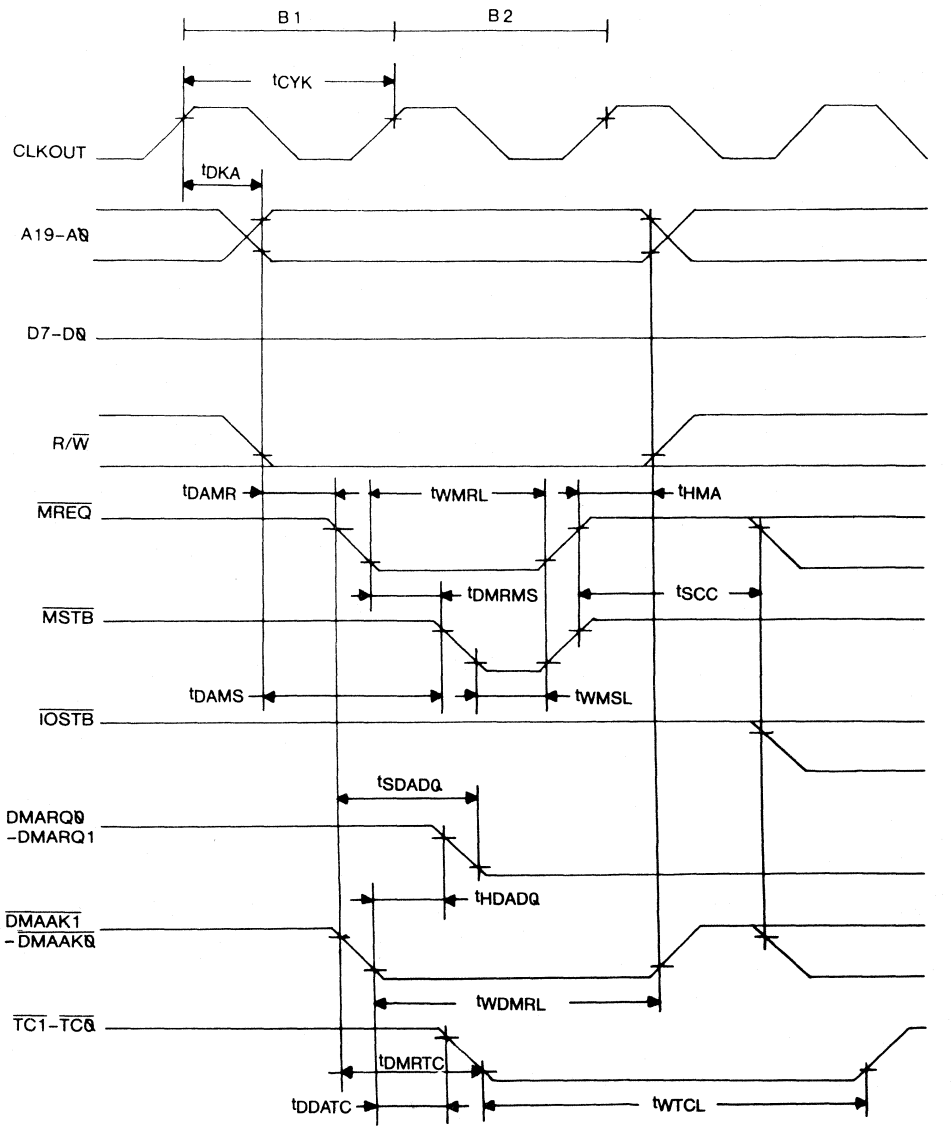
Memory Read Timing



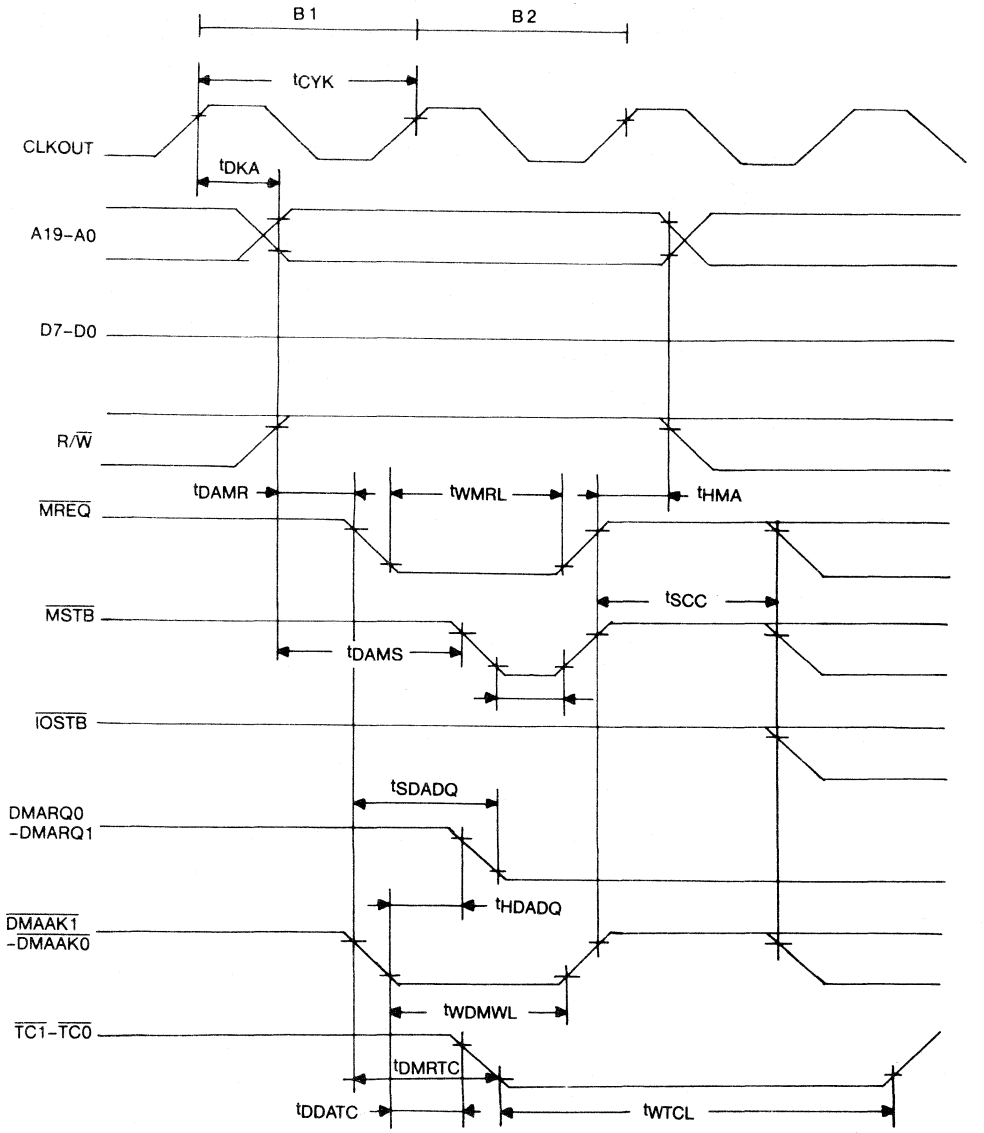
I/O Write Timing



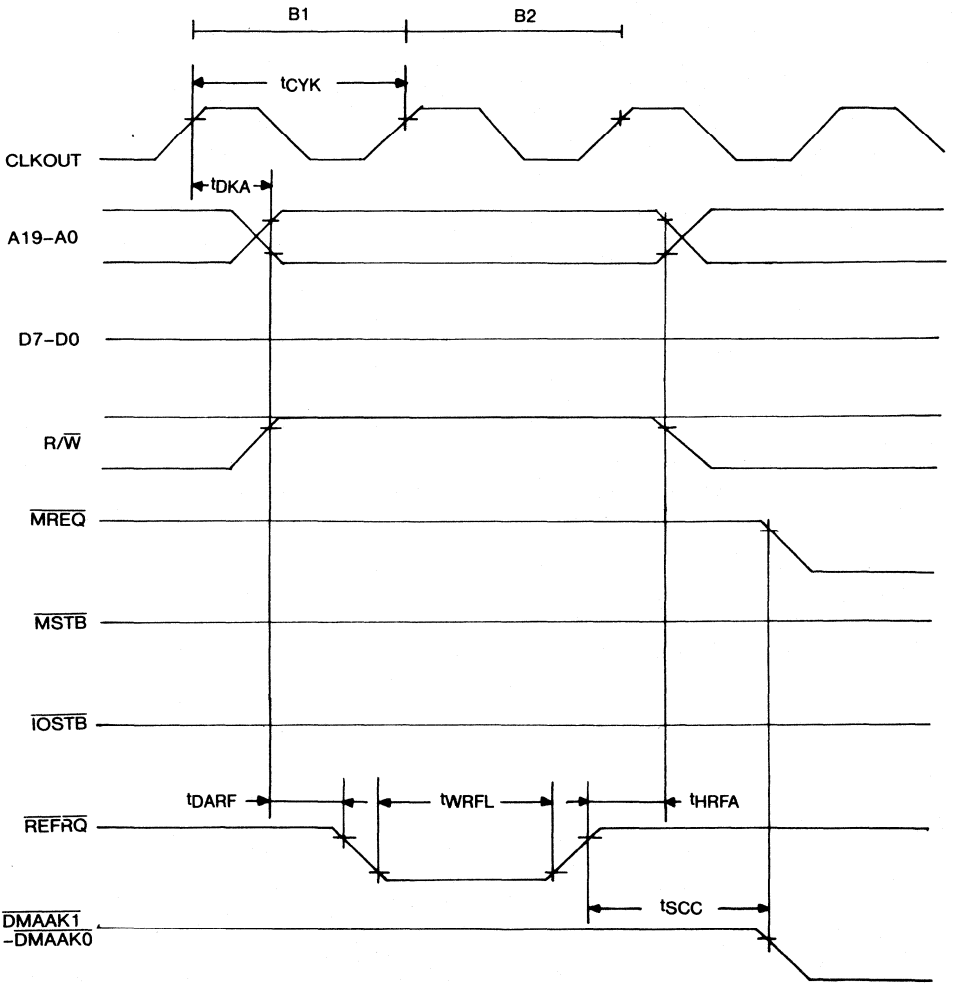
DMA(I/O →M) Timing



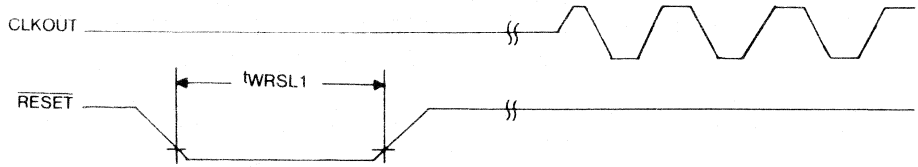
DMA(M→I/O) Timing



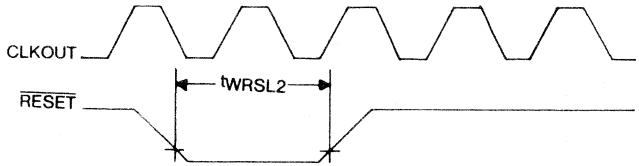
Refresh Timing



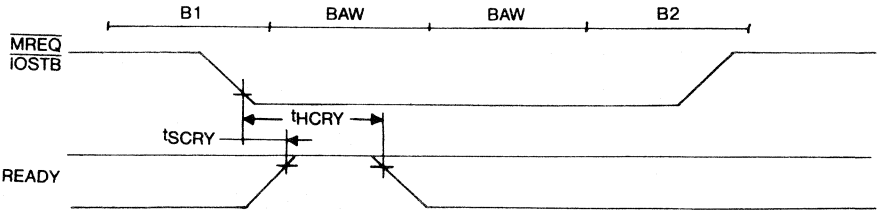
$\overline{\text{RESET}}$ Timing (1)



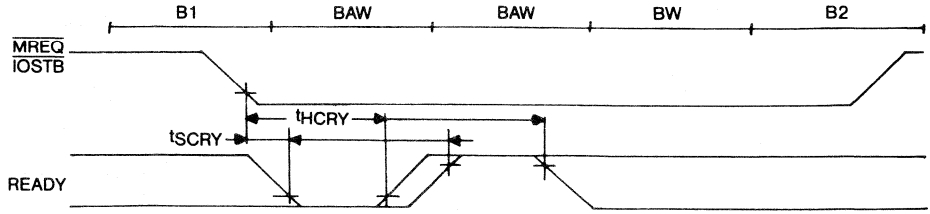
$\overline{\text{RESET}}$ Timing (2)

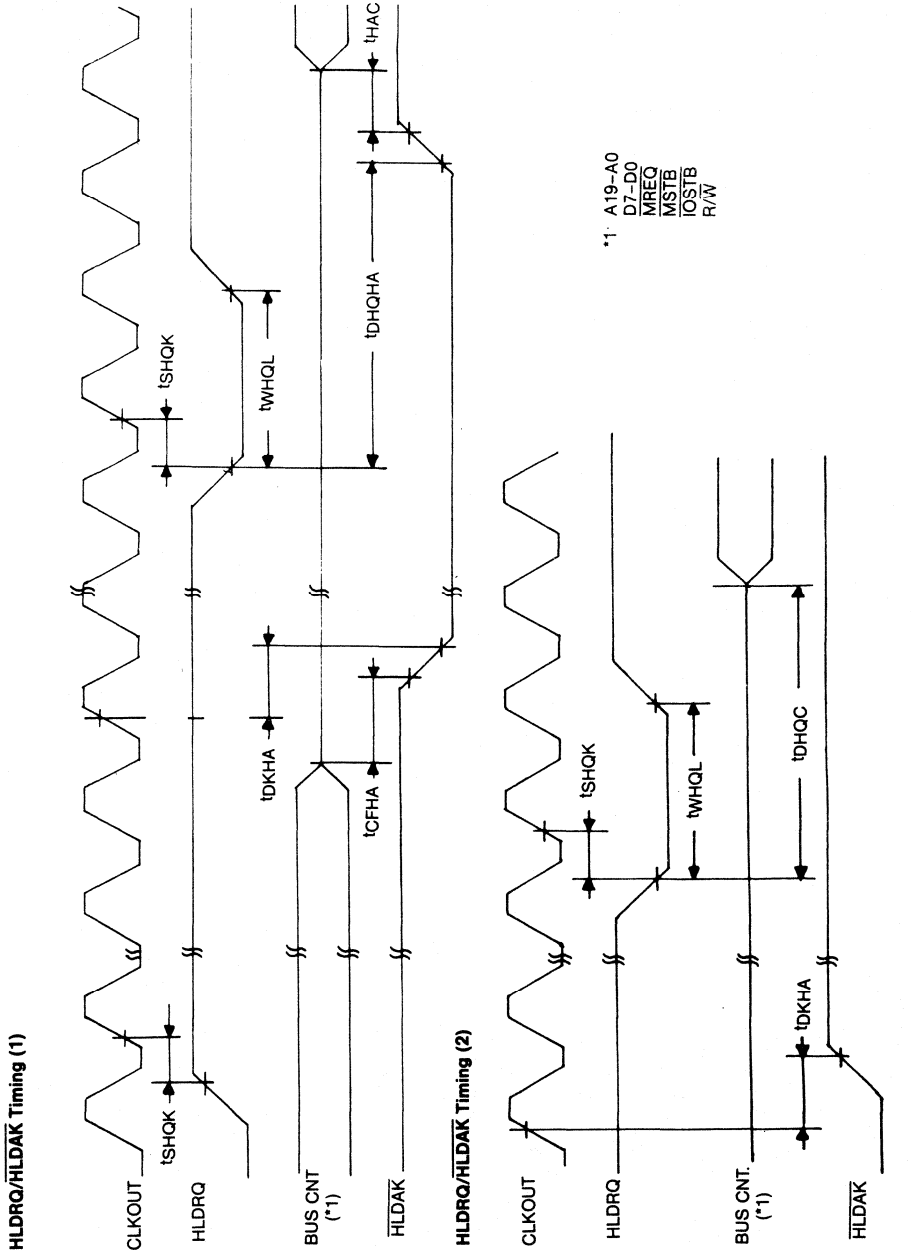


READY Timing (1)



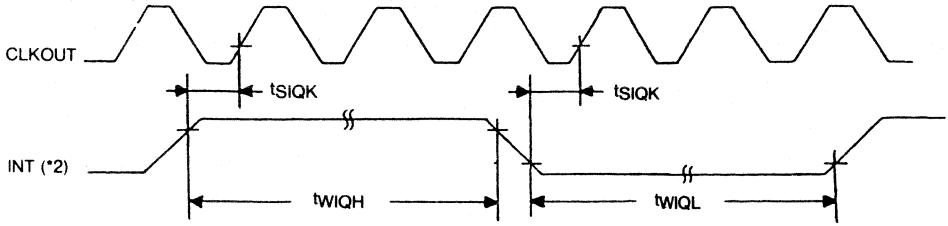
READY Timing (2)




HLDRQ/HLDAK Timing (1)
CLKOUT
HLDRQ
BUS CNT (*1)
HLDAK
tSHQK
tWHQL
tDHQHA
tHAC
tDKHA
tCFHA
HLDRQ/HLDAK Timing (2)
CLKOUT
HLDRQ
BUS CNT. (*1)
HLDAK
tSHQK
tWHQL
tDHQC
tDKHA
*1: A19-A0
D7-D0
MREQ
MSTB
IOSTB
R/W

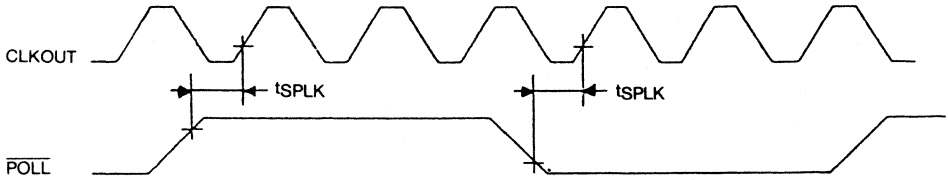
Timing Waveforms

$\overline{INTP2}$–$\overline{INTP0}$, DMARQ1–DMARQ0 Input Timing

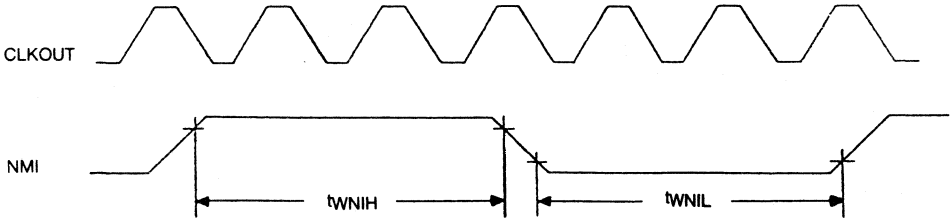


*2: $\overline{INTP2}$–$\overline{INTP0}$, DMARQ1–DMARQ0

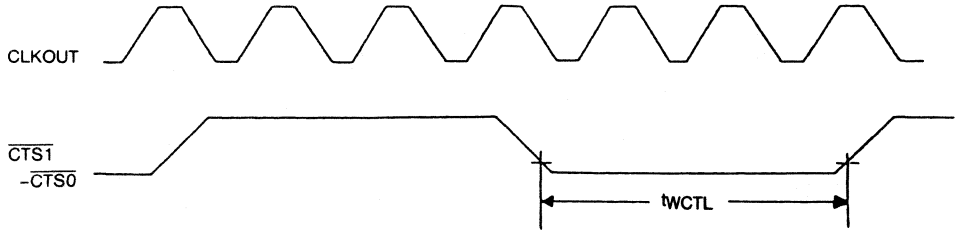
$\overline{POLL}$ Input Timing

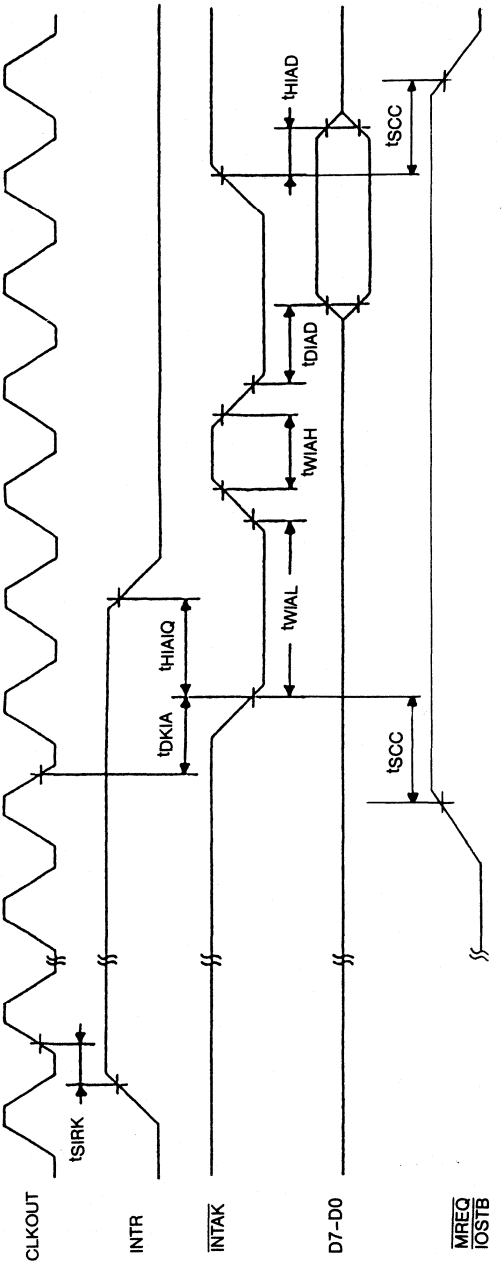


NMI Input Timing



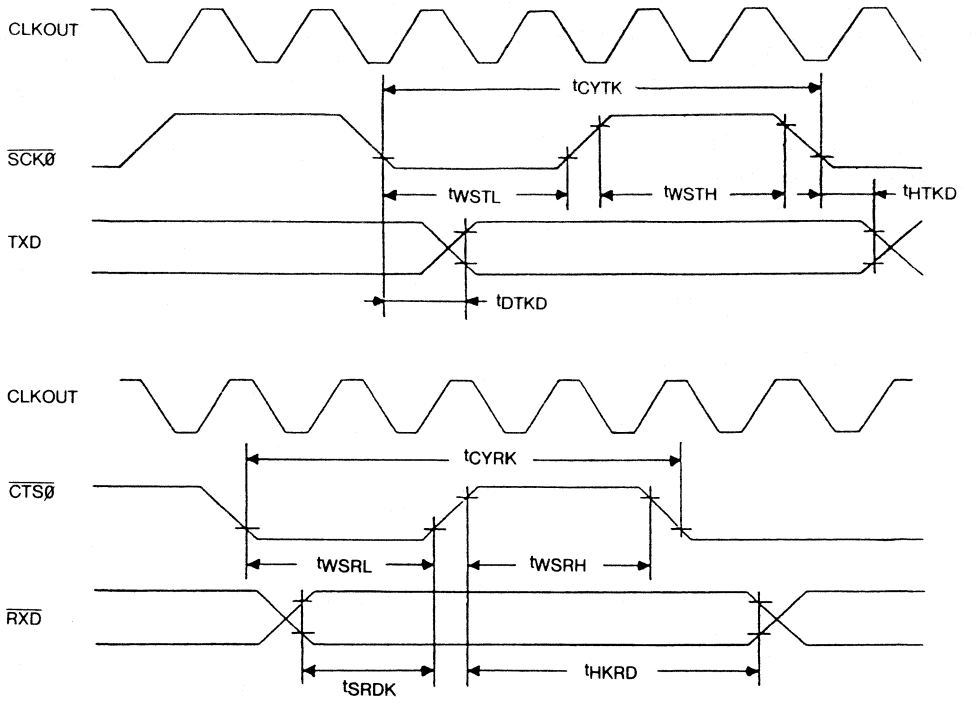
$\overline{CTS1}$–$\overline{CTS0}$ Input Timing





INTR/INTAK Timing

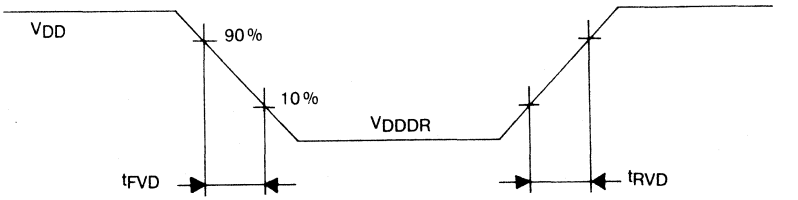
Timing Waveforms
SIO Timing



Stop Mode Data Retaintion Characteristics
(Ta = −10°C to +70°C)

Parameter	Symbol	Test Condition	Min.	Max	Unit
Data Retaintion Voltage	V_{DDDR}		2.5	5.5	V
V_{DD} Rise Time V_{DD} Fall Time	t_{RVD} t_{FVD}		200		μS

Stop Mode Data Retaintion Timing



APPENDIX J

μPD70320/70322

INSTRUCTION EXECUTION CLOCK LIST

V25 EA Clock Number Calculation

mod / mem	00	no clks	01	no clks	10	no clks
000	BW+IX	3	BW+IX+disp 8	3	BW+IX+disp 16	4
001	BW+IY	3	BW+IY+disp 8	3	BW+IY+disp 16	4
010	BP+IX	3	BP+IX+disp 8	3	BP+IX+disp 16	4
011	BP+IY	3	BP+IY+disp 8	3	BP+IY+disp 16	4
100	IX	3	IX+ disp 8	3	IX+ disp 16	4
101	IY	3	IY+ disp 8	3	IY+ disp 16	4
110	Direct	3	BP+ disp 8	3	BP+ disp 16	4
111	BW	3	BW+ disp 8	3	BW+ disp 16	4

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
data transfer instruction	MOV	reg, reg	1 0 0 0 1 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	1 0 0 0 1 0 0 W	mod reg mem	2-4	EA + 4 + W	EA + 2	EA + 6 + 2 W	EA + 2
		reg, mem	1 0 0 0 1 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		mem, imm	1 1 0 0 0 1 1 W	mod 0 0 0 mem	3-6	EA + 5 + W	EA + 5 + W	EA + 5 + 2 W	EA + 5 + 2 W
		reg, imm	1 0 1 1 W reg		2-3	5	5	6	6
		acc, dmem	1 0 1 0 0 0 0 W		3	9 + W	9 + W	11 + 2 W	11 + 2 W
		dmem, acc	1 0 1 0 0 0 1 W		3	7 + W	5 + W	9 + 2 W	5
		sreg, reg 16	1 0 0 0 1 1 1 0	1 1 0 sreg reg	2	–	–	4	4
		sreg, mem 16	1 0 0 0 1 1 1 0	mod 0 sreg mem	2-4	–	–	EA + 10 + 2 W	EA + 10 + 2 W
		reg 16, sreg	1 0 0 0 1 1 0 0	1 1 0 sreg reg	2	–	–	3	3
		mem 16, sreg	1 0 0 0 1 1 0 0	mod 0 sreg mem	2-4	–	–	EA + 7 + 2 W	EA + 3
		DS0, reg 16 mem 32	1 1 0 0 0 1 0 1	mod reg mem	2-4	–	–	EA + 19 + 4 W	EA + 19 + 4 W
		DS1, reg 16 mem 32	1 1 0 0 0 1 0 0	mod reg mem	2-4	–	–	EA + 19 + 4 W	EA + 19 + 4 W
		AH, PSW	1 0 0 1 1 1 1 1		1	2	2	–	–
		PSW, AH	1 0 0 1 1 1 1 0		1	3	3	–	–
	LDEA	reg 16, mem 16	1 0 0 0 1 1 0 1	mod reg mem	2-4	–	–	EA + 2	EA + 2
	TRANS	src-table	1 1 0 1 0 1 1 1		1	10 + W	10 + W	–	–
	XCH	reg, reg	1 0 0 0 0 1 1 W	1 1 reg reg	2	3	3	3	3
		mem, reg reg, mem	1 0 0 0 0 1 1 W	mod reg mem	2-4	EA + 10 + 2 W	EA + 8 + 2 W	EA + 14 + 4 W	EA + 10 + 4 W
		AW, reg 16 reg 16, AW	1 0 0 1 0 reg		1	–	–	4	4

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
repeat instructions	REPC		0 1 1 0 0 1 0 1		1	2	2	2	2
	REPNC		0 1 1 0 0 1 0 0		1	2	2	2	2
	REP REPE REPZ		1 1 1 1 0 0 1 1		1	2	2	2	2
	REPNE REPNZ		1 1 1 1 0 0 1 0		1	2	2	2	2
block transfer instructions	MOVBK	dst-block, src-block	1 0 1 0 0 1 0 W		1	20 + 2 W (rep.) 16 + (16 + 2 W) n	16 + 1 W 16 + (12 + 1 W) n	24 + 4 W 16 + (20 + 4 W) n	20 + 2 W 16 + (12 + 2 W) n
	CMPBK	src-block, dst-block	1 0 1 0 0 1 1 W		1	23 + 2 W (rep.) 16 + (21 + 2 W) n	19 + 2 W 16 + (21 + 2 W) n	27 + 4 W 16 + (25 + 2 W) n	21 + 4 W 16 + (25 + 2 W) n
	CMPM	dst-block	1 0 1 0 1 1 1 W		1	17 + W (rep.) 16 + (15 + W) n	17 + W 16 + (15 + W) n	19 + 2 W 16 + (17 + W) n	19 + 2 W 16 + (17 + W) n
	LDM	src-block	1 0 1 0 1 1 0 W		1	12 + W (rep.) 16 + (10 + W) n	12 + W 16 + (10 + W) n	14 + 2 W 16 + (12 + 2 W) n	14 + 2 W 16 + (12 + 2 W) n
	STM	dst-block	1 0 1 0 1 0 1 W		1	12 + W (rep.) 16 + (8 + W) n	10 16 + (6 + W) n	14 + 2 W 16 + (10 + 2 W) n	10 16 + (6 + 2 W) n

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			76543210	76543210		RAM enable	RAM disable	RAM enable	RAM disable
Bit field transfer instructions	INS	reg 8, reg 8	00001111 11 reg reg	00110001	3			63 - 155	63 - 155
		reg 8, imm 4	00001111 11000 reg	00111001	4			64 - 156	64 - 156
	EXT	reg 8, reg 8	00001111 11 reg reg	00110011	3			41 - 121	41 - 121
		reg 8, imm 4	00001111 11000 reg	00111011	4			42 - 122	42 - 122
I/O instructions	IN	acc, imm 8	1110010W		2	14 + W	14 + W	16 + 2 W	16 + 2 W
		acc, DW	1110110W		1	13 + W	13 + W	15 + 2 W	15 + 2 W
	OUT	imm 8, acc	1110011W		2	10 + W	10 + W	10 + 2 W	10 + 2 W
		DW, acc	1110111W		1	9 + W	9 + W	9 + 2 W	9 + 2 W
Primitive I/O instructions	INM	dst-block, DW	0110110W		1	19 + 2 W (rep.) 18 + (13 + 2 W) n	17 + 2 W 18 + (11 + 2 W) n	21 + 4 W 18 + (15 + 4 W) n	17 + 4 W 18 + (11 + 4 W) n
	OUTM	DW, src-block	0110111W		1	19 + 2 W (rep.) 18 + (13 + 2 W) n	17 + 2 W 18 + (11 + 2 W) n	21 + 4 W 18 + (15 + 4 W) n	17 + 4 W 18 + (11 + 4 W) n

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
addition/substraction instructions	ADD	reg, reg	0 0 0 0 0 0 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	0 0 0 0 0 0 0 W	mod reg mem	2-4	EA + 8 + 2 W	EA + 6 + 1 W	EA + 12 + 4 W	EA + 8 + 2 W
		reg, mem	0 0 0 0 0 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		reg, imm	1 0 0 0 0 0 S W	1 1 0 0 0 reg	3-4	5	5	6	6
		mem, imm	1 0 0 0 0 0 S W	mod 0 0 0 mem	3-6	EA + 9 + 2 W	EA + 7 + 2 W	EA + 14 + 4 W	EA + 10 + 4 W
		acc, imm	0 0 0 0 0 1 0 W		2-3	5	5	6	6
	ADDC	reg, reg	0 0 0 1 0 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	0 0 0 1 0 0 0 W	mod reg mem	2-4	EA + 8 + 2 W	EA + 6 + 1 W	EA + 12 + 4 W	EA + 8 + 2 W
		reg, mem	0 0 0 1 0 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		reg, imm	1 0 0 0 0 0 S W	1 1 0 1 0 reg	3-4	5	5	6	6
		mem, imm	1 0 0 0 0 0 S W	mod 0 1 0 mem	3-6	EA + 9 + 2 W	EA + 7 + 2 W	EA + 14 +4 W	EA + 10 + 4 W
		acc, imm	0 0 0 1 0 1 0 W		2-3	5	5	6	6
	SUB	reg, reg	0 0 1 0 1 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	0 0 1 0 1 0 0 W	mod reg mem	2-4	EA + 8 + 2 W	EA + 6 + 1 W	EA + 12 + 4 W	EA + 8 + 2 W
		reg, mem	0 0 1 0 1 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		reg, imm	1 0 0 0 0 0 S W	1 1 1 0 1 reg	3-4	5	5	6	6
		mem, imm	1 0 0 0 0 0 S W	mod 1 0 1 mem	3-6	EA + 9 + 2 W	EA + 7 + 2 W	EA + 14 + 4 W	EA + 10 + 4 W
		acc, imm	0 0 1 0 1 1 0 W		2-3	5	5	6	6
	SUBC	reg, reg	0 0 0 1 1 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	0 0 0 1 1 0 0 W	mod reg mem	2-4	EA + 8 + 2 W	EA + 6 + 1 W	EA + 12 + 4 W	EA + 8 + 2 W
		reg, mem	0 0 0 1 1 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		reg, imm	1 0 0 0 0 0 S W	1 1 0 1 1 reg	3-4	5	5	6	6
		mem, imm	1 0 0 0 0 0 S W	mod 0 1 1 mem	3-6	EA + 9 +2 W	EA + 7 + 2 W	EA + 14 + 4 W	EA + 10 + 4 W
		acc, imm	0 0 0 1 1 1 0 W		2-3	5	5	6	6

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			76543210	76543210		RAM enable	RAM disable	RAM enable	RAM disable
BCD operation instructions	ADD4S		00001111	00100000	2	22 + (27 + 3 W) n	22 + (25 + 3 W) n	–	–
	SUB4S		00001111	00100010	2	22 + (27 + 3 W) n	22 + (25 + 3 W) n	–	–
	CMP4S		00001111	00100110	2	22 + (23 + 2 W) n	22 + (23 + 2 W) n	–	–
	ROL4	reg 8	00001111 11000 reg	00101000	3	17	17	–	–
		mem 8	00001111 mod 000 mem	00101000	3-5	EA + 18 + 2 W	EA + 16 + 2 W	–	–
	ROR4	reg 8	00001111 11000 reg	00101010	3	21	21		
		mem 8	00001111 mod 000 mem	00101010	3-5	EA + 24 + 2 W	EA + 22 + 2 W		
increase/decrease instruction	INC	reg 8	11111110	11000 reg	2	5 [7)]	5 [7)]	5	5
		mem	1111111 W	mod 000 mem	2-4	EA + 11 + 2 W	EA + 9 + 2 W	EA + 15 + 4 W	EA + 11 + 4 W
		reg 16	01000 reg		1	–	–	2 [1)]	2 [1)]
	DEC	reg 8	11111110	11001 reg	2	5 [7)]	5 [7)]	5	5
		mem	1111111 W	mod 001 mem	2-4	EA + 11 + 2 W	EA + 9 + 2 W	EA + 15 + 4 W	EA + 11 + 4 W
		reg 16	01001 reg		1	–	–	2 [1)]	2 [1)]

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
multiplication operation	MULU	reg 8	1 1 1 1 0 1 1 0	1 1 1 0 0 reg	2	24	24	–	reg, reg
		mem 8	1 1 1 1 0 1 1 0	mod 1 0 0 mem	2-4	EA + 26 + W	EA + 26 + W	–	–
		reg 16	1 1 1 1 0 1 1 1	1 1 1 0 0 reg	2	–	–	32	32
		mem 16	1 1 1 1 0 1 1 1	mod 1 0 0 mem	2-4	–	–	EA + 34 + 2 W	EA + 34 + 2 W
	MUL	reg 8	1 1 1 1 0 1 1 0	1 1 1 0 1 reg	2	31 ~ 40	31 ~ 40	–	–
		mem 8	1 1 1 1 0 1 1 0	mod 1 0 1 mem	2-4	EA + 33 + W ~ EA + 42 + W	EA + 33 + W ~ EA + 42 + W	–	–
		reg 16	1 1 1 1 0 1 1 1	1 1 1 0 1 reg	2	–	–	39 ~48	39 ~ 48
		mem 16	1 1 1 1 0 1 1 1	mod 1 0 1 mem	2-4	–	–	EA + 43 + 2 W ~ EA + 52 + 2 W	EA + 43 + 2 W ~ EA + 52 + 2 W
		reg 16, (reg 16,) * imm 8	0 1 1 0 1 0 1 1	1 1 reg reg	3	–	–	39 ~ 49	39 ~ 49
		reg 16, mem 16, imm 8	0 1 1 0 1 0 1 1	mod reg mem	3-5	–	–	EA + 43 + 2 W ~ EA + 53 + 2 W	EA + 43 + 2 W ~ EA + 53 + 2 W
		reg 16, (reg 16,) * imm 16	0 1 1 0 1 0 0 1	1 1 reg reg	4	–	–	40 ~ 50	40 ~ 50
		reg 16, mem 16, imm 16	0 1 1 0 1 0 0 1	mod reg mem	4-6	–	–	EA +44 + 2 W ~ EA + 54 + 2 W	EA + 44 + 2 W ~ EA + 54 + 2 W

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Signed division instructions	DIV	reg 8	1 1 1 1 0 1 1 0	1 1 1 1 1 reg	2	46 ~ 56	46 ~ 56	–	–
		mem 8	1 1 1 1 0 1 1 0	mod 1 1 1 mem	2-4	EA + 48 + W ~ EA + 58 + W	EA + 48 + W ~ EA + 58 + W	–	–
		reg 16	1 1 1 1 0 1 1 1	1 1 1 1 1 reg	2	–	–	54 ~ 64	54 ~ 64
		mem 16	1 1 1 1 0 1 1 1	mod 1 1 1 mem	2-4	–	–	EA + 58 + 2 W ~ EA + 68 + 2 W	EA + 58 + 2 W ~ EA + 68 + 2 W

instruction group	mnemonic	operand	operation code			no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0			RAM enable	RAM disable	RAM enable	RAM disable
Unsigned division instructions	DIVU	reg 8	1 1 1 1 0 1 1 0	1 1 1 1 0 reg		2	31	31	–	–
		mem 8	1 1 1 1 0 1 1 0	mod 1 1 0 mem		2-4	EA + 33 + W	EA + 33 + W	–	–
		reg 16	1 1 1 1 0 1 1 1	1 1 1 1 0 reg		2	–	–	39	39
		mem 16	1 1 1 1 0 1 1 1	mod 1 1 0 mem		2-4	–	–	EA + 43 + 2 W	EA + 43 + 2 W

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
BCD complement instructions	ADJBA		0 0 1 1 0 1 1 1		1	17	17	–	–
	ADJ4A		0 0 1 0 0 1 1 1		1	9	9	–	–
	ADJBS		0 0 1 1 1 1 1 1		1	17	17	–	–
	ADJ4S		0 0 1 0 1 1 1 1		1	9	9	–	–
Data Conversion instructions	CVTBD		1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	2	20	20	–	–
	CVTDB		1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	2	19	19	–	–
	CVTBW		1 0 0 1 1 0 0 0		1	3	3	–	–
	CVTWL		1 0 0 1 1 0 0 1		1	–	–	8	8
Comparison instructions	CMP	reg, reg	0 0 1 1 1 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	0 0 1 1 1 0 0 W	mod reg mem	2-4	EA + 8 + 2 W	EA + 6 + 1 W	EA + 12 + 4 W	EA + 8 + 2 W
		reg, mem	0 0 1 1 1 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		reg, imm	1 0 0 0 0 0 S W	1 1 1 1 1 reg	3-4	5	5	6	6
		mem, imm	1 0 0 0 0 0 S W	mod 1 1 1 mem	3-6	EA + 9 + 2 W	EA + 7 + 2 W	EA + 14 + 4 W	EA + 10 + 4 W
		acc, imm	0 0 1 1 1 1 0 W		2-3	5	5	6	6
Complement instructions	NOT	reg	1 1 1 1 0 1 1 W	1 1 0 1 0 reg	2	5	5	5	5
		mem	1 1 1 1 0 1 1 W	mod 0 1 0 mem	2-4	EA + 11 + 2 W	EA + 9 + 1 W	EA + 15 + 4 W	EA + 11 + 2 W
	NEG	reg	1 1 1 1 0 1 1 W	1 1 0 1 1 reg	2	5	5	5	5
		mem	1 1 1 1 0 1 1 W	mod 0 1 1 mem	2-4	EA + 11 + 2 W	EA + 9 + 1 W	EA + 15 + 4 W	EA + 11 + 2 W

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Logical operation instructions	TEST	reg, reg	1 0 0 0 0 1 0 W	1 1 reg reg	2	4	4	4	4
		mem, reg reg, rem	1 0 0 0 0 1 0 W	mod reg mem	2-4	EA + 8 + W	EA + 8 + W	EA + 10 + 2 W	EA + 10 + 2 W
		reg, imm	1 1 1 1 0 1 1 W	1 1 0 0 0 reg	3-4	7	7	8	8
		mem, imm	1 1 1 1 0 1 1 W	mod 0 0 0 mem	3-6	EA + 11 + W	EA + 11 + W	EA + 11 + 2 W	EA + 11 + 2 W
		acc, imm	1 0 1 0 1 0 0 W		2-3	5	5	6	6
	AND	reg, reg	0 0 1 0 0 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	0 0 1 0 0 0 0 W	mod reg mem	2-4	EA + 8 + 2 W	EA + 6 + 1 W	EA + 12 + 4 W	EA + 8 + 2 W
		reg, mem	0 0 1 0 0 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		reg, imm	1 0 0 0 0 0 0 W	1 1 1 0 0 reg	3-4	5	5	6	6
		mem, imm	1 0 0 0 0 0 0 W	mod 1 0 0 mem	3-6	EA + 9 + 2 W	EA + 7 + 2 W	EA + 14 + 4 W	EA + 10 + 4 W
		acc, imm	0 0 1 0 0 1 0 W		2-3	5	5	6	6
	OR	reg, reg	0 0 0 0 1 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	0 0 0 0 1 0 0 W	mod reg mem	2-4	EA + 8 + 2 W	EA + 6 + 1 W	EA + 12 + 4 W	EA + 8 + 2 W
		reg, mem	0 0 0 0 1 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		reg, imm	1 0 0 0 0 0 0 W	1 1 0 0 1 reg	3-4	5	5	6	6
		mem, imm	1 0 0 0 0 0 0 W	mod 0 0 1 mem	3-6	EA + 9 + 2 W	EA + 7 + 2 W	EA + 14 + 4 W	EA + 8 + 2 W
		acc, imm	0 0 0 0 1 1 0 W		2-3	5	5	6	6
	XOR	reg, reg	0 0 1 1 0 0 1 W	1 1 reg reg	2	2	2	2	2
		mem, reg	0 0 1 1 0 0 0 W	mod reg mem	2-4	EA + 8 + 2 W	EA + 6 + 1 W	EA + 12 + 4 W	EA + 8 + 2 W
		reg, mem	0 0 1 1 0 0 1 W	mod reg mem	2-4	EA + 6 + W	EA + 6 + W	EA + 8 + 2 W	EA + 8 + 2 W
		reg, imm	1 0 0 0 0 0 0 W	1 1 1 1 0 reg	3-4	5	5	6	6
		mem, imm	1 0 0 0 0 0 0 W	mod 1 1 0 mem	3-6	EA + 9 + 2 W	EA + 7 + 2 W	EA + 14 + 4 W	EA + 8 + 2 W
		acc, imm	0 0 1 1 0 1 0 W		2-3	5	5	6	6

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Bit operation instructions	TEST1	reg 8, CL	0 0 0 1 0 0 0 0	1 1 0 0 0 reg	3	7	7	7	7
		mem 8, CL	0 0 0 0	mod 0 0 0 mem	3-5	EA + 11 + W	EA + 11 + W	EA + 13 + 2 W	EA + 13 + 2 W
		reg 16, CL	0 0 0 1	1 1 0 0 0 reg	3	7	7	7	7
		mem 16, CL	0 0 0 1	mod 0 0 0 mem	3-5	EA + 11 + W	EA + 11 + W	EA + 13 + 2 W	EA + 13 + 2 W
		reg 8, imm 3	1 0 0 0	1 1 0 0 0 reg	4	6	6	6	6
		mem 8, imm 3	1 0 0 0	mod 0 0 0 mem	4-6	EA + 8 + W	EA + 8 + W	EA + 10 + 2 W	EA + 10 + 2 W
		reg 16, imm 4	1 0 0 1	1 1 0 0 0 reg	4	6	6	6	6
		mem 16, imm 4	1 0 0 1	mod 0 0 0 mem	4-6	EA + 8 + W	EA + 8 + W	EA + 10 + 2 W	EA + 10 + 2 W
	NOT1	reg 8, CL	0 1 1 0	1 1 0 0 0 reg	3	7	7	7	7
		mem 8, CL	0 1 1 0	mod 0 0 0 mem	3-5	EA + 13 + 2 W	EA + 11 + 1 W	EA + 17 + 4 W	EA + 13 + 2 W
		reg 16, CL	0 1 1 1	1 1 0 0 0 reg	3	7	7	7	7
		mem 16, CL	0 1 1 1	mod 0 0 0 mem	3-5	EA + 13 + 2 W	EA + 11 + 1 W	EA + 17 + 4 W	EA + 13 + 2 W
		reg 8, imm 3	1 1 1 0	1 1 0 0 0 reg	4	6	6	6	6
		mem 8, imm 3	1 1 1 0	mod 0 0 0 mem	4-6	EA + 10 + 2 W	EA + 8 + 1 W	EA + 14 + 4 W	EA + 10 + 2 W
		reg 16, imm 4	1 1 1 1	1 1 0 0 0 reg	4	6	6	6	6
		mem 16, imm 4	1 1 1 1	mod 0 0 0 mem	4-6	EA + 10 + 2 W	EA + 8 + 1 W	EA + 14 + 4 W	EA + 10 + 2 W
	NOT1	CY	1 1 1 1 0 1 0 1		1	2	2	2	2

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Bit operation instructions	CLR1	reg 8, CL	0 0 0 1 0 0 1 0	1 1 0 0 0 reg	3	8	8	8	8
		mem 8, CL	0 0 1 0	mod 0 0 0 mem	3-5	EA + 14 + 2 W	EA + 12 + 1 W	EA + 18 + 4 W	EA + 14 + 2 W
		reg 16, CL	0 0 1 1	1 1 0 0 0 reg	3	8	8	8	8
		mem 16, CL	0 0 1 1	mod 0 0 0 mem	3-5	EA + 14 + 2 W	EA + 12 + 1 W	EA + 18 + 4 W	EA + 14 + 2 W
		reg 8, imm 3	1 0 1 0	1 1 0 0 0 reg	4	7	7	7	7
		mem 8, imm 3	1 0 1 0	mod 0 0 0 mem	4-6	EA + 11 + 2 W	EA + 9 + 1 W	EA + 15 + 4 W	EA + 10 + 2 W
		reg 16, imm 4	1 0 1 1	1 1 0 0 0 reg	4	7	7	7	7
		mem 16, imm 4	1 0 1 1	mod 0 0 0 mem	4-6	EA + 11 + 2 W	EA + 9 + 1 W	EA + 15 + 4 W	EA + 10 + 2 W
	SET1	reg 8, CL	0 1 0 0	1 1 0 0 0 reg	3	7	7	7	7
		mem 8, CL	0 1 0 0	mod 0 0 0 mem	3-5	EA + 13 + 2 W	EA + 11 + 1 W	EA + 17 + 4 W	EA + 13 + 2 W
		reg 16, CL	0 1 0 1	1 1 0 0 0 reg	3	7	7	7	7
		mem 16, CL	0 1 0 1	mod 0 0 0 mem	3-5	EA + 13 + 2 W	EA + 11 + 1 W	EA + 17 + 4 W	EA + 13 + 2 W
		reg 8, imm 3	1 1 0 0	1 1 0 0 0 reg	4	6	6	6	6
		mem 8, imm 3	1 1 0 0	mod 0 0 0 mem	4-6	EA + 10 + 2 W	EA + 8 + 1 W	EA + 14 + 4 W	EA + 10 + 2 W
		reg 16, imm 4	1 1 0 1	1 1 0 0 0 reg	4	6	6	6	6
		mem 16, imm 4	1 1 0 1	mod 0 0 0 mem	4-6	EA + 10 + 2 W	EA + 8 + 1 W	EA + 14 + 4 W	EA + 10 + 2 W
			2nd byte	3rd byte					
	CLR1	CY	1 1 1 1 1 0 0 0		1	2	2	2	2
		DIR	1 1 1 1 1 1 0 0		1	2	2	2	2
	SET1	CY	1 1 1 1 1 0 0 1		1	2	2	2	2
		DIR	1 1 1 1 1 1 0 1		1	2	2	2	2

in-struc-tion group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			76543210	76543210		RAM enable	RAM disable	RAM enable	RAM disable
Shift instructions	SHR	reg, 1	1101000W	11101 reg	2	8	8	8	8
		mem, 1	1101000W	mod101 mem	2-4	EA + 14 + 2W	EA + 12 + 1W	EA + 18 + 4W	EA + 14 + 2W
		reg, CL	1101001W	11101 reg	2	11 + 2n	11 + 2n	11 + 2n	11 + 2n
		mem, CL	1101001W	mod101 mem	2-4	EA + 17 + 2W + 2n	EA + 15 + 1W + 2n	EA + 21 + 4W + 2n	EA + 17 + 2W + 2n
		reg, imm 8	1100000W	11101 reg	3	9 + 2n	9 + 2n	9 + 2n	9 + 2n
		mem, imm 8	1100000W	mod101 mem	3-5	EA + 13 + 2W + 2n	EA + 11 + 1W + 2n	EA + 17 + 4W + 2n	EA + 13 + 2W + 2n
	SHRA	reg, 1	1101000W	11111 reg	2	8	8	8	8
		mem, 1	1101000W	mod111 mem	2-4	EA + 14 + 2W	EA + 12 + 1W	EA + 18 + 4W	EA + 14 + 2W
		reg, CL	1101001W	11111 reg	2	11 + 2n	11 + 2n	11 + 2n	11 + 2n
		mem, CL	1101001W	mod111 mem	2-4	EA + 17 + 2W + 2n	EA + 15 + 1W + 2n	EA + 21 + 4W + 2n	EA + 17 + 2W + 2n
		reg, imm 8	1100000W	11111 reg	3	9 + 2n	9 + 2n	9 + 2n	9 + 2n
		mem, imm 8	1100000W	mod111 mem	3-5	EA + 13 + 2W + 2n	EA + 11 + 1W + 2n	EA + 17 + 4W + 2n	EA + 13 + 2W + 2n

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Rotation instructions	ROL	reg, 1	1 1 0 1 0 0 0 W	1 1 0 0 0 reg	2	8	8	8	8
		mem, 1	1 1 0 1 0 0 0 W	mod 0 0 0 mem	2-4	EA + 14 + 2W	EA + 12 + 1 W	EA + 18 + 4 W	EA + 14 + 2 W
		reg, CL	1 1 0 1 0 0 1 W	1 1 0 0 0 reg	2	11 + 2 n	11 + 2 n	11 + 2 n	11 + 2 n
		mem, CL	1 1 0 1 0 0 1 W	mod 0 0 0 mem	2-4	EA + 17 + 2 W + 2 n	EA + 15 + 1 W + 2 n	EA + 21 + 4 W + 2 n	EA + 17 + 2 W + 2 n
		reg, imm 8	1 1 0 0 0 0 0 W	1 1 0 0 0 reg	3	9 + 2 n	9 + 2 n	9 + 2 n	9 + 2 n
		mem, imm 8	1 1 0 0 0 0 0 W	mod 0 0 0 mem	3-5	EA + 13 + 2 W + 2 n	EA + 11 + 1 W + 2 n	EA + 17 + 4 W + 2 n	EA + 13 + 2 W + 2 n
	ROR	reg, 1	1 1 0 1 0 0 0 W	1 1 0 0 1 reg	2	8	8	8	8
		mem, 1	1 1 0 1 0 0 0 W	mod 0 0 1 mem	2-4	EA + 14 + 2 W	EA + 12 + 1 W	EA + 18 + 4 W	EA + 14 + 2 W
		reg, CL	1 1 0 1 0 0 1 W	1 1 0 0 1 reg	2	11 + 2 n	11 + 2 n	11 + 2 n	11 + 2 n
		mem, CL	1 1 0 1 0 0 1 W	mod 0 0 1 mem	2-4	EA + 17 + 2 W + 2 n	EA + 15 + 1 W + 2 n	EA + 21 + 4 W	EA + 17 + 2 W + 2 n
		reg, imm 8	1 1 0 0 0 0 0 W	1 1 0 0 1 reg	3	9 + 2 n	9 + 2 n	9 + 2 n	9 + 2 n
		mem, imm 8	1 1 0 0 0 0 0 W	mod 0 0 1 mem	3-5	EA + 13 + 2 W + 2 n	EA + 11 + 1 W + 2 n	EA + 17 + 4 W + 2 n	EA + 13 + 2 W + 2 n

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			76543210	76543210		RAM enable	RAM disable	RAM enable	RAM disable
Rotate instruction	ROLC	reg, 1	1101000W	11010 reg	2	8	8	8	8
		mem, 1	1101000W	mod 010 mem	2-4	EA + 14 + 2 W	EA + 12 + 1 W	EA + 18 + 4 W	EA + 14 + 2 W
		reg, CL	1101001W	11010 reg	2	11 + 2 n	11 + 2 n	11 + 2 n	11 + 2 n
		mem, CL	1101001W	mod 010 mem	2-4	EA + 17 + 2 W + 2 n	EA + 15 + 1 W + 2 n	EA + 21 + 4 W + 2 n	EA + 17 + 2 W + 2 n
		reg, imm 8	1100000W	11010 reg	3	9 + 2 n	9 + 2 n	9 + 2 n	9 + 2 n
		mem, imm 8	1100000W	mod 010 mem	3-5	EA + 13 + 2 W + 2 n	EA + 11 + 1 W + 2 n	EA + 17 + 4 W + 2 n	EA + 13 + 2 W + 2 n

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Rotate instruction	RORC	reg, 1	1 1 0 1 0 0 0 W	1 1 0 1 1 reg	2	8	8	8	8
		mem, 1	1 1 0 1 0 0 0 W	mod 0 1 1 mem	2-4	EA + 14 + 2 W	EA + 12 + 1 W	EA + 18 + 4 W	EA + 14 + 2 W
		reg, CL	1 1 0 1 0 0 1 W	1 1 0 1 1 reg	2	11 + 2 n	11 + 2 n	11 + 2 n	11 + 2 n
		mem, CL	1 1 0 1 0 0 1 W	mod 0 1 1 mem	2-4	EA + 17 + 2 W + 2 n	EA + 15 + 1 W + 2 n	EA + 21 + 4 W + 2 n	EA + 17 + 2 W + 2 n
		reg, imm 8	1 1 0 0 0 0 0 W	1 1 0 1 1 reg	3	9 + 2 n	9 + 2 n	9 + 2 n	9 + 2 n
		mem, imm 8	1 1 0 0 0 0 0 W	mod 0 1 1 mem	3-5	EA + 13 + 2 W + 2 n	EA + 11 + 1 W + 2 n	EA + 17 + 4 W + 2 n	EA + 13 + 2 W + 2 n

in-struc-tion group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Shift instructions	SHL	reg, 1	1 1 0 1 0 0 0 W	1 1 1 0 0 reg	2	8	8	8	8
		mem, 1	1 1 0 1 0 0 0 W	mod 1 0 0 mem	2-4	EA + 14 + 2 W	EA + 12 + 1 W	EA + 18 + 4 W	EA + 14 + 2 W
		reg, CL	1 1 0 1 0 0 1 W	1 1 1 0 0 reg	2	11 + 2 n	11 + 2 n	11 + 2 n	11 + 2 n
		mem, CL	1 1 0 1 0 0 1 W	mod 1 0 0 mem	2-4	EA + 17 + 2 W + 2 n	EA + 15 + 1 W + 1 n	EA + 21 + 4 W + 2 n	EA + 17 + 2 W + 2 n
		reg, imm 8	1 1 0 0 0 0 0 W	1 1 1 0 0 reg	3	9 + 2 n	9 + 2 n	9 + 2 n	9 + 2 n
		mem, imm 8	1 1 0 0 0 0 0 W	mod 1 0 0 mem	3-5	EA + 13 + 2 W + 2 n	EA + 11 + 1 W + 2 n	EA + 17 + 4 W + 2 n	EA + 13 + 2 W + 2 n

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Subroutine control instructions	CALL	near proc	1 1 1 0 1 0 0 0		3	–	–	22 + 2 W	18 + 2 W
		regptr 16	1 1 1 1 1 1 1 1	1 1 0 1 0 reg	2	–	–	22 + 2 W	18 + 2 W
		memptr 16	1 1 1 1 1 1 1 1	mod 0 1 0 mem	2-4	–	–	EA + 26 + 4 W	EA + 24 + 4 W
		far proc	1 0 0 1 1 0 1 0		5	–	–	38 + 4 W	34 + 4 W
		memptr 32	1 1 1 1 1 1 1 1	mod 0 1 1 mem	2-4	–	–	EA + 36 + 8 W	EA + 24 + 8 W
	RET		1 1 0 0 0 0 1 1		1	–	–	20 + 2 W 5)	20 + 2 W 5)
		pop value	1 1 0 0 0 0 1 0		3	–	–	20 + 2 W	20 + 2 W
			1 1 0 0 1 0 1 1		1	–	–	29 + 4 W	29 + 4 W
		pop value	1 1 0 0 1 0 1 0		3	–	–	30 + 4 W 6)	30 + 4 W 6)

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Stack manipulation instructions	PUSH	mem 16	1 1 1 1 1 1 1 1	mod 1 1 0 mem	2-4	–	–	EA + 18 + 4 W	EA + 14 + 4 W
		reg 16	0 1 0 1 0 reg		1	–	–	10 + 2W 2) 14 + 2W 4)	6 2) 10 + 2W 8)
		sreg	0 0 0 sreg 1 1 0		1	–	–	11 + 2 W	7
		PSW	1 0 0 1 1 1 0 0		1	–	–	10 + 2 W	6
		R	0 1 1 0 0 0 0 0		1	–	–	82 + 16 W	50
		imm	0 1 1 0 1 0 S 0		2-3	–	–	14 + 2 W	10
	POP	mem 16	1 0 0 0 1 1 1 1	mod 0 0 0 mem	2-4	–	–	EA + 16 + 4 W	EA + 12 + 2 W
		reg 16	0 1 0 1 1 reg		1	–	–	12 + 2 W 3,4)	12 + 2 W 3,4)
		sreg	0 0 0 sreg 1 1 1		1	–	–	13 + 2 W	13 + 2 W
		PSW	1 0 0 1 1 1 0 1		1	–	–	14 + 2 W	14 + 2 W
		R	0 1 1 0 0 0 0 1		1	–	–	82 + 16 W	58
	PREPARE	imm 16, imm 8	1 1 0 0 1 0 0 0		4	– 9)	– 9)	– 9)	– 9)
	DISPOSE		1 1 0 0 1 0 0 1		1	–	–	12 + 2 W	12 + 2 W
Branch instructions	BR	near-label	1 1 1 0 1 0 0 1		3	–	–	12	12
		short-label	1 1 1 0 1 0 1 1		2	–	–	12	12
		regptr 16	1 1 1 1 1 1 1 1	1 1 1 0 0 reg	2	–	–	13	13
		memptr 16	1 1 1 1 1 1 1 1	mod 1 0 0 mem	2-4	–	–	EA + 17 + 2 W	EA + 17 + 2 W
		far-label	1 1 1 0 1 0 1 0		5	–	–	15	15
		memptr 32	1 1 1 1 1 1 1 1	mod 1 0 1 mem	2-4	–	–	EA + 25 + 4 W	EA + 25 + 4 W

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
Conditional branch instructions	BV	short-label	0 1 1 1 0 0 0 0		2	–	–	15/8	
	BNV	short-label	0 0 0 1		2	–	–	15/8	
	BC BL	short-label	0 0 1 0		2	–	–	15/8	
	BNC BNL	short-label	0 0 1 1		2	–	–	15/8	
	BE BZ	short-label	0 1 0 0		2	–	–	15/8	
	BNE BNZ	short-label	0 1 0 1		2	–	–	15/8	
	BNH	short-label	0 1 1 0		2	–	–	15/8	
	BH	short-label	0 1 1 1		2	–	–	15/8	
	BN	short-label	1 0 0 0		2	–	–	15/8	
	BP	short-label	1 0 0 1		2	–	–	15/8	
	BPE	short-label	1 0 1 0		2	–	–	15/8	
	BPO	short-label	1 0 1 1		2	–	–	15/8	
	BLT	short-label	1 1 0 0		2	–	–	15/8	
	BGE	short-label	1 1 0 1		2	–	–	15/8	
	BLE	short-label	1 1 1 0		2	–	–	15/8	
	BGT	short-label	1 1 1 1		2	–	–	15/8	
	DBNZNE	short-label	1 1 1 0 0 0 0 0		2	–	–	17/8	17/8
	DBNZE	short-label	0 0 0 1		2	–	–	17/8	17/8
	DBNZ	short-label	0 0 1 0		2	–	–	17/8	17/8
	BCWZ	short-label	0 0 1 1		2	–	–	15/8	15/8
	BTCLR *	mem 8 imm 3 short-label	0 0 0 0 1 1 1 1	1 0 0 1 1 1 0 0	5	29	29	–	–

*Newly added instruction for the μPD70322/70320.

Instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		RAM enable	RAM disable	RAM enable	RAM disable
CPU control instructions	HALT		1 1 1 1 0 1 0 0		1	–	–	–	–
	STOP *2		0 0 0 0 1 1 1 1	1 0 0 1 1 1 1 0	2	–	–	–	–
	POLL		1 0 0 1 1 0 1 1		1	–	–	–	4 + i 10)
	DI		1 1 1 1 1 0 1 0		1	4	4	4	4
	EI		1 1 1 1 1 0 1 1		1	12	12	12	12
	BUSLOCK		1 1 1 1 0 0 0 0		1	2	2	2	2
	FP01	fp-op	1 1 0 1 1 X X X	1 1 Y Y Y Z Z Z	2	–	–	60 + 10 W	48 + 10 W
	*3	fp-op, mem	1 1 0 1 1 X X X	mod Y Y Y mem	2-4	–	–	60 + 10 W	48 + 10 W
	FP02	fp-op	0 1 1 0 0 1 1 X	1 1 Y Y Y Z Z Z	2	–	–	60 + 10 W	48 + 10 W
	*3	fp-op, mem	0 1 1 0 0 1 1 X	mod Y Y Y mem	2-4	–	–	60 + 10 W	48 + 10 W
	NOP		1 0 0 1 0 0 0 0		1	4	4	4	4
	*1		0 0 1 sreg 1 1 0		1	2	2	2	2

*1: DS0:, DS1:, PS:, SS:
*2: Newly added instruction for the μPD70322/70320
*3: Does not execute on the μPD 70322/70320, but generates an interrupt.

Please refer to the notes on the last page.

instruction group	mnemonic	operand	operation code		no. of bytes	Byte		Word	
			76543210	76543210		RAM enable	RAM disable	RAM enable	RAM disable
Interrupt instructions	BRK	3	11001100		1	–	–	55 + 10 W	43 + 10 W
		imm 8 (≠3)	11001101		2	–	–	56 + 10 W	44 + 10 W
	BRKV		11001110		1	–	–	55 + 10 W	43 + 10 W
	RETI		11001111		1	–	–	43 + 6 W	35 + 2 W
	RETRBI *		00001111	10010001	2	–	–	12	12
	FINT *		00001111	10010010	2	2	2	2	2
	CHKIND	reg 16, mem 32	01100010	mod reg mem	2-4	–	–	–	–

*Newly added instruction for the μPD70322/70320.

Instruction execution timing with prefix instructions

- Repeating block transfer execution timing, which is the combination of primitive block transfer instruction and repeat prefix instruction, are indicated on the table.
  e.g. MOVBK (rep) 16 + (16 + 2W) n ⟩ cw registor value
  if (w = 0 ⇒ n = 10000 H

- All the other cases 2 clock execution time is necessary per each prefix instruction

  BUSLOCK: 2 clock

  Segment override prefix: 2 clock

  Repeat prefix (in the case not combining with primitive block transfer instruction)

  : 2 clock

e.g.	MOV ES : M1.AL:		2 + (EA + 4 + W)
	BUSLOCK REP MOVBK	S,D:	2 + {(16 + 2W)n} n = cw
	REP MOV BK	S,D:	(16 + 2W)n
	REP MOV	S1, AL:	2 + (EA + 4 + W)
	MOV BK	S1,D1:	20 + 2W

NOTES

NOTE 1: 1 Byte instruction INC/Dec. reg 16.

as for NOTE 7 2 Byte instruction INC/Dec. reg 16.

NOTE 2: 1 Byte instruction PUSH reg 16.

as for NOTE 8 2 Byte instruction starting from operating code FFH.

NOTE 3: 1 Byte instruction POP reg 16.

w for NOTE 4 2 Byte instruction starting from operating code 8 FH

Both number of Execution clock are the same (NOTE 3 and 4)

NOTE 4: 2 Byte instruction POP reg. starting operation code 8 FH. Refer NOTE 3

NOTE 5: RET near (ADJ) means the instruction which is return inside segment and moreover correct SP of ADJ portion after return (action)
Operating code starts from C2H.

NOTE 6: RET far (ADJ) means the instruction which is return outside segment and moreover correct SP of ADJ portion after return (action)
Operation code starts from CAH

NOTE 7: 2 Byte instruction INC/DEC reg 8.
refer to NOTE 1

NOTE 8: 2 Byte instruction PUSH reg. starting from operation code FFH.
refer to NOTE 2.

NOTE 9: PREPARE Number of execution clock in

CASE 1: i mm 8 = 0 27 + 2W

CASE 2: i mm 8 = 1 39 + 4W

CASE 3: i mm 8 = n.n >1 46 + 19 (n - 1) + 4 nw

NOTE 10: i = No. of Samplings

Instruction	Byte operation		Word operation	
	RAM enable	RAM disable	RAM enable	RAM disable
MS SFR <- M	22+(2+W)	20+(W-1) min: 20	22+(2+W)	21+2W
MS M <- SFR	22+(W-2) min: 22	20+W	22+2W	18+(2+W)+2
MS search SFR <- M	25+(2+W)	25+(2+W)	-	-
MS search M <- SFR	35+(2+W)	35+(W-1) min: 35	-	-
DMA SS	20+(W-1)+(W-2) min: 20	20+(W-1)+(W-2) min: 20	21+4W	21+4W
DMA DR MEM <- I/O I/O <- MEM	10+15n+(W-2)*n min: 10+15n	10+15n+(W-2)*n min: 10+15n	10+15n+2W*n	10+15n+2W*n
DMA ST MEM <- I/O I/O <- MEM	17+(W-2) min: 17	17+(W-2) min: 17	17+2W	17+2W
DMA B	12+12n+(W-1)+ (W-1)*(2n-1)+W min: 12+12n	12+12n+(W-1)+ (W-1)(2n-1)+W min: 12+12n	13+(14+4W)*n	13+(14+4W)*n
BTCLR	29	29	-	-
RETRBI	-	-	12	12
BREAK	-	-	58+10W	46+10W
INTR	-	-	62+10W	50+10W
CS INT	-	-	19	19

n = repitition mumber
CS INT = register bank interrupt
W = number of waits
SS = single step mode
DR = demand release mode
B = burst mode
ST = single transfer mode

Part 2.

μPD 70330/70332

(V35)

User´s Manual

CHAPTER 1 INTRODUCTION

The uPD70332 (or V35™) is a single-chip microcomputer like the uPD70322 (or V25™), but with the external data bus extended to 16 bits. This enables high speed 16-bit data processing and reduction of external circuits.

The uPD70330 is a product provided by excluding ROM from the uPD70332.

1.1 Features:

- o Internal 16-bit architecture and external 16-bit data bus
- o Software compatible with uPD70108 and uPD70116 (in the native mode). Additional instructions are included.
- o Minimum instruction cycle: 400 ns at 10 MHz
- o Internal memory
  - · ROM: 16384 bytes (uPD70332 only)
  - · RAM: 256 bytes
- o Memory space: 1M bytes
- o Input port with comparator (port T): eight bits
- o I/O lines (Input port: four bits,
  Input/output ports: 20 bits)
- o Serial interface: two channels
  - · Dedicated baud rate generator
  - · Asynchronous mode, I/O interface mode
- o Interrupt controller
  - · Programmable priority (eight levels)
  - · Three interrupt service functions
  - · Vectored interrupt, register bank switching, macro service
- o DRAM, pseudo SRAM refresh function
- o DMA controller: two channels
- o 16-bit timer: two channels
- o 20-bit time base counter: one channel
- o Clock generator
- o Programmable wait function
- o Standby function (STOP or HALT)

1.2 Ordering Information for V35 family:

Article	Ext.Data Bus	Package	Ext.Clock	Internal Rom
uPD70332L-xxx	16 bit	84pinPLCC	10 MHz	Mask Rom
uPD70330L	16 bit	84pinPLCC	10 MHz	Romless
uPD70332L-8-xxx	16 bit	84pinPLCC	16 MHz	Mask Rom
uPD70330L-8	16 bit	84pinPLCC	16 MHz	Romless
uPD70P322L	8/16bit	84pinPLCC	10 MHz	OTP
uPD70P322K	8/16bit	84pinCLCC	10 MHz	EPROM

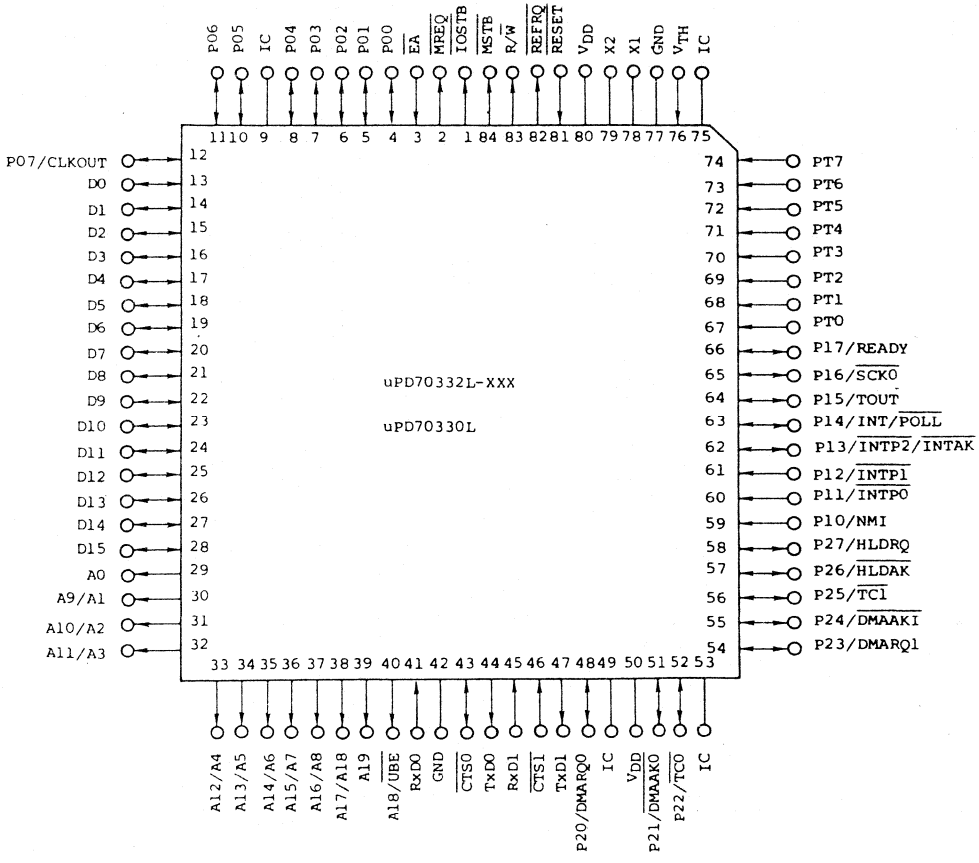
Note:

xxx = Rom Code

CLCC = Ceramic LCC with Window

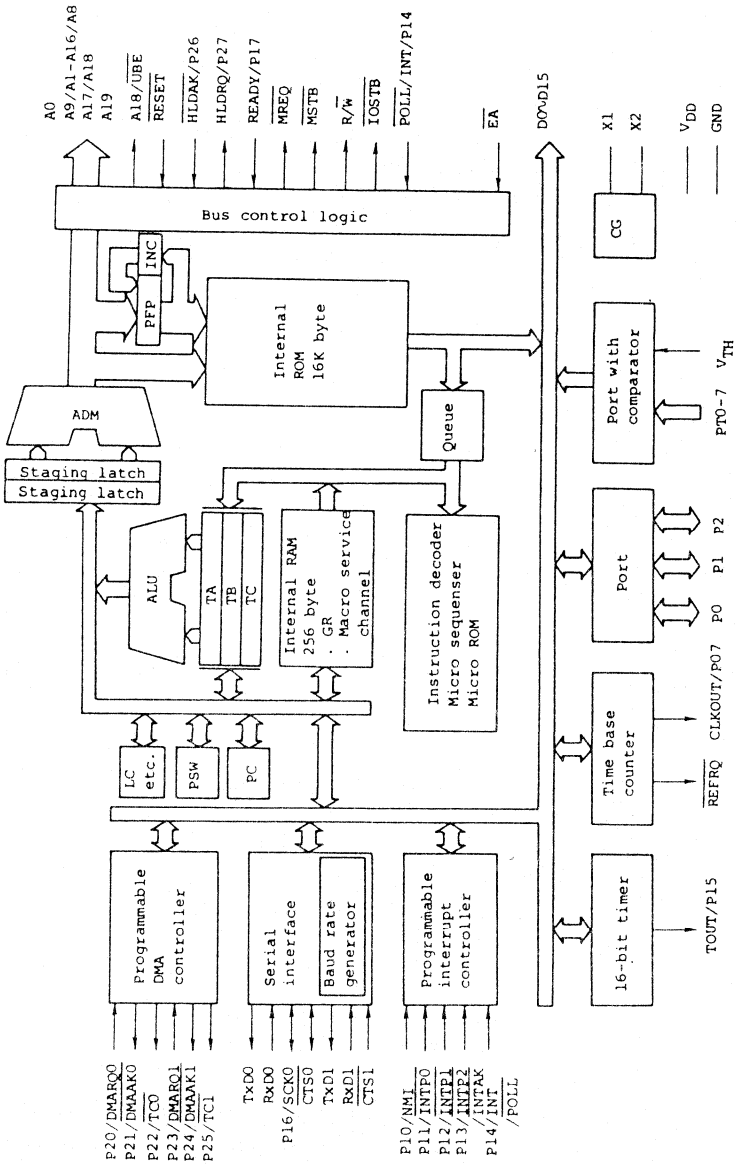
1.3 Pin Configuration

84-pin PLCC (top view)



Caution: IC pins should be tied to a high level through external pull-up resistors. Tie pin 9 to low level through a pull-down resistor.

1.4 uPD70332, 70330 Block Diagram



1.5 PIN FUNCTIONS : Port Pins

Pin name	I/O	Port function	Control function
P00-P06	I/O	Eight-bit bidirectional input/output port for which input or output mode can be specified bitwise.	-
P07/CLKOUT	I/O/O		System clock output
P10/NMI	I/I	Nonmaskable interrupt request input and input port	-
P11/$\overline{\text{INTP0}}$		External interrupt request input and input port	
P12/$\overline{\text{INTP1}}$			
P13/$\overline{\text{INTP2}}$/$\overline{\text{INTAK}}$	I/I/O		INT acknowledge signal output
P14/$\overline{\text{POLL}}$/INT	I/O/I/	Bidirectional input/output port and $\overline{\text{POLL}}$ input	External interrupt request input
P15/TOUT	I/O/I	Bidirectional input/output port; input or output mode can be specified bitwise	Timer output
P16/$\overline{\text{SCK0}}$			Serial clock output
P17/READY	I/O/I		READY input
P20/DMARQ0	I/O/I	Eight-bit bidirectional input/output port; input or output mode can be specified bitwise	DMA request input (CH0)
P21/$\overline{\text{DMAAK0}}$	I/O/O		DMA acknowledge output (CH0)
P22/$\overline{\text{TC0}}$			DMA termination output (CH0)
P23/DMARQ1	I/O/I		DMA request input (CH1)
P24/$\overline{\text{DMAAK1}}$	I/O/O		DMA acknowledge output (CH1)
P25/$\overline{\text{TC1}}$			DMA termination output (CH1)
P26/$\overline{\text{HLDAK}}$	I/O/O		HOLD acknowledge output
P27/HLDRQ	I/O/I		HOLD input
PT0-PT7	I	Input port with 8-bit comparator	-

1.6 Pin Functions: Pins Other Than Ports

Pin name	I/O	Function
TXD0	O	Serial data output
TXD1		
RXD0	I	Serial data input
RXD1		
$\overline{CTS0}$	I/O	CTS input in synchronous mode. Receive clock input/output in I/O interface mode.
$\overline{CTS1}$	I	CTS input
$\overline{REFRQ}$	O	DRAM refresh pulse output
V_{TH}	I	Comparator reference voltage input
$\overline{RESET}$		Reset signal input
$\overline{EA}$		Input to set ROM less mode
X1		Crystal connection pins for system clock oscillator. External clock is input with opposing phases to the X1 and X2 pins.
X2		
D0-D15	I/O	16-bit data bus
A0	O	Address LSB output used for low-order memory bank selection.
A9/A1-A16/A8, A17/A18, A19		19-bit address is output in a time division manner.
A18/$\overline{UBE}$		Address bit 18 output and high-order memory bank selection signal output in a time division manner.
$\overline{MREQ}$		Output indicating that memory bus cycle or I/O bus cycle is started. High-order address strobe output.

(Cont'd)

Pin name	I/O	Function
$\overline{MSTB}$	O	Memory read or memory write strobe output. Low-order address strobe output.
$R/\overline{W}$		Read or write cycle identification signal output.
$\overline{IOSTB}$		I/O read or I/O write strobe output. Low-order address strobe output.
V_{DD}	—	Positive power supply pin. Connect both the VDD pins.
GND		GND pin. Connect both the GND pins.

2. CPU

The uPD70332, uPD70330 has CPU which is software-compatible with uPD70116/uPD70108 in the native mode.

2.1 Registers

The uPD70332 and uPD70330 CPUs have general purpose register sets compatible with uPD70116/uPD70108. They also have various special function registers to control on-chip peripheral hardware. All the registers are mapped in memory space. Particularly, the general purpose register set is mapped in internal RAM, and a maximum of eight banks of register set can be provided on internal RAM.

The addresses of the registers can be relocated in 4K-byte units. The address is specified by using the internal data area base register (IDB), which is a special function register. (See 2.4.2.)

2.1.1 Register banks

The general purpose register set is mapped in the internal RAM area. The bank form is used for the general purpose register set, and up to eight banks can be set. Thirty-two bytes are used per bank. Banks 0 and 1 are also used for macro service channel (see 3.4.5) and DMA service channel (see 5.3.3). They can also be accessed as data memory (see 2.4.4).

Normally, the CPU executes a program by using register bank 7; when an interrupt occurs, an automatic switch is made to another register bank. To return to the former register bank from the register bank selected when an interrupt occurs, the return instruction from interrupt RETRBI (uPD70332/70330 additional instruction for uPD70108/70116) must be used. (See 3.4.2.)

Fig. 2-1 shows the register bank configuration. (+00H) and (+01H) in a register bank become a reserved area when the register bank is used. The general purpose register set is mapped in the area of (+08H) to (+1FH) offsets from the start address of each register bank. The (+02H) to (+07H) area is used when register bank switching is performed and cannot be used as general purpose.

The value loaded into PC when register bank switching is performed (that is, offset of the start address of the interrupt service routine) is set in the area (+02H).

(+04H) is an area in which PSW is saved when register bank switching is performed.

(+06H) is an area in which PC is saved when register bank switching is performed. (See 3.4.2.)

After reset, register bank 7 is automatically selected. Initialization of segment registers (see 2.1.4) after reset is made for register bank 7.

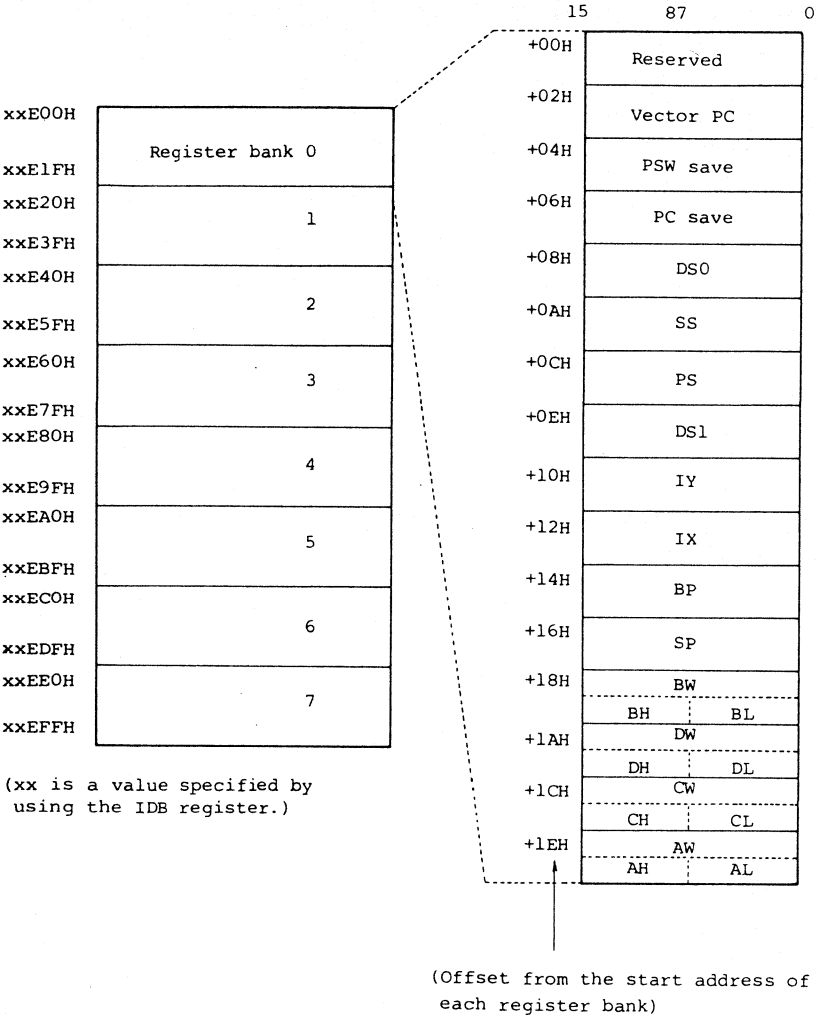


Fig. 2-1 Register Bank Configuration

2.1.2 General purpose registers (AW, BW, CW, and DW)

The general purpose registers include four 16-bit registers. The registers can be accessed not only as 16-bit registers, but also as 8-bit registers (AH, AL, BH, BL, CH, CL, DH, and DL) by dividing each register into the high-order and low-order portions (each eight bits).

The registers are used as 8-bit or 16-bit registers for a wide range of instruction such as transfer, arithmetic, and logical operations.

Each register can also be used as a default register in processing specific instructions as follows:

AW: Word multiplication/division, word input/output, data conversion
AL: Byte multiplication/division, byte input/output, translation, BCD rotation, data conversion
AH: Byte multiplication/division
BW: Translation
CW: Loop control branch, repeat prefix
CL: Shift instruction, rotation instruction, BCD operation
DW: Word multiplication/division, indirect addressing input/output

The registers are mapped in internal RAM. The register address is found by the following expression:
(IDB register(Note) value x 4096) + (0E00H) + (register bank number x 32) + (offset for each register)

Note: See 2.4.2 for the IDB register.

Table 2-1 General Purpose Register Offset

Register	Offset	Register	Offset
AW	1EH	AL	1EH
		AH	1FH
BW	18H	BL	18H
		BH	19H
CW	1CH	CL	1CH
		CH	1DH
DW	1AH	DL	1AH
		DH	1BH

2.1.3 Pointers (SP and BP) and index registers (IX and IY)

BP, IX, and IY are 16-bit registers used as a base pointer or index registers in memory access by using based addressing (BP), indexed addressing (IX, IY), based indexed addressing (BP, IX, IY), etc. SP is used as stack pointer. Like the general purpose registers, they are used for instructions such as transfer, arithmetic and logical operations, in which case they cannot be used as 8-bit registers. Each register can also be used as a default register in the following specific processes:

SP: Stack operation
IX: Block transfer, BCD string operation source
IY: Block transfer, BCD string operation destination

The registers are mapped in internal RAM. The register address is found by the following expression:

(IDB register(Note) valuex4096) + (0E00H) +
(register bank numberx32) + (offset for each register)

Note: See 2.4.2 for the IDB register.

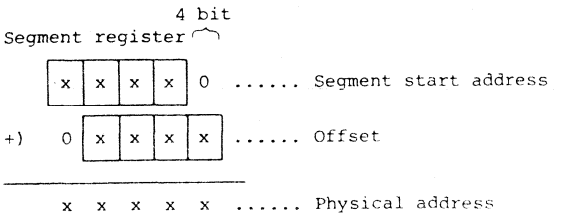
Table 2-2 Pointer and Index Register of Offset

Register	Offset
SP	16H
BP	14H
IX	12H
IY	10H

2.1.4 Segment registers (PS, SS, DS0 and DS1)

The CPU divides memory space into 64k-byte logical segments. Each segment register is used to specify the start address of the corresponding segment. Offset from the start address is specified by using another register or effective address.

Thus, the physical address is generated as shown below:



The four segment registers PS (Program Segment), SS (Stack Segment), DS0 (Data Segment 0), and DS1 (Data Segment 1) are included. The segments are used for:

PS: Program fetch
SS: Stack operation instruction, addressing with BP as base register
DS0: General variable access, source block data access with block transfer instruction, etc.
DS1: Destination block data access with block transfer instruction, etc.

Another segment can be used instead of DS0 by using a segment override prefix. Likewise, another segment can be used instead of SS in addressing with BP as the base register.

At reset, register bank 7 PS is initialized to FFFFH and SS, DS0, and DS1 are initialized to 0000H.

The registers are mapped in internal RAM. The register address is found by the following expression:

(IDB register(Note) valuex4096) + (0E00H) +
(register bank numberx32) + (offset for each register)

Note: See 2.4.2 for the IDB register.

Table 2-3 Segment Register Offset

Register	Offset
DS0	08H
DS1	0EH
SS	0AH
PS	0CH

2.1.5 Internal data area base register (IDB)

The IDB register is an 8-bit register for determining the address of the internal data area (see 2.4.1), which

includes internal RAM (also used for general registers) and special function registers (see 2.4.3) used for on-chip peripheral hardware control, etc. The IDB register can be referenced at two addresses FFFFFH and (IDB register value x 4096 + FFFH). (See 2.4.2.)

2.1.6 Special function registers

The uPD70330 and uPD70332 contain registers having special functions for on-chip peripheral hardware control. The registers are memory-mapped in the special function register section of the internal data area; reading and writing of the registers is done as with normal memory (See 2.4.3.).

The additional instruction BTCLR (see 13.1) can be used only for the special function registers.

2.2 Program Counter (PC)

The program counter (PC) is a 16-bit binary counter which holds the offset for the program memory address of the program to be executed by the CPU.

PC is incremented each time an instruction byte is fetched from the instruction queue. When a branch, call, return, or break instruction is executed, a new location is loaded into PC.

At reset, 0000H is loaded into PC. Since PS is initialized to FFFFH at reset, the CPU starts program execution at address FFFF0H after reset.

2.3 PSW (Program Status Word)

PSW (program status word) consists of the six status flags, five control flags, and two user flags:

o Status flags
  - V (Overflow)
  - S (Sign)
  - Z (Zero)
  - AC (Auxiliary Carry)
  - P (Parity)
  - CY (Carry)

o Control flags
  - RB0-RB2 (Register Bank 0-2)
  - DIR (Direction)
  - IE (Interrupt Enable)
  - BRK (Break)
  - $\overline{\text{IBRK}}$ (I/O Break)

o User flags
  - F0 (user Flag 0)
  - F1 (user Flag 1)

The status flags are automatically set to 1 or cleared according to the instruction execution result (data value). The CY flag can also be directly set, cleared, or inverted by using an instruction.

The control flags are set or cleared to control CPU operation by instructions. the IE and BRK flags are always cleared when interrupt service is started.
The user flags can be set, cleared, and tested by using instructions as the user desires.

The PSW bit configuration is shown below:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSW	1	RB2	RB1	RB0	V	DIR	IE	BRK	S	Z	F1	AC	F0	P	$\overline{\text{IBRK}}$	CY

The PSW contents can be saved into and restored from a stack by using PUSH and POP instructions. When the PSW contents are restored by using the POP PSW instruction, bits 12-14 (BR0-BR2) are not restored in PSW.

Also, the low-order eight bits of PSW can be saved into or restored from the AH register by using the MOV instruction. When an interrupt occurs the PSW contents are automatically saved in a stack before IE and BRK are cleared.

When RESET is input, RB0-RB2 and $\overline{IBRK}$ are set to 1, and other flags are cleared.

2.3.1 CY (Carry Flag)

(1) Binary addition and subtraction

In byte operations, when a carry from bit 7 of the operation result or a borrow is made, CY is set; when it is not made, CY is cleared. In word operations, when a carry from bit 15 of the operation result or a borrow is made, CY is set; when it is not made, CY is cleared. CY is not changed by execution of increment or decrement instructions.

(2) Logical operation

CY is cleared regardless of the operation result.

(3) Binary multiplication

If AH is set to 0 as a result of unsigned byte operation, CY is cleared; if it is not set to 0, CY is set.

If AH is used as an AL sign extension as a result of a signed byte operation, CY is cleared; when it is not used as an AL sign extension, CY is set.

If DW is set to 0 as a result of an unsigned word operation, CY is cleared; when it is not set to 0, CY is set.

If DW is used as an AW sign extension as a result of a signed word operation, CY is cleared; when it is not used as an AW sign extension, CY is set.

In 8-bit immediate operations, CY is cleared if the product is within 16 bits; if the product exceeds 16 bits, CY is set.

(4) Binary division

Undefined

(5) Shift/rotate

In shift and rotate containing CY, CY is set if the bit shifted to CY is set to 1; if 0, CY is cleared.

2.3.2 P (Parity Flag)

(1) Binary addition and subtraction, logical operation, and shift

If the number of the bits set to 1 in the low-order eight bits of the operation result is even, P is set; when it is odd, P is cleared. If the result is all 0, P is set.

(2) Binary multiplication and division

Undefined

2.3.3 AC (Auxiliary Flag)

(1) Binary addition and subtraction

In byte operations, AC is set if a carry from the low-order four bits to the high-order four bits or a borrow from the high-order four bits to the

low-order four bits is made; otherwise, AC is cleared.

In work operation, AC is set if a carry from the low-order four bits of the low-order byte to the four bits of the byte is made or if a borrow from the high-order four bits of the low-order byte to the low-order four bits of the byte is made; if it is not made, AC is cleared.

(2) Logical operation, binary multiplication and division, and shift/rotate

Undefined

2.3.4 Z (Zero Flag)

(1) Binary addition and subtraction, logical operation, and shift/rotate

If eight bits of the byte operation result or 16 bits of the word operation result are set to all 0, Z is set; if not all 0, Z is cleared.

(2) Binary multiplication and division

Undefined

2.3.5 S (Sign Flag)

(1) Binary addition and subtraction, logical operation, and shift/rotate

In byte operation, S is set if bit 7 of the result is set to 1; if it is set to 0, S is cleared.

In word operation, S is set if bit 15 of the result is set to 1; if it is set to 0, S is cleared.

(2) Binary multiplication and division

Undefined

2.3.6 V (Overflow Flag)

(1) Binary addition and subtraction

In byte operation, V is set if carry from bit 6 and bit 7 differ; if the same, V is cleared. In word operation, V is set carry from bit 14 and bit 15 differ; if the same, V is cleared.

(2) Logical operation

V is cleared regardless of the operation result.

(3) Binary multiplication

If AH is set to 0 as a result of unsigned byte operation, V is cleared; if AH is not set to 0, V is set.

If AH is used as an AL sign extension as a result of signed byte operation, V is cleared; if it is not used as AL sign extension, V is set.

If DW is set to 0 as a result of unsigned word operation, V is cleared; if it is not set to 0, V is set.

If DW is used as an AW sign extension as a result of signed word operation, V is cleared; if it is not used as AW sign extension, V is set.

If 8-bit immediate operation, if the product is within 16 bits, V is cleared; if it exceeds 16 bits, V is set.

(4) Binary division

V is cleared regardless of the operation result.

(5) Shift/rotate

In the operation result of left 1-bit shift/rotate,
when CY = most significant bit: V is cleared
when CY ≠ most significant bit: V is set

In the operation result of right 1-bit shift/rotate,
when most significant bit = second most significant bit: V is cleared
when most significant bit ≠ second most significant bit: V is set

In multibit shift/rotate, V is undefined

2.3.7 $\overline{\text{IBRK}}$ (I/O Break Flag)

$\overline{\text{IBRK}}$ controls software interrupt occurrence in input/output instruction execution.

When $\overline{\text{IBRK}}$ = 0, a software interrupt (interrupt vector 19) occurs if an attempt is made to execute an input/output instruction. Thus, input/output instruction can be emulated by using software.

When $\overline{\text{IBRK}}$ = 1, a software interrupt does not occur even if an input/output instruction is executed.

2.3.8 BRK (Break Flag)

If the BRK flag is set, a software interrupt (interrupt vector 1) occurs when one instruction is executed.

Thus, instructions can be traced one by one.

BRK is set by using memory operation instruction only when it is saved in the stack as a part of PSW. When BRK is restored in PSW after set, it becomes effective.

2.3.9 IE (Interrupt Enable Flag)

When IE is set by using the EI instruction, interrupt is enabled; when IE is cleared by using the DI instruction, interrupt is disabled.

2.3.10 DIR (Direction Flag)

DIR is set by using the SET1 DIR instruction. DIR is cleared by using the CLR1 DIR instruction.

When the DIR flag is set, processing is performed from the high-order address to low-order address in block transfer/input/output instructions. When the DIR flag is cleared, processing is performed from the low-order address to high-order address.

2.3.11 RB2-RB0 (Register Bank 2-0 Flag)

RB2-RB0 indicate the current register bank being used in the eight register banks set in internal RAM.

RB2-RB0 are not restored from the stack when the POP PSW instruction is executed.

Caution: Do not change the RB2-RB0 value saved in the stack or PSW save register in the interrupt service routine.

2.3.12 F1, F0 (user Flag 1, Flag 0)

The F1 and F0 flags can be used as the user desires.

The flags can be set and cleared by using PSW instructions. They can also be set, cleared, and tested by using the FLAG register (special function register).

Fig. 2-2 shows the Flag register bit configuration.

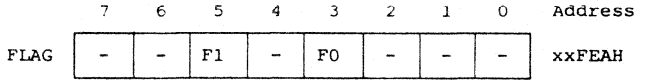


Fig. 2-2 User Flag Register (FLAG) Format

2.4 Memory Space

The uPD70332 and uPD70330 have 1M-byte memory space each. Fig. 2-3 shows a memory map.

Locations 00000H-003FFH are vector area. When the area is not used for vectors, it can be used for other purposes.

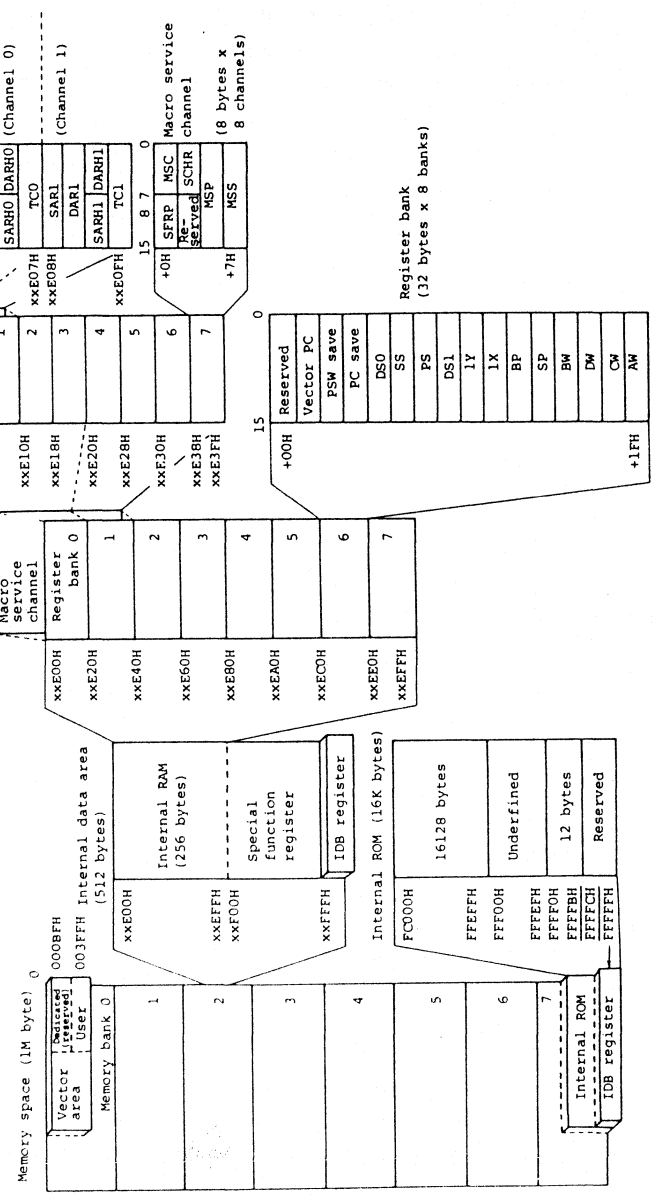
Locations xxE00H-xxFFFH (xx is the IDB register value) are internal data area. This area can be relocated in 4K-byte units.

The four bytes of FFFFCH-FFFFFH are reserved The IDB register is assigned to FFFFFH.

In memory space access, wait cycle insertion is programmable every 128K bytes.

See 2.1.4 for the physical addresses.

Caution: If word reference is made at address FFFFH (offset in segment), the reference address of the second byte becomes FFFFH + 1 rather than address 0000H in the segment (address 0000H in uPD70108/uPD70116 (V20™/V30™)).



Remarks 1: xx is a value specified by using the IDB register.
2: +□□ H is an address offset value. The actual address is provided by adding the register bank or macro service channel start address to the value.
3: Internal ROM is contained only in uPD70332.
4: Macro service channels are assigned to register banks 0, and DMA service channels are assigned to macro service channels 0 and 1.

Fig. 2-3 Memory Map

2.4.1 Internal data area

The internal data area is a 512-byte area containing internal RAM area and special function register area. It can be relocated in 4K-byte units in the 1M-byte memory space. The base address of the internal data area is set by using the IDB register (internal data area base register). The high-order eight bits of 20-bit internal data area base address are specified by using the IDB register and the low-order 12 bits are fixed to E00H.

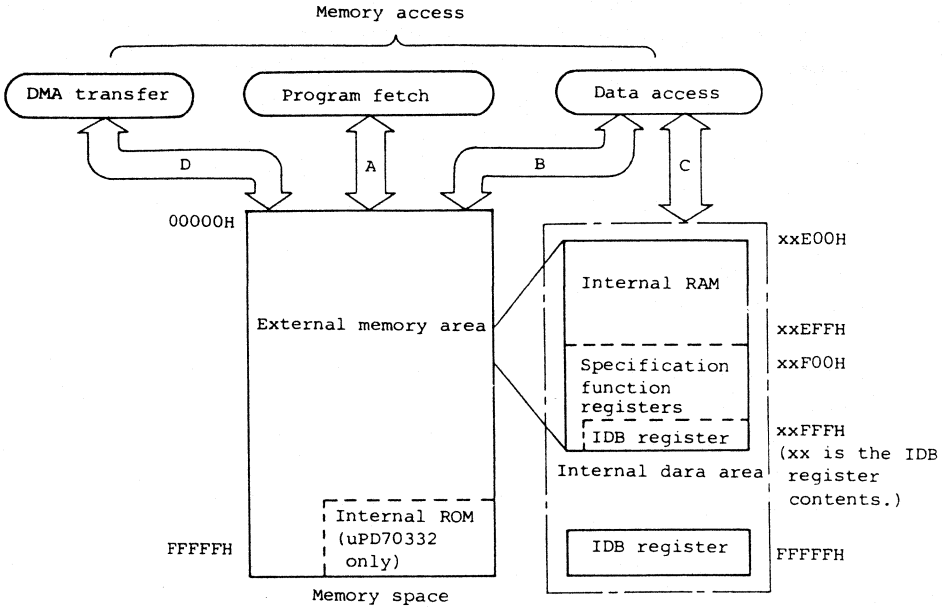
The internal data area is operated by using memory operation instructions.

The internal data area overlaps with the external memory area or internal ROM area (uPD70332 only) as shown in Fig. 2-4. In memory access other than program fetch, the internal data area is accessed. In program fetch, an area other than the internal data area is accessed.

The low-order 256 bytes of the internal data area (xxE00H-xxEFFH: xx is a value specified by using the IDB register) are the internal RAM area. In addition to normal RAM use, register banks, macro service channels, and DMA service channels are assigned functionally to the internal data area. Access to internal RAM as normal RAM can be disabled by clearing bit 6 (RAMEN) of the processor control register (PRC), which is a special function register.

The high-order 256 bytes of the internal data area (xxF00H-xxFFFH: xx is a value specified by using the IDB register) are used for the special function register area. Special function registers such as the on-chip peripheral hardware mode register and control register are mapped in this area.

When $\overline{\text{RESET}}$ is input, the IDB regsiter is initialized to FFH; thus, the internal data area is located at FFE00H-FFFFFH.



A. In program fetch, an area other than the internal data area is accessed.
B. Data access is disabled in areas other than the internal data area, or at addresses corresponding to the internal RAM area when internal RAM is accessed.
C. In data access, the internal data area will be accessed if condition B is not satisfied.
D. In DMA transfer, the external area is accessed (internal ROM is not accessed).

Fig. 2-4 Memory Space Access Conditions

2.4.2 Internal data area base register (IDB)

The internal data area base register (IDB) is used to determine the physical address of the internal data area (containing internal RAM and special function registers described below). The high-order eight bits of the

internal data area base address are specified. The low-order 12 bits are fixed to E00H.

IDB is assigned two addresses of xxFFFH in the special function register area (xx is the IDB register value) and fixed address FFFFFH. IDB can be changed or referenced by memory access to either of the addresses.

At reset, FFH is set in IDB. Thus, the internal data area base address becomes FFF00H.

2.4.3 Special function register area

Special function registers such as the on-chip peripheral hardware mode register and control register are mapped at xxF00H-xxFFFH (xx is the IDB register contents). Program fetch cannot be made from this area.

The special function registers are operated by memory access.

Table 2-4 lists the special function registers. The meanings of the columns in the table are as follows:

- Abbreviation : Symbol representing the special function register name and corresponding to the instruction operand.
- R/W : Indicates whether or not the function register can be read or written.
  R/W: Read and write can be made.
  R : Read only
  W : Write only
- Operation method: Indicates bit operations that can be performed for the register (16 bits, 8 bits, 1 bit).

· At RESET : Indicates the register value when $\overline{\text{RESET}}$ is input.

Addresses not listed in the table are reserved. In reading, the contents become undefined. In writeing the operation has no meaning.

Table 2-4 Special Function Register List

Address	Special function register name	Abbreviation	R/W	Operation method (bits)	At RESET
xxF00H	Port 0	P0	R/W	8/1	Undefined
xxF01H	Port 0 mode register	PM0	W	8	FFH
xxF02H	Port 0 mode control register	PMC0			00H
xxF08H	Port 1	P1	R/W	8/1	Undefined
xxF09H	Port 1 mode register	PM1	W	8	FFH
xxF0AH	Port 1 mode control register	PMC1			00H
xxF10H	Port 2	P2	R/W	8/1	Undefined
xxF11H	Port 2 mode register	PM2	W	8	FFH
xxF12H	Port 2 mode control register	PMC2			00H
xxF38H	Port T	PT	R	8	Undefined
xxF3BH	Port T mode register	PMT	R/W	8/1	00H
xxF40H	External interrupt mode register	INTM	R/W	8/1	00H
xxF44H	External interrupt macro service control register 0	EMS0			Undefined
xxF45H	External interrupt macro service control register 1	EMS1			
xxF46H	External interrupt macro service control register 2	EMS2			
xxF4CH	External interrupt request control register 0	EXIC0			47H
xxF4DH	External interrupt request control register 1	EXIC1			
xxF4EH	External interrupt request control register 2	EXIC2			

(Cont'd)

Address	Special function register name	Abbreviation	R/W	Operation method (bits)	At RESET
xxF60H	Receive buffer register 0	RxB0	R	8	Undefined
xxF62H	Transmit buffer register 0	TxB0	W		
xxF65H	Serial receive macro service control register 0	SRMS0	R/W	8/1	Undefined
xxF66H	Serial transmit macro service control register 0	STMS0			
xxF68H	Serial mode register 0	SCM0			00H
xxF69H	Serial control register 0	SCC0			
xxF6AH	Baud rate generator register 0	BRG0			
xxF6BH	Serial error register 0	SCE0	R	8	00H
xxF6CH	Serial error interrupt request control register 0	SEIC0	R/W	8/1	47H
xxF6DH	Serial receive interrupt request control register 0	SRIC0			
xxF6EH	Serial transmit interrupt request control register 0	STIC0			
xxF70H	Receive buffer register 1	RxB1	R	8	Undefined
xxF72H	Transmit buffer register 1	TxB1	W		
xxF75H	Serial receive macro service control register 1	SRMS1	R/W	8/1	Undefined
xxF76H	Serial transmit macro service control register 1	STMS1			Undefined
xxF78H	Serial mode register 1	SCM1			00H
xxF79H	Serial control register 1	SCC1			
xxF7AH	Baud rate generator register 1	BRG1			
xxF7BH	Serial error register 1	SCE1	R	8	00H
xxF7CH	Serial error interrupt request control register 1	SEIC1	R/W	8/1	47H
xxF7DH	Serial receive interrupt request control register 1	SRIC1			
xxF7EH	Serial transmit interrupt request control register 1	STIC1			
xxF80H	Timer register 0	TM0	R/W	16	Undefined
xxF82H	Modulo/timer register 0	MD0			
xxF88H	Timer register 1	TM1			
xxF8AH	Modulo/timer register 1	MD1			

(Cont'd)

Address	Special function register name	Abbreviation	R/W	Operation method (bits)	At RESET
xxF90H	Timer control register 0	TMC0	R/W	8/1	00H
xxF91H	Timer control register 1	TMC1			
xxF94H	Timer unit macro service control register 0	TMMS0	R/W	8/1	Undefined
xxF95H	Timer unit macro service control register 1	TMMS1			
xxF96H	Timer unit macro service control register 2	TMMS2			
xxF9CH	Timer unit interrupt request control register 0	TMIC0			47H
xxF9DH	Timer unit interrupt request control register 1	TMIC1			
xxF9EH	Timer unit interrupt request control register 2	TMIC2			
xxFA0H	DMA control register 0	DMAC0	R/W	8/1	Undefined
xxFA1H	DMA mode register 0	DMAM0			00H
xxFA2H	DMA control register 1	DMAC1			Undefined
xxFA3H	DMA mode register 1	DMAM1			00H
xxFACH	DMA interrupt request control register 0	DIC0			47H
xxFADH	DMA interrupt request control register 1	DIC1			
xxFE0H	Standby control register	STBC	(Notel) R/W	8/1	(Note 2) Undefined
xxFE1H	Refresh mode register	RFM	R/W	8/1	FCH
xxFE8H	Wait control register	WTC	R/W	16/8	FFFFH
xxFEAH	User flag register (Note 3)	FLAG	R/W	8/1	00H
xxFEBH	Processor control register	PRC	R/W	8/1	4EH
xxFECH	Time base interrupt request control register	TBIC			47H
xxFFFH	Internal data area base register	IDB			FFH

Note 1: The standby control register can be set (but not deared) to 1 by using an instruction.
(W: Only "1" enabled)

2: At power on reset: 00H
Others : No change

3: Only user flag register (FLAG) bits 3 and 5 can be operated. (Other bits: Don't care)
The user flag 0 or 1 (F0 or F1) contents in the FLAG register are also changed by operating PSW F0 or F1. (See 2.3.12.)

2.4.4 Internal RAM area

Internal 256-byte RAM comprises addresses xxE00H-xxEFFH (xx is the IDB register contents).

Eight register banks are assigned to internal RAM. Duplicately macro service channels and DMA service channels are also assigned.

Memory access to internal RAM can be disabled by clearing processor control register (PRC) bit 6 (RAMEN). Program fetch cannot be made from internal RAM. When memory access is disabled, internal RAM is accessed only as registers.

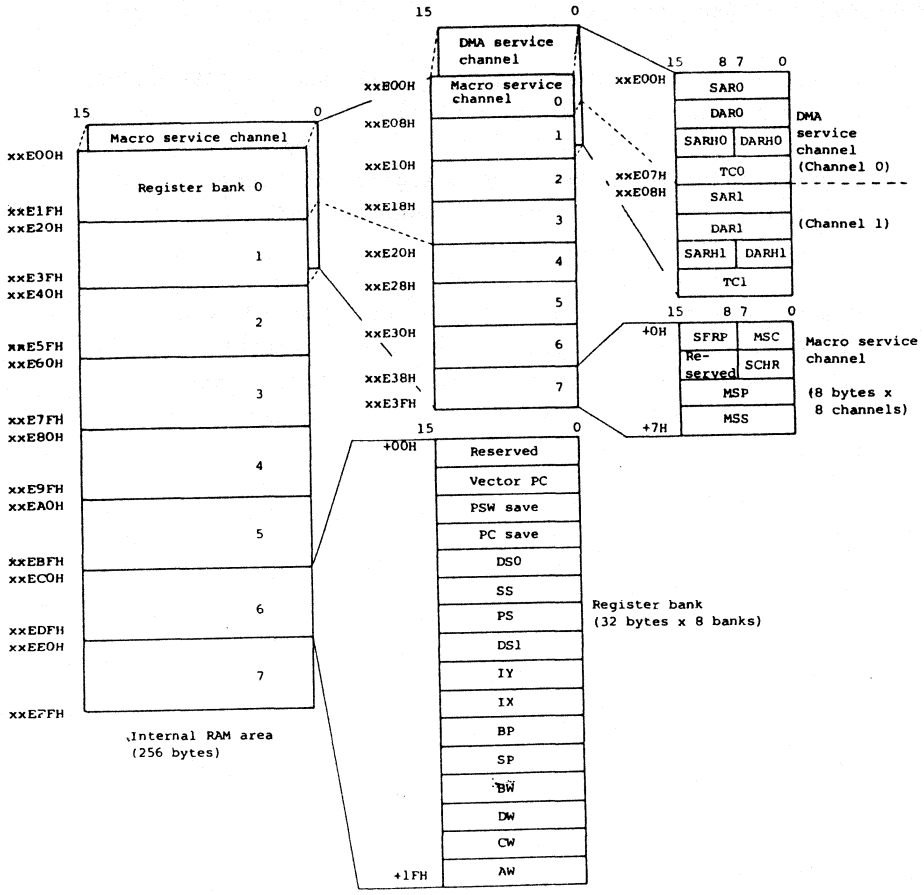


Fig. 2-5 Internal RAM Area Map

2.4.5 Vector table area

The 1K-byte area 00000H-003FFH comprises 256 vectors (each consisting of four bytes) for the start addresses of the interrupt routines initiated when an interrupt request or break instruction occurs.

```
Vector 0  (00000H) :  Divide error
Vector 1  (00004H) :  Single step
Vector 2  (00008H) :  NMI input
Vector 3  (0000CH) :  BRK 3 instruction
Vector 4  (00010H) :  BRKV instruction
Vector 5  (00014H) :  CHKIND instruction
Vector 6  (00018H) :  Reserved
Vector 7  (0001CH) :  FPO instruction
Vector 8  (00020H) :  Reserved
      |       |
Vector 11 (0002CH):  Reserved
Vector 12 (00030H):  INTSER0
Vector 13 (00034H):  INTSR0
Vector 14 (00038H):  INTST0
Vector 15 (0003CH):  Reserved
Vector 16 (00040H):  INTSER1
Vector 17 (00044H):  INTSR1
Vector 18 (00048H):  INTST1
Vector 19 (0004CH):  Input/output instruction
Vector 20 (00050H):  INTD0
Vector 21 (00054H):  INTD1
Vector 22 (00058H):  Reserved
Vector 23 (0005CH):  Reserved
Vector 24 (00060H):  INTP0
Vector 25 (00064H):  INTP1
Vector 26 (00068H):  INTP2
Vector 27 (0006CH):  Reserved
Vector 28 (00070H):  INTTU0
Vector 29 (00074H):  INTTU1
Vector 30 (00078H):  INTTU2
Vector 31 (0007CH):  INTTB
Vector 32 (00080H):  User area
      |       |        .
                       .  BRK imm 8 instruction
                       .
Vector 255(003FCH):  INT input
```

Vectors 0-31 are assigned specific use sources (some vectors are reserved) and cannot be used for general purposes.

Vectors 32-255 can be used for general purposes (2-byte break instruction and INT input). An unassigned area can also be used for purposes other than vectors.

A vector consists of four bytes. When an interrupt is acknowledged, the two high-order bytes of the vector are loaded into program segment PS and the two low-order bytes are loaded into the program counter.

Example:

Vector 0	000H	001H	PC ← (001H, 000H)
	002H	003H	PS ← (003H, 002H)

2.4.6 External memory area

External memory (such as ROM and RAM) for the uPD70332 is assigned to the 00000H-FBFFFH area.

Externally connected memory (such as ROM and RAM) for the uPD70330 is assigned to the 00000H-FFFFEH area. The FFF00H-FFFEFH and FFFFCH-FFFFEH areas are reserved.

The address bus (A0, A9/A1-A16/A8, A17/A18, A19), the data bus (D0-D15), and the $\overline{MREQ}$, $\overline{MSTB}$, R/$\overline{W}$, and A18/$\overline{UBE}$ signals are used to access external memory. Since the refresh pulse output pin ($\overline{REFRQ}$) is provided to refreshing pseudo-static memory, pseudo-static memory canbe easily connected. Since automatic output function of refresh addresses is also provided to refresh dynamic memory, dynamic memory can be connected easily. (See 4.3.) A wait cycle can be inserted in a memory cycle in 128-byte units by using software. (See 4.1.)

2.4.7 Internal ROM area

The uPD70332 contains internal masked ROM using the FC000H-FFFFFH area. However, the FFF00H-FFFEFH area is used for testing by NEC and is not available to the user. The four bytes of FFFFCH-FFFFFH are reserved. Thus, the user can use a 16140-byte area.

Internal ROM has a dedicated bus with an instruction queue. Thus, the uPD70332 can prefetch instructions independently of external memory space and at high speed; it can execute instructions at high speed. Prefetch can be made in one clock; at least three clocks are required for the external.

DMA transfer is not applied to the internal ROM area. External memory assigned the same addresses is accessed during DMA transfer.

2.5 I/O Space

The uPD70332, uPD70330 has a 64K-byte I/O space in addition to 1M-byte space. Fig. 2-6 shows an I/O space map.

The address bus (A0-A15), the data bus (D0-D7), and the $\overline{\text{IOSTB}}$, R/$\overline{\text{W}}$, $\overline{\text{DMAAK0}}$, and $\overline{\text{DMAAK1}}$ signals are used to access the I/O space. 0 is output from the unused high-order four bits of the address bus (A16-A19). A wait cycle can be included in an I/O cycle by using software. (See 4.1.)

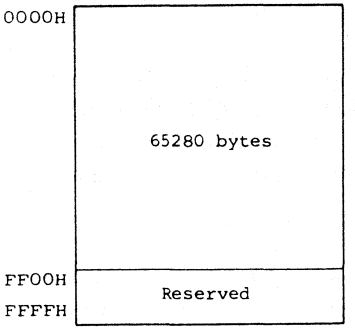


Fig. 2-6 I/O Map (64K Bytes)

3. INTERRUPTS

3.1 Interrupt Controller

The uPD70332, uPD70330 contains an internal interrupt controller which can control multiple interrupts from 17 sources.

The interrupt controller groups 17 sources of interrupts (five external and 12 internal) into eight groups for management and can perform programmable multiprocessing control in group units. In addition, an interrupt service function can be selected from the vectored interrupt function, register bank switching function, and macro service function according to the interrupt source characteristic.

The number of external interrupt sources can be easily increased by externally connecting an interrupt controller such as the uPD71059.

To set the interrupt controller, the interrupt control registers and macro service control registers are used. These registers are provided for each interrupt source. The interrupt control instructions include the EI and DI instructions to control enable and disable, the RETI and RETRBI instructions to indicate return from interrupt, and the FINT instruction to inform the internal interrupt controller of the termination of interrupt service.
(See 13.)

3.2 Interrupt Sources

There are 17 uPD70332 or uPD70330 interrupt sources (five external and 12 internal). The 17 interrupt sources are

separated into eight groups for management by the interrupt controller. The group configuration is fixed in hardware. By using software priorities 0-7 (0 is the highest) can be assigned as desired to interrupts of the groups except NMI, INT, or INTTB. The function supported by the interrupt controller varies according to the interrupt source. Table 3-1 lists the interrupt sources.

Table 3-1 Interrupt Source List

Interrupt source	External or internal	Vector	Macro service	Bank switching	Priority			Multiprocessing control
					Assignment	Intergroup	Intragroup	
NMI (Non Maskable Interrupt)	External	2	Not included	Not included	Cannot be made	0	-	Not applied
INT (INTerrupt)	External	External input	Not included	Not included	Cannot be made	7	-	Not applied
INTTU0 (INTerrupt from Timer Unit0)	Internal	28	Included	Included	Can be made	1	1	Applied
INTTU1 (INTerrupt from Timer Unit1)		29					2	
INTTU2 (INTerrupt from Timer Unit2)		30					3	
INTD0 (INTerrupt from DMA channel0)	Internal	20	Not included	Included	Can be made	2	1	Applied
INTD1 (INTerrupt from DMA channel1)		21					2	
INTP0 (INTerrupt from Peripheral#0)	External	24	included	Included	Can be made	3	1	Applied
INTP1 (INTerrupt from Peripheral#1)		25					2	
INTP2 (INTerrupt from Peripheral#2)		26					3	

(Cont'd)

Interrupt source	External or internal	Vector	Macro service	Bank switching	Priority			Multiprocessing control
					Assignment	Intergroup	Intragroup	
INTSER0 (INTerrupt from Serial ERror of channel0)	Internal	12	Not included	Included	Can be made	4	1	Applied
INTSR0 (INTerrupt from Serial Receiver of channel0)		13	Included				2	
INTST0 (INTerrupt from Serial Transmitter of channel0)		14	Included				3	
INTSER1 (INTerrupt from Serial ERror of channel1)	Internal	16	Not included	Included	Can be made	5	1	Applied
INTSR1 (INTerrupt from Serial Receiver of channel1)		17	Included				2	
INTST1 (INTerrupt from Serial Transmitter of channel1)		18	Included				3	
INTTB (INTerrupt from Time Base counter)	Internal	31	Not included	Not included	Cannot be made (fixed to 7)	6	-	Applied

3.3 Priority Control

3.3.1 Multi-interrupt priority control

Multi-interrupt priority control is performed in group units for interrupts except NMI or INT.

Interrupt multiprocessing control is performed in the EI state. Thus, in an interrupt service routine, the EI must be entered to perform multiprocessing. However, interrupt response by macro service is subjected to multiprocessing control even in the DI state. Multiprocessing control acknowledges an interrupt request assigned to higher priority than the current interrupt being processed, halts the current interrupt being processed, and processes the higher priority interrupt. It holds an interrupt request assigned the lower priority than the current interrupt being processed. The held interrupt is acknowledged at the termination of the current interrupt if the interrupt mask bit in the interrupt control register (provided for each interrupt source) is not set to 1 in the current interrupt service routing being executed and if the interrupt request flag is not cleared.

For interrupt response except NMI, INT, or software interrupt, the FINT instruction must be executed at the end of the interrupt service routine to inform the interrupt controller of the termination of the interrupt service routine. If the instruction is not executed, a subsequent interrupt will not be acknowledged if it is not assigned a higher priority than the interrupt for which the FINT instruction is not executed.

NMI and INT interrupt responses are not subjected to multiprocessing control. Therefore, they are not acknowledged if in the EI state (NMI is constant).

Eight priorities (0-7; 0 is the highest) can be assigned as desired for each interrupt group. The priority also indicates the number of the newly assigned register bank when the register bank switching function (described below) is used. The priority is set by using the three bits PR0-PR2 of the interrupt control register provided for each interrupt source. However, it can be set only in the interrupt control register corresponding to the interrupt source assigned the highest priority in an interrupt group; it cannot be set in other control registers (fixed to 7 at read). At reset, all priorities are initialized to 7.

3.3.2 Priority control when interrupts occur at the same time

When interrupts occur at the same time, NMI is assigned the highest acknowledgement priority and INT is assigned the lowest. The priorities other than the NMI or INT priority are the same as the multi-interrupt priorities. Among the groups assigned the same priority, the priority conforms to the priority fixed in hardware. Likewise, within a single group, the priority conforms to the priority of the group.

3.4 Interrupt Response System

The uPD70332 and uPD70330 have the three interrupt response methods; the vectored interrupt function, register bank switching function, and macro service function. One of these functions can be selected in accordance with the purpose of the interrupt. The interrupt controller handles an interrupt request conforming to the response method set in the interrupt control register.

When an interrupt is acknowledged by using the vectored interrupt or register bank switching function, the PC, PS, and PSW contents are saved by the method corresponding to

the function. After PSW is saved, the IE and BRK flags are cleared. Thus, interrupts except NMI or macro service interrupt response and single step interrupts are disabled; software interrupts except single step interrupts occur. (See 3.9.)

3.4.1 Vectored interrupt function

When an interrupt is acknowledged, the current PSW contents and the PC and PS contents are saved in the stack, one vector is selected from the vector table, and the interrupt service routine is executed starting at the address indicated by the vector. All vectors except INT interrupts are fixed. When an INT interrupt occurs, an interrupt acknowledge cycle is generated and interrupt vector is read from the data bus. (See 3.6.) Interrupt vectors other than INT are as listed in Table 3-1.

A return from an interrupt is made by using the RETI instruction; the FINT instruction must be executed for a return from an interrupt except NMI or INT. At a return from an interrupt, PC, PS, and PSW are restored from the stack.

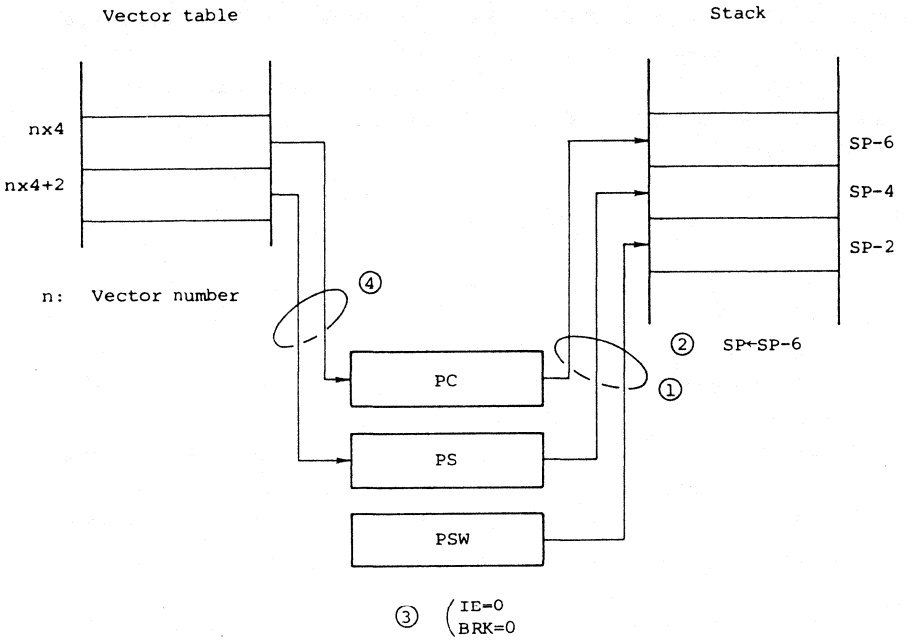


Fig. 3-1 Interrupt Acknowledgement Operation
(In the Order of ① to ④)

3.4.2 Register bank switching function

The uPD70332 and uPD70330 each map a general purpose register set in internal RAM and can have a maximum of eight register banks. Automatic register bank switching in an interrupt response eliminates the need for saving the registers in a stack by using software, thus enabling high-speed interrupt request handling.

To use the register bank switching function, the ENCS bit of the interrupt control register provided for each interrupt source is set to 1. One register bank can be specified for each interrupt group. The register bank number is the same as the multi-interrupt priority; it is specified by using interrupt control register bits PR0-PR2.

The register bank switching sequence is performed as follows: (See Fig. 3-2.)

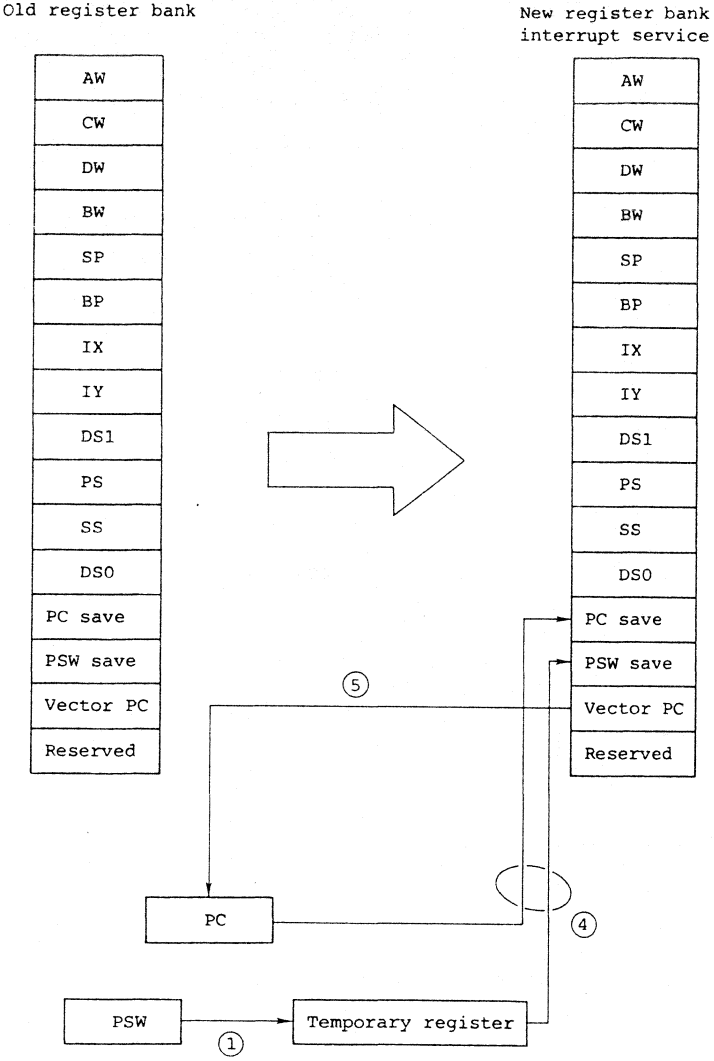
① Save the PSW contents in temporary register.
② Switch register bank.
③ Set IE = 0 and BRK = 0.
④ Save PC and the PSW contents kept in the temporary register in their respective save areas in the register bank.
⑤ Load the interrupt service routine start address offset from the vector PC area in the register bank into PC.

Now, register bank switching is complete and the interrupt service routine is executed.

A return from the register bank switching interrupt is made by executing the RETRBI instruction after the FINT instruction is executed. (The register bank switching function can be used only for interrupts subjected to multi-interrupt control.) when the RETRBI instruction is executed, PC and PSW are restored from the PC and PSW save areas in the register bank as shown in Fig. 3-3. (Since register bank restore cannot be performed if the RETI instruction is executed, a normal return to the main routine cannot be made.)

To use the register bank switching function, previously initialize PS, vector PC, SS, and SP in the new register bank. Initialize other registers as required. Do not change PS in the interrupt service routine.

The register bank switching function can be used only for one interrupt in the interrupt group assigned the same priority. (See 3-7 (1).)



(2) Register bank switching
(3) IE = 0, BRK = 0

Fig. 3-2 Register Bank Switching Sequence

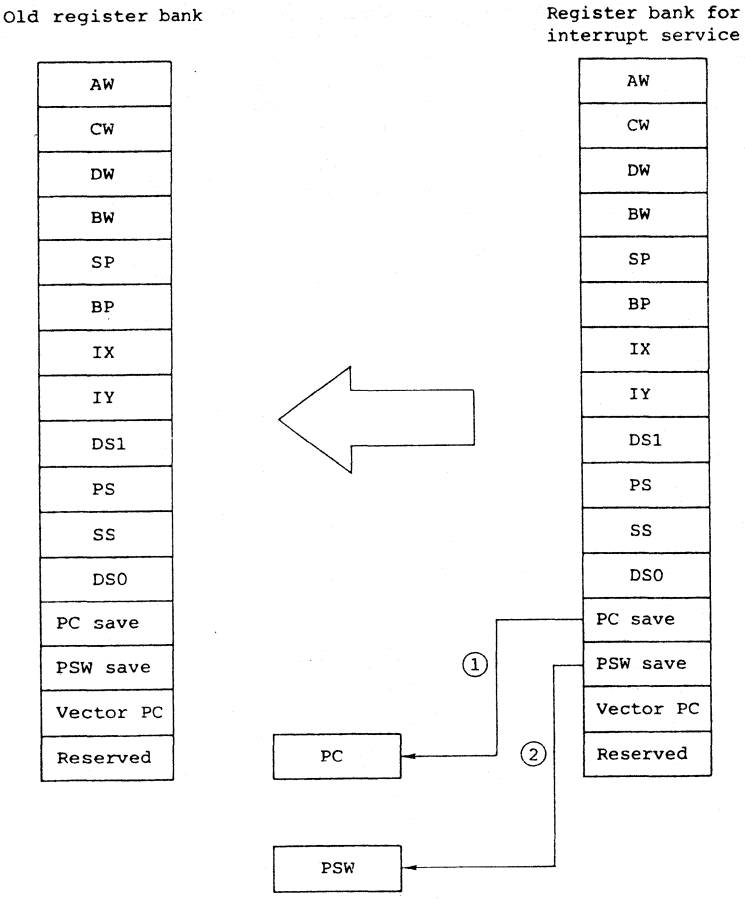


Fig. 3-3 Register Bank Return Sequence

3.4.3 Macro service function

The macro service function transfers data between the special function register area and memory space when an interrupt request is made. This function enables easy processing such as simple data transfers without software interrupt service: overhead involved in interrupt servicing (such as register save, initialization, and restore) can be reduced. Software need not be concerned with macro service function processing. Data formerly processed by software in byte units can be processed as a batch of data for programming is efficiency.

Unlike other interrupt response methods, the macro service function operates regardless of the EI or DI state if the IMK bit (interrupt mask bit) of the interrupt control register provided for each interrupt source is cleared and the MS/$\overline{\text{INT}}$ bit (macro service enable bit) is set. (See 3.7.) However, it is subjected to priority control.

The macro service function includes the following two operating modes:

(1) Normal mode

Each time an interrupt request occurs, data transfer one byte or word at a time is repeated as many times as the preset count.

(2) Character search mode

Each time an interrupt request occurs, 1-byte data transfer is repeated until the preset number of byte is reached or the transferred data matches the prespecified 8-bit data.

The macro service control register provided for each interrupt source for which macro service can be performed and the macro service channel specified in the macro service control register control the macro service function.

3.4.4 Macro service control register

The macro service control register is an 8-bit register used to control the macro service function.

The macro service control register bit configuration is shown below:

7	6	5	4	3	2	1	0
MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0

The bit functions are explained below:

CH0 - CH2 : Macro service channel specification bits. Any digit from 0 to 7 can be specified.

DIR : Data transfer direction specification bit.

0: Data is transferred from memory to special function register.

1: Data is transferred from special function register to memory.

MSM0 - MSM2 : Macro service mode specification bit. The normal operation mode, or character search and the number of transfer data bits in the normal mode (eight or 16 bits) are specified by setting the MSM0-MSM2 bits.

MSM2	MSM1	MSM0	Operating mode
0	0	0	Normal mode (8-bit transfer)
0	0	1	Normal mode (16-bit transfer)
1	0	0	Character search mode (8-bit transfer)
Other settings			Undefined

The macro service control register is contained in the special function register area. The register can be read or written by 8-bit or 1-bit operation memory access.

The macro service control register is provided for each interrupt source for which macro service can be performed. The interrupt sources for which macro service can be performed are timer interrupt (INTTU0-INTTU2), external interrupt (INTP0-INTP2), and serial receive and transmit interrupts (INTSR0, INTSR1 and INTST0, INTST1). For the macro service control register location for each interrupt source, see the appropriate item.

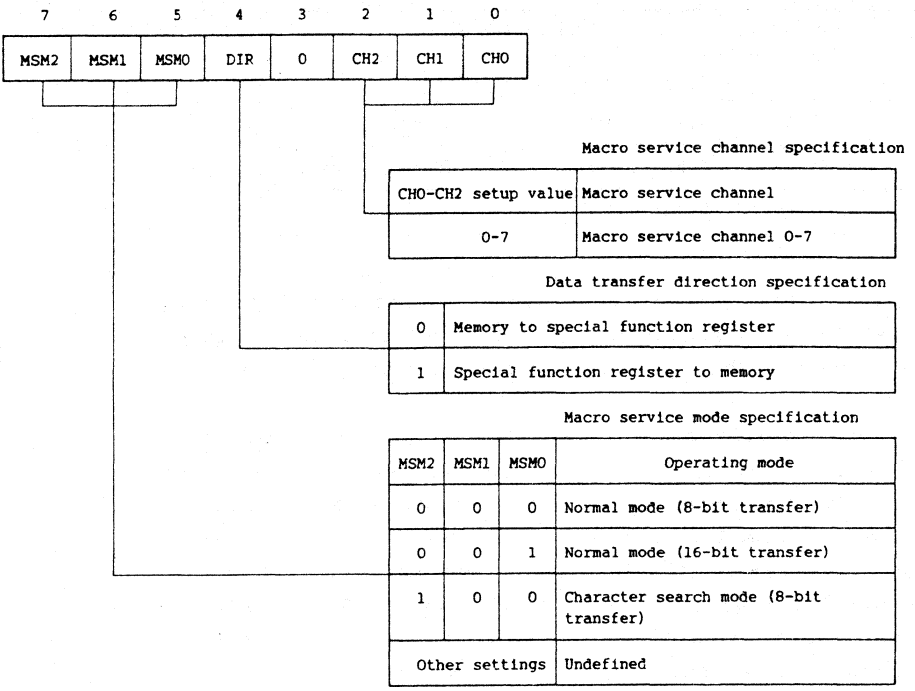
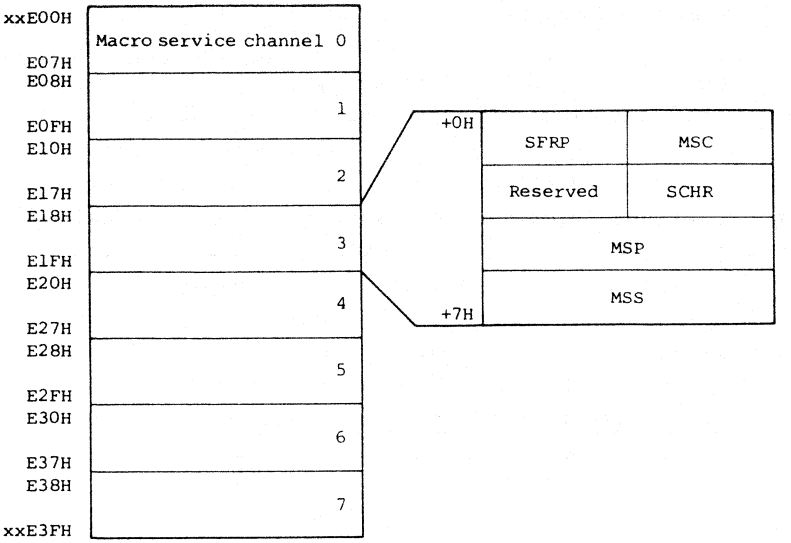


Fig. 3-4 Macro Service Control Register Format

3.4.5 Macro service channels

The macro service channels are assigned to internal RAM xxE00H-xxE3FH (xx is the IDB register value), as shown in Fig. 3-5. The macro service data transfer destination and source, transfer count, and comparison character are set by using the macro service channel. A maximum of eight channels can be used.



MSC (+0H): Macro service transfer count
SFRP (+1H): Special function register address offset value. xxF00H+SFRP (xx is the IDB register value) is the special function register address.
SCHR (+2H): 8-bit data compared in the character search mode.
MSP (+4H) : Offset value of memory address to which data transfer is applied in macro service
MSS (+6H) : Segment value of memory address to which data transfer is applied in macro service. The memory address to which data transfer is applied is MSS x 16 + MSP.

Fig. 3-5 Macro Service Channel Configuration

In the macro service channel MSC is decremented by one and MSP is incremented by one or two each time one data byte (eight or 16 bits) is transferred. After this, the interrupt request flag is cleared. However, if the MSC reaches 0 or the transfer data matches the comparison data (only in the character search mode), the interrupt request flag is not cleared and an interrupt is requested.

3.5 NMI (Non Maskable Interrupt)

NMI is assigned the highest priority and cannot be disabled. This interrupt is detected on the rising or falling edge according to the state of special function register (INTM) bit 0 (the ESNMI bit). When the ESNMI bit is set to 0, an NMI interrupt occurs on the falling edge; when the bit is set to 1, an NMI interrupt occurs on the rising edge. The interrupt enables vector response only and the vector type is fixed to 2. The NMI pin is also used for the P10. The input level can be examined by reading P10. When NMI is acknowledged, disable state (IE=0) is set and other interrupts are disabled (however, interrupt response by macro service is acknowledged).

3.6 INT (Interrupt)

INT is a maskable interrupt. This interrupt is detected at the active high level. INT is not subjected to multiprocessing control by the interrupt controller. It can always be acknowledged if IE is set to 1 (enable state). However, when a number of interrupts occur at a time, INT is assigned the lowest priority. INT enables vector response only. the vector type is read from the data bus in an interrupt acknowledge cycle. The interrupt acknowledge cycle can be checked by $\overline{\text{INTAK}}$ output. The INT pin is also used for P14 and $\overline{\text{POLL}}$. The pin function is selected by using a special function register:

port 1 mode control register (PMC1) bit 4. Thus, if the INT function is not selected, no interrupt occurs even if IE is set to 1 (enable state). $\overline{\text{INTAK}}$ is also used for P13 and $\overline{\text{INTP2}}$. The pin function is selected by using PMC1 bit 3.

Extension to a maximum of 64 external interrupt inputs can be made by connecting an interrupt controller such as the uPD71059.

When INT is acknowledged, the disable state (IE=0) is set.

3.7 Interrupt Request Control Register

The interrupt request control register is an 8-bit register used to control interrupts other than INT or NMI.

The interrupt request control register bit configuration is shown below:

7	6	5	4	3	2	1	0
IF	IMK	MS/$\overline{\text{INT}}$	ENCS	0	PR2	PR1	PR0

The bit functions are explained below:

PR0-PR2 : Interrupt group priority specification bits. Any digit from 0 to 7 can be specified. This priority specification can be made only in the control register for the interrupt assigned the highest priority in the group; it is not effective in other interrupt request control registers (fixed to 7 at read time). The priorities of other interrupt request control registers conform to the priority of the interrupt request control register assigned the highest priority in the group.

The priority specification also specifies a new register bank when the register bank switching function is executed.

ENCS : Bit to specify whether or not the register bank switching function is used.
1: Register bank switching function is used.
0: Vectored interrupt is used.

NS/$\overline{\text{INT}}$: Interrupt response method selection bit.
1: Macro service function is used.
0: Vectored interrupt or register bank switching function is used.

IMK : Interrupt masking bit.
1: Interrupt is masked.
0: Interrupt is unmasked.

IF : Bit indicating that an interrupt request is made.
1: Interrupt request is made.
0: No interrupt request is made.

When an interrupt request occurs, the IF bit is set to 1; when an interrupt is acknowledged or an instruction such as BTCLR (additional instruction from uPD70108/uPD70116) is executed, the IF bit is cleared.

The interrupt request control register exists in the special function register area. The register can be read or written by 8-bit or 1-bit memory access operation.

The interrupt request control register is provided for each interrupt source (except INT or NMI). For the interrupt request control register location for each interrupt source, see the appropriate item.

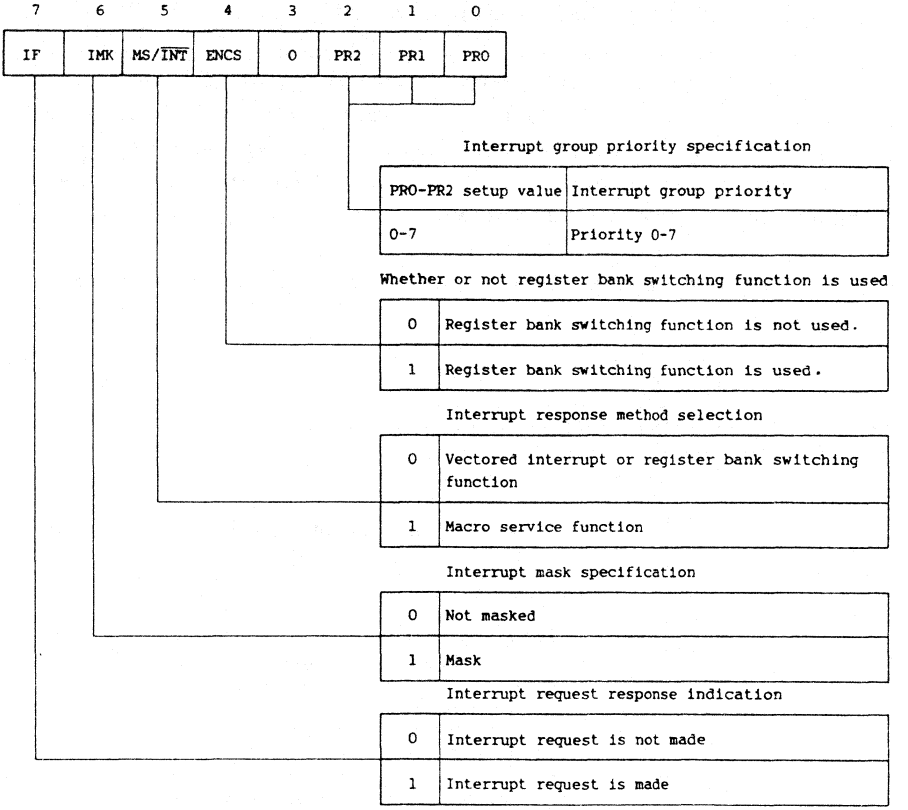


Fig. 3-6 Interrupt Request Control Register Format

3.8 External Interrupts

There are five external interrupt request sources. INT is level-detected. Interrupts other than INT are edge-detected. The effective edge for each external interrupt other than INT can be specified by using the external interrupt mode (INTM) special function register.

3.8.1 External interrupt mode register (INTM)

The 8-bit external interrupt mode register is used to specify the effective edge for each external interrupt request. The interrupts detected on the edge are NMI and INTP0-INTP2. The effective edges for these interrupts are specified in the INTM register.

Fig. 3-7 shows the INTM register bit configuration and functions.

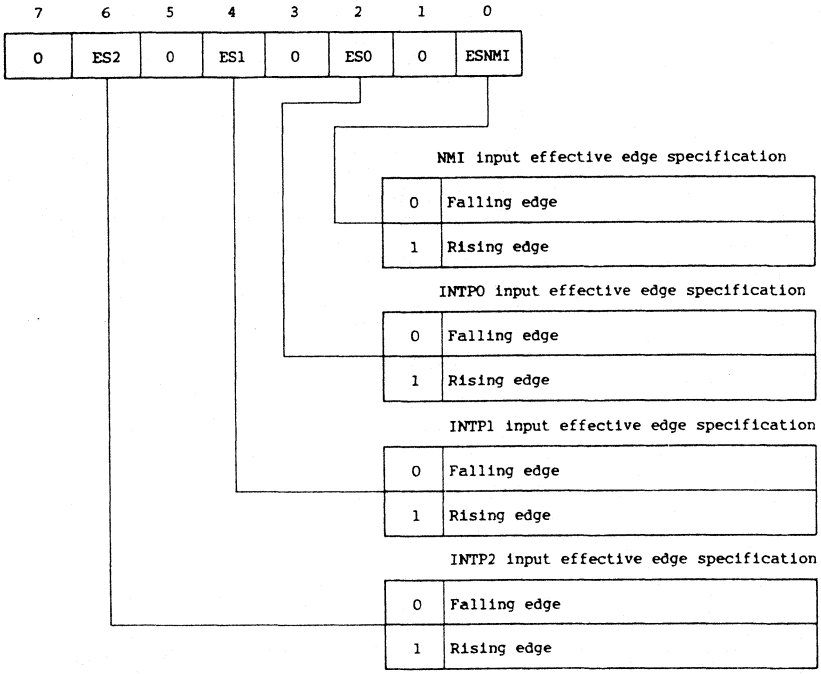


Fig. 3-7 External Interrupt Mode Register (INTM) Format

3.8.2 External Interrupt Request Control Register (EXIC0,EXIC1,EXIC2)

EXICn (n=0-2) register is a 8 bit register which controls interrupt requests generated from the 3 external interrupt request pins (INTP0 - INTP02).

These three interrupt requests consisting as one group for setting priorities. In the group the priority is fixed as below:

EXF0 > EXF1 > EXF2

	7	6	5	4	3	2	1	0
EXIC0	EXF0	EXMK0	MS/$\overline{INT}$	ENCS	0	PR2	PR1	PR0
EXIC1	EXF1	EXMK1	MS/$\overline{INT}$	ENCS	0	1	1	1
EXIC2	EXF2	EXMK2	MS/$\overline{INT}$	ENCS	0	1	1	1

Bit 2 - 0 of EXIC1 and EXIC2 are fixed to '1'.
Priority of interrupt request for EXIC1 and EXIC2 registers are following the setting of PR2 - PR0 in EXIC0 register.

As for each bit of EXICn registor, please refer to chapter 3.7.
EXICn registers can be read/written with 8 bit or 1 bit memory operation command. In this case one wait cycle is automatically inserted.
With RESET input, contents turn to 47H.

3.8.3 External Interrupt Macro Service Control Register (EMS0,EMS1,EMS2)

EMSn (n=0-2) are 8 bit registers, which control macro service initiated by the interrupt requests generated form the external interrupt request pins (INTP0 - INTP2).

EMS0 register controls macro service initiated by the EXF0 flag.

EMS1 register controls macro service initiated by the EXF1 flag.

EMS2 register controls macro service initiated by the EXF2 flag.

EMSn registors can be read/written with 8 bit or 1 bit memory operation command. In this case 1 wait will be inserted.

As for each bit of EMSn register, please refer to chapter 3.4.4.

EMS0, EMS1, EMS2

7	6	5	4	3	2	1	0
MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0

3.9 Software Interrupts

The uPD70332, uPD70330 contains eight software interrupts. (See Table 3-2.) Of these interrupts, six are compatible with uPD70108/uPD70116 software interrupts (however, interrupts related to the emulation mode are not included) and two are unique to the uPD70332/uPD70330.

The vectors of these interrupts are predefined.

Any software interrupt other than BRK (single step interrupt) is always acknowledged if the interrupt occurrence condition is satisfied (the interrupt takes precendence over a hardware interrupt). Although a BRK flag interrupt occurs when BRK = 1, the BRK flag is cleared automatically when a hardware or software interrupt is acknowledged; thus the BRK flag interrupt is assigned lower priority than other hardware or software interrupts and does not occur during another interrupt service.

Table 3-2 Software Interrupts

Interrupt source	Vector	Priority
DIVU divide error	0	1
DIV divide error		
CHKIND boundary over	5	
BRKV	4	
BRK 3	3	
BRK imm8	32-255	
BRK flag(single step)	1	2
Input/output instruction($\overline{\text{IBRK}}$ flag)	19	1
FPO instruction	7	

3.9.1 General software interrupts

The execution sequence for acknowledging software interrupts other than input/output instruction or FPO instruction interrupts is the same as for vectored interrupts. That is, the address of the next instruction and PSW are saved in the stack, IE and BRK are set to 0, and the vector contents are loaded into PS and PC.

The general software interrupts are explained below:

(1) DIVU divide error, DIV divide error

This interrupt always occurs if the quotient overflows when a divide instruction is executed.

(2) CHKIND boundary over

This interrupt occurs if the index value is judged to exceed the boundaries when the CHKIND instruction is executed to check whether or not the index value exceeds the predefined array boundaries.

(3) BRKV

This interrupt occurs if V (overflow flag) is set during BRKV instruction execution.

(4) BRK 3

This interrupt occurs by executing the BRK 3 instruction.

(5) BRK imm8

This interrupt occurs by executing the BRK imm8 instruction.

(6) BRK flag (single step)

This interrupt occurs each time one instruction is executed if BRK is set to 1.

3.9.2 Input/output instruction interrupts

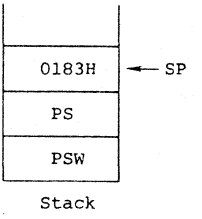
If an attempt is made to execute an input/output instruction when $\overline{IBRK}$ = 0, an interrupt occurs. Unlike general software interrupts (see 3.9.1), address information saved on the stack when the interrupt is acknowledged becomes the address at which the input/output instruction is stored. If a prefix is added to the input/output instruction, the address information becomes the address at which the prefix is stored. Other

operation is the same as in general software interrupts. In a return from an input/output instruction interrupts. the PC value on the stack must be adjusted for the normal return.

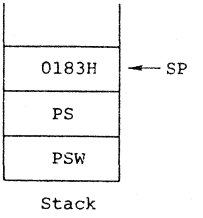
The specific instruction attempted to be executed in software to cause an interrupt can be known by making the address information saved on a stack the top address of the instruction. This function makes the former programs used with uPD70108/70116 easily portable.

The PSW contents immediately preceeding an interrupt occurs are saved on the stack. Then, IE = BRK = 0 and $\overline{IBRK}$ = 1 are automatically set. By setting $\overline{IBRK}$ = 1, the input/output instruction in interrupt service is executed as an input/output instruction, and when a return is made from the interrupt, automatically the state is restored to the former state ($\overline{IBRK}$=0).

```
Example 1:  No prefix
Address     Instruction
                ⋮
0183H           IN          AW, DW
                ⋮
```

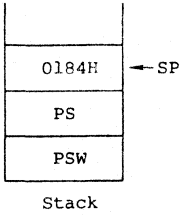



```
Example 2:  When prefix is added
Address     Instruction
                ⋮
0183H           REP
0184H           INM
                ⋮
```



Reference: General software interrupts

Address	Instruction
	⋮
0183H	BRK 3
	⋮



3.9.3 FPO instruction interrupt

Since the uPD70332 and uPD70330 differ from uPD70108/uPD70116 in external bus configuration, a coprocessor for floating-point operation for uPD70108/uPD70116 cannot be connected. Thus, an interrupt is generated to emulate FPO instruction operation when an attempt is made to execute the FPO instruction for the coprocessor. As with input/output instruction interrupts, the PC value saved on the stack for the interrupt becomes the top address of the instruction (when a prefix is added, the top address of the prefix). (See 3.9.2.) Thus, the FPO instruction can be decoded and emulated in software. As in input/output instruction interrupts, the PC value saved on the stack must be adjusted in a return from the FPO instruction interrupt.

4. BUS CONTROL

The uPD70332 and uPD70330 have the pins listed in Table 4-1 for bus control.

For the pins assigned multiple functions, the appropriate function must be selected by using the port mode control register (PMCn).

Table 4-1 Pin Functions for Bus Control

Pin name	I/O	Function	Remarks
A0	O	Address LSB output. It is used for low-order memory bank selection.	
A9/A1-A16/A8, A17/A18, A19	O	The 19-bit address is output in a time division manner.	
A18/$\overline{UBE}$	O	Address bit 18 output and high-order memory bank selection signal output done in a time division manner.	
D0-D15	I/O	Data bus	
R/$\overline{W}$	O	Distinction of read and write	
$\overline{MREQ}$	O	This indicates that a bus cycle is started. High-order address strobe signal.	
$\overline{MSTB}$	O	Memory read/memory write strobe signal. Low-order address strobe signal.	
$\overline{IOSTB}$	O	I/O cycle strobe signal. Low-order address strobe signal.	
$\overline{REFRQ}$	O	This indicates a refresh memory cycle.	
HLDRQ	I	Bus hold request signal	Also used for P27
$\overline{HLDAK}$	O	Bus hold acknowledge signal	Also used for P26
$\overline{DMAAK0}$	O	This indicates a DMA acknowledge cycle.	Also used for P2

Pin name	I/O	Function	Remarks
$\overline{DMAAK1}$	O	This indicates a DMA acknowledge cycle.	Also used for P24
$\overline{READY}$	I	Wait is inserted in a bus cycle from external.	Also used for P17
$\overline{INTAK}$	O	This indicates an interrupt acknowledge cycle.	Also used for P13 and $\overline{INTP2}$

Since both the uPD70332 and uPD70330 manage the memory address for each byte and have a 16-bit external data bus, memory is connected by being divided into high-order and low-order banks, as shown in Fig. 4-1.

The high-order 19 bits of a physical address except A0 are input to the address pins of the high-order and low-order memory banks. The A0 and A18/$\overline{UBE}$ signals are used to select the low-order and high-order memory banks, respectively.

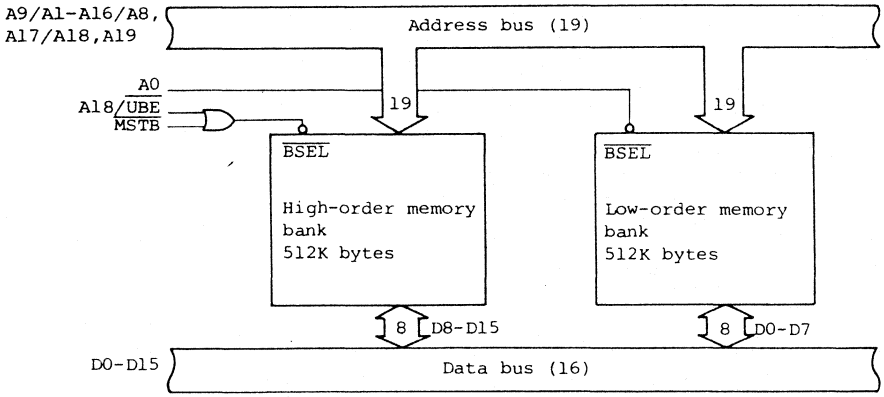


Fig. 4-1 Memory Bank Configuration

A uPD70332/uPD70330 memory cycle consists of the three states T1, T2 and T3. (See 4.5.)

In the T1 state, the first address (high-order address) of a 20-bit address is output to the external address bus. In the T2 state, the second address (low-order address) of a 20-bit address is output to the external address bus. In the T3 state, data is read or written.

The wait state TW is inserted between the T2 and T3 states in a read cycle. TW is inserted between T1 and T2 states in a write cycle.

The A0 pin outputs the least significant bit of a physical address in the T1 state to T3 state. The A18/$\overline{UBE}$ pin the outputs 18th bit of a physical address in the T1 state and the $\overline{UBE}$ signal in the next state to T1. Table 4-2 lists the relationship between the A0 and $\overline{UBE}$ signals.

Table 4-2 uPD70332, uPD70330 Data Access

Operand	$\overline{UBE}$	A0	Number of bus cycles
Even address word	0	0	1
Odd address word	0	1	2
	1	0	
Even address byte	1	0	1
Odd address byte	0	1	1

The I/O read and write cycle timings are the same as the memory read and write cycle timings except $\overline{IOSTB}$ rather than $\overline{MSTB}$ goes active. In a memory refresh cycle, the $\overline{MREQ}$ and $\overline{MSTB}$ signals are active.

In access to external memory and I/O, the uPD70332 and uPD70330 output a 20-bit physical address in a time division manner from the 11-bit address pins and the A18/$\overline{UBE}$ pin, as listed in Table 4-3.

Table 4-3 Address Time Division Output

Pin name	Memory cycle		I/O cycle		Refresh cycle
	First time	Second time	First time	Second time	First time
A0	A0	A0	A0	A0	"0"
A9/A1	A9	A1	A9	A1	A0
A10/A2	A10	A2	A10	A2	A1
A11/A3	A11	A3	A11	A3	A2
A12/A4	A12	A4	A12	A4	A3
A13/A5	A13	A5	A13	A5	A4
A14/A6	A14	A6	A14	A6	A5
A15/A7	A15	A7	A15	A7	A6
A16/A8	A16	A8	"0"	A8	A7
A17/A18	A17	A18	"0"	"0"	A8
A19	A19	A19	"0"	"0"	A9
A18/$\overline{UBE}$	A18	$\overline{UBE}$	"0"	$\overline{UBE}$	"0"
$\overline{REFRQ}$	"1"	"1"	"1"	"1"	"0"

4.1 Programmable Wait Function

The uPD70332 and uPD70330 enable wait state insertion during a bus cycle (except memory refresh cycle) specified by using software. Wait state insertion is specified for the 1M-byte memory space (eight 128K-byte blocks) and I/O space by using the wait control register (WTC), as shown in Fig. 4-1. However, the same setting is made in memory space block 6 (C0000H-DFFFFH) and block 7 (E0000H-FFFFFH)

Wait state specification can be selected from four types 0,1,2, and 3 or more (READY pin control) as desired

independently of other blocks, as listed in Table 4-4. To use READY pin control, port 1 mode control register (PMC1) bit 7 must be set to 1 because the READY pin is also used for P17. If PMC1 bit 7 is set to 0, the READY state is always set; that is, the number of wait states becomes 2. If READY pin control is selected, 2 states of the wait state TAW are inserted regardless of the READY pin state. The READY pin is level-sensed. While the pin is low, wait state is inserted.

Access to internal ROM (uPD70332 only) and internal data area is not affected by the programmable wait function. The wait state setting is applied to every access to the external area except during refresh.

At reset, the WTC register is initialized to FFFFH.

Fig. 4-2 Wait Control Register (WTC) Format

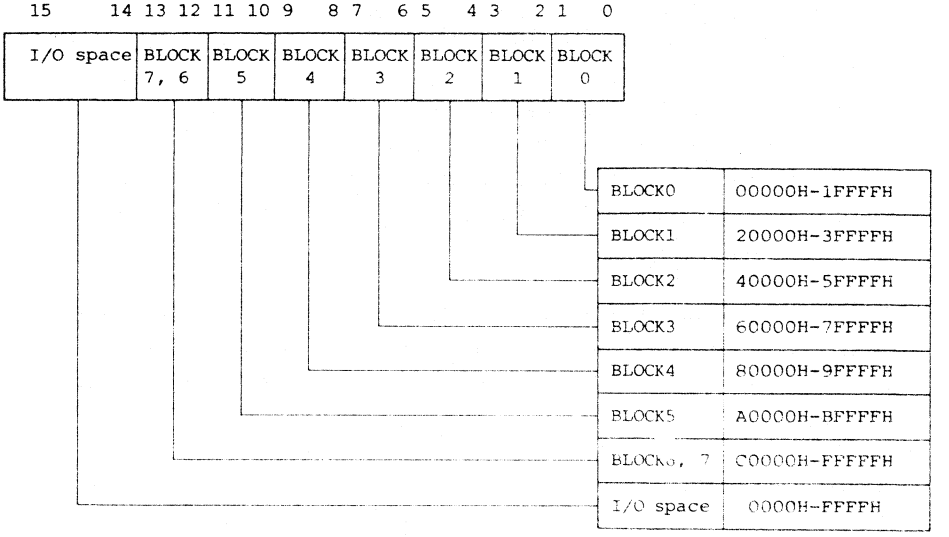


Table 4-4 Wait State Specification

BLOCKn, I/O space	Wait state
00	0 state
01	1 state
10	2 states
11	2 states+READY pin

4.2 Bus Hold Function

The uPD70332 and uPD70330 have a bus hold function. When a high level is applied to the HLDRQ pin from the external, it signals that an external device wants to use the bus. When the uPD70332/uPD70330 detects that the HLDRQ pin is high, it places A0-A19, D0-D7, $\overline{\text{REFRQ}}$, $\overline{\text{MREQ}}$, $\overline{\text{MSTB}}$, $\overline{\text{IOSTB}}$, and R/$\overline{\text{W}}$ output in the high impedance condition and makes the $\overline{\text{HLDAK}}$ pin low to indicate that the bus is open to the external device, then transits to the hold mode. In the hold mode, the uPD70332/uPD70330 stops instruction execution, prefetch interrupt acknowledge, etc., and only the on-chip peripheral hardware not using the bus operates. In the hold mode, the HLDRQ pin is checked, and when a low level is detected, the $\overline{\text{HLDAK}}$ signal is made high to indicate that the bus is not open to the external device; execution is restarted by delaying one clock.

A bus hold request can also be acknowledged in the HALT mode (one type of standby function, see 11-2). When the hold mode is released, a return is made to the HALT mode if the HLDRQ signal is low.

A bus hold request is not acknowledged during execution of one instruction following the BUSLOCK (Note) prefix or during interrupt acknowledge operation.

In the hold mode, the uPD70332 and uPD70330 can insert a memory refresh cycle by setting refresh mode (RFM) register HLDRF (bit 6) to 1. The $\overline{\text{HLDAK}}$ signal is forced to go high at each refresh timing, a check is made to ensure that HLDRQ goes low, then refresh cycle is executed. After this, if the HLDRQ signal is high, the transition to the hold mode is again made. If the HLDRQ signal remains low at that time, the hold mode is released and instruction execution is restarted.

Since the HLDRQ pin is also used for P27 and the $\overline{\text{HLDAK}}$ pin is also used for P26, port 2 mode control register (PMC2) bits 6 and 7 must be set to 1 to use the bus hold function.

Note: BUSLOCK
REP
MOVBK

A bus hold request is not acknowledged during block processing instruction execution in such a program.

4.3 Refresh Function

The uPD70332 and uPD70330 have various functions for refreshing DRAM and pseudo-static RAM:

- Periodic insertion of refresh cycle in a series of bus cycles
- Refresh address output and refresh pulse output to refresh DRAM and pseudo-static RAM
- Pseudo-static RAM power-down self-refresh mode support
- Refresh cycle generation in the hold or HALT mode
- Wait state insertion during refresh cycle

4.3.1 Refresh mode register (RFM)

The 8-bit RFM register controls the refresh function. The register can be written/read by 8-bit or 1-bit operation memory access.

At RESET, the RFM register is initialized to FCH.

The RFM register bit configuration is shown below:

7	6	5	4	3	2	1	0
RFLV	HLDRF	HLTRF	RFEN	RFW1	RFW0	RFT1	RFT0

The bit functions are explained below:

RFT0 and RFT1 : Refresh synchronization specification bits.

A refresh period can be selected out of time base counter (see 7-1) output taps 4-7. A refresh cycle is generated at the time intervals listed in Table 4-5.

Table 4-5 Refresh Period

f_{CLK}=5 MHz ($=\frac{1}{2} f_x$: f_x=10 MHz)

RFT 1	RFT 0	Refresh period
0	0	$2^4/f_{CLK}$ (3.2 us)
0	1	$2^5/f_{CLK}$ (6.4 us)
1	0	$2^6/f_{CLK}$ (12.8 us)
1	1	$2^7/f_{CLK}$ (25.6 us)

RFW0 and RFW1 : Bits to specify the number of wait states inserted in refresh cycle

The number of wait states in a refresh cycle is specified by using RFW0 and RFW1, as listed in Table 4-6, rather than the programmable wait function described above (see 4-1).

Table 4-6 Number of Wait States in Refresh Cycle

RFW 1	RFW 0	Number of wait states
0	0	0
0	1	1
1	0	2
1	1	2

RFEN : Bit to enable automatic insertion of refresh cycle.

1: Automatic insertion of refresh cycle is enabled.

0: Automatic insertion of refresh cycle is disabled. $\overline{\text{REFRQ}}$ pin output is controlled according to the RFLV bit contents. (For details, see below.)

HLTRF : Bit to enable automatic insertion of refresh cycle in the HALT mode.

1: Automatic insertion is enabled.

0: Automatic insertion is disabled. When the RFEN bit is set to 0, automatic insertion is disabled regardless of how the HLTRF bit is set.

HLDRF : Bit to enable automatic insertion of refresh cycle in the hold mode.

1: Automatic insertion is enabled.

0: Automatic insertion is disabled. When the HLDRF bit is set to 1, $\overline{\text{HLDAK}}$ output is forced to go high at every refresh timing, and refresh cycle is automatically inserted.

RFLV : $\overline{\text{REFRQ}}$ signal output level specification bit Fig. 4-3 shows the RFLV bit control circuit. Output is determined by the logic listed in Table 4-7.

The RFLV bit becomes the master RFLV output at read time and is written into slave RFLV at write time. Write to master RFLV is made when refresh timing is generated.

The power-down self-refresh mode of pseudo-static RAM can be supported by using the RFLV bit.

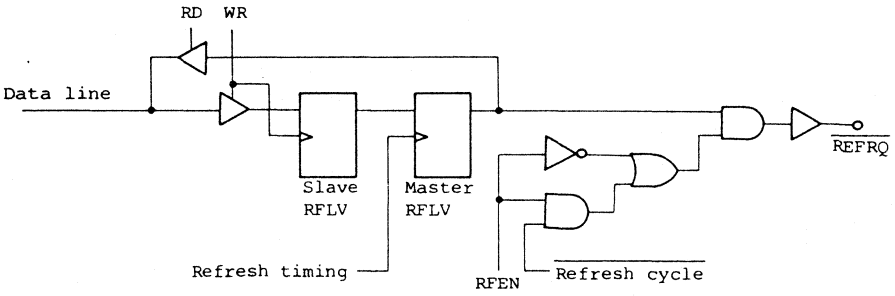


Fig. 4-3 RFLV Bit Control Circuit

Table 4-7 $\overline{\text{REFRQ}}$ Signal Output Level

RFLV	RFEN	$\overline{\text{REFRQ}}$ state
0	0	0
0	1	0
1	0	1
1	1	Refresh pulse output

A refresh cycle is inserted at the refresh timing when the RFEN bit is set to 1. At this time, $\overline{\text{MREQ}}$, $\overline{\text{MSTB}}$, and $\overline{\text{IOSTB}}$ go high, refresh address and high level are output to A0-A8 and A9-A19 respectively, and a refresh pulse is output from the $\overline{\text{REFRQ}}$ pin.

To use bit operation instructions, note that even if the RFLV bit is written, it does not become read data until the next refresh timing.

4.4 Bus Use Rights

The priorities for the right to use the uPD70332, uPD70330 bus are as follows:

(1) Refresh cycle (see 4.3)
Refresh cycle is always generated if refresh cycle insertion is enabled. In the hold mode, however, if refresh cycle insertion is enabled, the $\overline{\text{HLDAK}}$ signal is forced to go high, and a wait is made for the HLDRQ signal to go low before refresh cycle is executed.

(2) Hold mode (see 4.2)
The transition to the hold mode is not made during execution of one instruction following a BUSLOCK prefix or interrupt acknowledge cycle.

(3) DMA cycle (see 5)

(4) Other bus cycles

Refresh cycle, hold, and DMA cycle are temporarily held when an interrupt acknowledge cycle is executed.

In the stop mode, the bus does not operate. (See Table 11-2 for the the bus state.)

4.5 Bus Timing

Fig. 4-4 to 4-11 show main bus timings except for DMA.

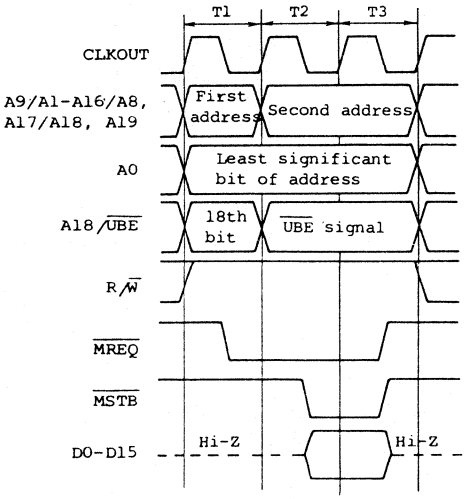


Fig. 4-4 Memory Read Cycle

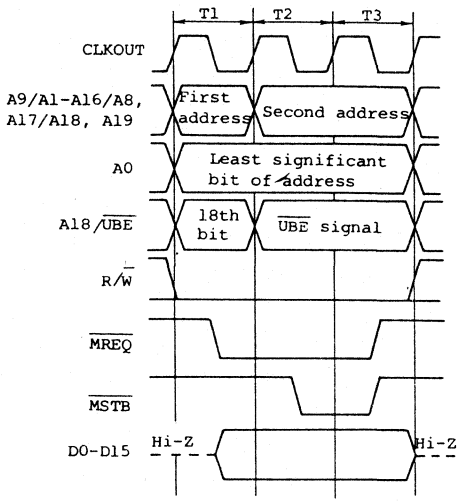


Fig. 4-5 Memory Write Cycle

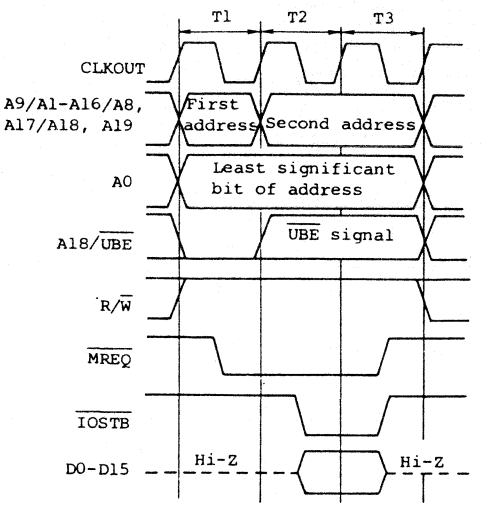


Fig. 4-6 I/O Read Cycle

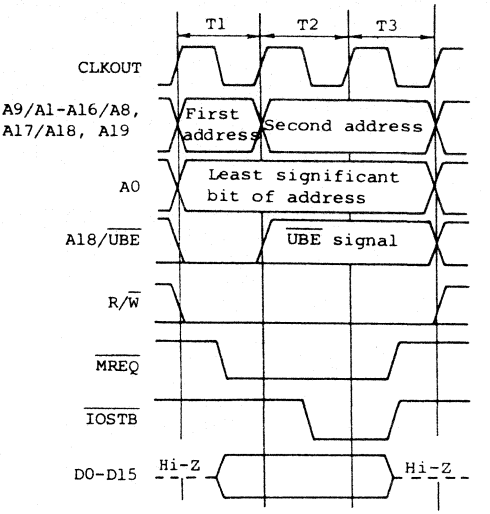


Fig. 4-7 I/O Write Cycle

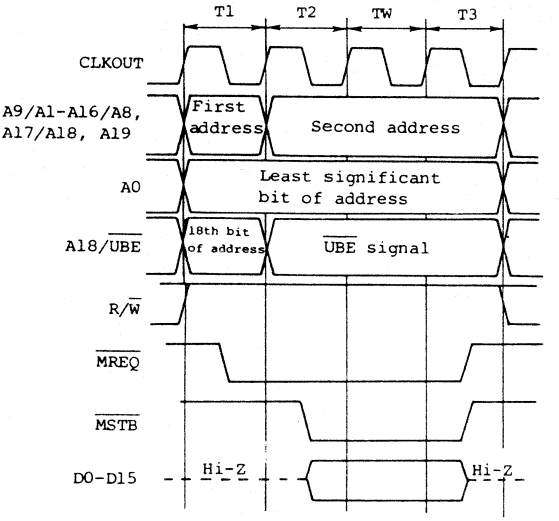


Fig. 4-8 Memory Read Cycle (When One Wait State is Inserted)

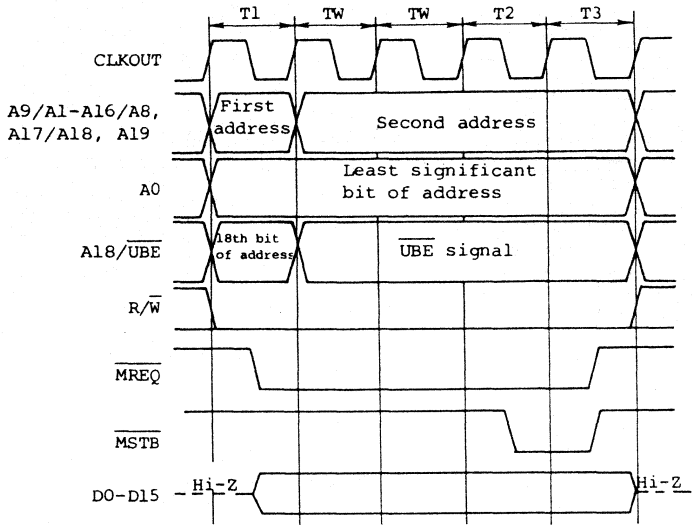


Fig. 4-9 Memory Write Cycle (When Two Wait States are Inserted)

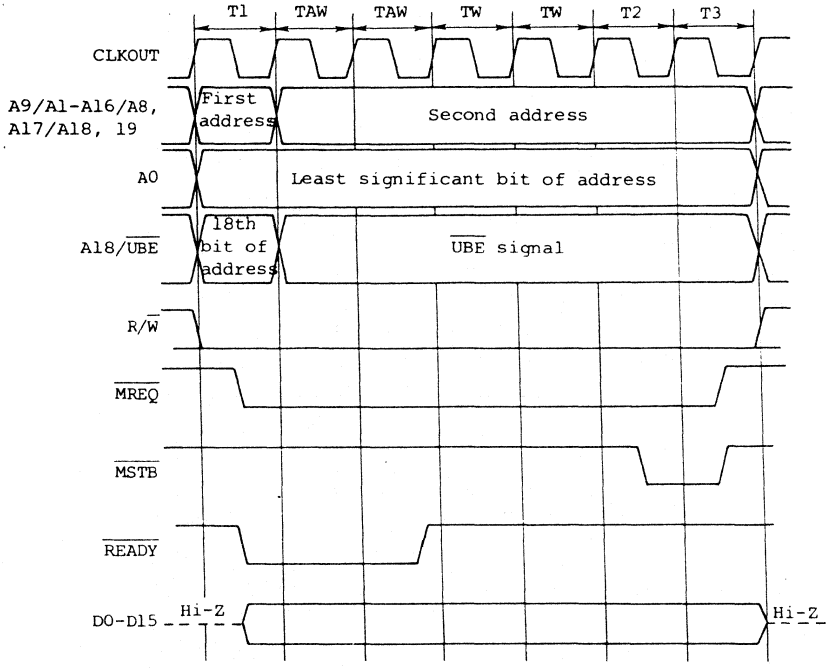


Fig. 4-10 Memory Write Cycle (During READY Pin Operation)

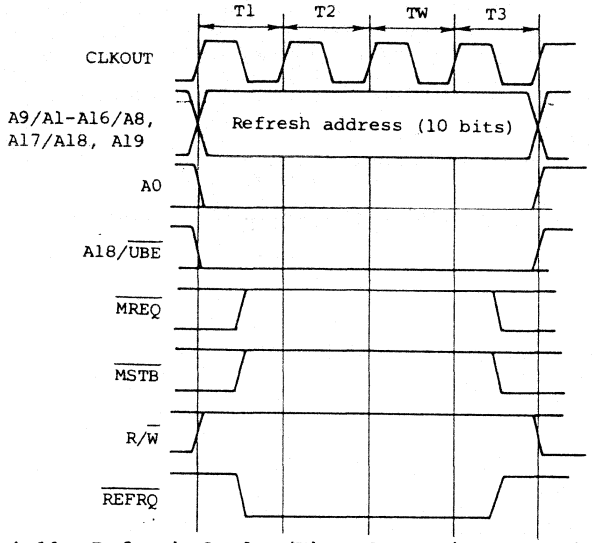


Fig. 4-11 Refresh Cycle (When One Wait State is Inserted)

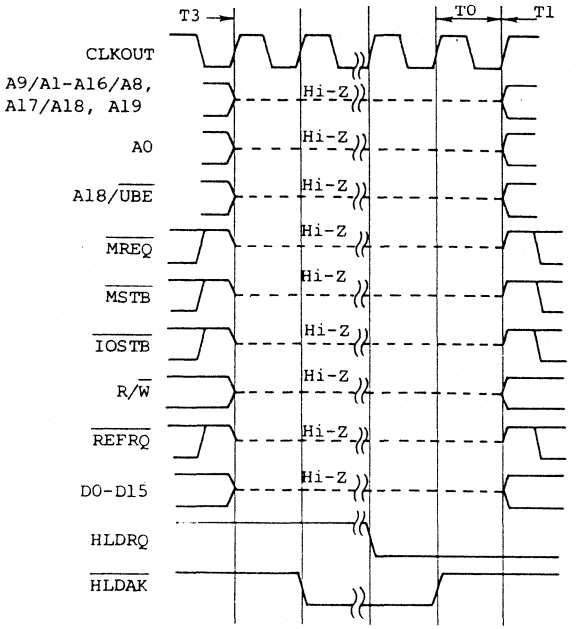


Fig. 4-12 Bus Hold Acknowledge Release Timing

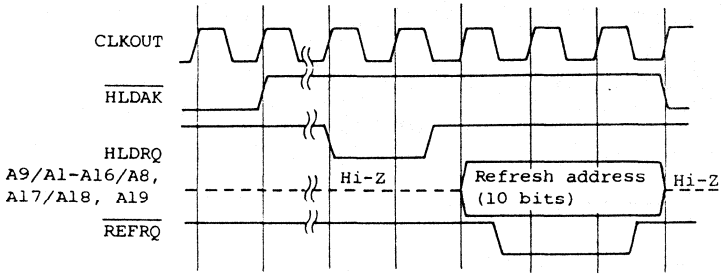


Fig. 4-13 Refresh Cycle in Hold Mode

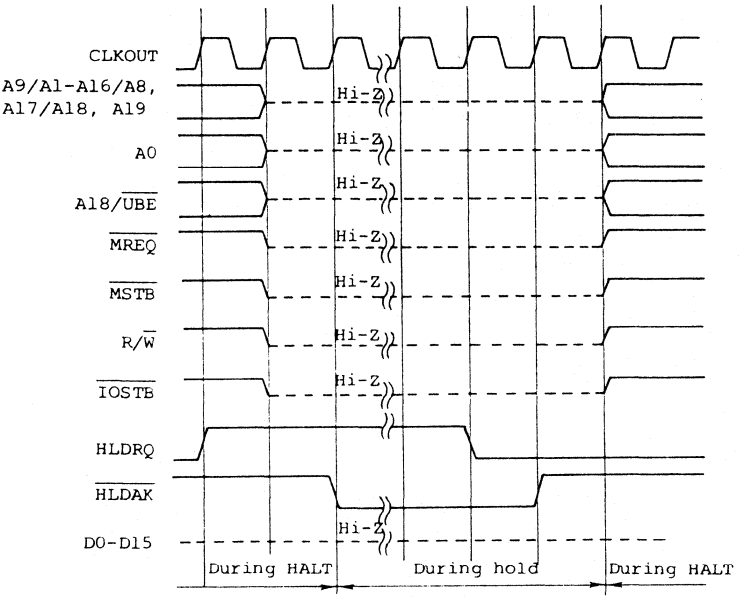


Fig. 4-14 Bus Hold Acknowledge Release Timing in HALT

5. DMA CONTROLLER

The uPD70332 and uPD70330 contain an internal 2-channel DMA controller to enable direct specification of the 1M-byte memory space.

5.1 Pin Functions

The pins described below are provided for the DMA controller. Since all the pins are also used as ports, set the appropriate port 2 mode control register (PMC2) bit to 1 to use the pin.

(1) DMARQ0 and DMARQ1 (P20 and P23)

Active high DMA request input pins

(2) $\overline{\text{DMAAK0}}$ and $\overline{\text{DMAAK1}}$ (P21 and P24)

Active low DMA response signal pins. However, no output is made in memory-to-memory DMA transfer (burst mode or single step mode).

(3) $\overline{\text{TC0}}$ and $\overline{\text{TC1}}$ (P22 and P25)

Active low DMA completion signal output pins. Output is made when TC0 and TC1 in the DMA service channel are set to 0.

5.2 DMA Operation

The uPD70332 and uPD70330 has four DMA transfer modes. Table 5-1 lists the transfer mode functions.

In DMA transfer, internal ROM and internal data area cannot be accessed. When addresses correponding to the internal data area and internal ROM are accessed, external memory assigned the same addresses will be accessed.

Table 5-1 Transfer Mode Functions

Mode name	Transfer between:	Function	DMA start	Stop method	Interrupt	During HALT	DMA request during operation
Single step	Memory and memory	When one DMA request is made, execution of one instruction and one DMA transfer are repeated alternately as many times as specified.	o DMARQ rising edge o DMA control register TDMA bit setting	By using soft-ware	Every interrupt is acknowledged.	Consecutive DMA transfer is made as many times as specified.	Channel 1 DMA is held or stopped, and channel 0 DMA is made.
Burst	Memory and memory	When one DMA request is made, consecutive DMA transfer is made as many times as specified.	o DMARQ rising edge o DMA control register TDMA bit setting	None	Not acknowledged during DMA transfer	Consecutive DMA transfer is made as many times as specified.	Another DMA is held until DMA transfer terminates.
One transfer	Memory and I/O	Each time a DMA request occurs, one DMA transfer is made.	o DMARQ rising edge ware	By using soft-ware	Every interrupt is acknowledged.	Same as normal	After one DMA transfer is made, requested DMA is made.
Demand release	Memory and I/O	While the DMARQ pin remains high, DMA transfer is made.	o DMARQ high level	o Stop at low level during DMA transfer. o In other cases, software is used	o Not acknowledged during DMA transfer. o In other cases, every interrupt is acknowledged.	Same as normal	While DMARQ remains high, other DMA is held.

In DMA transfer between memory and memory, the $\overline{\text{DMAAK}}$ signal is not output. In DMA transfer between memory and I/O, the $\overline{\text{DMAAK}}$ signal is output for each DMA cycle.

The programmable wait function (see 4.1) can also be used during DMA transfer. In DMA transfer between memory and memory, wait state specified for transfer destination and source is inserted. In DMA transfer between memory and I/O, one transfer is complete in a bus cycle. Thus the wait state for memory or I/O, whichever is the later, is inserted.

Wait state insertion timing is based on the memory side. In DMA transfer from I/O to memory, the wait state is inserted at a memory write cycle.

The bus hold function and refresh function can also be used during DMA transfer. When the function is executed, DMA transfer is temporarily halted.

DMA transfer is also made during execution of block processing (transfer, comparison, or retrieval input/ouput) instruction with a repeat prefix if DMA transfer is requested. At the time, block processing instruction execution is temporarily halted.

Likewise, DMA transfer is also made when a BUSLOCK prefix is added.

During DMA transfer, interrupts are not acknowledged, and all are held.

DMA transfer in the HALT mode is made if it is requested. When DMA transfer terminates, a return is made to the HALT mode. If a DMA transfer completion interrupt occurs when a return is made to the HALT mode, the HALT mode is released.

If DMA requests occur at the same time, channel 0 takes precedence.

When DMA transfer teminates (DMA transfer has bene made as many times as specified), an interrupt can be generated.

5.3 DMA Control Registers

The DMA mode registers and DMA control registers are provided to specify the DMA transfer mode, etc. The DMA service channels to specify the transfer destination, transfer source, and transfer count are mapped in internal RAM. Interrupt control registers are also provided. All the registers are provided for each channel.

5.3.1 DMA mode register (DMAM0, DMAM1)

The DMA mode register is an 8-bit register used to specify the DMA transfer mode, etc. The register can be written/read by 8-bit or 1-bit memory access operation. DMAM0 corresponds to channel 0; DMAM1 correponds to channel 1.

At RESET, the DMAMn register is initialized to 00H.

7	6	5	4	3	2	1	0
MD2	MD1	MD0	W	EDMA	TDMA	0	0

MD2-MD0: Transfer mode specification bits.

MD2	MD1	MD0	Transfer mode
0	0	0	Single step mode
0	0	1	Demand release mode (I/O to memory)
0	1	0	Demand release mode (memory to I/O)
0	1	1	Undefined
1	0	0	Burst mode
1	0	1	One transfer mode (I/O to memory)
1	1	0	One transfer mode (memory to I/O)
1	1	1	Undefined

W : Bit to specify whether transfer processing is performed in byte or word units.

0: Byte transfer

1: Word transfer

EDMA : Bit to specify whether DMA transfer is enabled or disabled.

1: DMA transfer is enabled.

0: DMA transfer is disabled.

The EDMA bit is automatically cleared when the DMA service channel terminal counter (TC) reaches 0.

TDMA : Transfer start bit

This bit is effective only in the single step or burst mode. When the bit is set to 1, DMA is started if the EDMA is set to 1. The TDMA bit read level is always 0. The TDMA bit is not significant in the demand release or one-transfer mode.

7	6	5	4	3	2	1	0
MD2	MD1	MD0	W	EDMA	TDMA	0	0

EDMA	TDMA	Transfer start
0	0	DMA disable state
1	0	State remains unchanged
1	1	DMA transfer start

EDMA	DMA transfer enable or disable
0	DMA disable state
1	DMA enable state

W	Transfer processing method
0	Byte transfer
1	Word transfer

MD2	MD1	MD0	Transfer mode
0	0	0	Single step mode
0	0	1	Demand release mode (I/O to memory)
0	1	0	Demand release mode (memory to I/O)
0	1	1	Undefined
1	0	0	Burst mode
1	0	1	One transfer mode (I/O to memory)
1	1	0	One transfer mode (memory to I/O)
1	1	1	Undefined

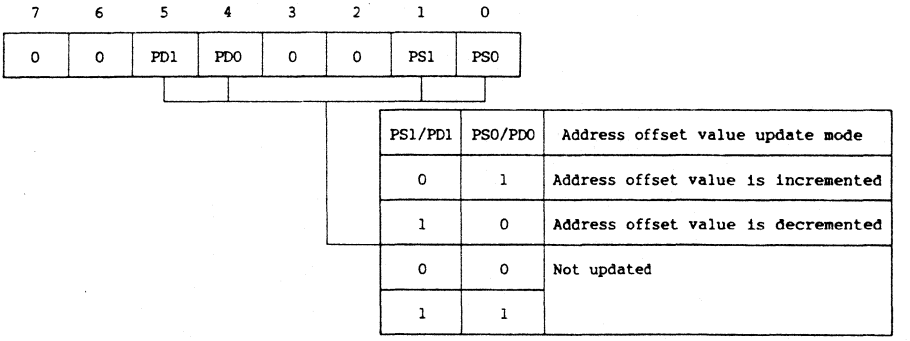
Fig. 5-1 DMA Mode Register (DMAM0,DMAM1) Format

5.3.2 DMA control register (DMAC0, DMAC1)

The 8-bit DMA control register specifies the source and destination address update modes in DMA transfer. The register can be written/read by 8-bit or 1-bit operation memory access.

At RESET, the DMACn register contents are held and are undefined.

As shown in Fig. 5-2, DMACn register bits 1 and 0 (PS1 and PS0) are used to specify the update mode of the source address offset value and bits 5 and 4 (PD1 and PD0) are used to specify the update mode of the destination address offset value.



PS1/PD1	PS0/PD0	Address offset value update mode
0	1	Address offset value is incremented
1	0	Address offset value is decremented
0	0	Not updated
1	1	

Fig. 5-2 DMA Control Register (DMAC0,DMAC1) Format

5.3.3 DMA service channels

The DMA service channels are used to specify the transfer source, destination, and count used for DMA transfer and are mapped in internal RAM. DMA service channel 0 is assigned to internal RAM addresses xxE00H-xxE07H, and channel 1 is assigned to xxE08H-xxE0FH (xx is a value specified in the IDB

register.) Macro service channels 0 and 1 and register bank 0 are also assigned to the same internal RAM area.

As in normal memory access, the DMA source and destination addresses are specified by using the segment and offset from the segment. Only eight bits of the segment can be specified, and the low-order eight bits are fixed to 0. If the address is updated when DMA transfer is made, only the offset is updated.

Fig. 5-3 shows the DMA service channel configuration Fig. 5-4 shows the DMA address generation method.

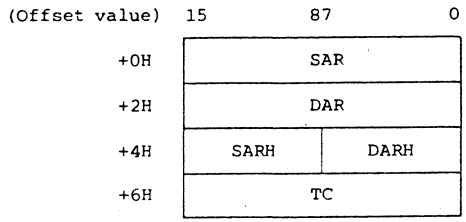
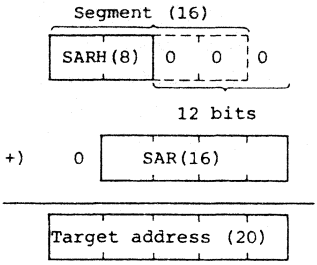


Fig. 5-3 DMA Service Channel Format

- o SAR (+0H): The DMA transfer source address offset (low-order 16 bits) is specified.
- o DAR (+2H): The DMA transfer destination address offset (low-order 16 bits) is specified.
- o DARH (+4H): The high-order eight bits of the DMA transfer destination address segment value are specified. The low-order eight bits of the segment value are fixed to 0.
- o SARH (+5H): The high-order eight bits of the DMA transfer source address segment value are specified. The low-order eight bits of the segment value are fixed to 0.
- o TC (+6H): The DMA transfer count is specified.

The value enclosed in parenthess is the offset from the DMA service channel start address.

o Source



o Destination

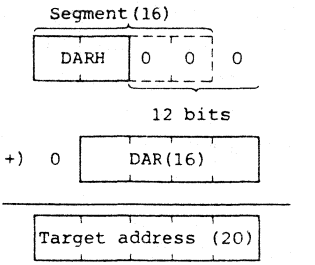


Fig. 5-4 DMA Address Generation Method

DMA service channel 0 is assigned to xxF00H, and channel 1 is assigned to xxE08H (xx is a value specified in the IDB register).

The DMA service channel is automatically updated when DMA operation is performed. TC is decremented by one each time one DMA transfer is made (byte or word data.) The address offset value is updated according to the mode specified in the DMA control register (DMACn). For byte data, the address offset value is incremented or decremented by one or remains unchanged; for word data, it is incremented or decremented by two or remains unchanged.

5.3.4 DMA interrupt request control register (DIC0, DIC1)

The 8-bit DMA interrupt request control register controls interrupts caused by the completion of DMA transfer. When the terminal counter (TC) is set to 0, an interrupt occurs.

The register can be written/read by 8-bit or 1-bit memory access operation. At RESET, the DICn regsiter is initialized to 47H.

The macro service function is not supported for DMA transfer completion interrupts. Channel 0 (INTD0) and channel 1 (INTD1) DMA transfer completion interrupts form one group interrupt; the higher interrupt priority is assigned to channel 0. INTD0 control is performed by using the DIC0 register and the vector is 36. INTD1 control is performed by using DIC1 and the vector is 37. (See 2.4.5.)

Interrupt priority: DF0 > DF1

7	6	5	4	3	2	1	0
DF0	DMK0	0	ENCS	0	PR2	PR1	PR0

DF1	DMK1	0	ENCS	0	1	1	1

Caution: DIC1 register bits 2-0 are fixed to 1 in hardware. The bits (PR2-PR0) are the interrupt request priority specificaion bit field for each group and form one group with the DIC0 register. The DIC1 register interrupt request priority conforms to the DIC0 register PR2-PR0 bit setting.

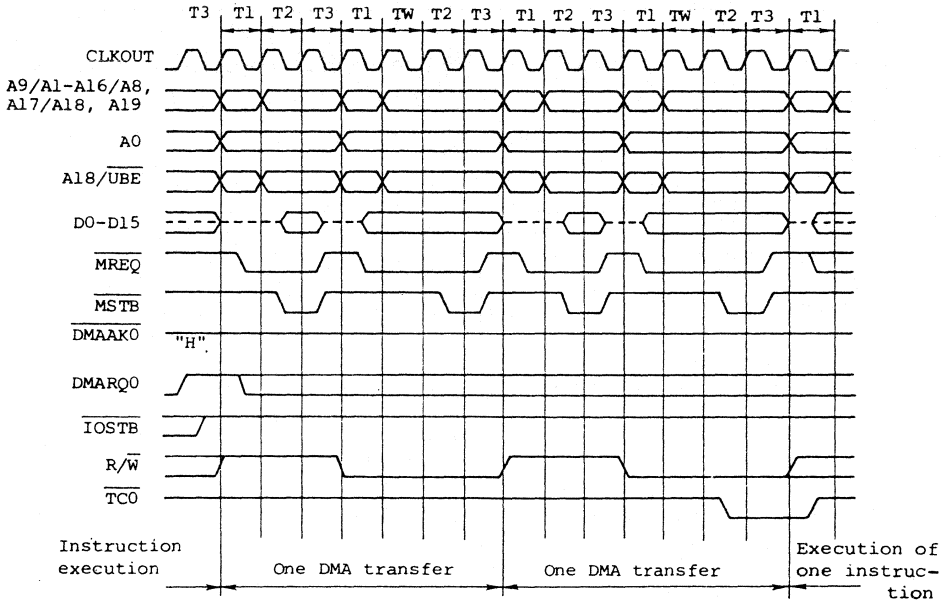
Fig. 5-5 DMA Interrupt Request Control Register (DIC0,DIC1) Format

The DF0 and DF1 bits are DMA transfer completion interrupt request flags. The DMK0 and DMK1 bits are DMA transfer completion interrupt mask bits.

For other bit fields, see 3.7.

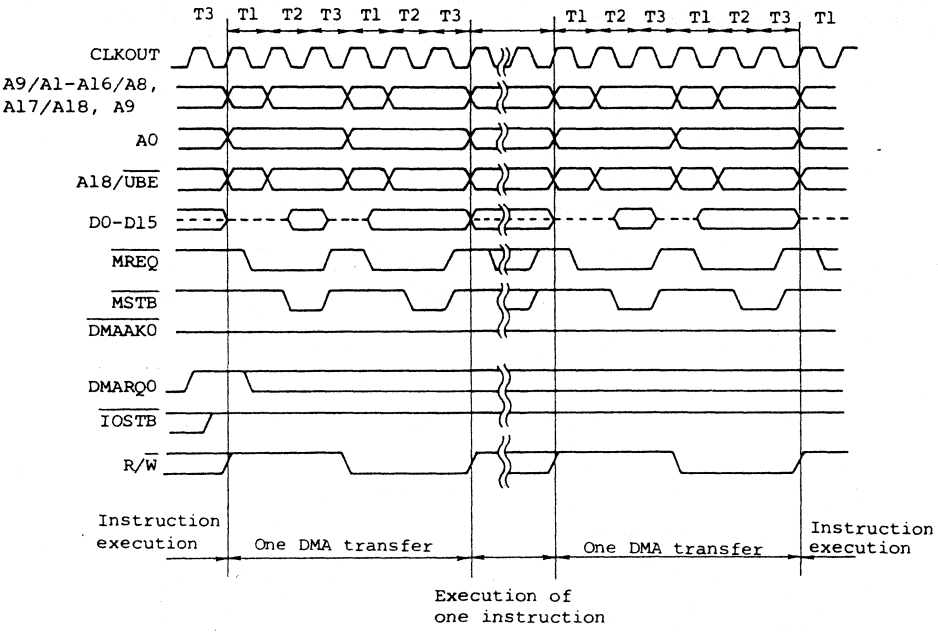
5.4 DMA Transfer Timing

Figures 5-6 to 5-9 show the main DMA transfer timings.



Remarks: The broken line denotes high impedance.

Fig. 5-6 In Burst Mode (DMA is started by using the DMARQ signal when TC=2. No wait is inserted in the transfer source memory bank; one wait state is inserted in the transfer destination. Byte transfer)



Remarks: The broken line denotes high impedance.

Fig. 5-7 In Single Step Mode

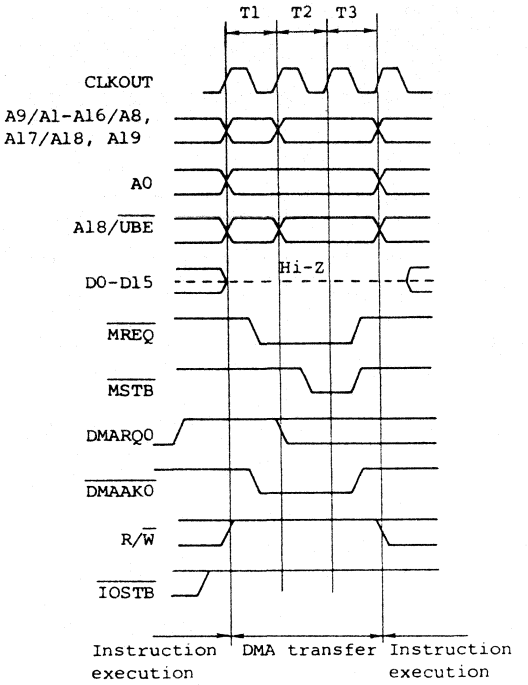


Fig. 5-8 One Transfer Mode (Memory to I/O with No Wait)

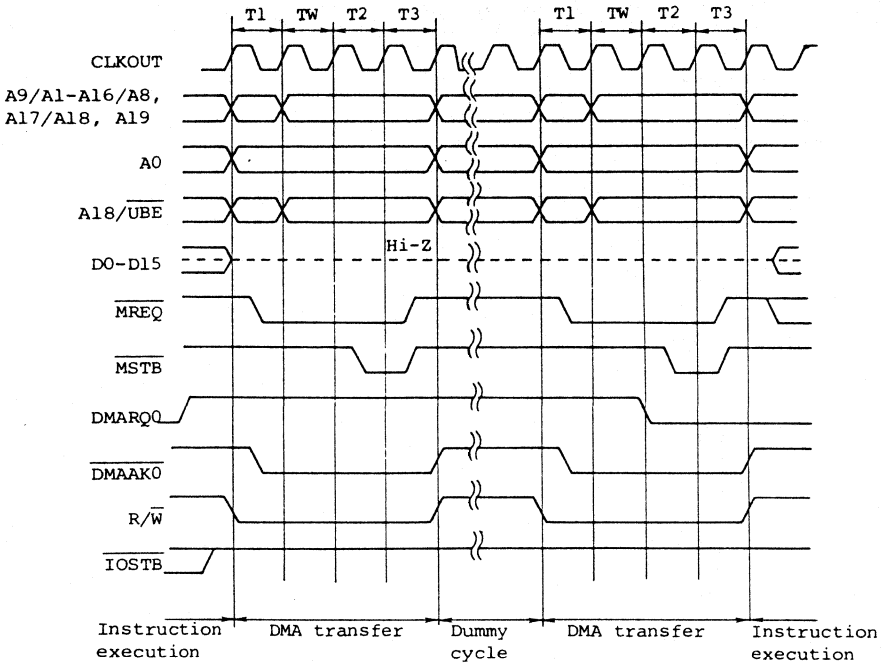


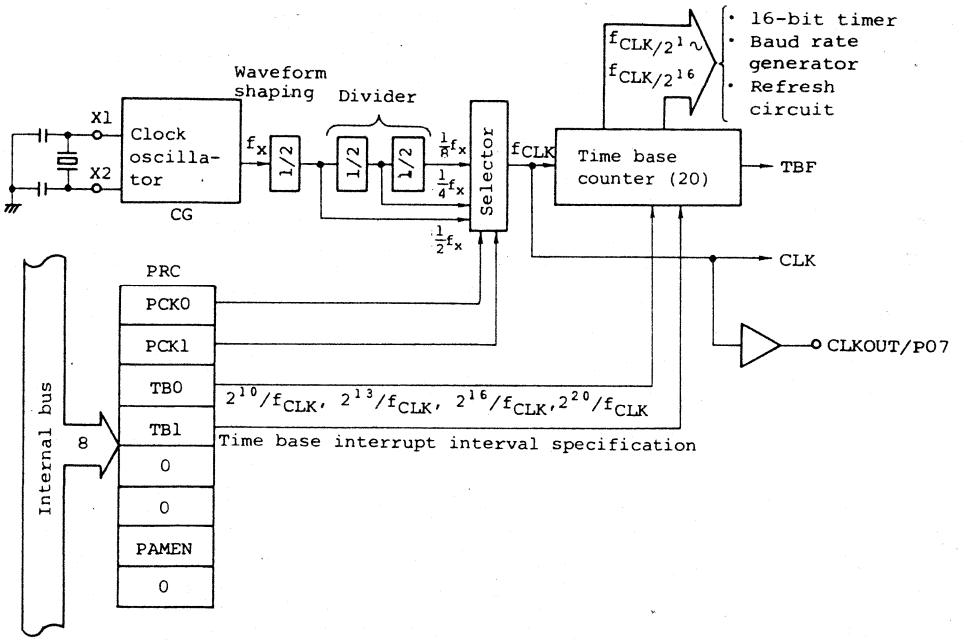
Fig. 5-9 Demand Release Mode (I/O to Memory, I/O = One Wait, Memory = No Wait)

6. CLOCK GENERATOR

The clock generator supplies various clocks to the CPU and peripheral hardware and controls the CPU operation mode.

6.1 Clock Generator Configuration

Fig. 6-1 shows the clock generator configuration.



f_X: Oscillation frequency
f_{CLK}: System clock frequency
PRC: Processor control register
TBF: Time base interrupt request flag
CLKOUT: System clock output pin

Fig. 6-1 Clock Generator Block diagram

The fundamental frequency of the clock generator is produced by the crystal oscillator connected to the X1 and X2 pins. The clock generator output waveform is shaped (divided by 2), divided by the value specified in PCK0 and PCK1, then used as system clock (CLK).

The CLK dividing ratio is selected from 1/2, 1/4, and 1/8 of the oscillation frequency (f_x) according to processor control register (PRC) bit 0, 1 (PCK0, PCK1) specification.

By placing CLK at low speed, the system clock frequency (f_{CLK}) is made low. Long-term operation is enabled even if the voltage of the battery drive system becomes lower.

6.2 Processor Control Register (PRC)

The 8-bit PRC register controls the CPU and internal system control items such as CPU operation clock, time base interrupt interval, and internal RAM memory reference enable.

The register can be written/read by 8-bit or 1-bit memory access operation.

When $\overline{\text{RESET}}$ is input, the register is initialized to 4EH.

The PCK0 and PCK1 bits are used to specify the system clock dividing ratio. The oscillator frequency is divided by the value specified in PCK0 and PCK1, then used as system clock (CLK).

The TB0 and TB1 bits are used to specify the time base interrupt interval. One of four long interval types can be selected.

The RAMEN bit is used to control internal RAM memory reference enable. When the RAMEN bit is set to 0 (disable state), internal RAM address decision is not made and

external memory is always accessed. When RAM is accessed as a register, internal RAM is always accessed.

7	6	5	4	3	2	1	0
0	RAMEN	0	0	TB1	TB0	PCK1	PCK0

PCK1	PCK0	System clock (CLK) dividing ratio specification
0	0	f_{CLK} = oscillator fundamental frequency (f_x) x 1/2
0	1	f_{CLK} = oscillator fundamental frequency (f_x) x 1/4
1	0	f_{CLK} = oscillator fundamental frequency (f_x) x 1/8
1	1	Undefined

TB1	TB0	Time base interrupt interval specification
0	0	An interrupt is generated at $2^{10}/f_{CLK}$ intervals
0	1	An interrupt is generated at $2^{13}/f_{CLK}$ intervals
1	0	An interrupt is generated at $2^{16}/f_{CLK}$ intervals
1	1	An interrupt is generated at $2^{20}/f_{CLK}$ intervals

RAMEN	Internal RAM access enable specification
0	Internal RAM access disabled
1	Internal RAM access enabled

Fig. 6-2 Processor Control Register (PRC) Format

7. TIME BASE COUNTER

The uPD70332 and uPD70330 have an internal long interval timer for the clock.

7.1 Time Base Counter Configuration

Fig. 7-1 shows the time base counter configuration.

The time base counter consists of 20 dividers for the system clock (CLK). The low-order bits of divider tap output are used for 16-bit timer count clock, baud rate generator input clock, refresh timing generation, and refresh address generation. Output taps 10, 13, 16 and 20 are used for time base interrupts.

The time base counter is cleared (00000H) only when $\overline{\text{RESET}}$ is input. After this, it is incremented continuously.

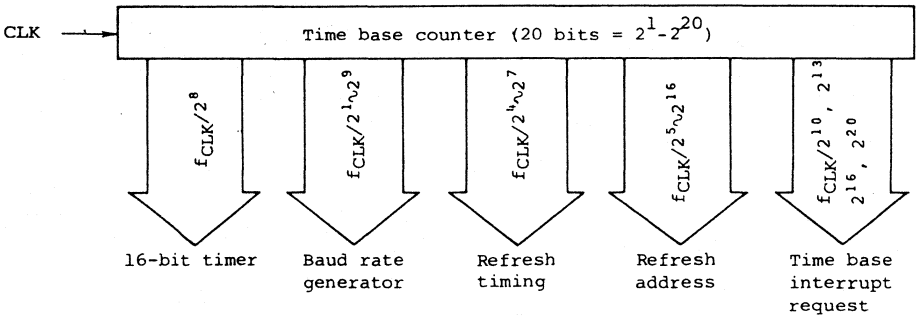


Fig. 7-1 Time Base Counter Configuration

7.2 Time Base Interval Specification

The interval time of an interrupt request generated from the time base counter can be selected from the four types shown in Fig. 7-2 by using processor control register (PRC) bits 2 and 3 (TB0 and TB1).

7	6	5	4	3	2	1	0
0	RAMEN	0	0	TB1	TB0	PCK1	PCK0

f_{CLK} = 5 MHz (= $\frac{1}{2}f_X$: f_X = 10 MHz)

TB1	TB0	Interval time
0	0	$2^{10}/f_{CLK}$ (205us)
0	1	$2^{13}/f_{CLK}$ (1.64ms)
1	0	$2^{16}/f_{CLK}$ (13.1ms)
1	1	$2^{20}/f_{CLK}$ (210ms)

Caution: The time to the first interrupt request generation immediately after TB0 and TB1 bits are set becomes undefined.

Fig. 7-2 Processor Control Register (PRC) Interval Timer Mode

7.3 Time Base Interrupt Request Control Register (TBIC)

The 8-bit TBIC register performs mask control of an interrupt request generated from the time base counter.

The TBIC register can be written/read by 8-bit or 1-bit memory access operation.

When $\overline{\text{RESET}}$ is input, the TBIC register is initialized to 47H.

7	6	5	4	3	2	1	0
TBF	TBMK	0	0	0	1	1	1

Fig. 7-3 Time Base Interrupt Request Control Register (TBIC) Format

When the time base counter output taps specified by using the processor control register (PRC) go high, the interrupt request flag (TBF) is set to 1 and an interrupt request is generated.

TBIC bits 4 and 5 are fixed to 0. The register bank switching function and macro service function are not contained in timer base counter interrupts. Since TBIC bits 0-2 are fixed to 1, the time base interrupt (INTTB) priority is fixed to 7. Even if another interrupt with priority 7 is made, the time base interrupt (INTTB) is fixed to the lowest priority. However, it is subjected to multiprocessing control.

8. SERIAL INTERFACE

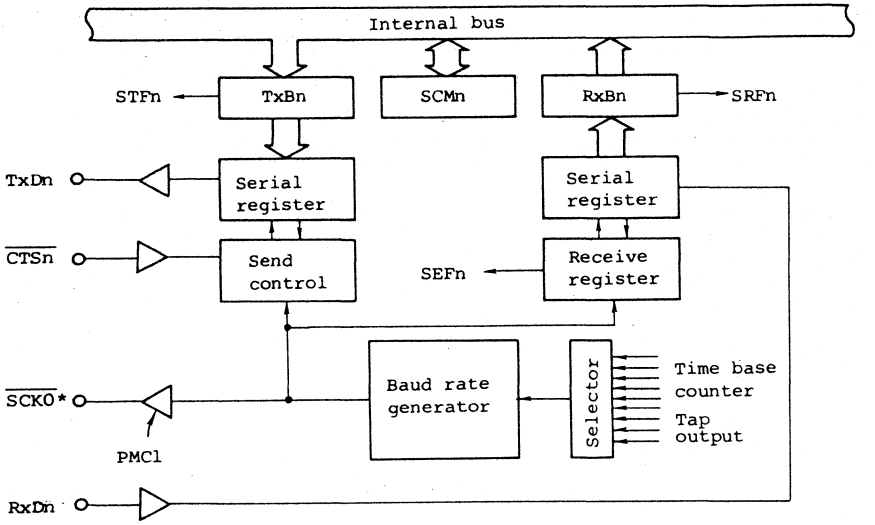
8.1 Serial Interface Configuration

The uPD70332 and uPD70330 include two serial interface channel with an internal dedicated baud rate generator.

A start/stop-bit transfer system is used in the serial interface. Two operating modes are provided: Asynchronous mode in which bit synchronization and character synchronization of data are taken by using the start bit, and the I/O interface mode in which data is transferred in synchronization with controlled serial clock as in the serial data transfer system of the uCOM-87 series. Fig. 8-1 shows the serial interface configuration when the asynchronous mode is set and when the I/O interface mode is set.

The serial interface consists of the serial data input (RxDn), serial data output (TxDn), serial clock output ($\overline{SCK0}$), send enable state control input ($\overline{CTSn}$) pins, the transfer control block, the 8-bit serial registers for send and receive, the transmit buffer (TxBn), the receive buffer (RxBn), and the baud rate generator. Since serial registers and buffers are provided for sending and receiving separately, sending and receiving can be performed independently (full duplex operation can be performed). Since the $\overline{CTSn}$ pin serves as receive clock input/output pin in the I/O interface mode, full duplex serial operation can also be performed in the I/O interface mode.

(a) When asynchronous mode is set (n=0, 1)



* Channel 0 only (fixed to high level in the synchronous mode)

(b) When I/O interface mode is set (channel 0 only)

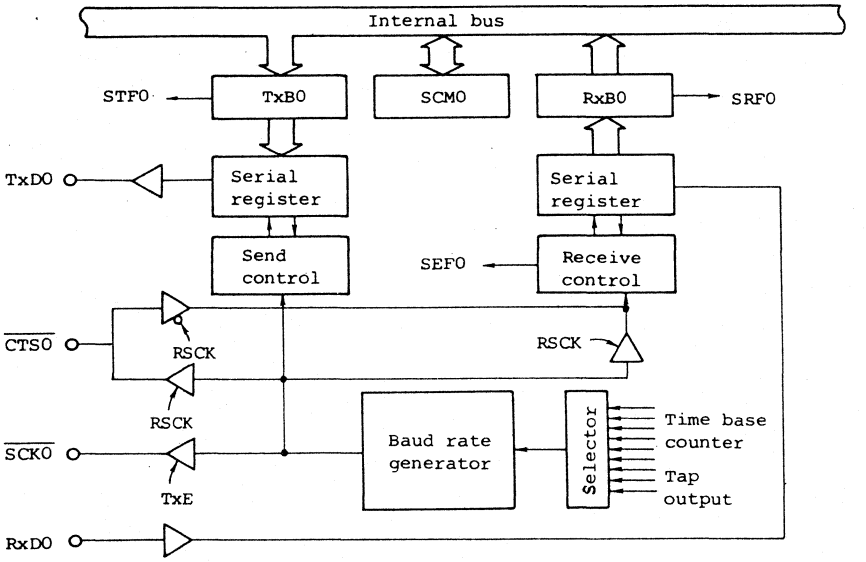


Fig. 8-1 Serial Interface Configuration

8.2 Asynchronous Mode

In the asynchronous mode, the character length, the number of stop bits, parity enable, and odd or even parity can be specified by using the serial mode register (SCMn).

(1) Send

Send operation is enabled when serial mode register (SCMn) bit 7 (TxRDY) is set to 1 and the $\overline{CTSn}$ pin is active low. (See Note.)

Send is started in any of the following three manners:

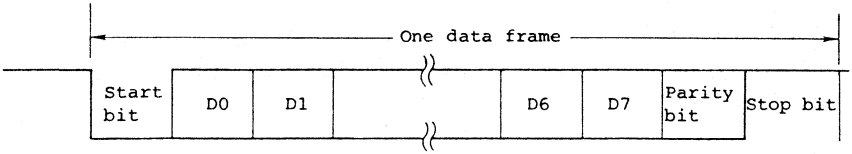
(i) By setting the send enable state when the transmit buffer (TxB) is empty, a send completion interrupt request is generated, and send data is written into the transmit buffer in the interrupt service.

(ii) When send data is transferred to the transmit buffer in the send enable state, the send data is consecutively sent after the preceding send operation terminates.

(iii) When the send enable state is set after send data is prewritten into the transmit buffer in the send disable state, the data stored in the transmit buffer is sent.

Note: There are no special limitations on the send enable state setting sequence. After TxRDY is set to 1, the $\overline{CTSn}$ pin can be made active (low); after the $\overline{CTSn}$ pin is made active (low), TxRDY can be set to 1.

One send data frame is made up of a start bit, character bits, parity bit, and a stop bit or bits as shown below. Data is sent starting at the least significant bit (LSB) from the TxDn pin. The TxDn pin is placed in the mark state (1) when the send disable state is set or the data to be sent is not contained in the serial register.



Number of start bits : One
Number of character bits: Seven or eight
Parity bit : Odd, even, 0, or not added
Number of stop bits : One or two

A second completion interrupt request occurs immediately when the transmit buffer (TxBn) becomes empty. When $\overline{\text{RESET}}$ is input, the transmit buffer (TxBn) becomes empty. If the send enable state is set at the time, a send completion interrupt request occurs. When send operation starts and the send data in the transmit buffer is transferred to the shift register. the transmit buffer becomes empty and a send completion interrupt request occurs.

Data can be sent consecutively without mark state (1) insertion by writing send data into the transmit buffer each time a send completion interrupt request occurs.

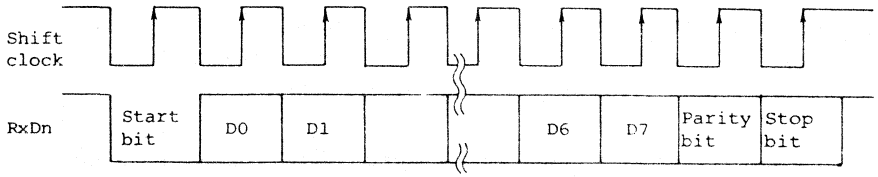
If a change is made to the send disable state while send operation is being performed, the complete frame of the data is sent. However, if new send data is

already written into the transmit buffer, data transfer from the transmit buffer to the shift register is disabled, and the send buffer contents are held. If again the send enable state is set, the transmit buffer contents are transferred in synchronization to the shift register, and a send completion interrupt request occurs at the same time as send starts.

(2) Receive

Receive operation is enabled when serial mode register (SCMn) bit 6 (RxE) is set to 1. (In the receive disable state (RxE=0), receive hardware stands by in the initial state.)

RxDn pin input is sampled on the input clock to the baud rate generator. When the falling edge is detected, receive operation is started and the baud rate generator for receive starts counting. If a low level input to the RxDn pin is detected with the first timing signal from the baud rate generator for receive, it is recognized as the start bit, and the subsequent receive operation is performed. If a high level is detected with the first timing signal, it is not recognized as the start bit. The baud rate generator is initialized and stops operation. Receive data is sampled in synchronization with the shift clock rising edge after the start bit is detected, as shown below:



Receive Data Sampling Timing

At the completion of character reception for the character length specified in serial mode register bit 3 (CL), the receive data in the shift register is transferred to the receive buffer (RxBn), and a receive completion interrupt request is generated.

During receive, if an odd/even parity check is made (when PRTY1 bit(Note) = 1). If a mismatch is detected (parity error), the stop bit level is low (framing error), or the next data is transferred to the receive buffer when the receive buffer is full (overrun error), the receive error flag is set and a receive error interrupt request occurs. (See 8.6.)

Note: The PRTY1 bit is serial mode register bit 5.

8.3 I/O Interface Mode

The I/O interface mode is the same as in the uCOM-87 serial interface mode. It is useful for extending I/O externally or for connecting an I/O controller such as an A/D converter or liquid crystal controller.

In the I/O interface mode, data is transferred starting at the most significant bit (MSB); the character length is fixed to eight bits with no parity bit.

The I/O interface mode can be used only on channel 0.

(1) Send

Send operation is enabled when serial mode register bit 7 (TxE) is set. In the I/O interface mode, the $\overline{SCK0}$ pin is used as the send clock output pin.

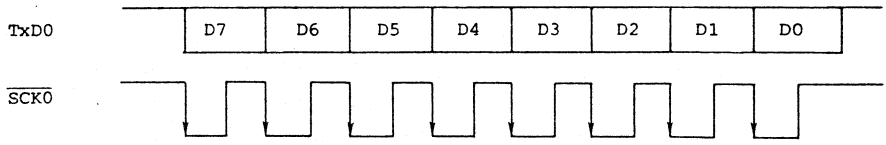
As in the asynchronous mode, send operation is started in any of the following three manners:

(i) By setting the send enable state when the transmit buffer (TxB0) is empty, a send completion interrupt request is generated and send data is written into the transmit buffer in the interrupt service.

(ii) When send data is transferred to the transmit buffer (TxB0) in the send enable state, the send data is sent consecutively after the preceding send operation terminates.

(iii) When the send enable state is set after send data is prewritten into the transmit buffer (TxB0) in the send disable state, the data stored in the transmit buffer (TxB0) is sent.

The send data format is shown below. The data length is fixed to eight bits and the data is sent starting at the most significant bit (MSB).



A send completion interrupt request occurs immediately when the transmit buffer (TxB0) becomes empty. When $\overline{\text{RESET}}$ is input, the transmit buffer (TxB0) becomes empty. If the send enable state is set at this time, a send completion interrupt request occurs. When send operation starts and the send data in the transmit buffer (TxB0) is transferred to the shift register, the transmit buffer becomes empty, and a send completion interrupt request occurs.

(2) Receive

Receive operation is enabled when serial mode register (SCM0) bit 6 (RxE) is set to 1. Receive data is input to the serial register on the rising edge of the receive clock. When the 8-bit data is received in the serial register, the data is transferred from the serial register to the receive buffer (RxB0), and receive completion interrupt request is generated.

The receive clock in the I/O interface mode can be selected from the external and internal receive clocks by using serial mode register (SCM0) bit 2 (RSCK).

In the I/O interface mode, the $\overline{\text{CTS0}}$ pin serves as the receive clock input/output pin.

While receiving, if the next data is transferred to the receive buffer (RxB0) when the receive buffer is full (overrun error), the receive error flag is set, and a receive error interrupt request occurs.

8.4 Serial Mode Register (SCM0, SCM1)

The 8-bit SCMn register (n=0 or 1) specifies the serial interface transfer mode. It can be set for each of channels 0 and 1 (SCM0 and SCM1). The meanings of SCMn bits 2 to 7 vary depending on how bits 1 and 0 (MD1 and MD0) are set.

7	6	5	4	3	2	1	0
						MD 1	MD 1

MD1, MD0 = 0, 1 (asynchronous mode)

	7	6	5	4	3	2	1	0
SCM0/ SCM1	TxRDY	RxE	PRTY 1	PRTY 0	CL	SL	0	1

MD1, MD0 = 0, 0 (I/O interface mode)

	7	6	5	4	3	2	1	0
SCM0	TxE	RxE	0	0	TSK	RSCK	0	0

The MD1, MD0 bit field is used to specify the serial interface transfer mode. If MD1, MD0 = 0, 1 is set, the asynchronous mode is specified; if MD1, MD0 = 0, 0 is set, the I/O interface mode is specified. However, the I/O interface mode can be specified only in SCM0.

SCMn can be read/written in 8-bit or 1-bit operation by memory access.

When $\overline{\text{RESET}}$ is input, the SCMn register is cleared (00H).

(1) When asynchronous mode is set

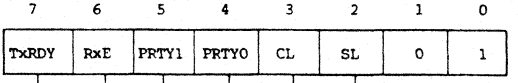
RxE : Receive enable control
When the RxE bit is set to 0 (receive disable state) during receive operation, receive processing is stopped and no receive completion interrupt occurs.

SL : Stop bit specification
When the SL bit is cleared, the number of stop bits is set to 1; when SL is set to 1, the number is set to 2.

CL : Character length specification
When the CL bit is cleared, the character length is set to seven bits; when CL is set to 1, the character length is set to eight bits.

PRTY0 and PRTY1 :
Parity addition specification
The PRTY0 and PRTY1 bits are used to specify no parity, odd parity, even parity, or 0 parity. If 0 parity is specified, the parity bit is set to 0 for send and ignored for receive.

TxRDY : Send enable state control bit
When the $\overline{\text{CTSn}}$ pin level is low and $\overline{\text{TxRDY}}$ is set to 1, the send enable state is set.



Stop bit length specification

0	One stop bit
1	Two stop bits

Character length specification

0	Seven bits
1	Eight bits

Parity addition specification

PRTY 1	PRTY 0	Parity specification
0	0	No parity
0	1	0 parity*
1	0	Odd parity
1	1	Even parity

* For the 0 parity, the parity bit is set to 0 for send and ignored for receive.

Receive enable control

0	Receive disable
1	Receive enable

Send enable control

0	Send disable
1	Send enable when the $\overline{CTSn}$ pin level is low.

Fig. 8-2 Serial Mode Register (SCM0, SCM1) Format (When Asynchronous Mode is Set)

(2) When I/O interface mode is set

[RSCK] : Serial receive clock source specification
When the RSCK bit is cleared, receive operations use the external receive clock; when RSCK is set to 1, receive operation is performed using the internal receive clock. Receive clock is input/output by using the $\overline{CTS0}$ pin.

[TSK] : Receive clock output trigger
The TSK bit is effective only when the RSCK bit is set to 1. Eight shift clocks for receive are output from the $\overline{CTS0}$ pin by setting the TSK bit to 1.

When a serial clock is output, the TSK bit is automatically cleared.

[RxE] : Receive enable control
When the RxE bit is set to 1, the receive enable state is specified; when RxE is cleared, the receive disable state is specified.

If the receive disable state is set during a receive operation, receive processing is stopped at the time and no receive completion interrupt request occurs.

[TxE] : Send enable control
When the TxE bit is set to 1, the send enable state is specified; when TxE is cleared, the send disable state is specified.

When transfer data is written into the transmit buffer in the send enable state (TxE=1), if data is being sent, the corresponding serial send is started after send processing is complete. If data is not sent, the corresponding serial send is started immediately.

When send data is written into the transmit buffer in the send disable state (TxE=0), serial send is not performed and the data in the transmit buffer remains held. Then, sending the send data stored in the buffer is started at the same time a change is made to the send enable state.

Even if the TxE bit is cleared (send disable state) during send operation, the send operation is performed to completion. However, if the next send data is already stored in the transmit buffer when the disable state is set, sending is postponed and the data remains held in the buffer.

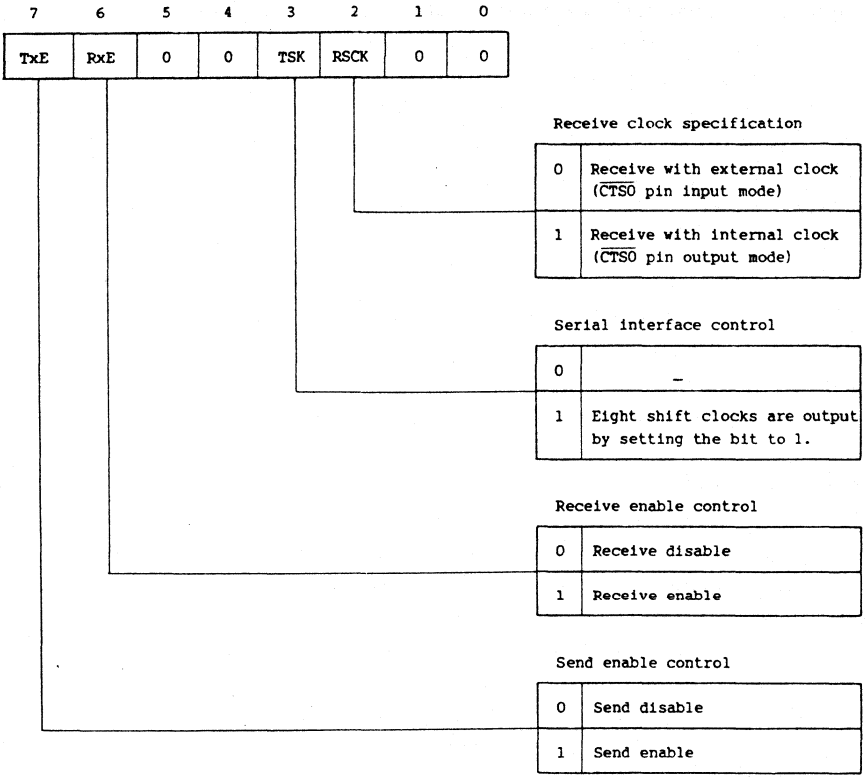


Fig. 8-3 Serial Mode Register (SCM0) Format when I/O Interface Mode is Set.

8.5 Baud Rate Generator

The baud rate generator is an 8-bit timer dedicated to the serial interface to generate send and receive shift clocks. The send and receive baud rate generators are provided for each channel. The same baud rate is used for both sending and receiving. The baud rate is determined by writing a

value into the baud rate generator register (BRGn). The maximum baud rate is 750 kbps. Data send or receive at baud rates exceeding 750 kbps is enabled by inserting idle time of one clock or more between data segments.

Input clock to the baud rate generator is specified by selecting a time base counter (See 7.1) output tap in serial control register (SCCn) PRS3-PRS0. The baud rate generator output signal is used for serial interface shift clock. To set the baud rate generator for the transfer rate, the parameter values should be set so as to satisfy the following expression:

$$B \cdot G = 10^6 \times \frac{f_{CLK}}{2^{n+1}}$$

Where:

B: Transfer baud rate (bps)
B = 110, 150, ..., 9600, 19200, ...

G: Baud rate generator register (BRGn) setup value
$(2 \leq G \leq 255)$

n: Input clock specification number to the baud rate generator specified in the serial control register
$(0 \leq n \leq 8)$

CLK: System clock frequency (MHz)

Table 8-1 lists the baud rate generator setup values for the standard transfer baud rates based on the expression when an external 10-MHz crystal is used.

Table 8-1 Baud Rate Generator Setup Values (for reference)

f_{CLK} = 5 MHz ($=\frac{1}{2}f_X$: f_X=10 MHz)

Transfer baud rate	n	BRGn register setup value n	Error (%)
110	7	178	0.25
150	7	130	0.16
300	6	130	0.16
600	5	130	0.16
1200	4	130	0.16
2400	3	130	0.16
4800	2	130	0.16
9600	1	130	0.16
19200	0	130	0.16
38400	0	65	0.16
1.25M	0	2	0

n: Specification number for input clock to the baud rate generator

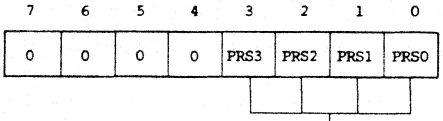
8.5.1 Serial control register (SCC0, SCC1)

The SCCn register (n=0 or 1) is used to control the serial interface transfer rate.

The register can be written/read by 8-bit or 1-bit memory access operation.

When $\overline{\text{RESET}}$ is input, the register is initialized to 00H.

Inpact of the time base counter output tap to the baud rate generator is specified in the PRS3-PRS0 bit field.



PRS3	PRS2	PRS1	PRS0	Input clock to baud rate generator
0	0	0	0	f_{CLK}
0	0	0	1	$f_{CLK}/2$
0	0	1	0	$f_{CLK}/4$
0	0	1	1	$f_{CLK}/8$
0	1	0	0	$f_{CLK}/16$
0	1	0	1	$f_{CLK}/32$
0	1	1	0	$f_{CLK}/64$
0	1	1	1	$f_{CLK}/128$
1	0	0	0	$f_{CLK}/256$

f_{CLK}: System clock frequency

Fig. 8-4 Serial Control Register (SCC0, SCC1) format

8.6 Serial Error Handling

The following three types of serial interface receive errors can be detected:

i) Parity error (in asynchronous mode)
Send parity and receive parity do not match.

ii) Framing error (in asynchronous mode)
Stop bit is not detected.

iii) Overrun error (in asynchronous or I/O interface mode)
The next receive is complete before the preceding data is received from RxB.

8.6.1 Serial error register (SCE0, SCE1)

The serial error register is an 8-bit register indicating the error flag state for each type of error. Registers are provided for both channels 0 and 1.

The register can only be read by 8-bit memory access operation.

When RESET is input, the register is initialized to 00H.

[ERPn]: Parity error flag
When send parity and receive parity do not match, the ERP flag is set to 1. When receive data is read from the receive buffer, the flag is cleared.

[ERFn]: Framing error flag
When stop bit is not detected, the ERF flag is set to 1. When receive data is read from the receive buffer, the flag is cleared.

[EROn]: Overrun error flag
When the next receive is complete before the preceding receive data is received from RxB, the ERO flag is set to 1. When receive data is read from the receive buffer, the flag is cleared.

[RxDn]: Bit to check the receive pin input state with the RxB bit.
When $\overline{\text{RESET}}$ is input, the serial error register (SCEn) is initialized to 00H.

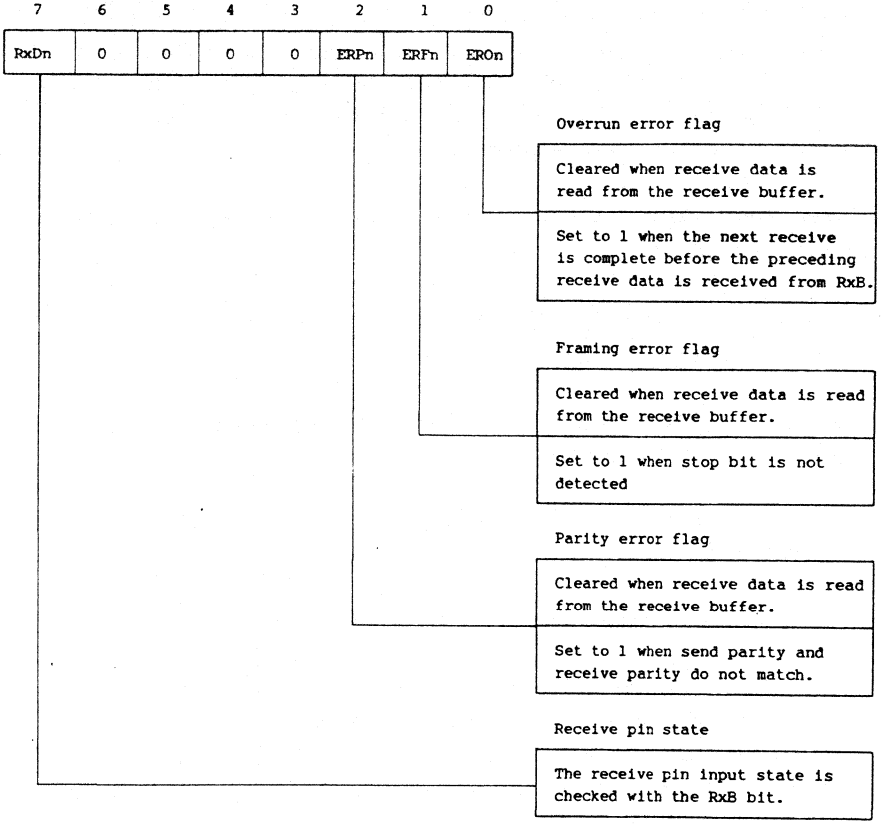


Fig. 8-5 Serial Error Register (SCM0, SCE1) Format (When n = 0 or 1)

8.7 Break Detection Function

The uPD70332 and uPD70330 can detect the line break state by software processing (only in the asynchronous mode). The break state detection sequence is described below:

1. Receive error interrupt occurrence caused by the first framing error

   Check the receive data in the receive error handling routine to ensure that the data is 00H.

   At the same time, check the receive error flags to ensure that the error is a framing error.

2. Receive error interrupt occurrence caused by the second framing error

   In the break state, again a framing error occurs.

   To determine the line to be in the break state, check again that the receive data is 00H and data of 00H is consecutively received with a framing error, and check directly the pin state using serial error register (SCEn) bit 7 (RxDn).

8.8 Serial Interface Interrupt Requests

The interrupt requests generated from the serial interface are send completion interrupt, receive completion interrupt, and receive error interrupt requests for each channel (0 and 1).

8.8.1 Interrupt request control registers (SEICn, SRICn, and STICn where n is 0 or 1)

The interrupt request control registers are used to control a receive error interrupt request (SEFn), receive completion interrupt request (SRFn), and send completion interrupt request (STFn) generated at the serial interface. The three registers form one group, and the serial interface interrupt request priority can be specified. The priorities in the group are defined in the hardware as follows:

SEFn > SRFn > STFn

	7	6	5	4	3	2	1	0
SEIC0/SEIC1	SEFn	SEMKn	MS/$\overline{INT}$	ENCS	0	PR2	PR1	PR0
SRIC0/SRIC1	SRFn	SRMKn	MS/$\overline{INT}$	ENCS	0	1	1	1
STIC0/STIC1	STFn	STMKn	MS/$\overline{INT}$	ENCS	0	1	1	1

Caution: SRICn and STICn bits 2-0 are fixed to 1 in hardware. The bit 2-0 field (PR2-PR0) is used to specify the interrupt request priority for each group and form one group with SEICn. The SRICn and STICn interrupt request priorities conform to the setting of the SEICn PR2-PR0 bit.

Fig. 8-6 Interrupt Control Register (SEICn, SRICn STICn) Format (When n = 0 or 1)

The SEFn, SRFn, and STFn bits are interrupt request flags which are set to 1 at receive error occurrence, receive completion, and send completion, respectively, and are cleared when an interrupt request is acknowledged or by using software.

For other bit fields, see 3.7.

The registers can be written/read by 8-bit or 1-bit memory access operation.

When $\overline{\text{RESET}}$ is input, the SEICn, SRICn, and STICn registers are initialized to 47 H.

8.8.2 Macro service control registers (SRMSn and STMSn where n = 0 or 1)

The 8-bit SRMSn register is used to specify the macro service processing mode and channel accompanying serial interface receive completion. The 8-bit STMSn register is used to specify the macro service processing mode and channel accompanying serial interface send completion. The SRMSn and STMSn registers are provided for each channel (0 and 1).

The registers can be written/read by 8-bit or 1-bit memory access operation.

See 3.4.3 for the macro service control register bits.

7	6	5	4	3	2	1	0
MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0

MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0

Fig. 8-7 Macro Service Control Register (SRMSn, STMSn) Format (When n = 0 or 1)

9. TIMER UNIT

The uPD70332, uPD70330 timer unit can be used for an interval timer, one-shot timer, and square wave output.

9.1 Timer Unit Configuration and Operation

The timer unit consists of two 16-bit timer registers, two 16-bit modulo/timer registers, and one 8-bit timer control register. The timer unit configuration and operation are explained in each operation mode.

(1) Interval timer mode

When the timer unit is set to the interval timer mode, timers 0 and and 1 can be used as shown in Fig. 9-1.

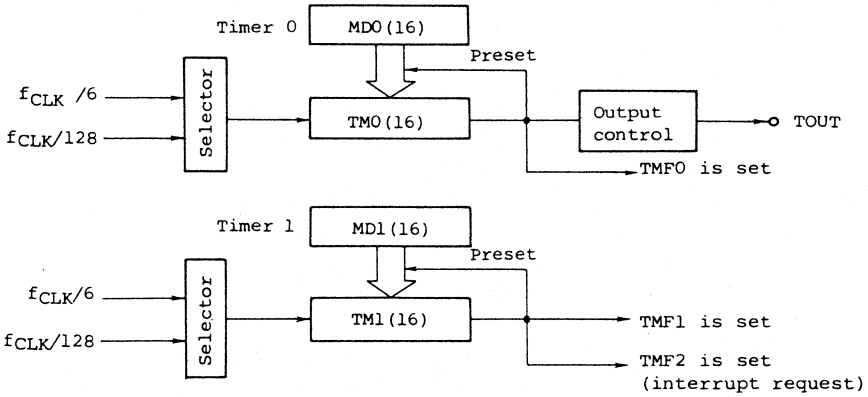


Fig. 9-1 Timer Unit Configuration in Interval Timer Mode

When the interval timer mode is specified in the timer control register (TMC0) and the TS0 bit is set to 1, the MD0 register value is set in the TM0 register and the clock specified in the TCLK0 bit is counted down. if an underflow occurs during counting down, again the

MD0 register value is set in the TM0 register an again counting down is repeated.

For the timer 1 register, similar count operation is performed.

(2) One-shot timer mode

When the timer unit is set to the one-shot timer mode, timer 0 is used as shown in Fig. 9-2. However, timer 1 can also be operated as an interval timer at the same time.

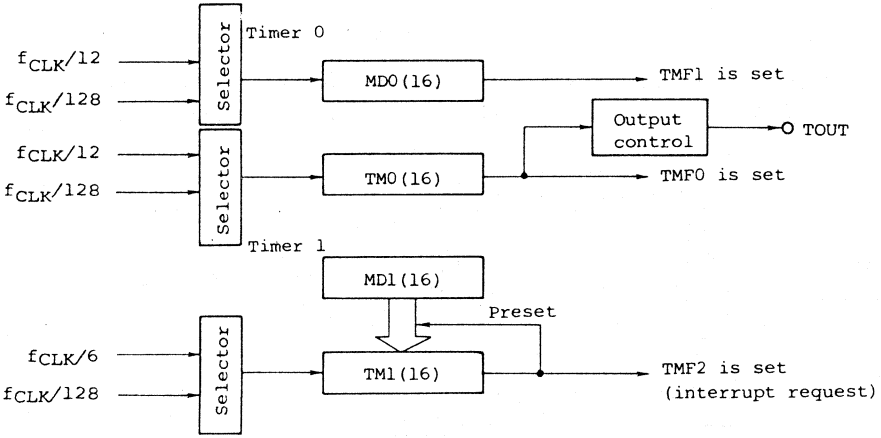


Fig. 9-2 Timer Unit Configuration in One-Shot Timer Mode

When the one-shot timer mode is specified in the timer control register (TMC0), and the TS0/MS0 bit is set to 1, the TM0/MD0 register counts down the clock specified in the TCLK0/MCLK0 bit. if an underflow occurs during down counting, the count is stopped, and the TM0/MD0 register holds 0000H.

9.2 Timer Control Registers (TMC0 and TMC1)

The 8-bit TMC0 register controls the operation of the TM0 and MD0 registers. The 8-bit TMC1 register controls the operation of the TM1 and MD1 registers.

The registers can be written/read by 8-bit or 1-bit memory access.

When $\overline{\text{RESET}}$ is input, the registers are initialized to 00H. TMC0 and TMC1 differ in format as shown below:

(Interval or single-shot timer mode)

	7	6	5	4	3	2	1	0
TMC0	TS0	TCLK0	MS0	MCLK0	ENTO	ALV	MOD1	MOD0

(Interval timer mode)

	7	6	5	4	3	2	1	0
TMC1	TS1	TCLK1	0	0	0	0	0	0

The operating modes of timer 0 (consisting of TM0 and MD0) and timer 1 (consisting of TM1 and MD1) are specified by using TMC0 and TMC1 register bits 0 and 1 (MOD0 and MOD1).

MOD0 and MOD1: Timer 0 or 1 operating mode specification bits

MOD0 = 0, MOD1 = 0: Interval timer operating mode is specified.

MOD0 = 1, MOD1 = 0: One-shot timer operating mode is specified.

In the interval time operating mode, TM0 and TM1 are used as timer registers to count down the setup value, and MD0 and MD1 are used as modulo registers which hold the interval setup value. In the one-shot timer operating mode, TM0 and MD0 are used as timer registers to count down the setup value. TMC1 bits 0 and 1 are fixed to 0. Timer 1 performs interval timer operation only.

Timer 0 can operate as a 16-bit interval timer (consisting of TM0 and MD0) by setting the TMC0 register. Timer 1 can operate as a 16-bit interval timer (consisting of TM1 and MD1) by setting the TMC1 register.

Timer 0 can provide a square wave to the output pin TOUT. Since the TOUT pin is also used for P15, the pin must be placed in the control mode by setting port 1 mode control, register bit 5 (PMC15) to 1.

ALV : Active level specification bit for TOUT pin output

When the ENTO bit is cleared, the active level of TOUT pin output becomes low when the ALV bit is cleared; high when the bit is set to 0.

ENTO : Specification bit of square wave output operation to the TOUT pin

When the ENTO bit is cleared, the active level of the TOUT pin is defined according to the ALV bit. When the ENTO bit is set to 1, the TOUT pin level is inverted when the timer unit interrupt request flag (TMF0) is set to 1.

Other bits of the TMC0 and TMC1 registers are explained for each operating mode.

(1) Interval timer mode (MOD0 = 0, MOD1 = 0): Timers 0 and 1

TCLKn : TMn register count clock specification bit
Table 9-1 lists the reference values when system clock frequency (f_{CLK}) 5 MHz is used.

TSn : Timer n operation control bit
When the TSn bit is set to 1, the MDn register value is set in the TMn register, and the TMn register starts counting down. When the TSn bit is cleared, the TMn register holds the TMn and MDn register contents and stops counting down.

If an underflow occurs or if the TSn bit is again set to 1, during countdown again the MDn register value is set in the TMn register, and again the counting down operation is started.

Table 9-1 Timer Register n (TMn) Count Time in Interval Timer Mode (n = 0 or 1)

f_{CLK}=5 MHz ($=\frac{1}{2}f_x$: f_x=10 MHz)

TCLKn	Count clock	Resolution	Full count
0	$f_{CLK}/6$	1.2 us	78.6 ms
1	$f_{CLK}/128$	25.6 us	1.7 s

(2) One-shot timer mode (MOD0 = 1, MOD1 = 0): Timer 0 only

TCLK0 : TM0 register count clock specification bit

Table 9-2 lists the reference values when system clock frequency (f_{CLK}) 5 MHz is used.

TM0 : TM0 register operation control bit

When the TS0 bit is set to 1, countdown starts at the value currently held in the TM0 register. When an underflow occurs, the TS0 bit is cleared and the count is stopped. When the TS0 bit is cleared, the TM0 register value is held and counting is stopped.

MCLK0 : MD0 register count clock specification bit

Table 9-2 lists the reference values when 5 MHz is used as the system clock frequency (f_{CLK}). If the interval timer mode is specified, the MCLK0 bit does not affect the count.

MS0 : MD0 register count operation control bit

When the MS0 bit is set to 1, the countdown starts at the value currently held in the MD0 register. When an underflow occurs, the MS0 bit is cleared, and the count is stopped. When the MS0 bit is cleared, the MD0 register value is held, and counting is stopped.

The MS0 bit does not affect the count during interval timer operation.

Table 9-2 Timer Register 0 (TM0), Modulo Timer Register 0 (MD0) Count Timer in the One-Shot Timer Mode

f_{CLK}=5 MHz ($=\frac{1}{2}f_x$: f_x=10 MHz)

TCLK0 /MCLK0	Count clock	Resolution	Full count
0	$f_{CLK}/12$	2.4 us	157.3 ms
1	$f_{CLK}/128$	25.6 us	1.7 s

Caution: The TM0 register count clock varies according to which mode (interval timer or one-shot timer) is specified.

7	6	5	4	3	2	1	0
TSO	TCLKO	MSO	MCLKO	ENTO	ALV	MOD1	MOD0

Operating mode specification

MOD 1*	MOD 0	Specification
0	0	Interval timer mode
0	1	One-shot timer mode
0	x	Undefined

* Be sure to set MOD1 to 0.

TOUT pin effective level specification

0	Active level of TOUT pin is low.
1	Active level of TOUT pin is high.

TOUT pin operation specification

0	TOUT pin level is fixed to ALV bit specification.
1	TOUT pin level is inverted when TMF0 is set.

MD0 register clock source specification (in one-shot timer mode)

0	$f_{CLK}/12$ is specified for MD0 register count clock
1	$f_{CLK}/128$ is specified for MD0 register count clock

MD0 register operation control (in one-shot timer mode)

0	Count stop (MD0 is held)
1	MD0 countdown start

TM0 register clock source specification

In interval timer mode	0	$f_{CLK}/6$ is specified for TM0 count clock.
	1	$f_{CLK}/128$ is specified for TM0 count clock.
In one-shot timer mode	0	$f_{CLK}/12$ is specified for TM0 count clock.
	1	$f_{CLK}/128$ is specified for TM0 count clock.

TM0 register operation control

In interval timer mode	0	Count stop (TM0, MD0 holding)
	1	The MD0 contents are loaded into TM0, and TM0 countdown is started.
In one-shot timer mode	0	Count stop (TM0 holding)
	1	TM0 countdown start

Fig. 9-3 Timer Control Register 0 (TMC0) Format

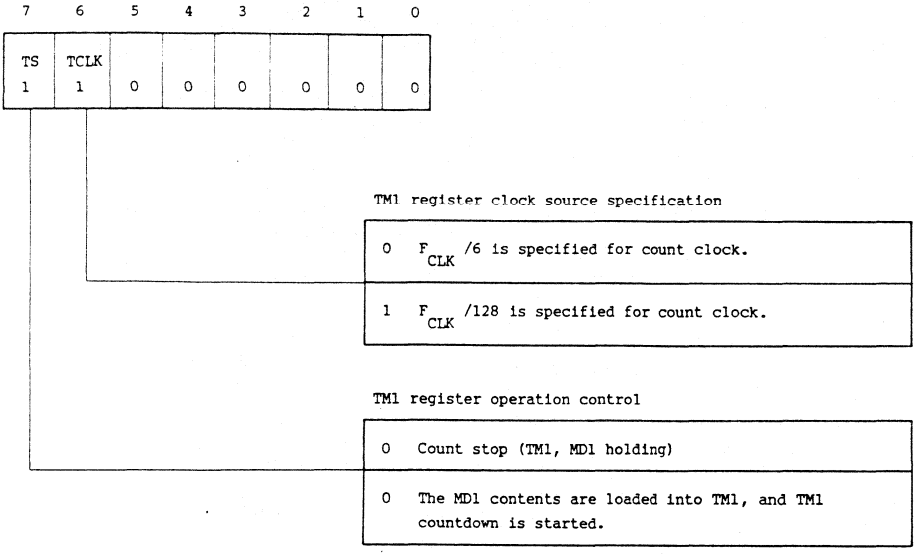
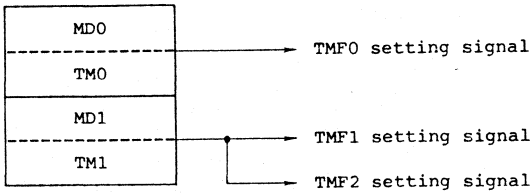
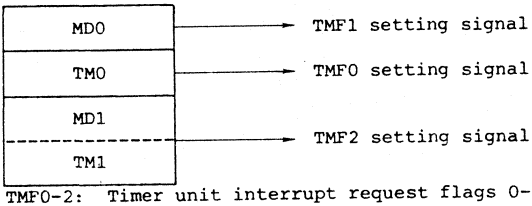


Fig. 9-4 Timer Control Register 1 (TMC1) Format

(a) If interval timer mode is specified for TM0, MD0



(b) If one-shot timer mode is specified for TM0, MD0



TMF0-2: Timer unit interrupt request flags 0-2

Fig. 9-5 Timer Unit Interrupt Requests

9.3.1 Timer unit interrupt request control registers (TMIC0, TMIC1, and TMIC2)

The 8-bit TMICn register (n=0-2) register controls each of the three types of interrupt requests generated in the timer unit. The three interrupt requests form one group; timer unit interrupt request priority can be specified in a program. In the group, the priorities are fixed in hardware as follows:

TMF0 > TMF1 > TMF2

	7	6	5	4	3	2	1	0
TMIC0	TMF0	TMMK0	MS/$\overline{INT}$	ENCS	0	PR2	PR1	PR0
TMIC1	TMF1	TMMK1	MS/$\overline{INT}$	ENCS	0	1	1	1
TMIC2	TMF2	TMMK2	MS/$\overline{INT}$	ENCS	0	1	1	1

Caution: TMIC1 and TMIC2 bits 2-0 are fixed to 1 in hardware. The bit 2-0 field (PR2-PR0) is used to specify the interrupt request priority for each group and forms a group with TMIC0. The TMIC1 and TMIC2 interrupt request priorities conform to the setting of TMIC0 PR2-PR0.

Fig. 9-6 Timer Unit Interrupt Request Control

For the TMICn register bits, see 3.7.

The registers can be written/read by 8-bit or 1-bit memory access operation.

When $\overline{RESET}$ is input, the TMICn register is initialized to 47H.

9.3.2 Timer unit macro service control registers (TMMS0, TMMS1, and TMMS2)

The 8-bit TMMSn register (n=0-2) controls the macro service routine started when the corresponding interrupt request is generated from the timer unit.

The TMMS0 register controls the macro service routine started when the TMF0 flag is set. The TMMS1 register controls the macro service routine started when the TMF1 flag is set; and the TMMS2 register controls the macro service routine started when the TMF2 flag is set.

The registers can be written/read by 8-bit or 1-bit memory access operation.

7	6	5	4	3	2	1	0
MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0

Fig. 9-7 Timer Unit Macro Service Control Register (TMMS0, TMMS1, TMMS2) Format

For the TMMSn register bits, see 3.4.3.

10. PORT FUNCTION

10.1 Ports 0-2

10.1.1 Hardware configuration

Basically, uPD70332 or uPD70330 ports 0-2 comprise 3-state bidirectional ports shown in Fig. 10-1.

When $\overline{\text{RESET}}$ is input, the port mode register bits are set to 1 and input port mode is specified. All port pins are placed in high impedance. The output latch contents are not affected by $\overline{\text{RESET}}$ input.

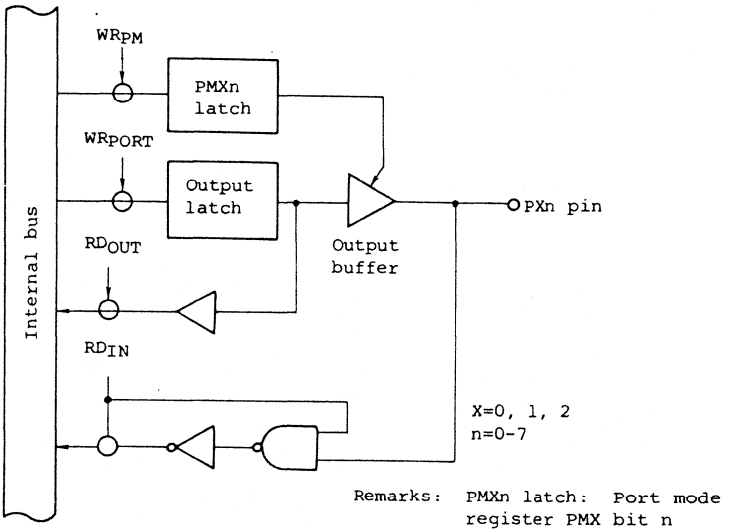


Fig. 10-1 Port 0-2 Configuration

(1) When output port mode is specified (PMXn = 0)

The output latch is made effective, and data can be transferred between the output latch and an accumulator by using a transfer instruction. The output latch contents can be set as desired by using a logical operation instruction. The data written into the output latch is held until another port operation instruction is executed.

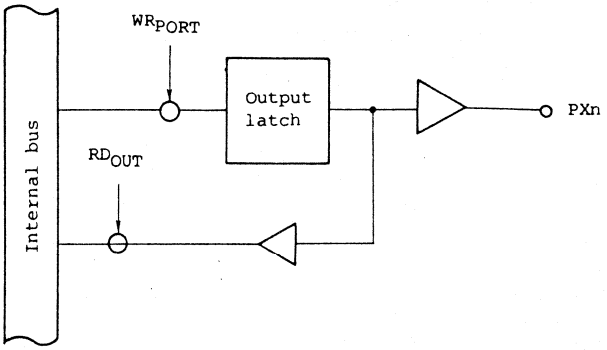


Fig. 10-2 Port Specified as Output Port

(2) When input port mode is specified (PMXn = 1)

The port pin level can be loaded into an accumulator by using a transfer instruction. In this case, data can also be written into output latch. The data transferred from the accumulator by using a transfer instruction is stored in all output latches regardless of input or output mode specification for the port. However, since the output buffer for bits specified as input ports is placed in high impedance, no data is output to the port pins (when the bits specified as input ports are changed to output mode, the output latch contents are output through the port pins).

The output latch contents for bits specified as input ports cannot be loaded into an accumulator.

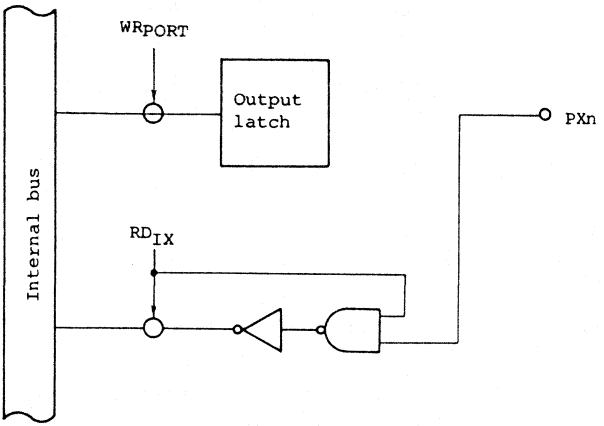


Fig. 10-3 Port Specified as Input Port

(3) When control specification is made (PMCXn = 1)

Ports 0-2 can be used bitwise for control signal input or output by setting port mode control register (PMCX) bit to 1 regardless of how the port mode register (PMX) is set. When the pins are used as control signals, the control signal state can be read by executing a port access instruction.

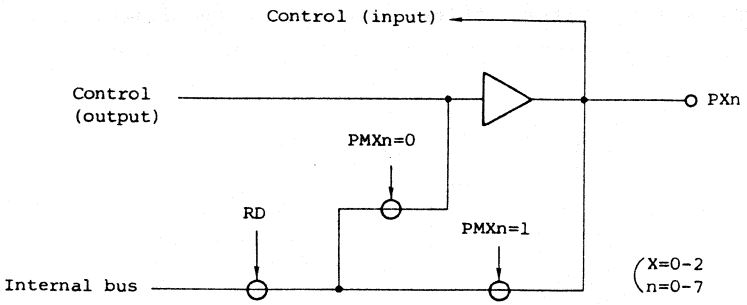


Fig. 10-4 When Control Specification is Made

(i) When port is used for control signal output

If a port read instruction is executed when the port mode register (PMXn) is set to 1, the state of the control signal pin can be read.

If a port read instruction is executed when the port mode register is cleared, the state of the internal control signal can be read.

(ii) When port is used for control, signal input

If a port read instruction is executed only when the port mode register is set to 1, the state of the control signal pin can be read.

10.1.2 Port functions

(1) P00-P07 (port 0) -- 3-state input/output

Port 0 is an 8-bit special input/output port. It serves as not only a general purpose input/output port for which input or output mode can be

specified bitwise, but also system clock output pin CLKOUT (also used for P07). The function can be selected bitwise by using the port 0 mode register (PM0) and port 0 mode control register (PMC0).

Table 10-1 Port 0 Operation (n=0-7)

	PMC0n = 1	PMC0n = 0	
		PM0n=1	PM0n=0
P00		Input port	Output port
P01		Input port	Output port
P02		Input port	Output port
P03		Input port	Output port
P04		Input port	Output port
P05		Input port	Output port
P06		Input port	Output port
P07	CLKOUT output	Input port	Output port

(i) Port 0 mode control register (PMC0)

The 8-bit PMC0 register specifies port or system clock output mode bitwise for port 0.

The register can only be written by 8-bit memory access operation. If the corresponding bit of the PMC0 register is set to 1, the system clock output mode (P07) is specified; if it is cleared, the port mode is specified. When $\overline{\text{RESET}}$ is input, all the PMC0 register bits are cleared and the port mode is set.

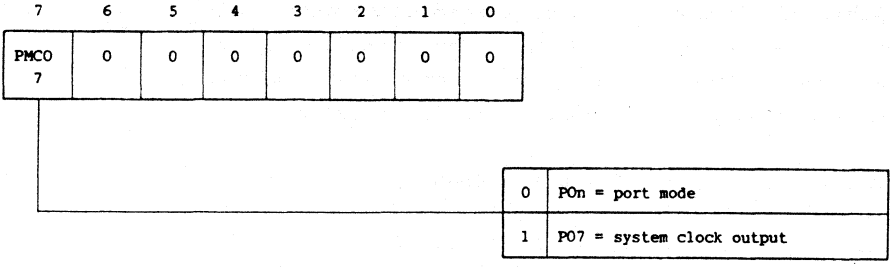


Fig. 10-5 Port 0 Mode Control Register (PMC0) Format

(ii) Port 0 mode register (PM0)

The PM0 register in an 8-bit register to specify the input or output mode bitwise for port 0. The register can only be written by 8-bit memory access operation.

When the corresponding bit of the PMC0 register is set to 0, PM0 specification is made effective. When $\overline{\text{RESET}}$ is input, all the PM0 register bits are set to 1.

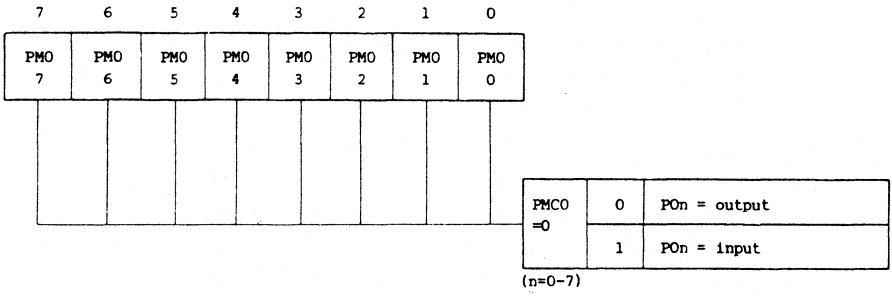


Fig. 10-6 Port 0 Mode Register (PM0) Format

(2) P10-P17 (port 1) - 3-state input/output

Port 1 is an 8-bit special input/output port. Like port 0, port 1 serves as not only a general purpose port for which input or output mode can be specified bitwise, but also as control pins. A function can be selected bitwise by using port 1 mode register (PM1) and port 1 mode control register (PMC1). The state of the P10-P13 pins can be determined by directly reading (accessing) port 1 (P1).

Table 10-2 Port 1 Operation (n = 0-7)

	PMC1n = 1	PMC1n = 0	
		PM1n=1	PM1n=0
P10		NM1 input	
P11		$\overline{\text{INTP0}}$ input	
P12		$\overline{\text{INTP1}}$ input	
P13	$\overline{\text{INTAK}}$ output	$\overline{\text{INTP2}}$ input	
P14	INT input	Input port ($\overline{\text{POLL}}$ input)	Output port
P15	TOUT output	Input port	Output port
P16	$\overline{\text{SCK0}}$ output	Input port	Output port
P17	READY input	Input port	Output port

(i) Port 1 mode control register (PMC1)

The 8-bit PMC1 register specifies the port or control signal input/output mode bitwise for port 1.

The register can only be written by 8-bit memory access operation. If the

corresponding bit of the PMC1 register is set to 1, the control signal input/output mode is specified; if it is cleared, the port mode is specified. When $\overline{\text{RESET}}$ is input, all the PMC1 register bits are cleared and the port mode is set; however, P10-P12 are fixed to the port mode.

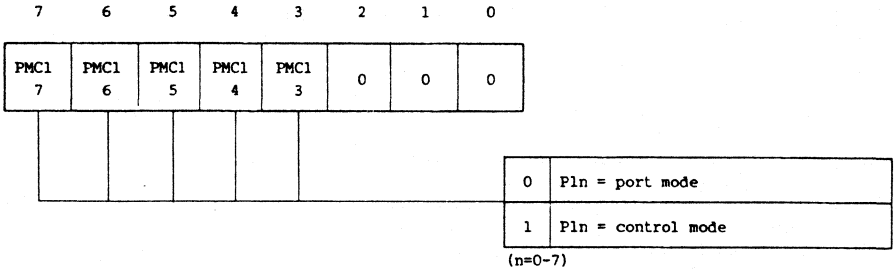


Fig. 10-7 Port 1 Mode Control Register (PMC1) Format

(ii) Port 1 mode register (PM1)

The 8-bit PM1 register specifies input or output mode bitwise for port 1. The register can only be written by 8-bit memory access operation.

When the corresponding bit of the PCM1 register is set to 0, PM1 specification is made effective. When $\overline{\text{RESET}}$ is input, all the PM1 register bits are set to 1.

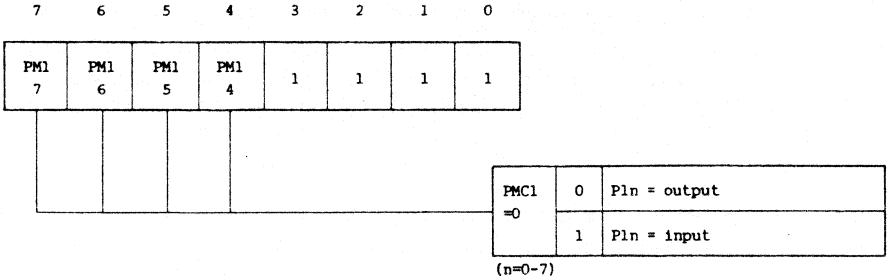


Fig. 10-8 Port 1 Mode Register (PM1) Format

(3) P20-P27 (port 2) - 3-state input/output

Port 2 is an 8-bit special input/output port. Like port 0, port 2 serves as not only as a general purpose port for which input or output mode can be specified bitwise, but also as control pins. The function can be selected bitwise by using port 2 mode register (PM2) and port 2 mode control register (PMC2).

Table 10-3 Port 2 Operation

	PMC2 = 1	PMC2 = 0	
		PM2n=1	PM2n=0
P20	DMARQ0 input	Input port	Output port
P21	$\overline{\text{DMAAK0}}$ output	Input port	Output port
P22	$\overline{\text{TC0}}$ output	Input port	Output port
P23	DMARQ1 input	Input port	Output port
P24	$\overline{\text{DMAAK1}}$ input	Input port	Output port
P25	$\overline{\text{TC1}}$ output	Input port	Output port
P26	$\overline{\text{HLDAK}}$ output	Input port	Output port
P27	HLDRQ input	Input port	Output port

(i) Port 2 mode control register (PMC2)

The 8-bit PMC2 register specifies the port or control signal input/output mode bitwise for port 2.

The register can only be written by 8-bit memory access operation.

If the corresponding bit of the PMC2 register is set to 1, the control signal input/output mode is specified; if it is cleared, the port mode is specified. When $\overline{\text{RESET}}$ is input, all the PMC2 register bits are cleared and the port mode is set.

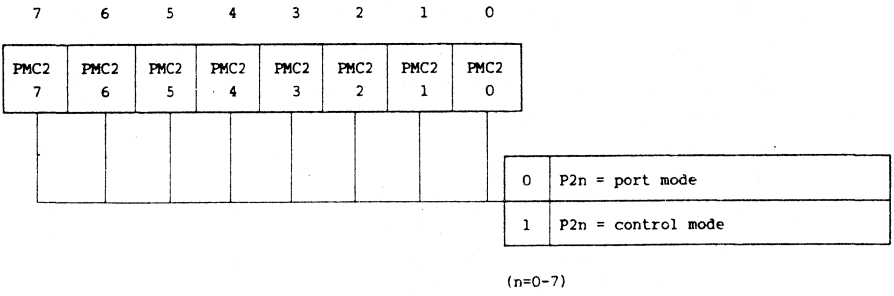


Fig. 10-9 Port 2 Mode Control Register (PMC2) Format

(ii) Port 2 mode register (PM2)

The PM2 register is an 8-bit register to specify the input or output mode for port 2 bitwise. The register can only be written by 8-bit memory access operation.

When the corresponding bit of the PMC2 register is set to 0, PM2 specification is made effective.

When RESET is input, all the PM2 register bits are set to 1.

7	6	5	4	3	2	1	0
PM2 7	PM2 6	PM2 5	PM2 4	PM2 3	PM2 2	PM2 1	PM2 0

PMC2 =0	0	P2n = output
	1	P2n = input

Fig. 10-10 Port 2 Mode Register (PM2) Format

10.2 Port T (PT0-PT7)

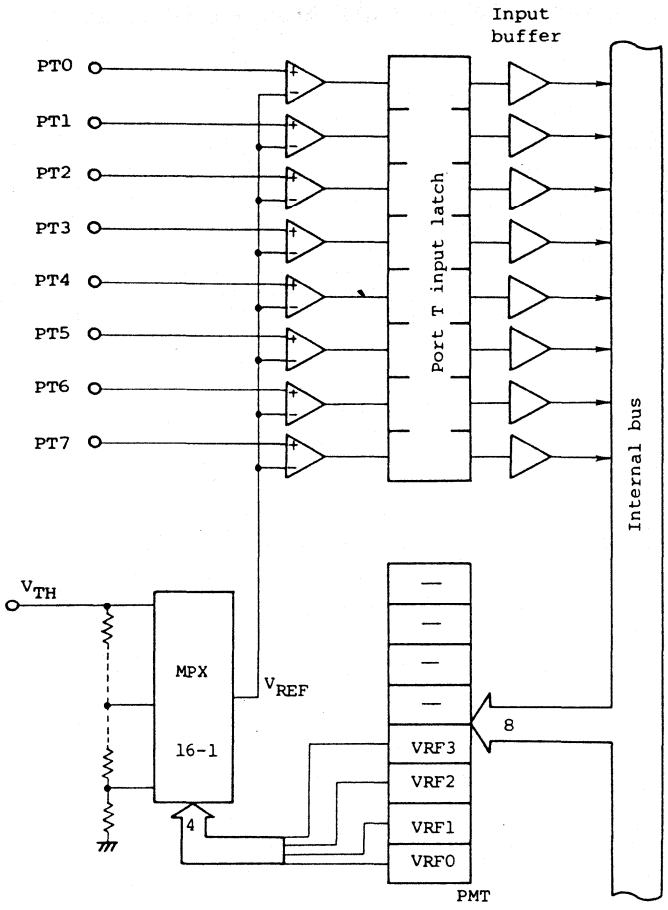
Port T is an 8-bit input port which enables the threshold voltage (reference voltage) to be changed in 16 stages. Comparator operation is performed by analog input to the port.

10.2.1 Hardware configuration

Port T consists of PT0-PT7 comparator input, a reference power supply input pin V_{TH}, a multiplex circuit (MPX) to select one of sixteen comparison voltages (V_{REF}) from 1/16 x V_{TH} to 16/16 x V_{TH}, the port mode T register (PMT) to control MPX, and eight latches.

V_{REF} selected by the PMT register setting is compared with PT0-PT7 input by the comparator. The result is latched in the port T input latch.

$$\begin{cases} V_{REF} > PTn \rightarrow 0 \\ V_{REF} < PTn \rightarrow 1 \end{cases}$$



Caution: The V_{TH} pin is connected to the GND pin through a high resistance. Thus, if voltage is applied to the V_{TH} pin in the standby mode, current consumption increase.

Fig. 10-11 Port T Block Diagram

10.2.2 Port T mode register (PMT)

The port T mode register (PMT) is used to select the comparison voltage (V_{REF}) from 16 types as shown in Fig. 10-12.

The register can be written/read by 8-bit or 1-bit memory access operation.

When $\overline{RESET}$ is input, all the PMT register bits are cleared.

7	6	5	4	3	2	1	0
-	-	-	-	VREF3	VREF2	VREF1	VREF0

VREF3	VREF2	VREF1	VREF0	V_{REF}
0	0	0	0	V_{TH} x 16/16
0	0	0	1	V_{TH} x 1/16
0	0	1	0	V_{TH} x 2/16
0	0	1	1	V_{TH} x 3/16
0	1	0	0	V_{TH} x 4/16
0	1	0	1	V_{TH} x 5/16
0	1	1	0	V_{TH} x 6/16
0	1	1	1	V_{TH} x 7/16
1	0	0	0	V_{TH} x 8/16
1	0	0	1	V_{TH} x 9/16
1	0	1	0	V_{TH} x 10/16
1	0	1	1	V_{TH} x 11/16
1	1	0	0	V_{TH} x 12/16
1	1	0	1	V_{TH} x 13/16
1	1	1	0	V_{TH} x 14/16
1	1	1	1	V_{TH} x 15/16

Fig. 10-12 Port T Mode Register (PMT) Format

11 STANDBY FUNCTION

The uPD70332 and uPD70330 contain two operating clock the standby function control modes for low power consumption:

o HALT mode: Only the CPU operation clock is stopped. All the CPU status, data, and internal RAM contents are held. Peripheral hardware continues operation. The total power consumption in the system can be decreased by intermittent operations which use HALT mode and normal operating mode in combination.

o STOP mode: The oscillator is stopped and the entire internal circuit is disabled. Power consumption is very low. The internal RAM contents and output data on ports are held.

The mode is set by using the HALT or STOP instruction.

11.1 Standby Control Register (STBC)

The 8-bit STBC register contains a standby flag (SBF). The high-order seven bits are fixed to 0.

SBF is used for, returning from the STOP state.

SBF is set to 1 only by using an instruction. It is cleared only when the supply voltage (V_{DD}) rises from 0 V; it is not cleared by an instruction.

Thus, reset start with power on (SBF=0) or return from the STOP mode (SBF=1 (Note)) can be decided by testing SBF.

The STBC register is not initialized when the RESET signal is input.

Note: Preset SBF to 1 before entering the SOTP mode.

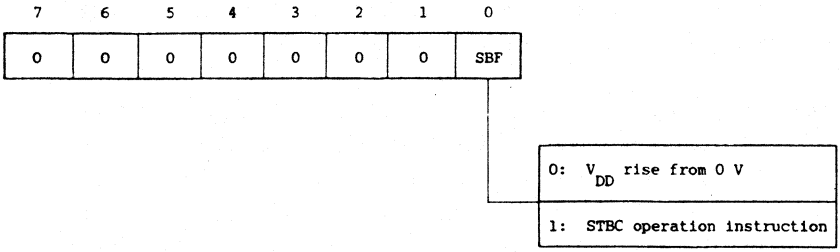


Fig. 11-1 Standby Control Register (STBC) Format

11.2 HALT Mode

The HALT mode is used to stop the CPU clock.

The total power consumption for the system can be decreased by setting the HALT mode during CPU idle time. The HALT state is set by executing the HALT instruction.

In the HALT mode, the CPU clock stops and program execution stops, but all register and the contents of internal RAM immediately before execution stops are held. Table 11-2 the lists hardware state.

11.2.1 HALT mode release

The HALT mode is released when a nonmaskable interrupt (NMI) or unmasked maskable interrupt request is made, or when $\overline{\text{RESET}}$ is input. (See Fig. 11-2.)

When a macro service or DMA processing request is made, macro service or DMA processing is started from the HALT mode. (See Fig. 11-3.) When the macro service routine or DMA processing terminates, a return is made to the HALT mode. However, if the condition listed in

Table 11-1 is set while executing the macro service routine or DMA processing, the HALT mode is released.

(1) Release when an interrupt request is made

(i) When the HALT mode is set in interrupt service routine

The HALT mode is released when a nonmaskable interrupt request or unmasked maskable interrupt request assigned a higher priority than the interrupt being processed occurs.

(ii) Other than (i)

The HALT mode is released when a nonmaskable interrupt request or unmasked maskable interrupt request occurs regardless of the priority.

(2) Release when $\overline{\text{RESET}}$ is input

The HALT mode is released in the same manner as the normal reset operation.

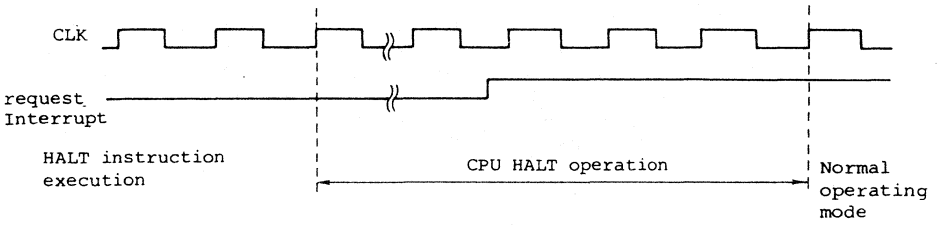


Fig. 11-2 HALT Mode Release when Interrupt Request is Made

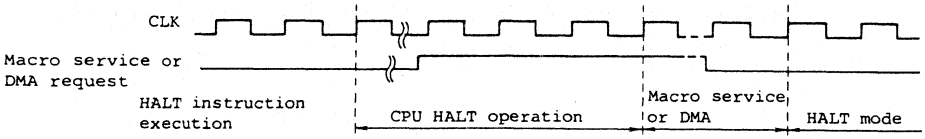


Fig. 11-3 Macro Service or DMA Start during the HALT Mode

Table 11-1 Operation after the HALT Mode is Released

Release source	EI state	DI state
Nonmaskable interrupt request	After the HALT mode is released, a branch is made to the vector address.	After the HALT mode is released, a branch is made to the vector address.
Maskable interrupt request	After the HALT mode is released, a branch is made to the vector address.	After the HALT mode is released, the next instruction is executed.
Macro service request	When macro service is started and the macro service counter reaches OH, a branch is made to the vector address. If a transfer data match is found in the character search mode, a branch is also made to the vector address. If the macro service counter does not reach OH, again the HALT mode is entered.	When the macro service routine is started and the macro service counter reaches OH, the HALT mode is released and the next instruction is executed. If a transfer data match is found in the character search mode, the HALT mode is also released.
DMA request	When DMA is started and the terminal counter reaches OH, a branch is made to the vector address. If the terminal counter does not reach OH, again the HALT mode is entered.	When DMA is started and the terminal counter reaches OH, the HALT mode is released and the next instruction is executed.

11.3 STOP Mode

The STOP mode is used to stop the oscillator.

When the entire application system stops, power consumption is very low. The STOP state is set by executing the STOP instruction. In the STOP mode, all clocks stop. Program execution is stopped, but all the contents of the registers and internal RAM immediately before stop are held. Table 11-2 lists the hardware state.

11.3.1 STOP mode release

The STOP mode is released when an NMI request is made or $\overline{\text{RESET}}$ is input.

(1) Release when NMI request is made (Fig. 11-4)

When an effective edge is input to the NMI pin, the oscillator restarts. The time base counter (TBC) also starts operation. After the STOP mode is released, clock supply is not made immediately but is started after a count for stabilization of TBC oscillation. The count for stabilization of TBC oscillation is half of the value specified in processor control register (PRC) bits 2 and 3 (TB0 and TB1). (However, an interrupt request generated from the TBC is disabled at this time.)

(2) Release when $\overline{\text{RESET}}$ is input

The STOP mode is released in the same manner as the normal reset operation.

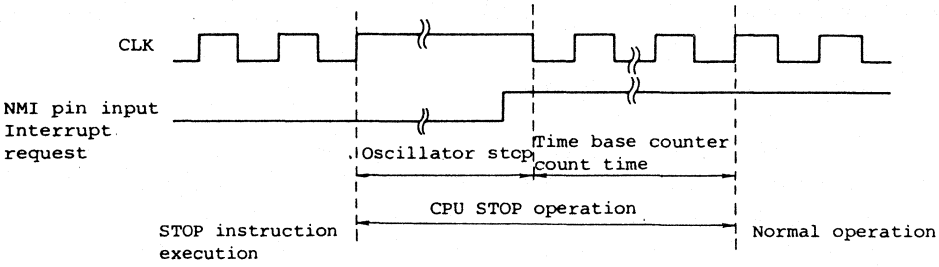


Fig. 11-4 STOP Mode Release when NMI Pin Input is Made

Table 11-2 Differences between HALT and STOP Mode

Item		HALT mode	STOP mode
Oscillator		Operating	Stop
Internal system clock		Stop	
16-bit timer		Operating	
Time base counter			
HOLD circuit			
Serial interface			
Interrupt request controller			
DMA controller			
I/O lines		Held	Held
Bus lines	A0-A19	Held	Held
	D0-D7	High impedance	High impedance
R/$\overline{W}$ output		High level	High level
Refresh operation		Operating or stop	Stop
Data hold		All internal data such as the CPU status and RAM contents are held.	All internal data such as the CPU status and RAM contents are held.
Release method		o Nonmaskable interrupt request o Maskable interrupt request o $\overline{RESET}$ input o Macro service request (Note) o DMA (Note)	o Nonmaskable interrupt request o $\overline{RESET}$ input

Note: After the macro service routine or DMA processing terminates, a return is made to the HALT mode.

12. RESET FUNCTION

When a low level is input to the $\overline{RESET}$ input pin, the system is reset and the hardware is placed in the state listed in Table 12-1. when the $\overline{RESET}$ input goes high, the reset state is released and program execution is started. Initialize the register contents in a program as required.

Table 12-1 Hardware State after Reset

Hardware (abbreviation)			Address (Note 1) (Low-order 12 bits: xx H)	State after reset
Program counter		PC		0000H
Program status word		PSW		F002H
Internal RAM	Data memory			Undefined
	General purpose register	AW, CW, DW, BW, SP, BP, IX, IY	EFEH-EF0H	Undefined
	Segment register	DS1, SS, DS0	EEEH, EEAH, EE8H	0000H
		PS	EECH	FFFFH
Port	Port register	P0, P1, P2	F00H, F08H, F10H	Undefined
		PT	F38H	
	Port mode register	PM0, PM1, PM2	F01H, F09H, F11H	FFFFH
		PMT	F3BH	00H
	Port mode control register	PMC0, PMC1, PMC2	F02H, F0AH, F12H	00H
Timer unit	Timer register	TM0, TM1	F80H, F88H	Undefined
	Modulo/timer register	MD0, MD1	F82H, F8AH	Undefined
	Timer control register	TMC0, TMC1	F90H, F91H	00H

(Cont'd)

Hardware (abbreviation)			Address (Note 1) (Low-order 12 bits: xx □□□ H)	State after reset
Timer unit	Interrupt request control register	TMIC0-TMIC2	F9CH-F9EH	47H
	Macro service control register	TMMS0-TMMS2	F94H-F96H	Undefined
DMA controller	DMA mode register	DMAM0, DMAM1	FA1H, FA3H	00H
	DMA control register	DMAC0, DMAC1	FA0H, FA2H	Undefined
	Interrupt request control register	DIC0, DIC1	FACH, FADH	47H
Serial interface	Serial mode register	SCM0, SCM1	F68H, F78H	00H
	Serial control register	SCC0, SCC1	F69H, F79H	00H
	Baud rate generator setup value	BRG0, BRG1	F6AH, F7AH	00H
	Receive buffer register	RxB0, RxB1	F60H, F70H	Undefined
	Transmit buffer register	TxB0, TxB1	F62H, F72H	Undefined
	Serial error register	SCE0, SCE1	F6BH, F7BH	00H
	Interrupt request control register	(Error) SEIC0, SEIC1	F6CH, F7CH	47H
		(Receive) SRIC0, SRIC1	F6DH, F7DH	
		(Send) STIC0, STIC1	F6EH, F7EH	
	Macro service control register	(Receive) SRMS0, SRMS1	F65H, F75H	Undefined
		(Send) STMS0, STMS1	F66H, F76H	
Timer base insterrupt request control register		TBIC	FECH	47H
User flag register		FLAG	FEAH	00H

(Cont'd)

Hardware (abbreviation)			Address (Note 1) (Low-order 12 bits: xx□□□ H)	State after reset
Internal data area base register		IDB	FFFH	FFH
Processor control register		PRC	FEBH	4EH
Wait control register		WTC	FE8H	FFFFH
Refresh mode register		RFM	FE1H	FCH
Standby control register		STBC	FE0H	Undefined (Note 2)
External interrupt	External interrupt mode register	INTM	F40H	00H
	Interrupt request control register	EXIC0-EXIC2	F4CH-F4EH	47H
	Macro service control register	EMS0-EMS2	F44H-F46H	Undefined

Note 1: xx in the high-order eight bits of the address is a value specified in the IDB register.

2: At power on reset: 00H
Other than the above: No change

Pay attention for!!!!

During active RESET (RESET pin is low level) all pins are in high impedance state except the following pins:

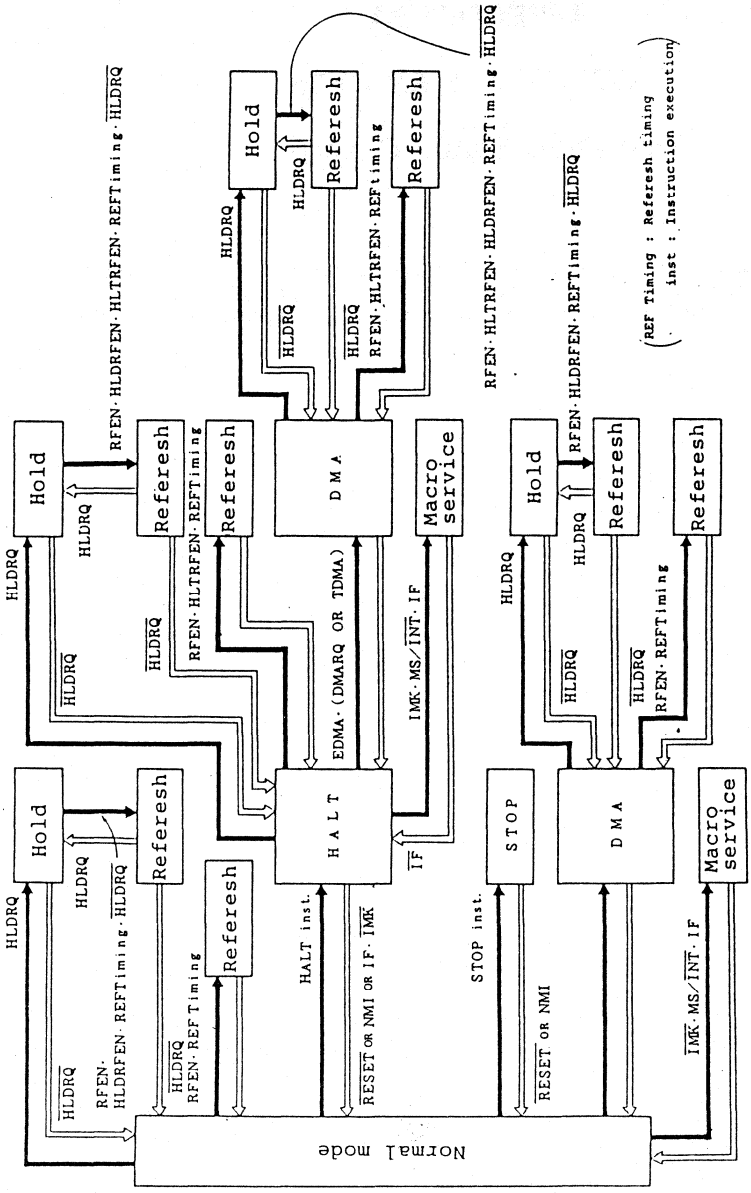
X1
X2
GND
VDD
RESET
I.C.

APPENDIX 2A

FLOW OF V25 AND V35

Operation Status Change

The operating mode (macro service, DMA, refresh, hold, HALT, STOP) of the uPD70320/322 and uPD70330/332 changes as shown in the following figure according to the conditions indicated in the figure.



Flow of Operating Status Change

APPENDIX B
μPD 70330 / 70332
Input / Output Circuits

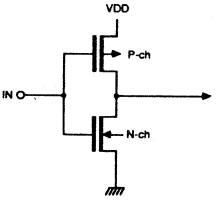
PIN	TYPE
P00–P06	5
P07/CLKOUT	5
NMI	2
P11/INTP0	1
P12/INTP1	1
P13/INTP2/INTAK	5
P14/INT/POLL	5
P15/TOUT	5
P16/SCK0	5
P17/READY	5

PIN	TYPE
P20/DMARQ0	5
P21/$\overline{\text{DMAAK0}}$	5
P22/$\overline{\text{TC0}}$	5
P23/DMARQ1	5
P24/$\overline{\text{DMAAK1}}$	5
P25/$\overline{\text{TC1}}$	5
P26/$\overline{\text{HLDAK}}$	5
P27/HLDRQ	5
PT0–PT7	7

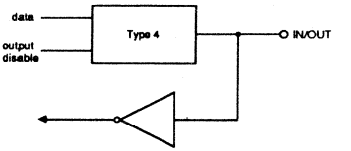
PIN	TYPE
TxD0	4
TxD1	4
RxD0	1
RxD1	1
$\overline{\text{CTS0}}$	5
$\overline{\text{CTS1}}$	1
$\overline{\text{REFRQ}}$	4
$\overline{\text{RESET}}$	2
$\overline{\text{EA}}$	1
D0–D15	5

PIN	TYPE
A0	4
A9/A1–A16/A8	4
A17/A18	4
A19	4
A18/$\overline{\text{UBE}}$	4
$\overline{\text{MREQ}}$	4
$\overline{\text{MSTB}}$	4
R/$\overline{\text{W}}$	4
$\overline{\text{IOSTB}}$	4

Type 1

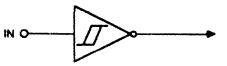


Type 5



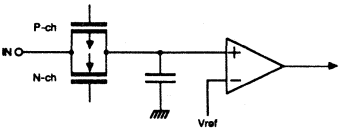
(I/O power circuit consit of Push-Pull output (Type 4) and Input Buffer (Type 1)

Type 2



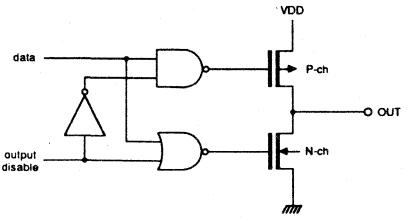
(Schmitt trigger Input which has hysterisys charcateristic)

Type 7



(Threshhold voltage)

Type 4



(Push-Pull output which can make output high impedance both P-ch ,N-ch off)

APPENDIX C

μPD 70330/70332

Unosed pin connections (1)

pin name	recommendation
p00 - p06 p07 /clkout	input mode : pull up to vdd level through resistor output mode: open
p10 /nmi	pull down to gnd level through resistor
p11 /- intp0 p12 /- intp1	pull up to vdd level through resistor or pull down to gnd level through resistor
p13 /- intp2/- intak p14 / int/- poll p15 /tout p16 /- sck0 p17 /ready p20 /dmarq0 p21 /- dmaak0 p22 /- tc0 p23 /dmarq1 p24 /- dmaak1 p25 /- tc1 p26 /- hldak p27 /hldrq	input mode : pull up to vdd level through resistor output mode: open
pt0 - pt7	pull down to gnd level through resistor
txd0 txd1	open
rxd0 rxd1	pull up to vdd level through resistor or pull down to gnd level through resistor
-cts0	input mode : pull up to vdd level through resistor output mode: open

Unused pin connections (2)

pin name	recommendation
-cts1	pull up to vdd level through resistor or pull down to gnd level through resistor
-refrq	open
vth	pull down to gnd level through resistor
d0 - d15	input mode : pull up to vdd level through resistor output mode: open
a0/ a19 -mreq -mstb r/ -w -iostb a9/a1 - a16/a8 a17 - a18 a18/-ube	open

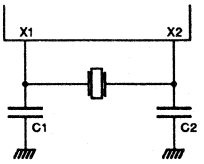
Use for pull - up and pull - down: 5 to 100 kohm

APPENDIX D

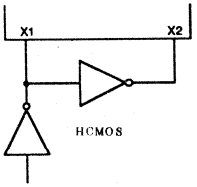
μPD 70330/70332

Clock Generation Circuits

a. Cristal / Ceramic OSC



b. External Clock input



Note:

1. Set OSC circuit to the nearest position to X1 and X2
2. Do not attach any other connections to X1 and X2
3. When using Cristal OSC, set C1 = C2 = 15 pF
4. Following chart shows recommended OSC and values of C1 and C2 when using Ceramic OSC:

Device	Supplier	Type	C1(pF)	C2(pF)
μPD70322-8,70320-8	Murata	CSA16.00MX040	30	30
μPD70322,70320 μPD70322-8,70320-8 μPD70332,70330 μPD70P322		CSA10.0MT	47	47
μPD70322,70320 μPD70322-8,70320-8 μPD70332,70330 μPD70P322	Kyocera	KBR-10.0M	33	33
μPD70322-8,70320-8	T D K	FCR16.0M2S	15	6

APPENDIX E

μPD70330/70332

Wait Control with READY

Choosing the wait control from READY pin with WTC, the CPU inserts 2 waits between T1 state and T2 state.
CPU starts sampling the READY pin condition at falling edge of CLKOUT in TW (TAW) state.

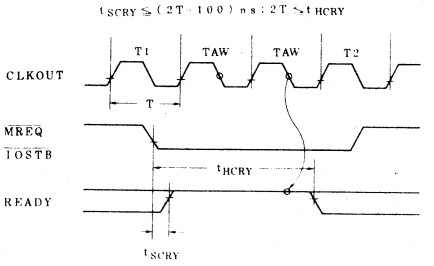
During TW (TAW) sampling, if READY pin is at the second TW (TAW) on 'L' level, CPU inserts wait states with the same number of times as sampling.

During wait states, there is no insertion of refresh cycle. This would mean, that the DRAM controller would not be able to maintain the refresh interval, if the wait states become too long.

The following figures shows the corresponding timings:

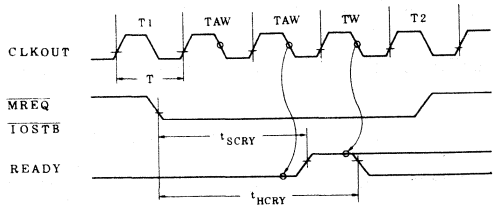
Wait by READY pin

a. without supplement

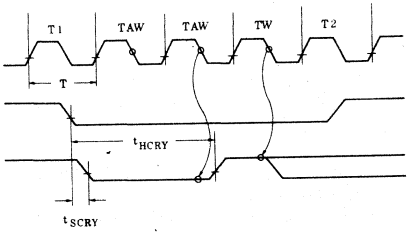


b. with one wait supplement

(I) $t_{SCRY} \geqq (3T - 100)$ ns : $3T \leqq t_{HCRY}$

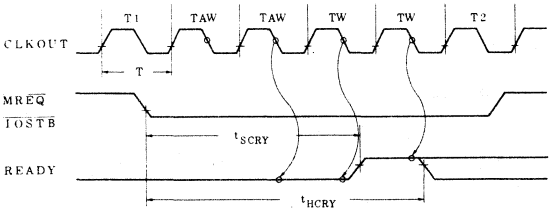


(II) $t_{SCRY} \leqq (2T - 100)$ ns : $2T \leqq t_{HCRY}$

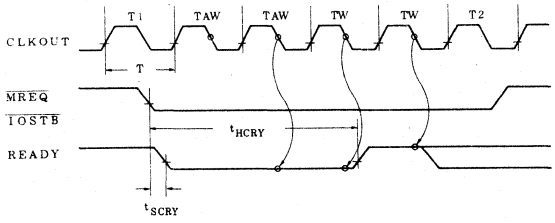


c. with two wait supplement

(I) $t_{SCRY} \geqq (4T - 100)$ ns : $4T \geqq t_{HCRY}$



(II) $t_{SCRY} \leqq (3T - 100)$ ns : $3T \geqq t_{HCRY}$



PRELIMINARY

APPENDIX 2F

μPD70330 (70332)

V35

ELECTRICAL SPECIFICATION

μPD70330/70332
μPD70330-8/70332-8 Electrical Specification

ABSOLUTE MAXIMUM RATINGS (Ta=25℃)

Parameter	Symbol	Test Condition	Ratings	Unit
Power Supply Voltage	VDD		-0.5 to +7.0	V
	VTH		-0.5 to VDD+0.5 ≦+7.0	V
Input Voltage	VI		-0.5 to VDD+0.5 ≦+7.0	V
Output Voltage	VO		-0.5 to VDD+0.5 ≦+7.0	V
Output Current Low	IOL	All Output Pin	4.0	mA
		All Output Pin Total	50	mA
Output Current High	IOH	All Output Pin	-2.0	mA
		All Output Pin Total	-20	mA
Operating Temperature	Topt		-40 to +85	℃
Storage Temperature	Tstg		-85 to +150	℃

CAPACITANCE (Ta=25℃,VDD=0V)

Parameter	Symbol	Test Condition	MIN	TYP	MAX	Unit
Input Capacitance	CI	fc=1MHZ			10	PF
Output Capacitance	CO	Unmeasured Pins			20	PF
I/O Capacitance	CIO	Returned to 0V			20	PF

D. C. CHARACTERISTICS (Ta = −10℃ to +70℃, VDD = +5.0V ± 10%)

Parameter	Symbol	Test Condition	MIN	TYP	MAX	Unit
Input Low Voltage	VIL		0		0.8	V
Input High Voltage	VIH1	All except $\overline{\text{RESET}}$, P10/NMI, X1, X2	2.2		VDD	V
	VIH2	$\overline{\text{RESET}}$, P10/NMI, X1, X2	0.8 VDD		VDD	V
Output Low Voltage	VOL	IOL=1.6mA			0.45	V
Output High Voltage	VOH	IOH=−0.4mA	VDD−1.0			V
Input Current	II	$\overline{\text{EA}}$, P10/NMI; 0≦VI≦VDD			±20	uA
Input Leakage Current	ILI	All except $\overline{\text{EA}}$, P10/NMI; 0≦VI≦VDD			±10	uA
Output Leakage Current	ILO	0≦VO≦VDD			±10	uA
VTH Supply Current	ITH	0≦VTH ≦VDD		0.5	1.0	mA
VDD Supply Current	IDD1	Operation Mode fCLK =5MHZ/8MHZ		50/65	100/120	mA
	IDD2	HALT Mode fCLK =5MHZ/8MHZ		20/25	40/50	mA
	IDD3	STOP Mode		10	30	uA

COMPARATOR CHARACTERISTICS

(Ta = −10℃ to +70℃, VDD = +5.0V ± 10%)

Parameter	Symbol	Test Condition	MIN	TYP	MAX	Unit
Comparison Accuracy	VACOMP				±100	mV
Threshold Voltage	VTH		0		VDD+0.1	V
Comparison Time	tCOMP		64		65	TCYK
PT Input Voltage	VIPT		0		VDD	V

A. C. CHARACTERISTICS (Ta = −10℃ to +70℃, VDD = +5.0V±10%)

PARAMETER	SYMBOL	TEST CONDITION	MIN	MAX	UNIT	
Input Rise Time Input Fall Time	tIR tIF	EXCEPT X1, X2, RESET, NMI		20	ns	
Input Rise Time (SCHMITT) Input Fall Time (SCHMITT)	tIRS tIFS	$\overline{\text{RESET}}$, NMI		30	ns	
Output Rise Time Output Fall Time	tOR tOF	Except CLKOUT		20	ns	
X1 Input Cycle Time	tCYX	fCLK=5MHZ/8MHZ	98/62	250	ns	
X1 Width Low	tWXL		35/20		ns	
X1 Width High	tWXH		35/20		ns	
X1 Rise Time X1 Fall Time	tXR tXF			10	ns	
CLKOUT Cycle Time	tCYK	fCLK=5MHZ/8MHZ fX/2, T=tCYK	200/125	2000	ns	
CLKOUT Width Low	tWKL	fCLK=5MHZ/8MHZ	0.5T-15		ns	
CLKOUT Width High	tWKH		0.5T-15		ns	
CLKOUT Rise Time CLKOUT Fall Time	tKR tKF			15	ns	
Address Delay Time	tDKA			80	ns	
Address to Data Input	tDADR			(n+1.5)T-120	ns	
$\overline{\text{MREQ}}$ to Address Hold Time	tHMRA		0.5T-50		ns	*
$\overline{\text{MSTB}}$ to Data Delay Time	tDMSD			(n+1)T-90	ns	*
$\overline{\text{MREQ}}$ to $\overline{\text{MSTB}}$ Read Delay Time	tDMRMSR		1T-50	1T+50	ns	*
$\overline{\text{MREQ}}$ Width Low	tWMRL		(n+2)T-40		ns	*
$\overline{\text{MREQ}}$, $\overline{\text{MSTB}}$ to Address Hold Time	tHMA		0.5T-50		ns	
Input Data Hold Time	tHMD		0		ns	
Next Control Setup Time	tSCC		T-25		ns	
$\overline{\text{MREQ}}$ to $\overline{\text{TC}}$ Delay Time	tDMRTC			0.5T+50	ns	
$\overline{\text{TC}}$ Width Low	tWTCL		2T-40		ns	

A. C. CHARACTERISTICS (Continued)

PARAMETER	SYMBOL	TEST CONDITION	MIN	MAX	UNIT	
$\overline{MREQ}$ Read Delay Time	tDAMR		0.5T-50		ns	
$\overline{MSTB}$ Read Delay Time	tDAMSR		0.5T-50		ns	*
$\overline{MSTB}$ Read Width Low	tWMSLR		(n+1)T-40		ns	*
Address to Data Output	tDADW			0.5T+50	ns	
Data Output Setup Time	tSDW		(n+2)T-50		ns	*
$\overline{MSTB}$ Write Delay Time	tDAMSW		(n+0.5)T-50		ns	*
$\overline{MREQ}$ to $\overline{MSTB}$ Write Delay Time	tDMRMSW		(n+1)T-50	(n+1)T+50	ns	*
$\overline{MSTB}$ Write Width Low	tWMSLW		1T-40		ns	*
$\overline{IOSTB}$ Delay Time	tDAIS		0.5T-50		ns	
$\overline{IOSTB}$ to Data Input	tDISD			(n+1)T-120	ns	
$\overline{IOSTB}$ Width Low	tWISL		(n+1)T-40		ns	
$\overline{MREQ}$ to $\overline{IOSTB}$ Delay Time	tDMRIS		1T-50		ns	*
Next DMARQ Setup Time	tSDADQ	Demand Mode		1T	ns	
DMARQ Hold Time	tHDARQ	Demand Mode	0		ns	
$\overline{DMAAK}$ Read Width Low	tWDMRL		(n+2.5)T-40		ns	*
$\overline{DMAAK}$ Write Width Low	tWDMWL		(n+2)T-40		ns	*
$\overline{REFRQ}$ Delay Time	tDARF		0.5T-50		ns	
$\overline{REFRQ}$ Width Low	tWRFL		(n+2)T-40		ns	*
Address Hold Time	tHRFA		0.5T-50		ns	

A. C. CHARACTERISTICS (Continued)

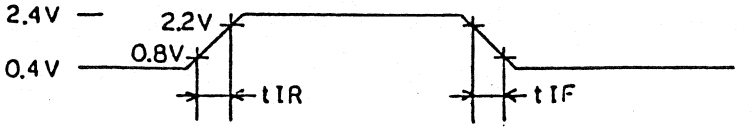
PARAMETER	SYMBOL	TEST CONDITION	MIN	MAX	UNIT	
$\overline{\text{RESET}}$ Width Low (STOP/POR.)	tWRSL1		30		ms	
$\overline{\text{RESET}}$ Width Low (SYTEM RESET)	tWRSL2		5		μs	
$\overline{\text{MREQ}}$, $\overline{\text{IOSTB}}$ to READY Setup Time	tSCRY	n ≧2		nT-100	ns	*
$\overline{\text{MREQ}}$, $\overline{\text{IOSTB}}$ to READY Hold Time	tHCRY	n ≧2	nT		ns	*
HLDRQ Setup Time	tSHQK		30		ns	
$\overline{\text{HLDAK}}$ Output Delay Time	tDKHA			80	ns	
BUS Contorol Float to $\overline{\text{HLDAK}}$ ↓	tCFHA		1T-50		ns	
$\overline{\text{HLDAK}}$ ↑ to Control Output Time	tDHAC		1T-50		ns	
HLDRQ to $\overline{\text{HLDAK}}$ Delay Time	tDHQHA			3T+160	ns	
HLDRQ ↓ to Control Float	tDHQC		3T+30		ns	
HLDRQ Width Low	tWHQL		1.5T		ns	
$\overline{\text{HLDAK}}$ Width Low	tWHAL		1 T		ns	
INTP, DMARQ Setup Time	tSIQK		30		ns	
INTP, DMARQ Width High	tWIQH		8T		ns	
INTP, DMARQ Width Low	tWIQL		8T		ns	
$\overline{\text{POLL}}$ Setup Time	tSPLK		30		ns	
NMI Width High	tWNIH		5		μs	
NMI Width Low	tWNIL		5		μs	
$\overline{\text{CTS}}$ Width Low	tWCTL		2T		ns	
INTR Setup Time	tSIRK		30		ns	
$\overline{\text{INTAK}}$ Delay Time	tDKIA			80	ns	
INTR Hold Time	tHIAIQ		0		ns	
$\overline{\text{INTAK}}$ Width Low	tWIAL		2T-40		ns	
$\overline{\text{INTAK}}$ Width High	tWIAH		1T-40		ns	
$\overline{\text{INTAK}}$ to DATA Delay Time	tDIAD			2T-130	ns	
$\overline{\text{INTAK}}$ to DATA Hold Time	tHIAD		0	0.5T	ns	

A. C. CHARACTERISTICS (Continued)

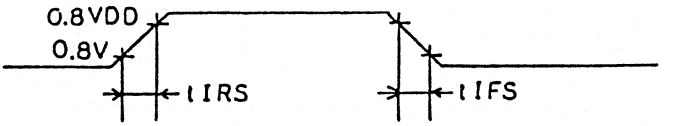
PARAMETER	SYMBOL	TEST CONDITION	MIN	MAX	UNIT
$\overline{SCKO}$ Cycle Time	tCYTK		1000		ns
$\overline{SCKO}$ (TSCK) Width High	tWSTH		450		ns
$\overline{SCKO}$ (TSCK) Width Low	tWSTL		450		ns
TXD Delay Time	tDTKD			210	ns
$\overline{CTSO}$ (RSCK) Cycle Time	tCYRK		1000		ns
$\overline{CTSO}$ (RSCK) Width High	tWSRH		420		ns
$\overline{CTSO}$ (RSCK) Width Low	tWSRL		420		ns
RXD Setup Time	tSRDK		80		ns
RXD Hold Time	tHKRD		80		ns

CL=100pF

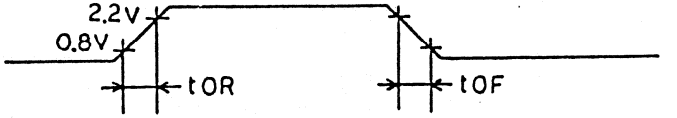
AC INPUT WAVEFORM(1) (EXCEPT X1, X2, $\overline{\text{RESET}}$, $\overline{\text{NMI}}$)



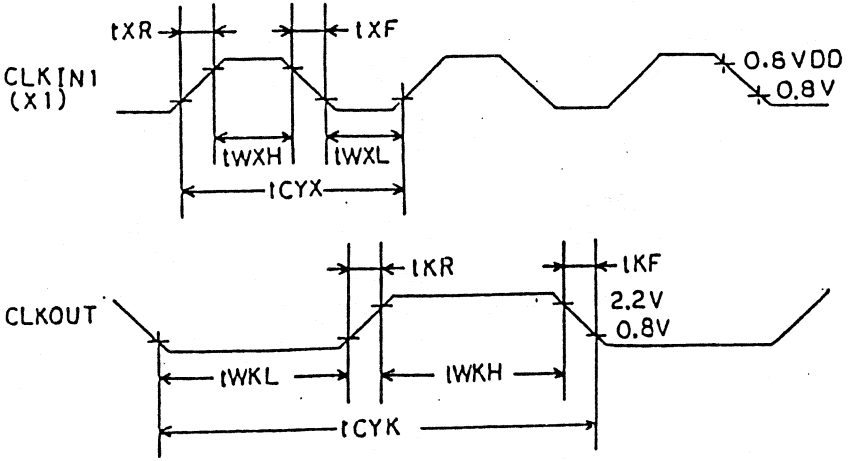
AC INPUT WAVEFORM(2) ($\overline{\text{RESET}}$, $\overline{\text{NMI}}$)

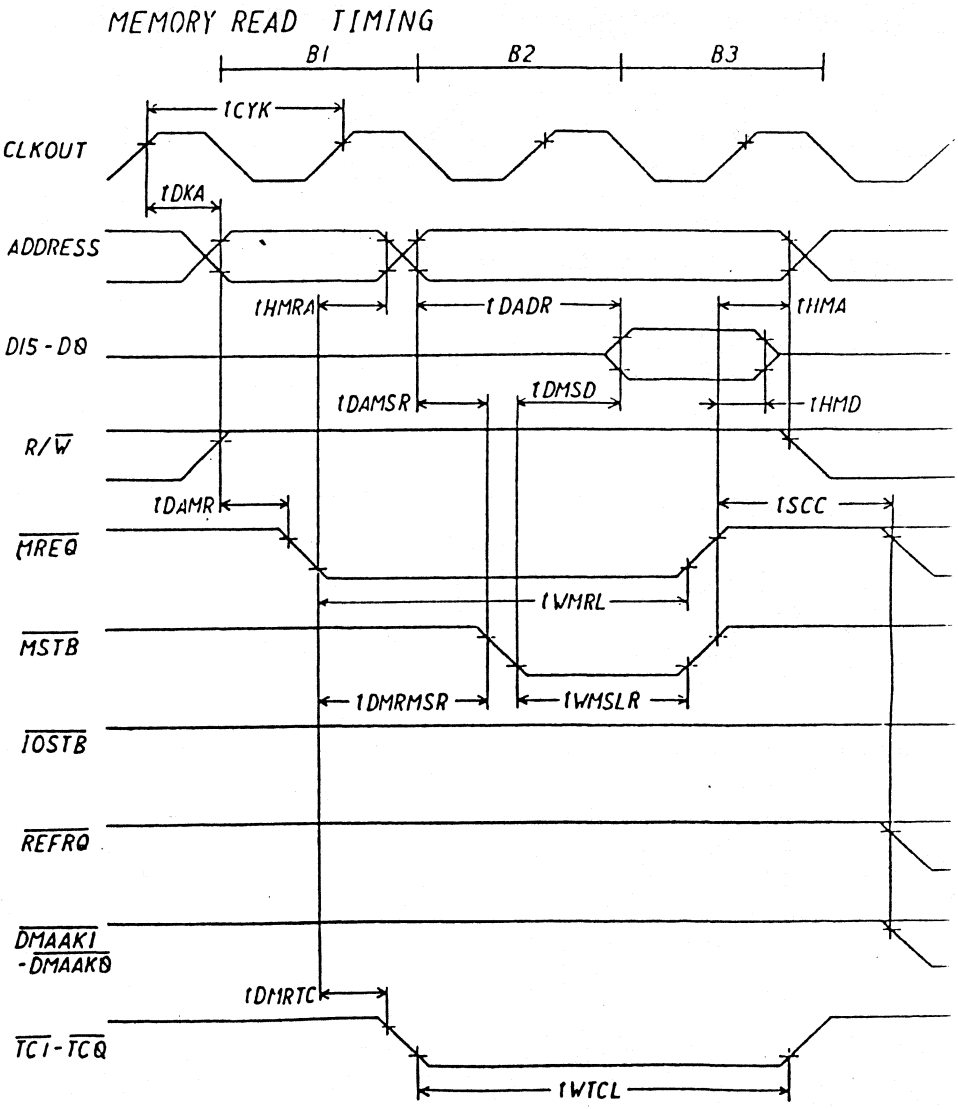


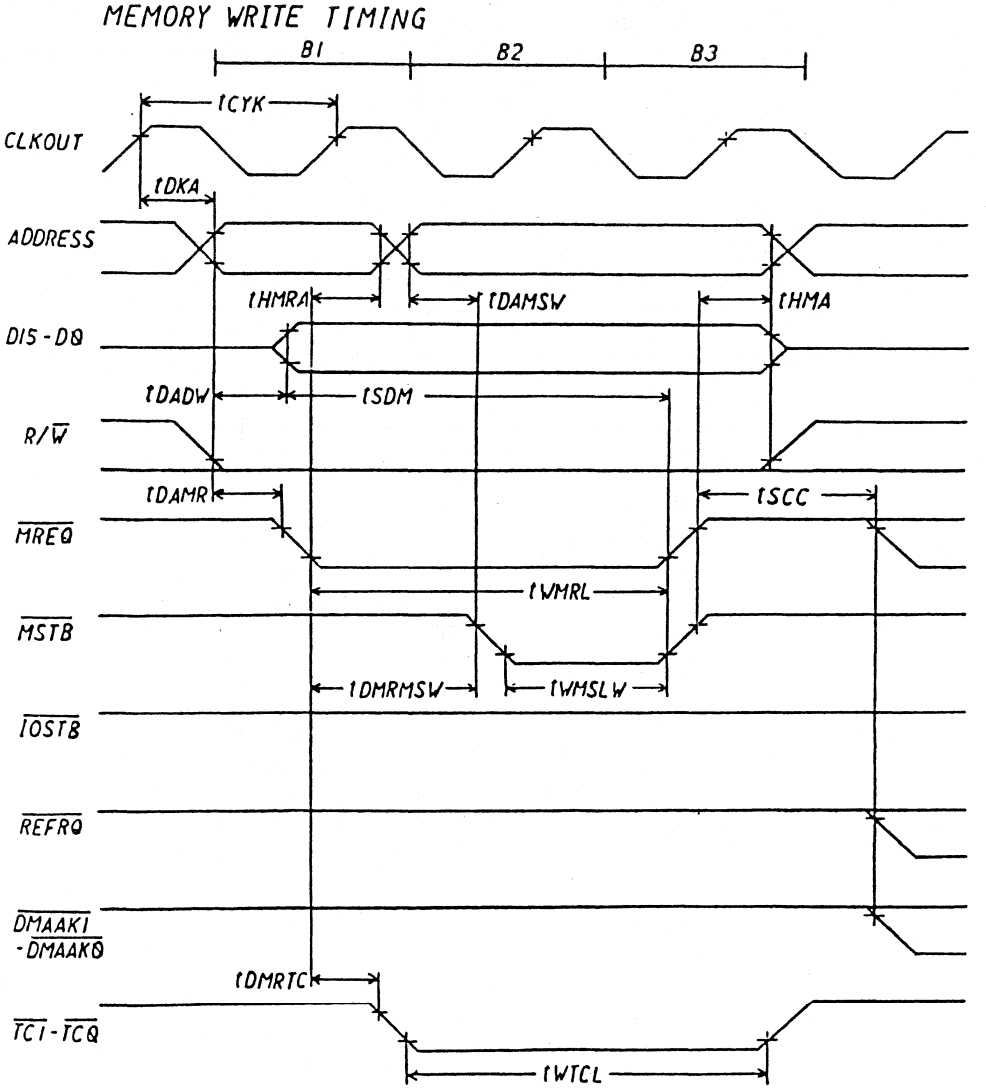
AC OUTPUT TEST POINT (EXCEPT CLKOUT)

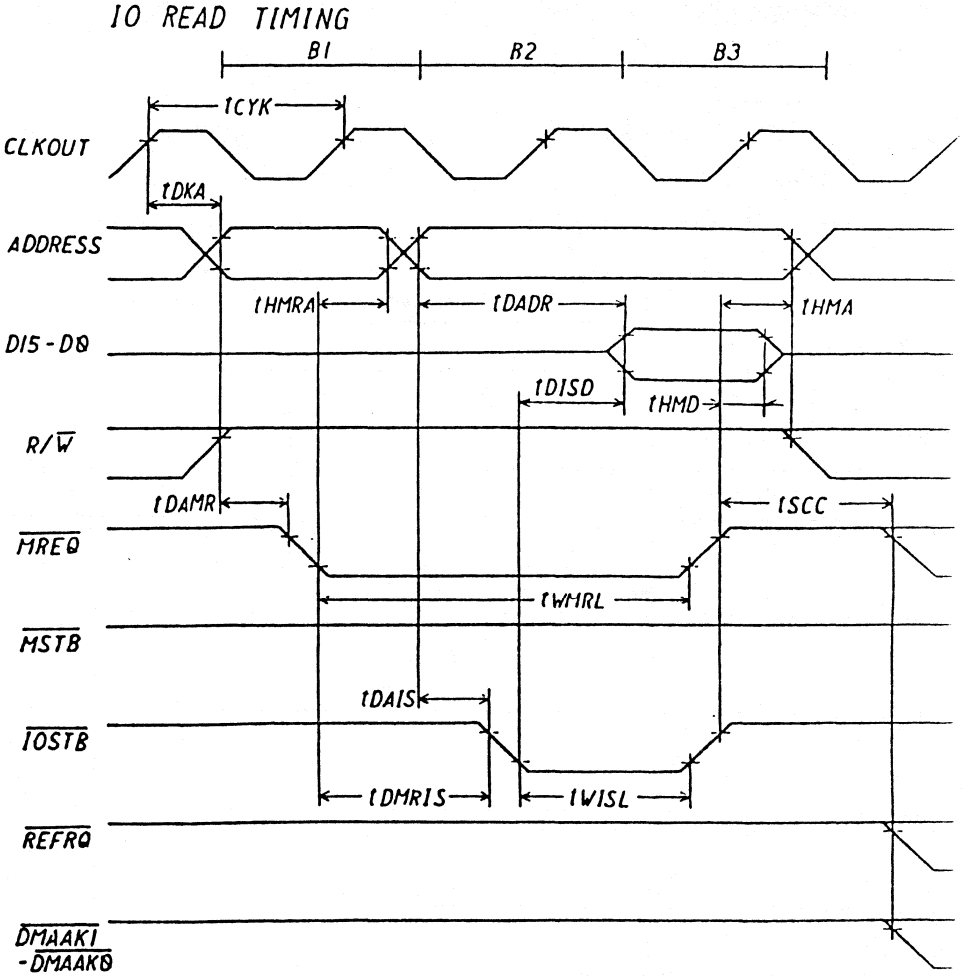


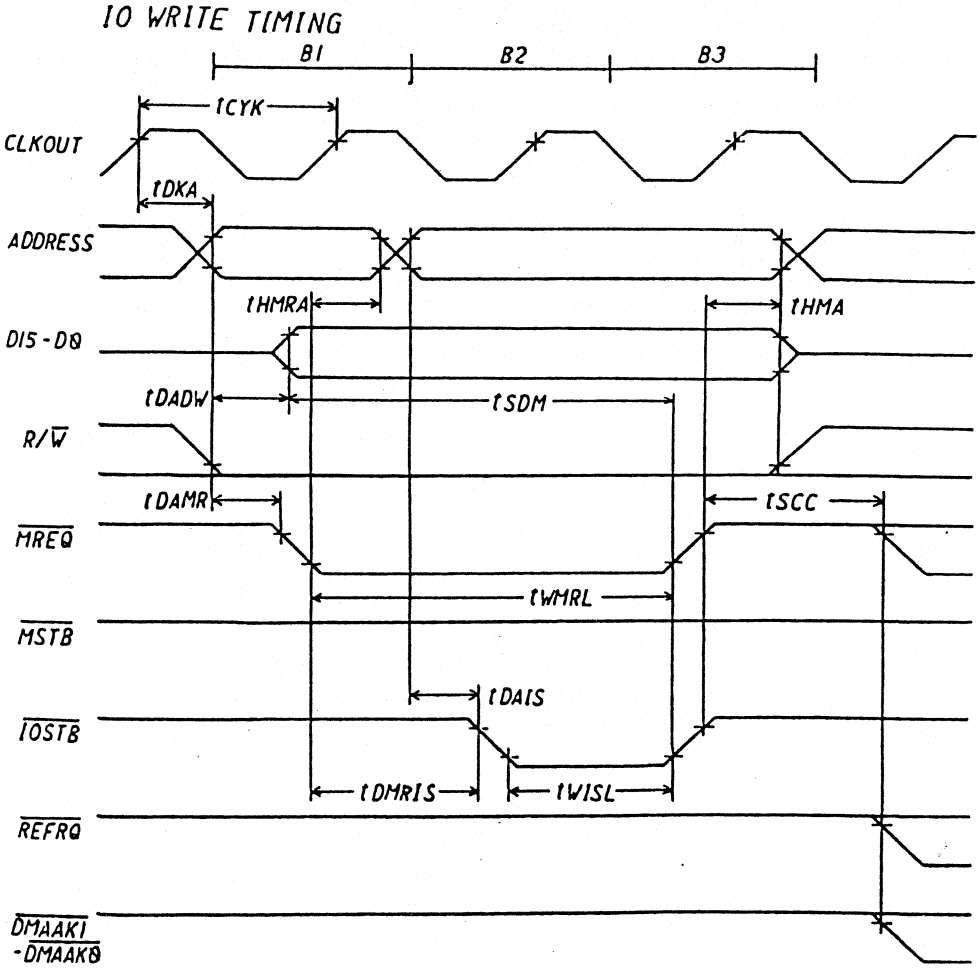
CLOCK TIMING

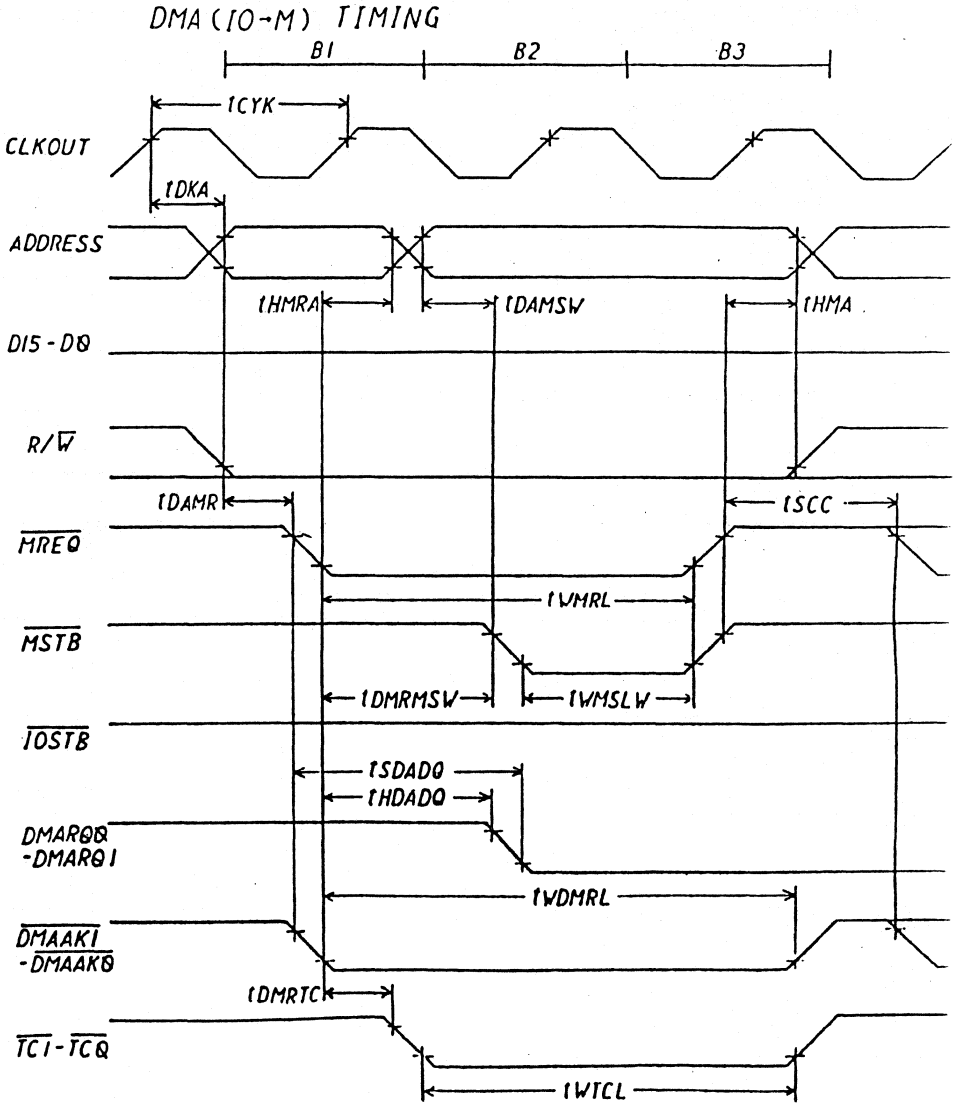


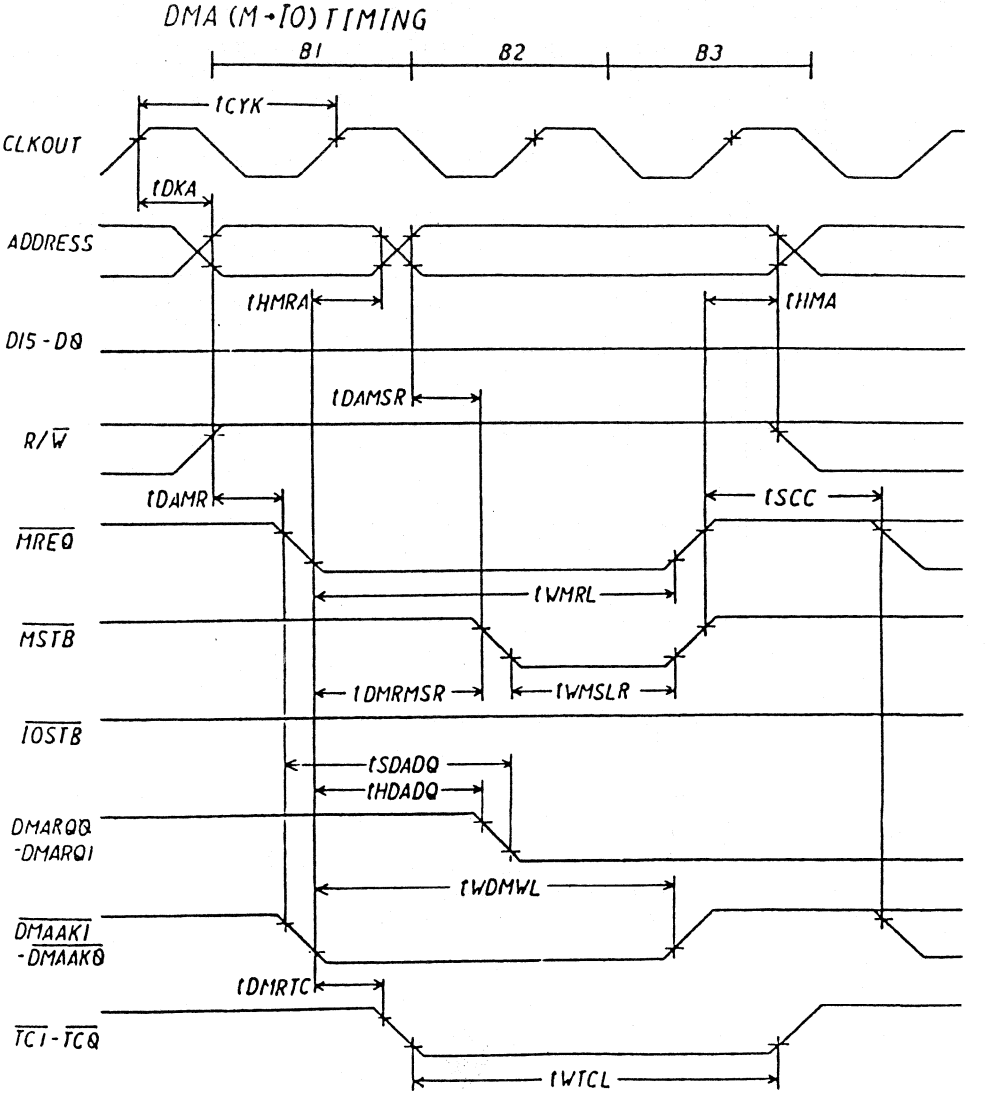

MEMORY READ TIMING
B1
B2
B3
tCYK
CLKOUT
tDKA
ADDRESS
tHMRA
tDADR
tHMA
DI5 - DI0
tDAMSR
tDMSD
tHMD
R/W
tDAMR
tSCC
MREQ
tWMRL
MSTB
tDMRMSR
tWMSLR
IOSTB
REFRQ
DMAAK1
- DMAAK0
tDMRTC
TC1 - TC0
tWTCL

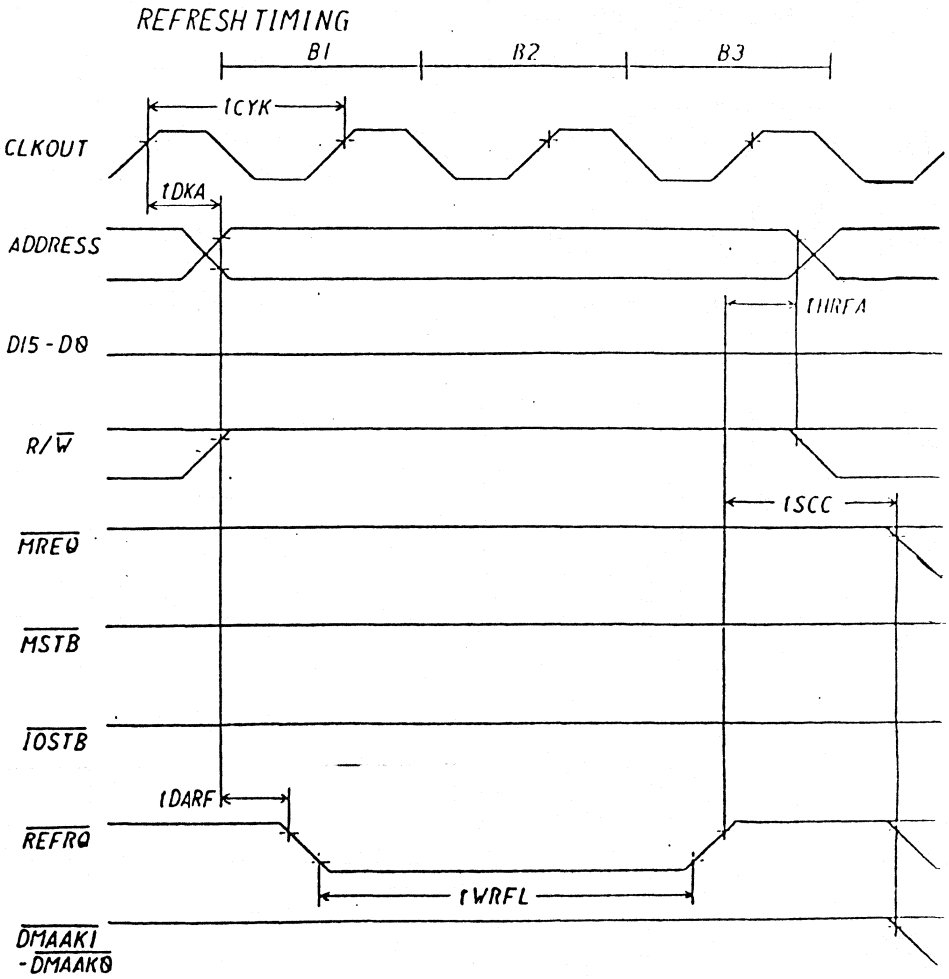

MEMORY WRITE TIMING
B1
B2
B3
tCYK
CLKOUT
tDKA
ADDRESS
tHMRA
tDAMSW
tHMA
DI5 - D0
tDADW
tSDM
R/W
tDAMR
tSCC
MREQ
tWMRL
MSTB
tDMRMSW
tWMSLW
IOSTB
REFRQ
DMAAK1
- DMAAK0
tDMRTC
TC1 - TC0
tWTCL


IO READ TIMING
B1
B2
B3
tCYK
CLKOUT
tDKA
ADDRESS
tHMRA
tDADR
tHMA
DI5-D0
tDISD
tHMD
R/W
tDAMR
tSCC
MREQ
tWMRL
MSTB
tDAIS
IOSTB
tDMRIS
tWISL
REFRQ
DMAAKI
-DMAAK0

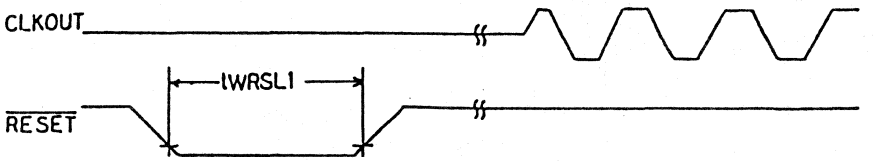

IO WRITE TIMING
B1
B2
B3
tCYK
CLKOUT
tDKA
ADDRESS
tHMRA
tHMA
DI5 - D0
tDADW
tSDM
R/W
tDAMR
tSCC
MREQ
tWMRL
MSTB
tDAIS
IOSTB
tDMRIS
tWISL
REFRQ
DMAAK1
- DMAAK0


DMA (IO→M) TIMING
B1
B2
B3
tCYK
CLKOUT
tDKA
ADDRESS
tHMRA
tDAMSW
tHMA
DI5 - DI0
R/W
tDAMR
tSCC
MREQ
tWMRL
MSTB
tDMRMSW
tWMSLW
IOSTB
tSDADQ
tHDADQ
DMARQ3
-DMARQ1
tWDMRL
DMAAK1
-DMAAK0
tDMRTC
TC1 - TC0
tWTCL

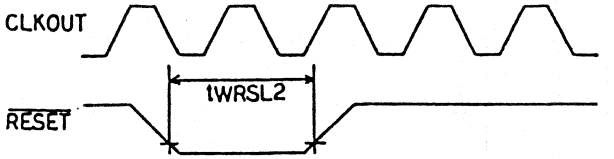

DMA (M→IO) TIMING
B1
B2
B3
tCYK
CLKOUT
tDKA
ADDRESS
tHMRA
tHMA
DI5 - DI0
tDAMSR
R/W
tDAMR
tSCC
MREQ
tWMRL
MSTB
tDMRMSR
tWMSLR
IOSTB
tSDADQ
tHDADQ
DMARQ3
-DMARQ1
tWDMWL
DMAAK1
-DMAAK0
tDMRTC
TC1 - TC0
tWTCL


REFRESH TIMING
B1
B2
B3
tCYK
CLKOUT
tDKA
ADDRESS
tHRFA
D15 - D0
R/W
tSCC
MREQ
MSTB
IOSTB
tDARF
REFRQ
tWRFL
DMAAK1
- DMAAK0

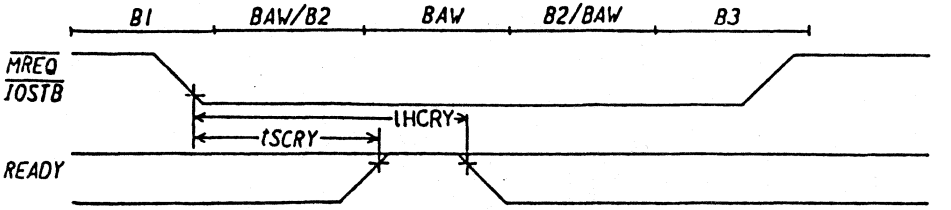
$\overline{\text{RESET}}$ TIMING (1)



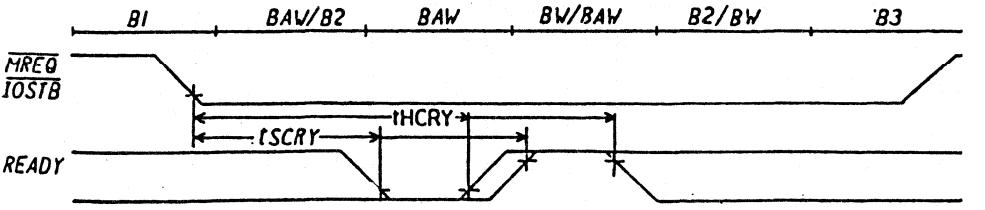
$\overline{\text{RESET}}$ TIMING (2)

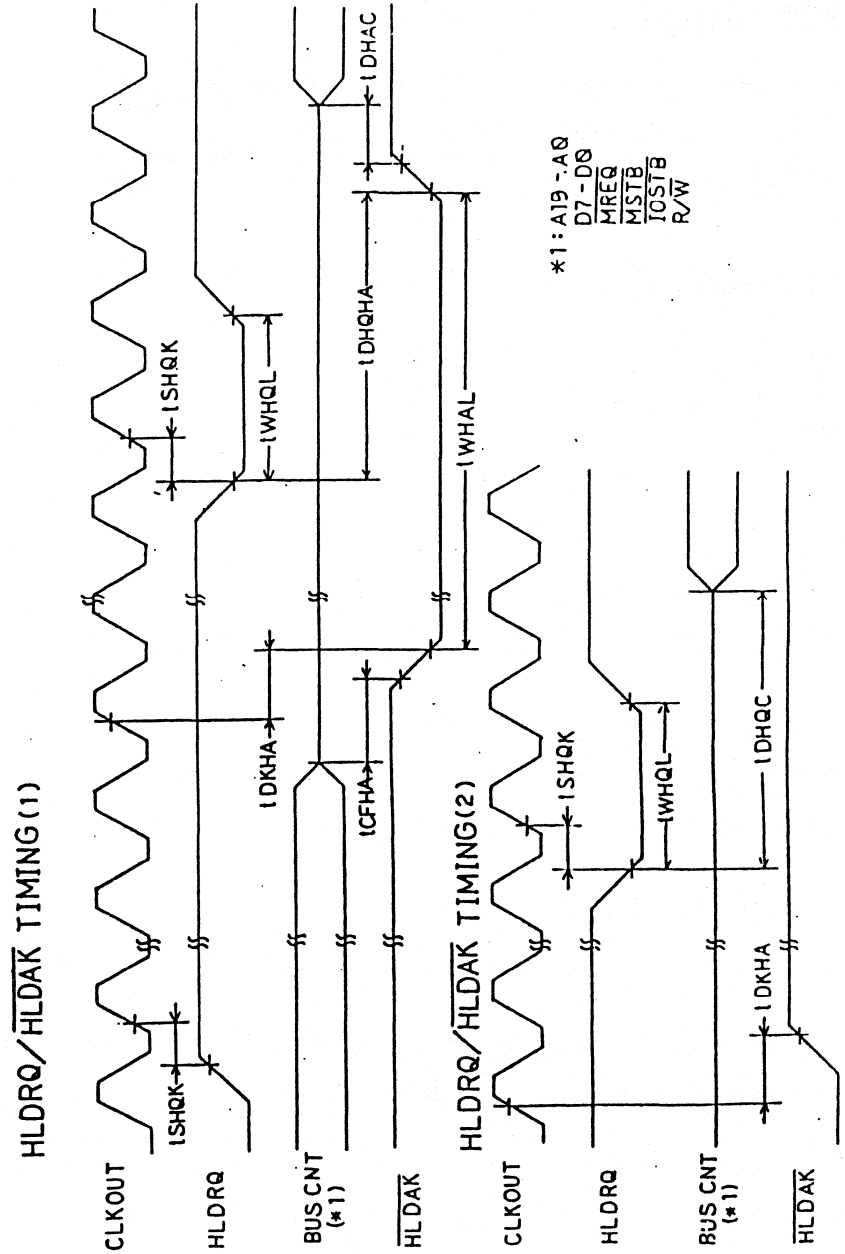


READY TIMING (1)

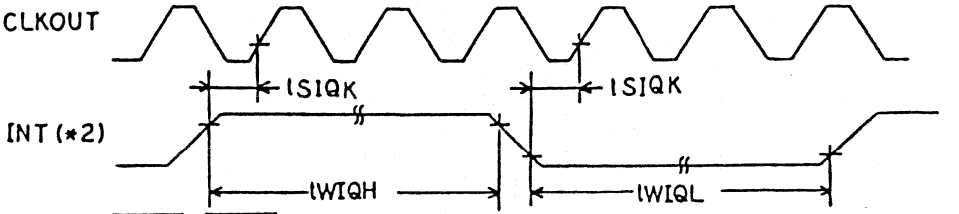


READY TIMING (2)



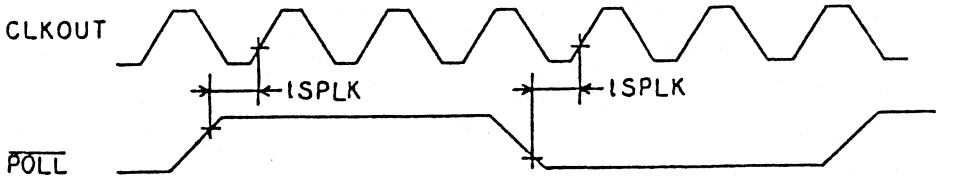

HLDRQ/HLDAK TIMING(1)
CLKOUT
HLDRQ
BUS CNT (*1)
HLDAK
tSHQK
tSHQK
tWHQL
tDHQHA
tDHAC
tDKHA
tCFHA
tWHAL
*1: A19-A0
D7-D0
MREQ
MSTB
IOSTB
R/W
HLDRQ/HLDAK TIMING(2)
CLKOUT
HLDRQ
BUS CNT (*1)
HLDAK
tSHQK
tWHQL
tDHQC
tDKHA

$\overline{INTP2}$ - $\overline{INTP0}$, DMARQ1-DMARQ0 INPUT TIMING

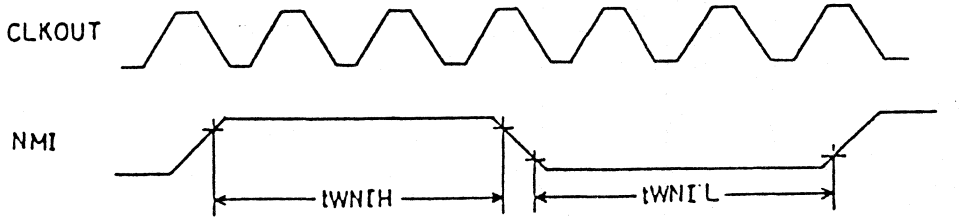


*2 : $\overline{INTP2}$ - $\overline{INTP0}$, DMARQ1-DMARQ0

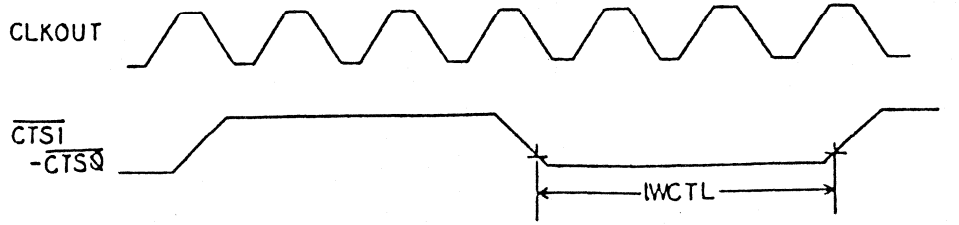
$\overline{POLL}$ INPUT TIMING



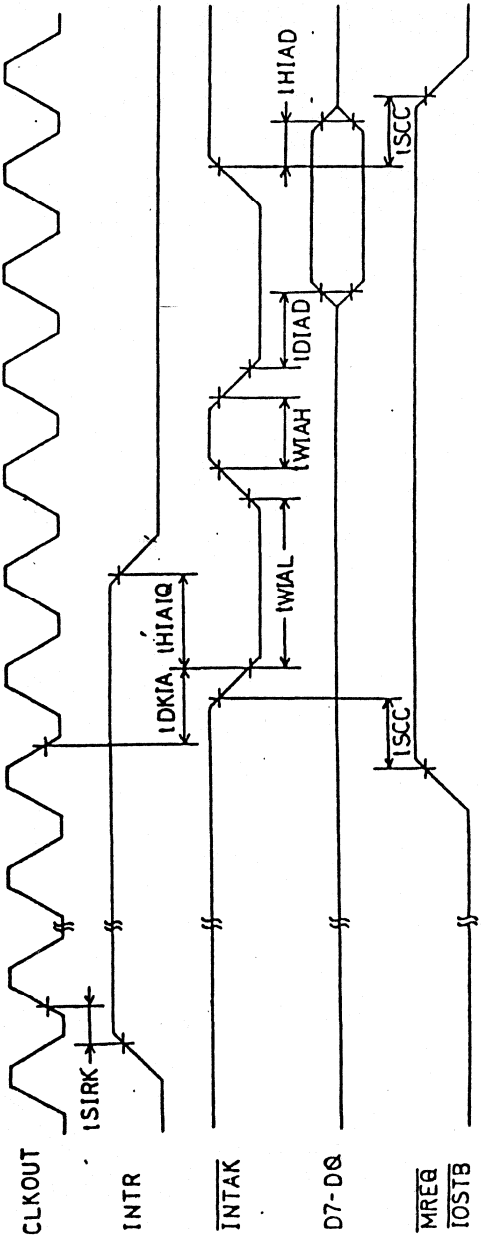
NMI INPUT TIMING



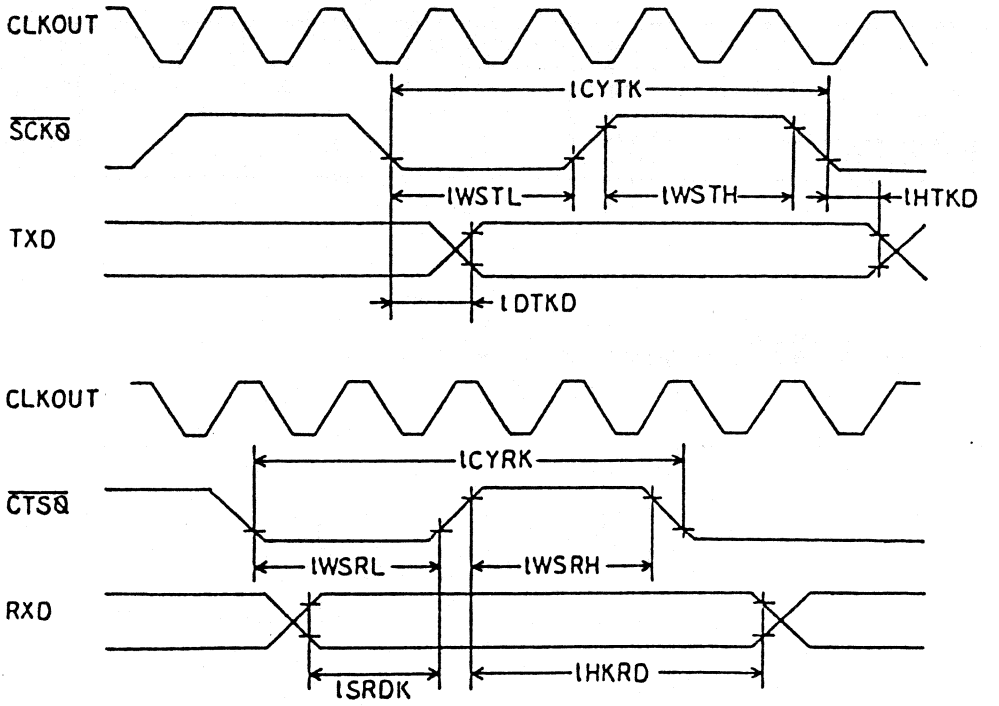
$\overline{CTS1}$ - $\overline{CTS0}$ INPUT TIMING



INTR/$\overline{\text{INTAK}}$ TIMING



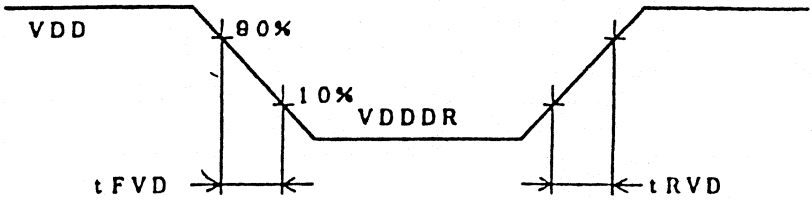
SIO TIMING


CLKOUT
tCYTK
SCKQ
tWSTL
tWSTH
tHTKD
TXD
tDTKD
CLKOUT
tCYRK
CTSQ
tWSRL
tWSRH
RXD
tSRDK
tHKRD

STOP MODE DATA RETAINTION CHARACTERISTISCS (Ta = −10℃ to +70℃)

PARAMETER	SYMBOL	TEST CONDITION	MIN	MAX	UNIT
Data Retaintion Voltage	VDDDR		2.5	5.5	V
VDD Rise Time VDD Fall Time	tRVD tFVD		200		μS

STOP MODE DATA RETAINTION TIMING

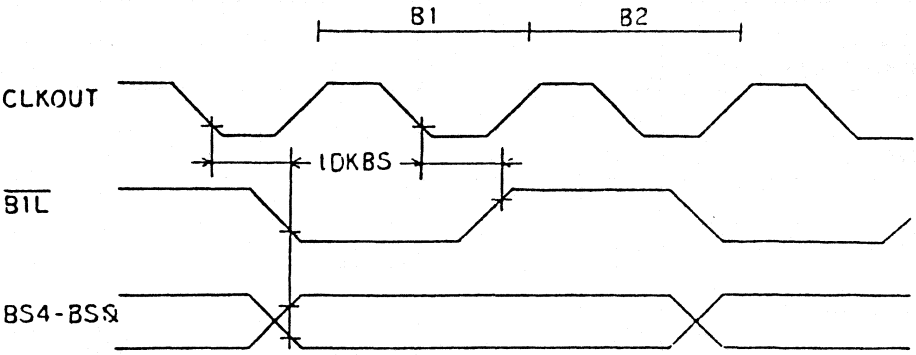


μPD70329/339-8
A. C. CHARACTERISTICS (Ta=-10 ℃ to +70 ℃, VDD=+5.0V±10%)

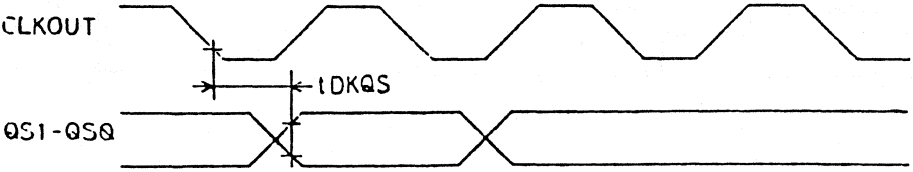
PARAMETER	SYMBOL	TEST CONDITION	MIN	MAX	UNIT
$\overline{BIL}$, BUS STATTUS Delay Time	tDKBS			60	ns
QUEUE STATTUS Delay Time	tDKQS			60	ns
$\overline{INTSV}$ Setup Time	tSSQK		30		ns
$\overline{SVACK}$ Delay Time	tDKSA			60	ns
$\overline{SVACK}$ to $\overline{INTSV}$ Hold Time	tHSASQ		0		ns
$\overline{IPINC}$ Delay Time	tDKIP			60	ns
INSTRUCTION Setup Time	tSIK		30		ns
INSTRUCTION Hold Time	tHKI		60		ns

μPD70329/339-8

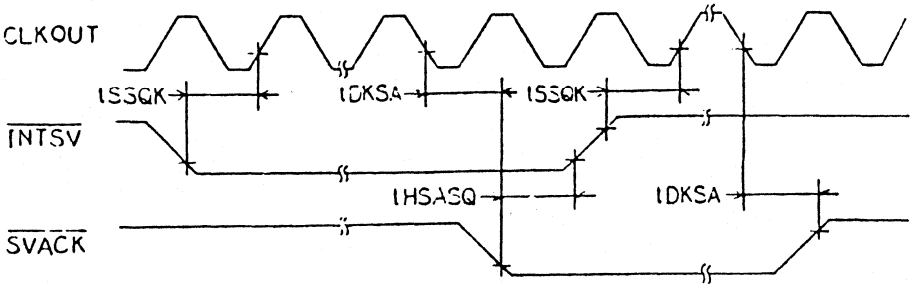
BUS STATUS TIMING



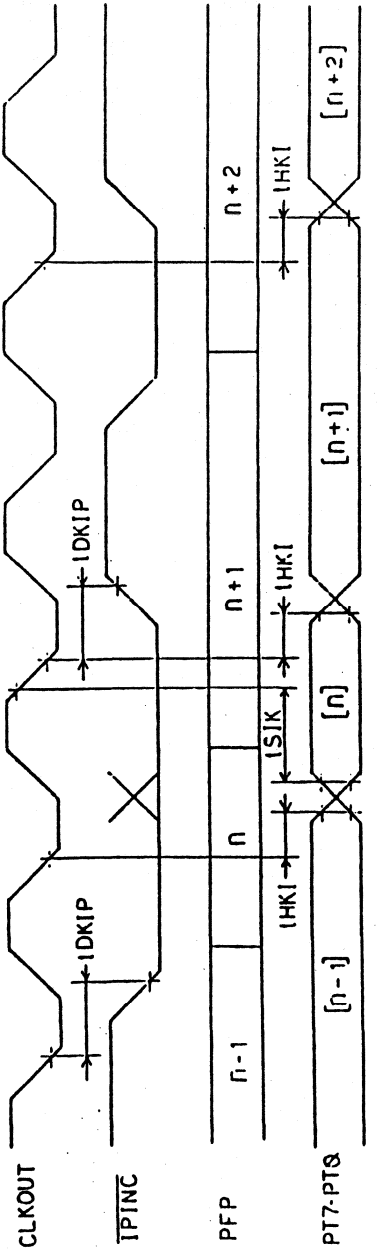
QUEUE STATUS TIMING



INTSV/SVACK TIMING



μPD70329/339-8
INSTRUCTION FETCH TIMING



PRELIMINARY

APPENDIX 2G

μPD70330/70332

INSTRUCTION

EXECUTION

CLOCK LIST

V35 INSTRUCTION EXECUTION CLOCK LIST

INSTRUCTION	BYTE operation		WORD operation		Note
	RAM enable	RAM disable	RAM enable	RAM disable	
ADD reg,reg	2	2	2	2	
ADD mem,reg	EA+10+2W	EA+7+1W	EA+10+2W	EA+7+1W	
CMP mem,reg	EA+7+1W	EA+7+1W	EA+7+1W	EA+7+1W	
ADD reg,mem	EA+7+1W	EA+7+1W	EA+7+1W	EA+7+1W	
ADD AL,imm8	5	5	-	-	
ADD AW,imm16	-	-	6	6	
PUSH sreg	-	-	10+1W	7	
POP sreg	-	-	12+1W	12+1W	
TEST1 reg,CL	7	7	7	7	
TEST1 mem,CL	EA+12+1W	EA+12+1W	EA+12+1W	EA+12+1W	
CLR1 reg,CL	8	8	8	8	
CLR1 mem,CL	EA+16+2W	EA+13+1W	EA+16+2W	EA+13+1W	
SET1 reg,CL	7	7	7	7	
SET1 mem,CL	EA+15+2W	EA+12+1W	EA+15+2W	EA+12+1W	
NOT1 reg,CL	7	7	7	7	
NOT1 mem,CL	EA+15+2W	EA+12+1W	EA+15+2W	EA+12+1W	
TEST1 reg,imm8	6	6	6	6	
TEST1 mem,imm8	EA+9+1W	EA+9+1W	EA+9+1W	EA+9+1W	
CLR1 reg,imm8	7	7	7	7	
CLR1 mem,imm8	EA+13+2W	EA+10+1W	EA+13+2W	EA+9+1W	
SET1 reg,imm8	6	6	6	6	
SET1 mem,imm8	EA+12+2W	EA+9+1W	EA+12+2W	EA+9+1W	
NOT1 reg,imm8	6	6	6	6	
NOT1 mem,imm8	EA+12+2W	EA+9+1W	EA+12+2W	EA+9+1W	
ADD4S	22+(30+3W)n	22+(28+3W)n	-	-	n= (no.of BCD figures) / 2
SUB4S	22+(30+3W)n	22+(28+3W)n	-	-	n= (no.of BCD figures) / 2
CMP4S	22+(25+2W)n	22+(25+2W)n	-	-	n= (no.of BCD figures) / 2
ROL4 reg	17	17	-	-	
ROL4 mem	EA+20+2W	EA+18+2W	-	-	
ROR4 reg	21	21	-	-	
ROR4 mem	EA+26+2W	EA+24+2W	-	-	
INS reg,reg	-	-	63~155	63~155	
INS reg,imm8	-	-	64~156	64~156	
EXT reg,reg	-	-	41~121	41~121	
EXT reg,imm8	-	-	42~122	42~122	
ADJ4A	9	9	-	-	
ADJ4S	9	9	-	-	
ADJBA	17	17	-	-	
ADJBS	17	17	-	-	
INC/DEC reg16	-	-	2	2	
PUSH reg16	-	-	9+1W	6	
POP reg16	-	-	11+1W	11+1W	
PUSH R	-	-	74+8W	50	
POP R	-	-	74+8W	58	
CHKIND reg,mem	-	-	EA+24+2W	EA+24+2W	
FPO1/FPO2	-	-	55+5W	43+5W	
PUSH imm16	-	-	13+1W	10	
MUL r1,r2,imm16	-	-	40~50	40~50	
MUL r1,mem,imm16	-	-	EA+43+1W ~EA+53+1W	EA+43+1W ~EA+53+1W	
PUSH imm8	-	-	12+1W	9	
MUL r1,r2,imm8	-	-	39~49	39~49	
MUL r1,mem,imm8	-	-	EA+42+1W ~EA+52+1W	EA+42+1W ~EA+52+1W	
INM	19+2W	17+2W	21+2W	17+2W	
(rep)	18+(13+2W)n	18+(11+2W)n	18+(15+2W)n	18+(11+2W)n	n= repetition no.
OUTM	21+2W	19+2W	19+2W	15+2W	
(rep)	18+(15+2W)n	18+(13+2W)n	18+(13+2W)n	18+(9+2W)n	n= repetition no.
BV	-	-	15/8	15/8	
BNV	-	-	15/8	15/8	
BC	-	-	15/8	15/8	
BL	-	-	15/8	15/8	
BNC	-	-	15/8	15/8	
BNL	-	-	15/8	15/8	
BE	-	-	15/8	15/8	
BZ	-	-	15/8	15/8	
BNE	-	-	15/8	15/8	

INSTRUCTION	BYTE operation		WORD operation		Note
	RAM enable	RAM disable	RAM enable	RAM disable	
BNZ	-	-	15/8	15/8	
BNH	-	-	15/8	15/8	
BH	-	-	15/8	15/8	
BN	-	-	15/8	15/8	
BP	-	-	15/8	15/8	
BPE	-	-	15/8	15/8	
BPO	-	-	15/8	15/8	
BLT	-	-	15/8	15/8	
BGE	-	-	15/8	15/8	
BLE	-	-	15/8	15/8	
BGT	-	-	15/8	15/8	
ADD reg,imm8	5	5	-	-	
ADD mem,imm8	EA+11+2W	EA+9+2W	-	-	
CMP mem,imm8	EA+8+1W	EA+8+1W	-	-	
ADD reg,imm16	-	-	6	6	
ADD mem,imm16	-	-	EA+12+2W	EA+8+2W	
CMP mem,imm16	-	-	EA+9+1W	EA+9+1W	
TEST reg,reg	4	4	4	4	
mem,reg	EA+9+1W	EA+9+1W	EA+9+1W	EA+9+1W	
XCH reg,reg	3	3	3	3	
mem,reg	EA+12+2W	EA+9+1W	EA+12+2W	EA+9+1W	
MOV reg,reg	2	2	2	2	
MOV mem,reg	EA+5+1W	EA+2	EA+5+1W	EA+2	
reg,mem	EA+7+1W	EA+7+1W	EA+7+1W	EA+7+1W	
MOV reg,sreg	-	-	3	3	
MOV mem,sreg	-	-	EA+6+1W	EA+3	
LDEA	-	-	EA+2	EA+2	
MOV sreg,reg	-	-	4	4	
MOV sreg,mem	-	-	EA+9+1W	EA+9+1W	
POP mem	-	-	EA+14+2W	EA+11+1W	
XCH AW,reg	-	-	4	4	
CVTBW	3	3	-	-	
CVTWL	-	-	8	8	
CALL far	-	-	36+2W	32+2W	
POLL	-	-	-	-	
PUSH PSW	-	-	9+1W	6	
POP PSW	-	-	13+1W	13+1W	
MOV PSW,AH	3	3	-	-	
MOV AH,PSW	2	2	-	-	
MOV A,mem	10+1W	10+1W	10+1W	10+1W	
MOV mem,A	8+1W	5	8+1W	5	
MOVBK	22+2W	17+1W	22+2W	19+1W	
(rep)	16+(18+2W)n	16+(13+1W)n	16+(18+2W)n	16+(13+1W)n	n= repetition no.
(rep CW=0)	12	12	12	12	
CMPBK	25+2W	21+2W	25+2W	19+2W	
(rep)	16+(23+2W)n	16+(23+2W)n	16+(23+2W)n	16+(23+2W)n	n= repetition no.
(rep CW=0)	12	12	12	12	
TEST AL,imm8	5	5	-	-	
TEST AW,imm16	-	-	6	6	
STM	13+1W	10	13+1W	10	
(rep)	16+(9+1W)n	16+(7+1W)n	16+(7+1W)n	16+(5+1W)n	n= repetition no.
(rep CW=0)	12	12	12	12	
LDM	13+1W	13+1W	13+1W	13+1W	
(rep)	16+(11+1W)n	16+(11+1W)n	16+(11+1W)n	16+(11+1W)n	n= repetition no.
(rep CW=0)	12	12	12	12	
CMPM	18+1W	18+1W	18+1W	18+1W	
(rep)	16+(16+1W)n	16+(16+1W)n	16+(16+1W)n	16+(16+1W)n	n= repetition no.
(rep CW=0)	12	12	12	12	
SHL reg,imm8	9+2n	9+2n	9+2n	9+2n	
SHL mem,imm8	EA+15+2W+2n	EA+12+1W+2n	EA+15+2W+2n	EA+12+1W+2n	n= repetition no.
RET near(pop_value)	-	-	19+1W	19+1W	
RET near	-	-	19+1W	19+1W	

INSTRUCTION	BYTE operation		WORD operation		Note
	RAM enable	RAM disable	RAM enable	RAM disable	
MOV DS0,reg,mem	-	-	EA+17+2W	EA+17+2W	
MOV DS1,reg,mem	-	-	EA+17+2W	EA+17+2W	
MOV reg,imm8	5	5	-	-	
MOV mem,imm8	EA+6+1W	EA+6+1W	-	-	
MOV reg,imm16	-	-	6	6	
MOV mem,imm16	-	-	EA+6+1W	EA+6+1W	
PREPARE (imm8=0)	26+1W	26+1W	26+1W	26+1W	
(imm8=1)	37+2W	37+2W	37+2W	37+2W	
(imm8>1)	44+19(n-1)2nW	44+19(n-1)2nW	44+19(n-1)2nW	44+19(n-1)2nW	n= value of 2nd oprand
DISPOSE	-	-	11+1W	11+1W	
RET far(pop_value)	-	-	28+2W	28+2W	
RET far	-	-	27+2W	27+2W	
BRK 3	-	-	50+5W	38+5W	
BRK n	-	-	51+5W	39+5W	
BRKV	-	-	50+5W	38+5W	
RETI	-	-	40+3W	34+1W	
SHL reg,1	8	8	8	8	
SHL mem,1	EA+16+2W	EA+13+1W	EA+16+2W	EA+13+1W	
SHL reg,CL	11+2n	11+2n	11+2n	11+2n	n= no.of shifts (CL)
SHL mem,CL	EA+19+2W+2n	EA+16+1W+2n	EA+19+2W+2n	EA+16+1W+2n	n= no.of shifts (CL)
CVTDB	20	20	-	-	
CVTBD	19	19	-	-	
TRANS	11+1W	11+1W	-	-	
DBNZNE /DBNZE	-	-	17/8	17/8	
DBNZ	-	-	17/8	17/8	
DCWZ	-	-	15/8	15/8	
IN A,imm8	15+1W	15+1W	15+1W	15+1W	
OUT imm8,A	11+1W	11+1W	11+1W	11+1W	
CALL near	-	-	12	12	
BR far	-	-	15	15	
BR near short	-	-	12	12	
IN A,DW	14+1W	14+1W	14+1W	14+1W	
OUT DW,A	10+1W	10+1W	10+1W	10+1W	
HALT					
TEST reg,imm8	7	7	-	-	
mem,imm8	EA+9+1W	EA+9+1W	-	-	
TEST reg,imm16	-	-	8	8	
mem,imm16	-	-	EA+10+1W	EA+10+1W	
NOT/NEG reg	5	5	5	5	
mem	EA+13+2W	EA+10+1W	EA+13+2W	EA+10+1W	
MULU reg	24	24	-	-	
MULU mem	EA+27+1W	EA+27+1W	-	-	
MULU reg	-	-	32	32	
MULU mem	-	-	EA+33+1W	EA+33+1W	
MUL reg	31~40	31~40	-	-	
MUL mem	EA+34+1W ~EA+43+1W	EA+34+1W ~EA+43+1W	-	-	
MUL reg	-	-	39~48	39~48	
MUL mem	-	-	EA+42+1W ~EA+51+1W	EA+42+1W ~EA+51+1W	
EI	12	12	12	12	
DI	4	4	4	4	
DIVU reg	31	31	-	-	
DIVU mem	EA+34+1W	EA+34+1W	-	-	
DIVU reg	-	-	39	39	
DIVU mem	-	-	EA+43+2W	EA+43+2W	
DIV reg	46~56	46~56	-	-	
DIV mem	EA+49+1W ~EA+59+1W	EA+49+1W ~EA+59+1W	-	-	
DIV reg	-	-	54~64	54~64	
DIV mem	-	-	EA+57+1W ~EA+67+1W	EA+57+1W ~EA+67+1W	
INC/DEC reg8	5	5	-	-	
mem	EA+13+2W	EA+10+1W	EA+13+2W	EA+10+1W	
CALL near reg	-	-	21+1W	17+1W	
mem	-	-	EA+24+2W	EA+22+2W	
CALL far mem	-	-	EA+32+4W	EA+20+4W	

INSTRUCTION	BYTE operation		WORD operation		Note
	RAM enable	RAM disable	RAM enable	RAM disable	
BR near reg	-	-	13	13	
mem	-	-	EA+16+1W	EA+16+1W	
BR far mem	-	-	EA+23+2W	EA+23+2W	
PUSH mem	-	-	EA+16+2W	EA+12+2W	
REPC	2	2	2	2	
REPNC	2	2	2	2	
REP/REPE/REPZ	2	2	2	2	
REPNE/REPNZ	2	2	2	2	
BUSLOCK	2	2	2	2	
NOP	4	4	4	4	
NOT1	2	2	2	2	
CLR1 CY	2	2	2	2	
DIR	2	2	2	2	
SET1 CY	2	2	2	2	
DIR	2	2	2	2	
FINT	2	2	2	2	
DS1:	2	2	2	2	
PS:	2	2	2	2	
SS:	2	2	2	2	
DS0:	2	2	2	2	

INSTRUCTION	BYTE operation		WORD operation		Note
	RAM enable	RAM disable	RAM enable	RAM disable	
MS SFR<-mem	22+(3+1W)	20+1W	22+(3+1W)	20+1W	
MS mem<-SFR	22+(1W-1) min=22	21+1W	22+(1W-1) min=22	21+1W	
MS/CH SFR<-mem	25+(3+1W)	25+(3+1W)	-	-	
MS/CH mem<-SFR	35+(3+1W)	35+1W	-	-	
CS INT	-	-	19	19	
RETRBI	-	-	12	12	
PR INT	-	-	53+5W	41+5W	
INTR	-	-	57+5W	45+5W	
BREAK	-	-	53+5W	41+5W	
BTCLR	29	29	-	-	
STOP	-	-	-	-	
DMA0 (S.S.)	20+1W-(1W-1) min=20	20+1W-(1W-1) min=20	20+1W-(1W-1) min=20	20+1W-(1W-1) min=20	
DMA0 (D.R.) mem<-I/O I/O<-mem	10+15n +(1W-1)n min=10+15n	10+15n +(1W-1)n min=10+15n	10+15n +(1W-1)n min=10+15n	10+15n +(1W-1)n min=10+15n	n= repetition no.
DMA0 (B.)	13+(12+2W)n	13+(12+2W)n	13+(12+2W)n	13+(12+2W)n	n= repetition no.
DMA0 (S.T.) mem<-I/O I/O<-mem	17+(1W-1) min=17	17+(1W-1) min=17	17+(1W-1) min=17	17+(1W-1) min=17	
DMA1 (S.S.)	20+1W-(1W-1) min=20	20+1W-(1W-1) min=20	20+1W-(1W-1) min=20	20+1W-(1W-1) min=20	
DMA1 (D.R.) mem<-I/O I/O<-mem	10+15n +(1W-1)n min=10+15n	10+15n +(1W-1)n min=10+15n	10+15n +(1W-1)n min=10+15n	10+15n +(1W-1)n min=10+15n	n= repetition no.
DMA1 (B.)	13+(12+2W)n	13+(12+2W)n	13+(12+2W)n	13+(12+2W)n	n= repetition no.
DMA1 (S.T.) mem<-I/O I/O<-mem	17+(1W-1) min=17	17+(1W-1) min=17	17+(1W-1) min=17	17+(1W-1) min=17	

V35 EA_Calculation Execution Clock

mem \ mod	00	clocks	01	clocks	10	clocks
000	BW+IX	3	BW+IX+disp8	3	BW+IX+disp16	4
001	BW+IY	3	BW+IY+disp8	3	BW+IY+disp16	4
010	BP+IX	3	BP+IX+disp8	3	BP+IX+disp16	4
011	BP+IY	3	BP+IY+disp8	3	BP+IY+disp16	4
100	IX	3	IX + disp8	3	IX + disp16	4
101	IY	3	IY + disp8	3	IY + disp16	4
110	direct	3	BP + disp8	3	BP + disp16	4
111	BW	3	BW + disp8	3	BW + disp16	4

Part 3.

μPD 70P322

User´s Manual

CHAPTER 1 INTRODUCTION

The uPD70P322 is a single-chip microcomputer provided by replacing internal mask ROM of uPD70322 (or V25™), uPD70332 (or V35™) with EPROM.

The V25 mode and V35 mode can be changed by using a given pin.

1.1 Features

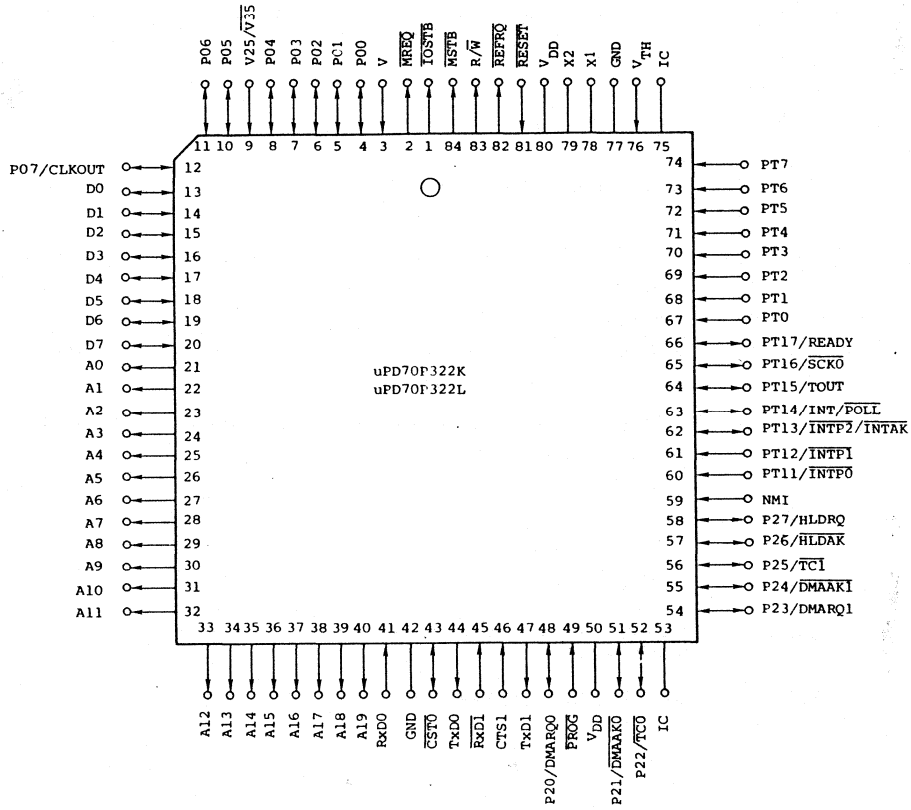
- o Internal EPROM
  - uPD70P322K: Reprogrammable (appropriate for system evaluation)
  - uPD70P322L: Only once programmable (appropriate for small quantity production)
- o V25 mode/V35 mode change function
  - V25 mode: Internal 16-bit architecture, external 8-bit data bus (uPD70332 equivalent)
  - V35 mode: Internal 16-bit architecture, external 16-bit data bus (uPD70332 equivalent)

Order information:

Order name	Package
uPD70P322K	84-pin LCC (ceramic leadless chip carrier) with window
uPD70P322L	84-pin PLCC (plastic leaded chip carrier)

1.2 Pin Connection (Top View)

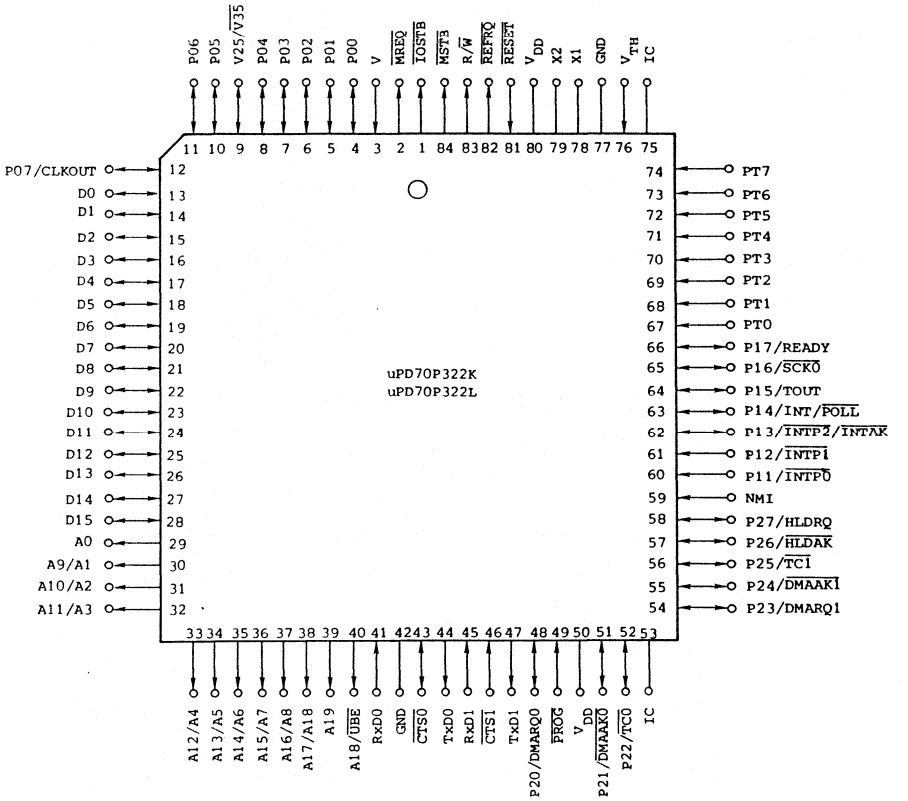
(1) V25 mode ($\overline{PROG}$ = H, V25/$\overline{V35}$ = H)



Caution 1: IC: Fix the pin to a high level with external pull-up resistor.

2: V: Connect the pin to V_{DD}.

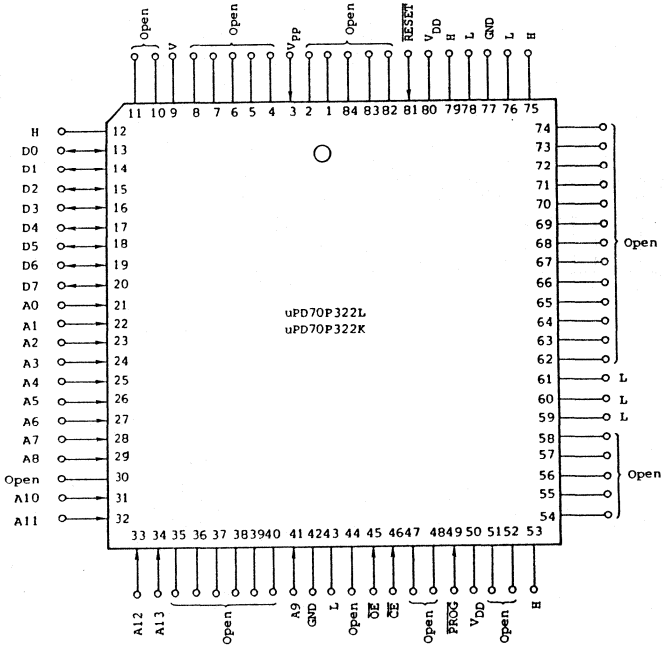
(2) V35 mode ($\overline{PROG}$ = H, V25/$\overline{V35}$ = L)



Caution 1: IC: Fix the pin to a high level with external pull-up resistor.

2: V: Connect the pin to VDD.

(3) EPROM programming mode (PROG = L)



Caution 1: H : Fix the pin to a high level with external pull-up resistor.

2: L : Fix the pin to a low level with external pull-up resistor.

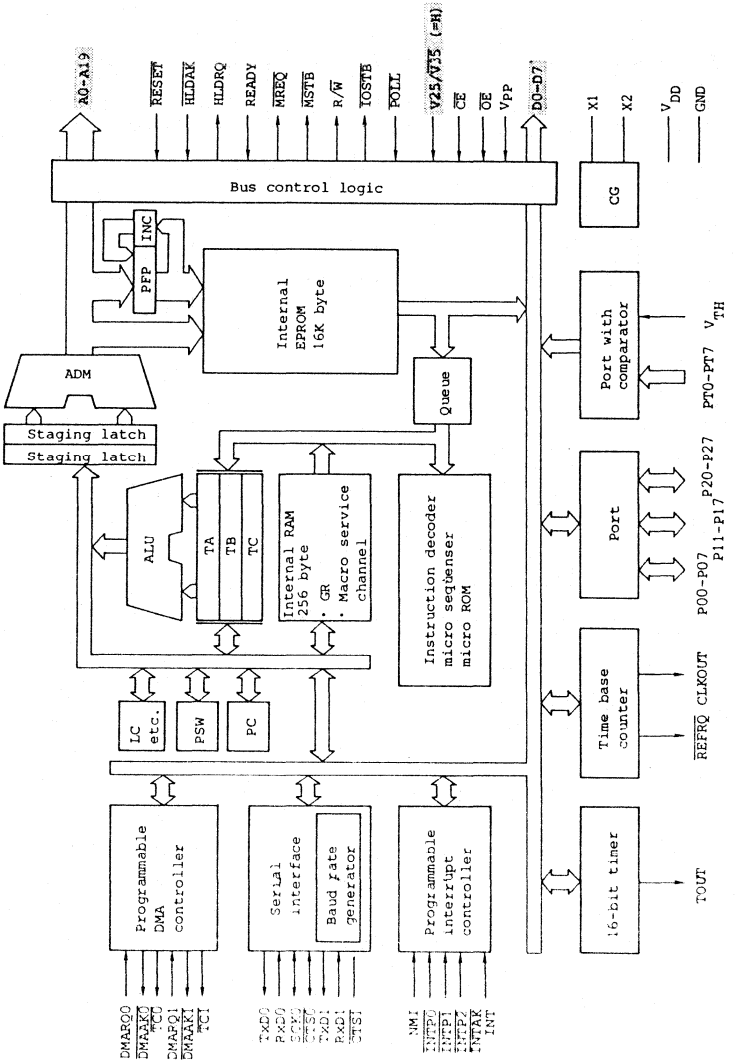
3: V : Connect the pin to V_{DD}.

4: Open: Do not connect the pin.

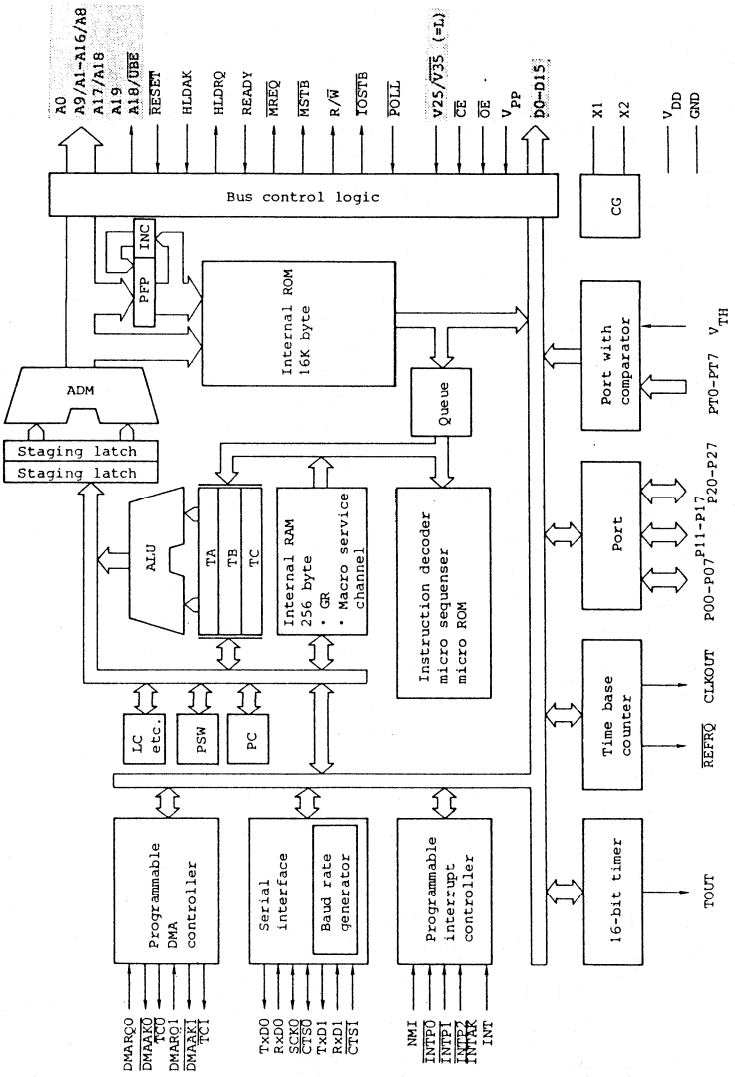
$\overline{IOSTB}$	I/O Strobe	HLDRQ	Hold Request
$\overline{MREQ}$	Memory Request	NMI	Non-Maskable Interrupt Request
P00-P07	Port 0	$\overline{INTP0}$-$\overline{INTP2}$	Interrupt From Peripherals
P11-P17	Port 1	$\overline{INTAK}$	Interrupt Acknowledge
P20-P27	Port 2	INT	Interrupt Request
V25/$\overline{V35}$	V25 Mode/V35 Mode	$\overline{POLL}$	Polling
CLKOUT	Clock Out	TOUT	Timer Output
D0-D15	Data Bus	$\overline{SCK0}$	Serial Clock
A0-A19	Address Bus	READY	Ready
RxD0, RxD1	Receive Data	PT0-PT7	Port T
$\overline{CTS0}$, $\overline{CTS1}$	Clear To Send	X1, X2	Crystal
TxD0, TxD1	Transfer Data	$\overline{RESET}$	Reset
DMARQ0, DMARQ1	DMA Request	$\overline{REFRQ}$	Refresh Request
$\overline{PROG}$	Program	R/$\overline{W}$	Read/Write
$\overline{DMAAK0}$, $\overline{DMAAK1}$	DMA Acknowledge	$\overline{MSTB}$	Memory Strobe
$\overline{TC0}$, $\overline{TC1}$	Terminal Count	$\overline{OE}$	Output Enable
$\overline{HLDAK}$	Hold Acknowledge	$\overline{CE}$	Chip Enable

1.3 Internal Block Diagram

(1) In V25 mode



(2) In V35 mode


A0
A9/A1-A16/A8
A17/A18
A19
A18/UBE
RESET
HLDAK
HLDRQ
READY
MREQ
MSTB
R/W
IOSTB
POLL
V25/V35 (=L)
CE
OE
VPP
D0-D15
X1
X2
VDD
GND
Bus control logic
CG
INC
PFP
Internal ROM
16K byte
ADM
Queue
Port with comparator
PT0-PT7
VTH
Staging latch
Staging latch
ALU
TA
TB
TC
Internal RAM
256 byte
· GR
· Macro service channel
Instruction decoder
micro sequenser
micro ROM
Port
P00-P07
P11-P17
P20-P27
LC etc.
PSW
PC
Time base counter
CLKOUT
REFRQ
Programmable DMA controller
Serial interface
Baud rate generator
Programmable intercupt controller
16-bit timer
TOUT
DMARQ0
DMAAK0
TC0
DMARQ1
DMAAK1
TC1
TxD0
RxD0
SCK0
CTS0
TxD1
RxD1
CTS1
NMI
INTP0
INTP1
INTP2
INTAK
INT

1.4 PIN FUNCTIONS

1.4.1 V25 Mode ($\overline{PROG}$ = H, V25/$\overline{V35}$ = H)

(1) Port Pins (PMC corresponding bit = 0)

Pin name	I/O	Also used for	Remarks
P00-P06	I/O	-	Input or output mode can be specified bitwise.
P07		CLKOUT	
NMI (P10)	I	-	This pin cannot be used as a general-purpose port pin (non-maskable interrupt)
P11	I	$\overline{INTP0}$	Input or output mode can be specified bitwise.
P12		$\overline{INTP1}$	
P13		$\overline{INTP2}$	
P14	I/O	$\overline{POLL}$/INT	
P15		TOUT	
P16		$\overline{SCK0}$	
P17		READY	
P20	I/O	DMARQ0	Input or output mode can be specified bitwise.
P21		$\overline{DMAAK0}$	
P22		$\overline{TC0}$	
P23		$\overline{DMARQ1}$	
P24		$\overline{DMAAK1}$	
P25		$\overline{TC1}$	
P26		$\overline{HLDAK}$	
P27		HLDRQ	
PT0-PT7	I	-	Comparator input

(2) Pins other than ports

Pin name	I/O	Function	Also used for
$\overline{PROG}$	I	EPROM programming mode setting. (=H: Pull-up resistor is required.)	-
V25/$\overline{V35}$	I	V25 or V35 mode selection. (=L: Pull-up resistor is required.)	
V_{TH}	I	Comparator reference voltage	
TxD0, TxD1	I	Serial data	
RxD0, RxD1	O	Serial data	
$\overline{CTS0}$	I/O	Asynchronous mode : Send instruction input I/O interface mode: Receive clock input/ output	
$\overline{CTS1}$	I	Send instruction input	
$\overline{RESET}$		Chip reset	
X1, X2	-	System clock oscillating crystal is connected. (Opposite phase clock can be input to X1, X2.)	
D0-D7	I/O	8-bit data bus	
A0-A19	O	Address bus	
$\overline{MREQ}$		Memory bus cycle start indication	
$\overline{MSTB}$		Memory access strobe	
R/$\overline{W}$		Read cycle/write cycle indication	
$\overline{REFRQ}$		DRAM refresh pulse	
$\overline{IOSTB}$		I/O access strobe	
V_{DD}	-	Positive power supply pin (both the V_{DD} bins are connected)	
GND		Ground pin (both the GND pins are connected)	

(Cont'd)

Pin name	I/O	Function	Also used for
CLKOUT	O	System clock	P07
$\overline{\text{INTP0}}$	I	External interrupt request	P11
$\overline{\text{INTP1}}$			P12
$\overline{\text{INTP2}}$			P13/$\overline{\text{INTAK}}$
$\overline{\text{INTAK}}$	O	Interrupt enable	P13/$\overline{\text{INTP2}}$
$\overline{\text{POLL}}$	I	Wait insertion	P14/INT
INT		External interrupt request	P14/$\overline{\text{POLL}}$
TOUT	O	Timer output	P15
$\overline{\text{SCK0}}$		Serial clock	P16
READY	I	External ready	P17
DMARQ0	I	DMA request	P20
DMARQ1			P23
$\overline{\text{DMAAK0}}$	O	DMA enable	P21
$\overline{\text{DMAAK1}}$			P24
$\overline{\text{TC0}}$		DMA termination instruction	P22
$\overline{\text{TC1}}$			P25
$\overline{\text{HLDAK}}$		Hold enable	P26
HLDRQ	I	Hold request	P27

1.4.2 V35 mode ($\overline{PROG}$=H, V25/$\overline{V35}$=L)

(1) Port pins (PMC corresponding bit=0)

Pin name	I/O	Also used for	Remarks
P00-P06	I/O	-	Input or output mode can be specified bitwise.
P07		CLKOUT	
NMI (P10)	I	-	This pin cannot be used as a general-purpose port pin (non-maskable interrupt)
P11	I	$\overline{INTP0}$	Input or output mode can be specified bitwise.
P12		$\overline{INTP1}$	
P13		$\overline{INTP2}$	
P14	I/O	$\overline{POLL}$/INT	
P15		TOUT	
P16		$\overline{SCK0}$	
P17		READY	
P20	I/O	DMARQ0	Input or output mode can be specified bitwise.
P21		$\overline{DMAAK0}$	
P22		$\overline{TC0}$	
P23		$\overline{DMARQ1}$	
P24		$\overline{DMAAK1}$	
P25		$\overline{TC1}$	
P26		$\overline{HLDAK}$	
P27		HLDRQ	
PT0-PT7	I	-	Comparator input

(2) Pins other than ports

Pin name	I/O	Function	Also used for
$\overline{PROG}$	I	EPROM programming mode setting. (=H: Pull-up resistor is required.)	
V25/$\overline{V35}$	I	V25 or V35 mode selection. (=L: Pull-up resistor is required.)	
V_{TH}	I	Comparator reference voltage	
TxD0, TxD1	O	Serial data	
RxD0, RxD1	I	Serial data	
$\overline{CTS0}$	I/O	Asynchronous mode : Send instruction input I/O interface mode: Receive clock input/ output	
$\overline{CTS1}$	I	Send instruction input	
$\overline{RESET}$		Chip reset	
X1, X2	-	System clock oscillating crystal is connected. (Opposite phase clock can be input to X1, X2.)	
D0-D15	I/O	16-bit data bus	
A0	O	Low-order memory bank is selected	
$\overline{UBE}$		High-order memory bank is selected.	A18
A1-A19		Address bus	
$\overline{MREQ}$		Memory or I/O bus cycle start indication, (high-order address strobe)	
$\overline{MSTB}$		Memory access strobe (low-order address strobe)	
R/$\overline{W}$		Read cycle/write cycle indication	
$\overline{REFRQ}$		DRAM refresh pulse	
$\overline{IOSTB}$		I/O access strobe (low-order address strobe)	
V_{DD}	-	Positive power supply pin (both the V_{DD} pins are connected)	
GND		Ground pin (both the GND pins are connected)	

(Cont'd)

Pin name	I/O	Function	Also used for
CLKOUT	O	System clock	P07
$\overline{\text{INTP0}}$	I	External interrupt request	P11
$\overline{\text{INTP1}}$			P12
$\overline{\text{INTP2}}$			P13/$\overline{\text{INTAK}}$
$\overline{\text{INTAK}}$	O	Interrupt enable	P13/$\overline{\text{INTP2}}$
$\overline{\text{POLL}}$	I	Wait insertion	P14/INT
INT		External interrupt request	P14/$\overline{\text{POLL}}$
TOUT	O	Timer output	P15
$\overline{\text{SCK0}}$		Serial clock	P16
READY	I	External ready	P17
DMARQ0	I	DMA request	P20
DMARQ1			P23
$\overline{\text{DMAAK0}}$	O	DMA enable	P21
$\overline{\text{DMAAK1}}$			P24
$\overline{\text{TC0}}$		DMA termination instruction	P22
$\overline{\text{TC1}}$			P25
$\overline{\text{HLDAK}}$		Hold enable	P26
HLDRQ	I	Hold request	P27

1.4.3 EPROM Programming Mode ($\overline{PROG}$=L)

Pin name	I/O	Function
D0-D7	I/O	8-bit data bus
A0-A13	I	Address bus
$\overline{PROG}$		EPROM programming mode setting (=L: Pull-down resistor is required)
$\overline{CE}$		Chip enable
$\overline{OE}$		Output enable
$\overline{RESET}$		EPROM mode setting
V_{PP}		Write power supply pin
V_{DD}	-	Positive power supply pin (both the GND pins are connected)
GND		Ground pin (both the GND pins are connected)

2. DIFFERENCES BETWEEN uPD70P322 AND uPD70322, uPD70332

Since the uPD70P322 is a product provided by replacing internal mask ROM of uPD70322, uPD70332 with EPROM (reprogrammable ROM), it has the same functions as uPD70322, uPD70332 except for EPROM specifications such as write/verify. Table 2-1 lists the differences between the uPD70P322 and uPD70322, uPD70332.

For detailed information on the CPU function and internal hardware, refer to documents such as the uPD70322, uPD70332 User's Mannual.

Table 2-1 Differences Between uPD70P322 and uPD70322, uPD70332

Item	uPD70P322	uPD70322, uPD70332
Internal program memory	EPROM	Mask ROM
EPROM programming pin	Included	Not included
Package	o 84-pin LCC with ceramic window (Note 1) o 84-pin PLCC (Note 2)	84-pin PLCC

Note 1: Reprogrammable (uPD70P322K)
2: Only once programmable (uPD70P322L)

3. EPROM PROGRAMMING

The uPD70P322 internal program memory is EPROM which is reprogrammable ROM. Normal operation and the EPROM programming mode are changed by changing the $\overline{RESET}$ and $\overline{PROG}$ input levels.

Caution 1: Put a protection seal on the erasion window of uPD70P322K except when EPROM is erased.

2: uPD70P322L which is programmable only once does not have an erasion window and cannot be erased with ultraviolet rays.

3.1 EPROM Programming Operation Mode

When the $\overline{RESET}$ and $\overline{PROG}$ pins are made low, EPROM programming (write/verify/read) is enabled on the uPD70P322. EPROM programming is set to the operation mode according to how the $\overline{CE}$ and $\overline{OE}$ pins are set as listed in Table 3-1.

Table 3-1 EPROM Programming Mode

Operation mode	$\overline{CE}$	$\overline{OE}$	V_{PP}	V_{DD}	$\overline{RESET}$	$\overline{PROG}$
Read	L	L	+5 V	+5 V	L	L
Output disable	L	H				
Standby	H	X				
Program	L	H	+12.5 V	+6 V		
Program verify	X	L				
Program inhibit	H	H				

X: L or H

3.2 Recommended Conditions for Unused Pins

Treat the pins assigned no function in the EPROM programming mode as listed in Table 3-2.

Table 3-2 Recommended Conditions for Unused Pins

Treatment	Pin No.
Pull up via resistor	12, 53, 75, 79
Pull down via resistor	43, 59, 61, 76, 78
Pull up or down via resistor	9
Do not connect	1, 2, 4-8, 10, 11, 30, 35-40, 47, 48, 51, 52, 54-58, 62-74, 82-84

3.3 EPROM Write Procedure

The EPROM write procedure is described below (high speed write can be made):

(1) Supply 6 V to the V_{DD} pin and +12.5 V to the V_{PP} pin.

(2) Supply initial address.

(3) Supply write data.

(4) Supply 1 ms program pulse (active low) to the $\overline{CE}$ pin.

(5) Perform verify mode operation. If data can be written normally, proceed to (7); if not, repeat (3) to (5). If data cannot be written normally after 25 repetitions, proceed to (6).

(6) Decide the device to be faulty. Stop write operation.

(7) Supply write data. Supply X (number of (3) - (5) repetitions) x 3 ms program pulse (additional write).

(8) Increment the address.

(9) Repeat (3) to (8) until the end address is reached.

Fig. 3-1 shows flowchart of the procedure.

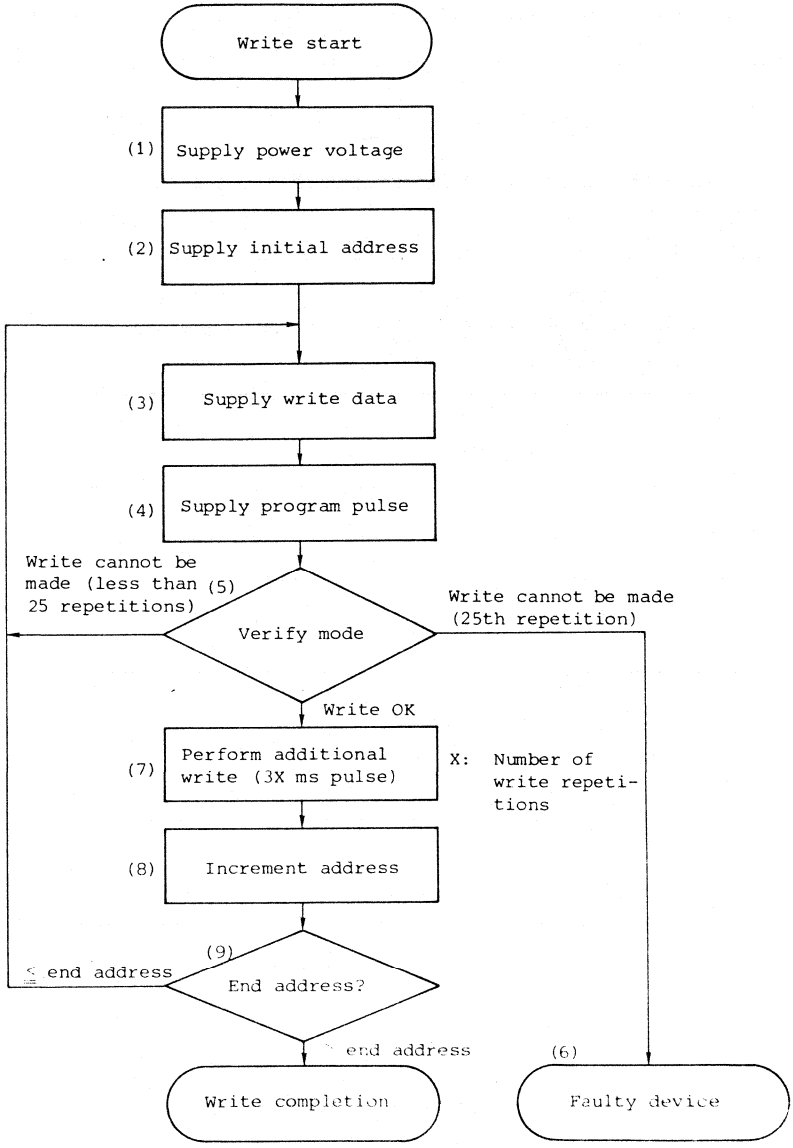


Fig. 3-1 Write Procedure Flowchart

Fig. 3-2 shows the timing of (1) - (7) in Fig. 3-1

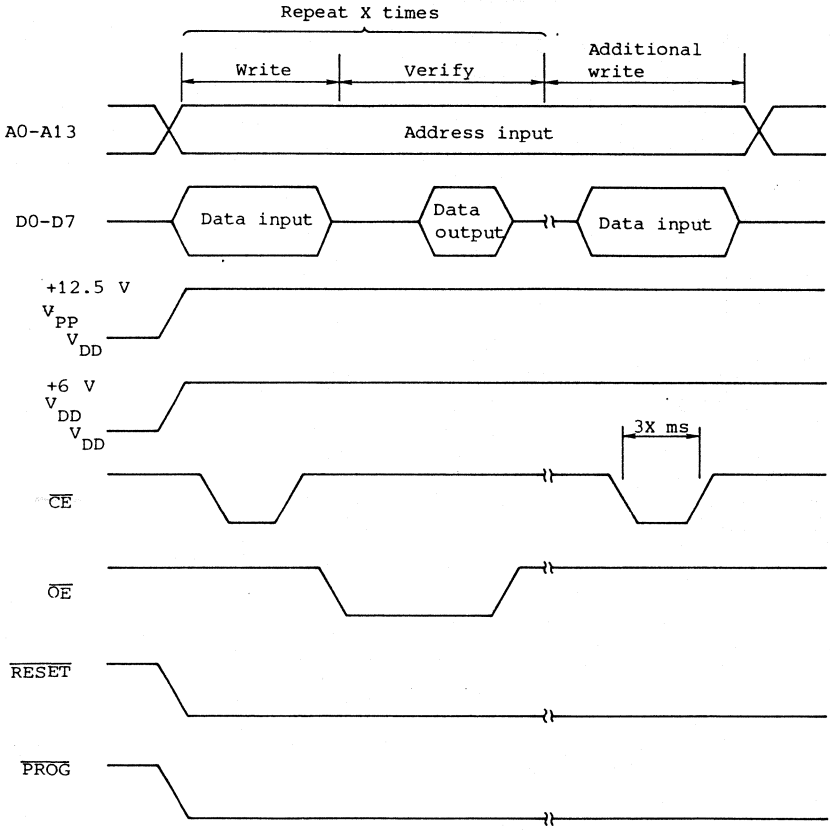


Fig. 3-2 EPROM Write and Verify Timing

3.4 EPROM Read Procedure

The EPROM contents can be read into the external data bus (D0-D7) according to the following procedure:

(1) Supply 5 V to the V_{DD} and V_{PP} pins.

(2) Input the address of the data to be read to the A0-A13 pins.

(3) Perform read mode operation.

(4) Output data to the D0-D7 pins.

Fig. 3-3 shows the timing of the procedure.

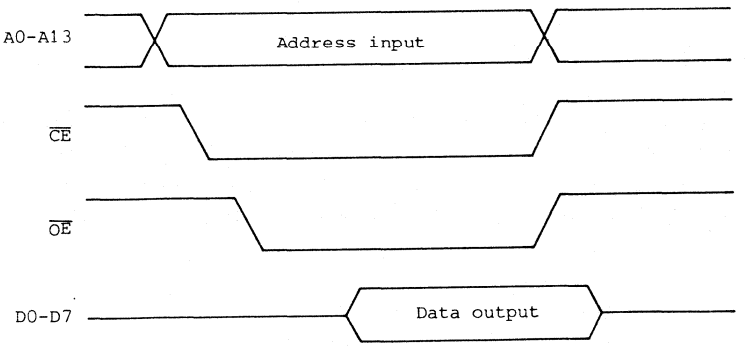


Fig. 3-3 EPROM Read Timing

4. ERASION CHARACTERISTICS (uPD70P322K ONLY)

The programmed data contents on the uPD70P322K can be erased (FFH) by applying rays whose wave length is shorter than about 400 nm.

To erase the uPD70P322K program memory contents, normally apply ultraviolet rays whose wave length is 254 nm. All irradiation amount required to erase the uPD70P322K completely is 15 $W \cdot s/cm^2$ (ultraviolet ray length x erasion time) in minimum. The erasion time is about 15 to 20 minutes (when 12,000 uW/cm^2 ultraviolet ray lamp is used). However, the erasion time may be prolonged due to ultraviolet ray lamp performance degradation, dirty package window, or another reason. For erasion, place the uPD70P322K at position within 2.5 cm from the ultraviolet ray lamp. If a filter is attached to the ultraviolet ray lamp, remove the filter before erasion.

5. ERASION WINDOW SEAL (uPD70P322K ONLY)

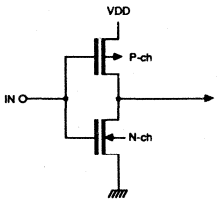
To prevent erroneous erasion of the EPROM contents by light source other than erasion lamp or prevent light from causing internal circuit other than EPROM to malfunction, put protection seal on the erasion window except when the EPROM contents are erased.

APPENDIX A
uPD 70P322
Input/Output Circuis

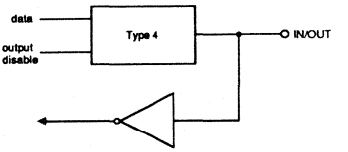
PIN	TYPE	PIN	TYPE
P00-P06	5	P20/DMARQ0	5
P07/CLKOUT	5	P21/$\overline{DMAAK0}$	5
NMI	2	P22/$\overline{TC0}$	5
P11/$\overline{INTP0}$	1	P23/DMARQ1	5
P12/$\overline{INTP1}$	1	P24/$\overline{DMAAK1}$	5
P13/INTP2/$\overline{INTAK}$	5	P25/$\overline{TC1}$	5
P14/INT/$\overline{FOLL}$	5	P26/$\overline{HLDAK}$	5
P15/TOUT	5	P27/HLDRQ	5
P16/SCK0	5	PT0-PT7	7
P17/READY	5		

PIN	TYPE	PIN	TYPE
TxD0	4	A9/A1-A16/A8	4
TxD1	4	A17/A18	4
RxD0	1	A19	4
RxD1	1	A18/$\overline{UBE}$	4
$\overline{CTS0}$	5	$\overline{MREQ}$	4
$\overline{CTS1}$	1	$\overline{MSTB}$	4
$\overline{REFRQ}$	4	R/$\overline{W}$	4
$\overline{RESET}$	2	$\overline{IOSTB}$	4
D0-D15	5	$\overline{PROM}$	1
A0	4	V25/$\overline{V35}$	1

Type 1

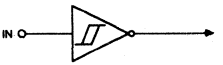


Type 5



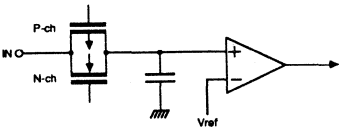
(I/O power circuit consit of Push-Pull output (Type 4) and Input Buffer (Type 1)

Type 2



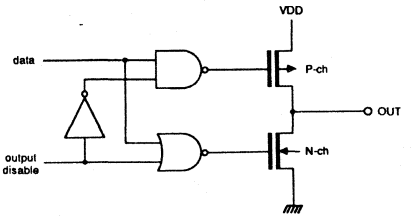
(Schmitt trigger Input which has hysterisys charcateristic)

Type 7



(Threshhold voltage)

Type 4



(Push-Pull output which can make output high impedance both P-ch ,N-ch off)

PART 4.

μPD79011

SINGLE CHIP MICROCOMPUTER (V25)

WITH

REAL TIME OPERATING SYSTEM

USER'S MANUAL

CHARACTERISTICS

μPD79011 is a 16-bit CMOS single-chip microcomputer which has a kernel of Operating System (OS) in the internal 16KB ROM of the V25.

- **Realtime multitask environment**
  This OS provides realtime multitask environment which requires a quick response on the various external events. To be usable as an essential kernel of an embedded control system such as process control and communication control equipments as well as data processing field such as personal computer and typewriter.

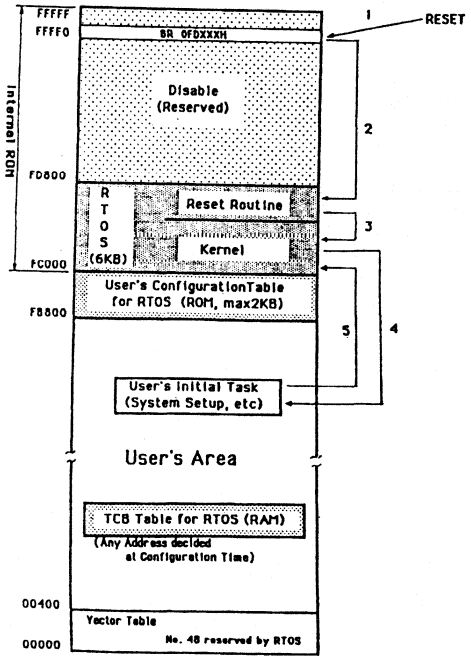
- **More simplified specifications, higher processing speed**
  This OS which is called as RTOS (Real Time Operating System) adopts the simplified specifications which can minimize processing time of each system call, and also to make it be possible to minimize the overhead by achieving high-speed task switching operation through utilization of built-in register bank.
  Overhead (processing time for system call) range is 41 - 123 usec on bank resident task basis, and 41 - 294 usec on the basis including not resident task (at 8MHz clock rate of the μPD79011).

- **Programming in high level language C**
  In addition to assembler language, program development is possible by using high level language C. System call can be issued as a form of function call of language C.

1. Memory map

Please refer to fig. 1.
uPD79011 is available to access memory space of 1 MB. The highest position of 16KB to be internal ROM space where RTOS is built-in.
Configuration information of RTOS to be loaded at the space between such address as 2 KB from FB800 to FBFFF so that it can start at address FFFF0 by system reset, and also control to be performed on each user's task definitively after completion of set of configuration information.
The space between such address as 1 K from 00000 to 003FF is the area to be reserved for vector address so that user's task to be available at the space of address starting from 00400 through to FB7FF.

FIG. 1 Memory Map



2. Basic function of RTOS

RTOS is able to handle 64 pieces of task in total.
Designable task numbers are from 0 through to 63.

Priority level is 64 level, and task number and priority level is mutually corresponding at the ratio of 1 : 1.
For instance, task priority of task number 3 is 3.
The highest priority is level 0 and the lowest priority is level 63.
Scheduling method for priority basis to be adopted.
RTOS to select the highest priority task upon request, and to make it be possible to run.

uPD79011 has internally 8 register banks, and perform the high-speed task switching operation through function of these register bank switch.

RTOS to monopolize bank 7 properly so that designation for tasks are on the remaining bank from 0 through to 6.
Tasks designated as 0 - 5 (task number 0-5) are resident task which are able of high-speed switching due to unnecessity to save or resave task information at task switching time.
The remaining tasks (task number 6-63) to be operated only on bank 6 so that necessary to reserve a little time to save or resave task information at task switching time accordingly.

2.1 Task states and state transition

State transition shall be referred as per Fig. 2.1.
States of task operation shall be classified into the below-mentioned 4 type (6 types more precisely).
All tasks to be in READY or DORMANT at configuration time.

(1) RUN

Right to utilize CPU shall be given, and under running operation state.
Only one to be available at a certain period in the system.

(2) READY

State available for operation.
All preparation necessary for RUN to ready but still in waiting because of the higher priority task under operation.

(3) WAIT

Condition in waiting for the forthcoming event.
READY condition to come over when release of all the waiting factors shall be realized. More precisely to be classified into the following 3 stages :

(3-1) WAIT

Waiting state resulting from such system call as requirement for resource, which is to be one of the followings : resource waiting by semaphore, message waiting (2 cases of mailbox or direct communication) and interrupt waiting.

(3-2) SUSPEND

Suspend state by system call SUS_TSK forcibly.
Possible to transit directly from RUN task to this state too.
To keep waiting until resume requirement by system call.

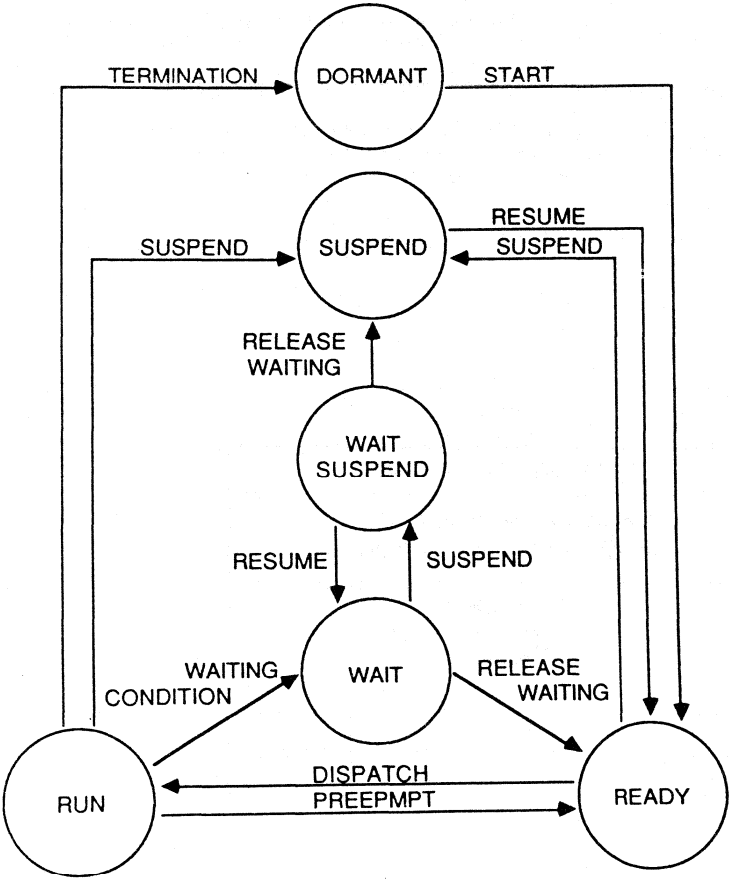
(3-3) WAIT SUSPEND

State of WAIT and Suspend.
Transition to WAIT by release of SUSPEND under system call RSM_TSK, and transition to SUSPEND by release of WAIT.

(4) DORMANT

Only one task to be automatically into READY state at configuration time but all the other tasks to be kept in DORMANT.
Transition to DORMANT to be performed by EXI_TSK to be issued by task under RUN condition.

FIG. 2.1 Task States



2.2 Task operation

Such task operations are available as termination, start, suspend, resume and restart address set.

Task state transition from DORMANT to READY to be performed by system call STA_TSK.

From RUN to DORMANT upon task termination by EXI_TSK.

Task state come into SUSPEND by SUS_TSK, which can be released by RES_TSK reopening.

Possible also to set up restart address by system call SET_ADR.

This function to be available under incorporation with system call RSM_INT which shall complete interrupt handler. (Please refer to 2.5 Interrupt control)

FIG. 2.2.1 RTOS System Call Processing

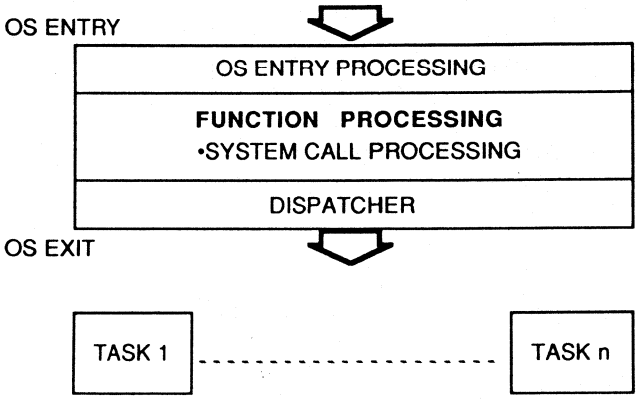
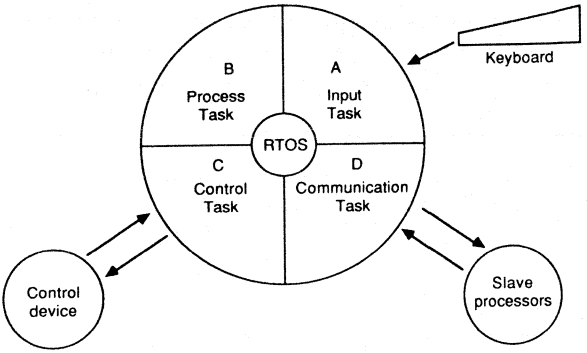
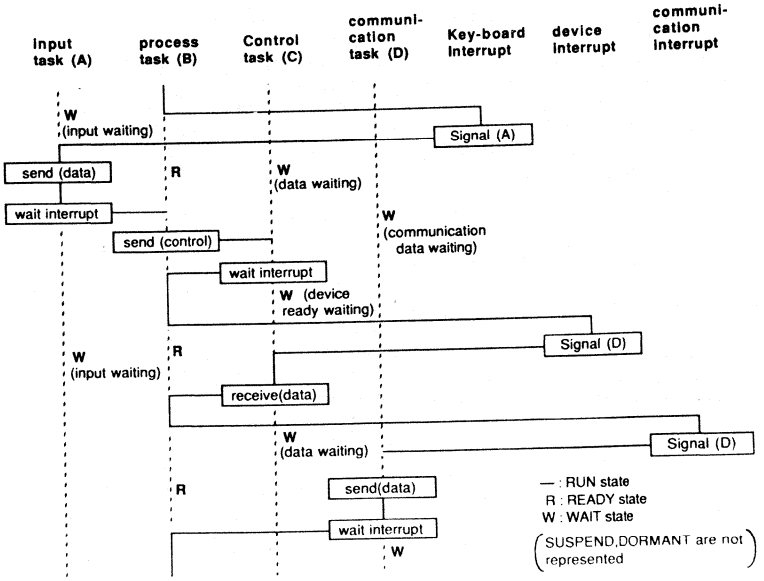


FIG. 2.2.2 Typical Example of RTOS Multi Tasking





2.3 Synchronization / Communication

Two functions such as waiting and mutual exclusion.

Waiting : To suspend the other task operation until termination of a certain task under operation.

Mutual exclusion : To prohibit simultaneous access by plural tasks to the common resource (memory, device etc.).

For realization of synchronization / communication, semaphore to be for synchronization, and mailbox and direct communication for communication respectively.

(1) Semaphore

Usable for waiting and mutual exclusion.
To fix requirement resource number of P-instruction and V-instruction against semaphore at 1.
In case of P-instruction, 2 different system calls such as (REQ_RSC) which perform transition to WAIT state upon non-fulfilment of requirement, and (POL_RSC) which inform only the fact of non-fulfilment of requirement, are supported.
V-instruction to release the resource (REL_RSC).
Fig. 2.3.1 is an example of mutual exclusion case, which shows a process to avoid the collision between WRITE and READ resulting from control exchange between task A and tack B by interrupt under the common memory :

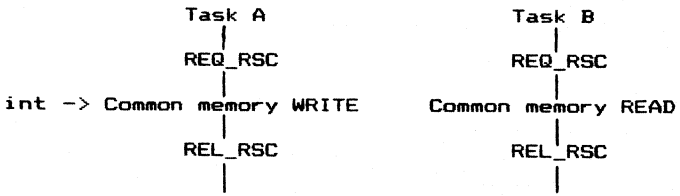
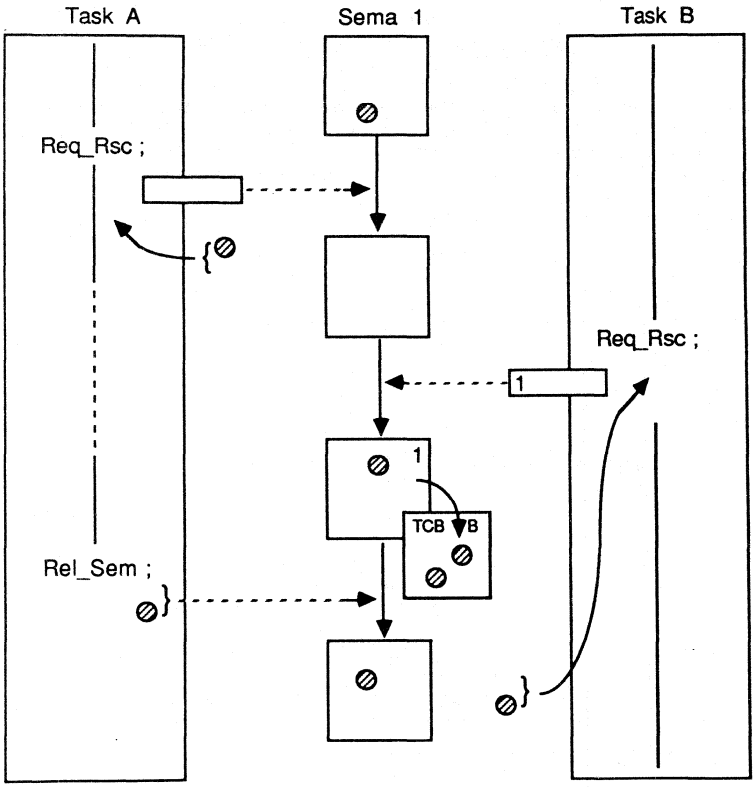


FIG. 2.3.1 Mutual exclusion function

FIG. 2.3.2 RTOS Semaphore Example:



(2) Mailbox and Direct communication

Mailbox has a function to send / receive the message from task to task, which consists of task queue for message receiving and message queue for send.

Direct communication is used to send / receive the message from a task to a designated task without usage of mailbox.

Two different kinds of system calls group i. e. (RCV_MSG, RCV_DIR) which make a transition to WAIT state upon non-receiving message, and another (POL_MSG, POL_DIR) which inform only the fact of non-receiving message, on both cases of mailbox and direct communication respectively.

FIG. 2.3.3 RTOS Mailbox Example 1:

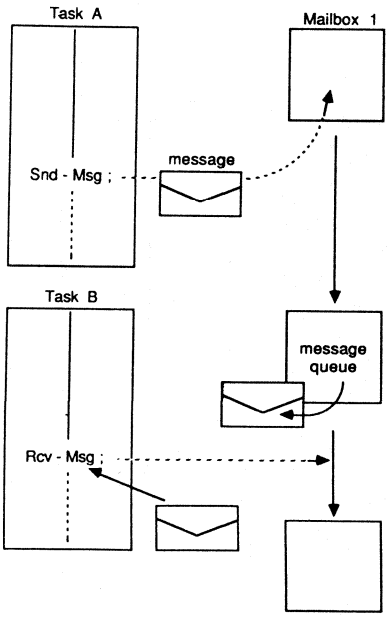


FIG. 2.3.4 RTOS Mailbox Example 2:

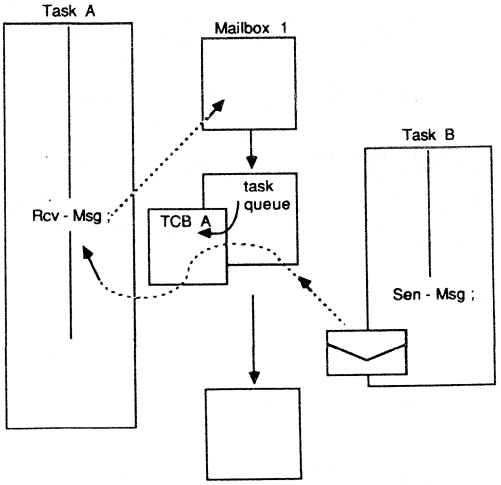


FIG 2.3.5 RTOS Direct CommunicationExample

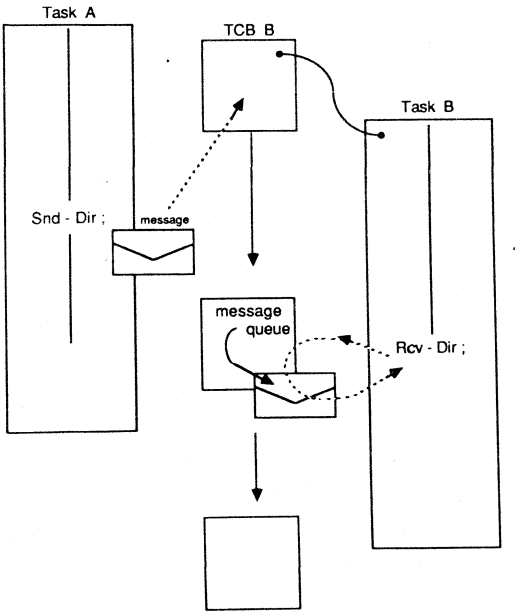
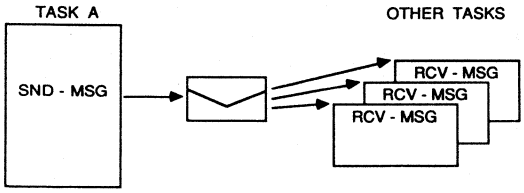
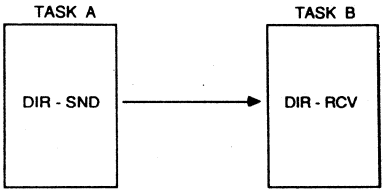


FIG 2.3.6 Differencebetween Mailbox and Direct Communication



MAILBOX



DIRECT COMMUNICATION

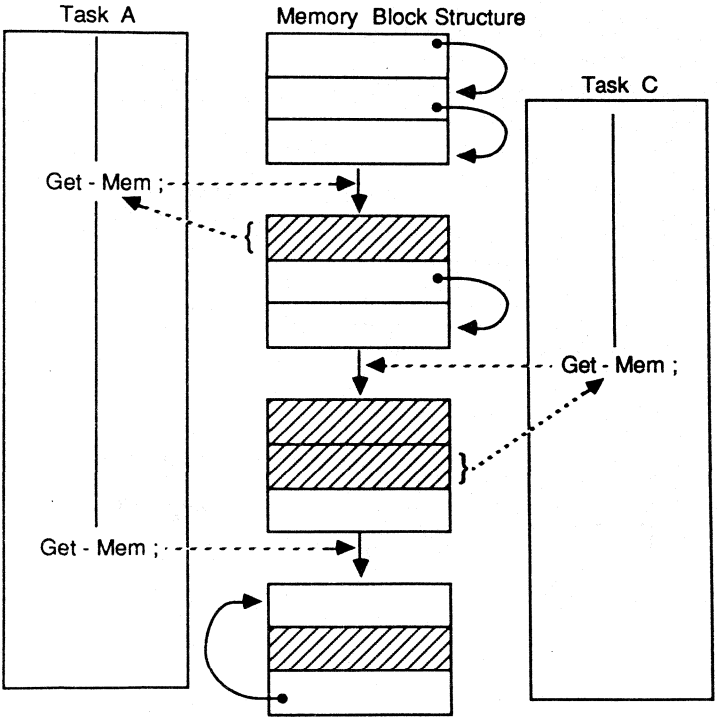
2.4 Memory management

To keep the buffer and area for message communication, and to obtain and return memory block by system call actively.

Memory block size shall be designated and fixed at configuration time.

No transition of task state upon no obtaining of memory block when system call GET_MEM shall be issued, but only return of error code.

FIG. 2.4 Fixed Type Memory Block Management



2.5 Interrupt control

To support interrupt process routine (interrupt handler) to be operated internal / external interrupt.
Also including the following functions :
To set up interrupt handler, to return from interrupt handler, permission / prohibition of interrupt and waiting for interrupt.

Setting up or releasing interrupt handler shall be possible by system call DEF_INT, actually, through matching interrupt level of controller uPD71059 with top address of interrupt handler, or matching interrupt reguest from an internal peripheral with top address of interrupt handler.
Permission for interrupt shall be possible by system call DIS_INT, and prohibition by ENA_INT.
2 system calls, SIG_INT and RES_INT, to transfer control to task after termination of interrupt handler.
SIG_INT can select and invocate one task under WAIT state, which resulted from WAI_INT, immediately after termination of interrupt handler.
Please refer to fig. 2.5.1 as an example of control transition onto task by SIG_INT.
In this case, task B is under interrupt WAIT state by system call WAI_INT, and interrupt comes at position of int during task A's running period so that SIG_INT (B) shall be issued at the end of interrupt handler.
When task A's priority is higher than task B's one, control returns to the position of task A's interrupt occurrence point, meanwhile when task B's priority is higher than task A's one, control shall be transferred to task B.

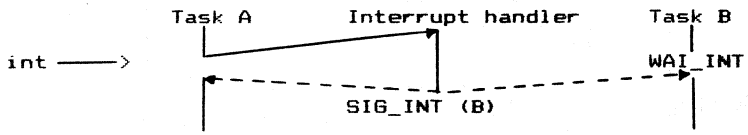


Fig. 2.5.1 Example of SIG_INT function

Fig. 2.5.2 is an example of system call RES_INT for return from interrupt handler.

This system call shall be usable in combination with system call for restart address setting (SET_ADR).

When SET_ADR shall be issued by the task which was interrupted by interrupt handler to issue RES_INT, control shall be transferred onto restart address which was set up, not onto the address at interrupt position.

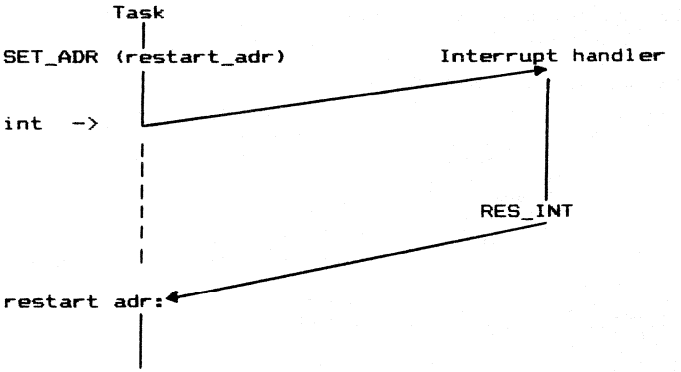
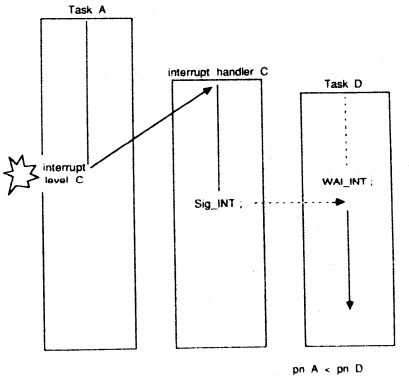


Fig. 2.5.2 Example of RES_INT

FIG 2.5.3 Interrupt Processing by Interrupt Task



3. Development process

Fig. 3 showing the chart to complete user's application system.

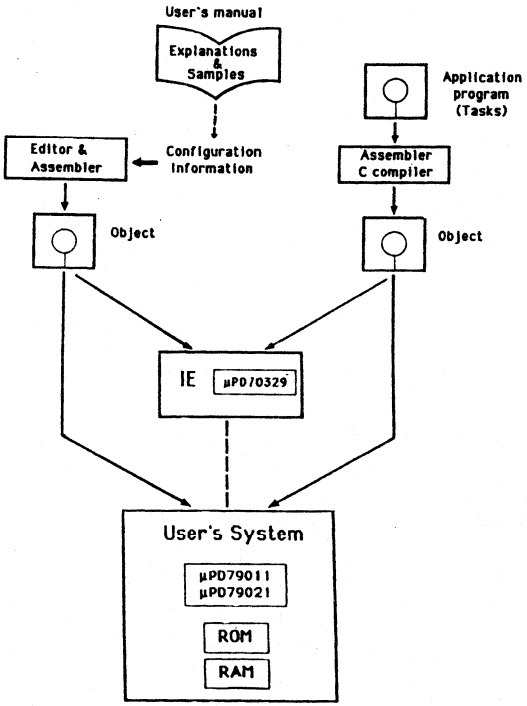
As the first step, making up the application task on the development machine, and secondarily to assemble or compile it and link to the necessary library.

Meanwhile making up configuration informations source program in assembly language on the machine by using editor, and to assemble it.

Configuration information shall be down-loaded to address FB800, and the other application tasks shall be down-loaded onto each load address by IE-79011 (In Circuit Emulator).

Finally completing the system through the process of debugging under multi task environment by adopting IE.

FIG. 3 User's Application System



4. System call prospectus

Here is the description of each system call of RTOS in the following order :

- System call name
- System call number
- Function
- Parameter
- Condition code
- C format
- Explanation

The definition of the above items are as follows :

- System call name :
  Name of each system call
- System call number :
  System call ID number corresponding to each system call
- Function :
  Summarized explanation of function of each system call
- Parameter :
  Significants of each system call
- Condition code :
  Function results to be expected from system call operation and to be returned in function value in case of C-language, meanwhile in the form of return parameter on register AW in case of assembler.
  If some value is returned, this value to be understood as function value instead of Conditon code.
- C format :
  Description form by C-language
- Explanation :
  Commentary of system call function

Please refer to reserved vector table and error code table which shall be listed at the end of chapters.

– Interface with C-language

High-level language to support for application program is C-language, which can utilize COMPACT MODEL and LARGE MODEL. Assembler routine shall be necessary for getting interface with OS in case of system call by C-language. Refer to the next item (Interface with Assembler) with regard to assembler routine.

System call form in C-language is as follows :

cc = <name>([<parameter>]..);

cc : Function value
name : System call name in 7 letters
parameter : Input parameter

The followings shall be adopted as data type and size of each parameter :

short : 16 bit integral number
pointer : 32 bit integral number

– Interface with Assembler

In case of direct system call by Assembler, internal system of RTOS is installed corresponding to C-language so that all the parameters are passed via stack.

System call program and stack condition are as below-mentioned.

In case of pointer type parameter, off-set value shall be stacked at the lower part of stack, and segment value at the upper part.

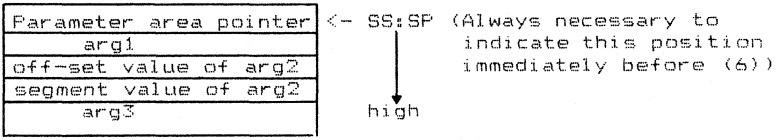
```
cc = <name>(arg1,arg2,arg3);

  short     arg1;
pointer     arg2;
  short     arg3;
```

The program based on the above conditions are as follows :

(1) Stacking parameter 3
(2) Stacking parameter 2
(3) Stacking parameter 1
(4) Stacking pointer for parameter area
(5) Setting system call number in AW register
(6) Inter-segment calling RTOS_ENTRY (F000:C000 address)

Stacking conditions are as follows :



Stack condition should be as the above-mentioned immediately before inter-segment calling RTOS_ENTRY.

Also necessary to inter-segment call even in case of RTOS_ENTRY address shall be located in the same segment.

However, not necessary to stack parameter area pointer in case of system call without parameter.

Only in case of system call SIG_INT and RES_INT, program shall be changed, which shall be referred to each chapter.

4.1 Task operation

System calls relating to task operation are as follows :

(1) **STA_TSK** starting
(2) **EXI_TSK** completion
(3) **SUS_TSK** suspend
(4) **RSM_TSK** reopen
(5) **SET_ADR** setting restart address

4.1.1 STA_TSK (START TASK)

System call number 0

Function

To start task and transition from DORMANT to READY

Parameter

Task number (0 - 63) (Word type)

Condition code

Normal termination	0	E_OK
Corresponding task under non-DORMANT condition	1	E_DMT

C format

```
cc = STA_TSK(task_no);
short           task_no ;
```

Explanation

Impossible to issue this system call against task not to be in DORMANT state (E_DMT).

Running action of started task shall be different in case of original start and another case of restart after completion once.

In case of original start, starting from initial stage address and simultaneously such values as stack pointer, stack segment and data segment shall be set up on the value of initial stage.

The other register values are infinitive.

In case of restart, also starting from initial stage address, however, all the values such as stack pointer, stack segment, data segment and other register values shall remain unchanged from the values immediately before system call EXI_TSK issuing.

4.1.2 EXI_TSK (EXIT TASK)

System call number 1

Function

Terminating task and transition from RUN to DORMANT

Parameter

nil

Condition code

Normal termination	0	E_OK

C format

```
cc = EXI_TSK();
```

Explanation

In case of task which was transited once onto DORMANT shall be restarted by system call STA_TSK, start address returns to initial stage address, but the other register values remain unchanged from ones at EXI_TSK issuing time.

The vales of stack pointer, stack segment and data segment therefore, are not always similar to ones at configuration time.

4.1.3 SUS_TSK (SUSPEND TASK)

System call number 2

Function

Suspending task to be in SUSPEND state

Parameter

Task number (0 - 63) (Word type)

Condition code

Normal termination	0	E_OK
Corresponding task under DORMANT	1	E_DMT
Corresponding task under SUSPEND	2	E_SUS

C format

```
cc = SUS_TSK(task_no);
short           task_no ;
```

Explanation

Impossible to issue this system call against task not to be in DORMANT state (E_DMT).
Also impossible to issue against task to be already in SUSPEND state (E_SUS).
Transition onto WAIT_SUSPEND state by issuing against task to be in WAIT state.

4.1.4 RSM_TSK (RESUME TASK)

System call number 3

Function

Restart task which is in SUSPEND state

Parameter

Task number (0 - 63) (Word type)

Condition code

Normal termination	0	E_OK
Corresponding task under DORMANT	1	E_DMT
Corresponding task under non-SUSPEND	2	E_SUS

C format

```
cc = RSM_TSK(task_no);
short            task_no ;
```

Explanation

Impossible to issue this system call against task which is in DORMANT state (E_DMT).
Also impossible to issue against task which is not in SUSPEND state (E_SUS).
Only SUSPEND state shall be released by issuing against task which is in WAIT_SUSPEND state, and transition onto WAIT state accordingly.

4.1.5 SET_ADR (SET RESTART ADDRESS)

System call number 4

Function

Setting task restart address

Parameter

Task restart address (Pointer type)

Condition code

Normal termination	0	E_OK

C format

```
cc = SET_ADR(restart_adr);
pointer          restart_adr ;
```

Explanation

This system call is usable incorporated with RES_INT. On return from interrupt handler, control shall return to designated restart address by system call RET_INT.

Also possible to issue plural times and the finally issued restart address shall be effective value.

Possible to release restart address which was set up by designating restart address at 0.

4.2 Synchronization / Communication

System call relating to Synchronization / Communication are as follows :

(1) **REQ_RSC**
(2) **POL_RSC**
(3) **REL_RSC**
(4) **RCV_MSG**
(5) **POL_MSG**
(6) **SND_MSG**
(7) **RCV_DIR**
(8) **POL_DIR**
(9) **SND_DIR**

4.2.1 REQ_RSC (REQUEST RESOURCE)

System call number 5

Function

Resource requirement for semaphore

Parameter

Semaphore number (0 - designated number) (Word type)

Condition code

Normal termination	0	E_OK

C format

```
cc = REQ_RSC(semaphore_no);
short           semaphore_no ;
```

Explanation

Transition onto WAIT in case of resource number to be 0. In case of resource number to be more than 1, resource number shall be decreased by 1.

Wait state task by issuing this system call shall be queued at the tail of task waiting queue of semaphore with regardless of task priority.

4.2.2 POL_RSC (POLL RESOURCE)

System call number 6

Function

Resource requirement for semaphore

Parameter

Semaphore number (0 - designated number) (Word type)

Condition code

Normal termination	0	E_OK
Resource number 0	6	E_RSC

C format

```
cc = POL_RSC(semaphore_no);
short              semaphore_no ;
```

Explanation

In case of resource number at 0, error code E_RSC returns and no transition onto WAIT state.
In case of resource number more than 1, resource number shall be decreased by 1.
This system call shall be usable for judging whether resource is still left in semaphore or not.
Only reflection of resource existence or non-existence without transition onto WAIT state.

4.2.3 REL_RSC (RELASE RESOURCE)

System call number 7

Function

To release resource to semaphore

Parameter

Semaphore number (0 - designated number) (Word type)

Condition code

Normal termination	0	E_OK

C format

```
cc = REL_RSC(semaphore_no);
short              semaphore_no ;
```

Explanation

Resource number shall be increased 1 more.
In case of waiting task to exist, Task which came into WAIT at the first of all shall be selected and released.
To set the initial value with semaphore resource number at resetting time, however, no error comes even in case of exceeding the initial value by this system call.
Also no error resulting from automatic clearance into 0 in case of exceeding OFFFFH.

4.2.4 RCV_MSG (RECEIVE MESSAGE)

System call number 8

Function

Requirement for receiving message to mailbox

Parameter

Mailbox number (0 - designated number) (Word type)

Condition code

Nil

C format

```
cc = RCV_MSG(mailbox_no);
short              mailbox_no ;
```

Explanation

In case of message to exist, one message which came into mailbox at the first of all shall be selected and the segment value of its message area shall be returned as function value.

In case of message not to exist, transition onto WAIT state so that queuing at the tail end of task waiting queue of mailbox.

4.2.5 POL_MSG (POLL MESSAGE)

System call number 9

Function

Requirement for receiving message to mailbox

Parameter

Mailbox number (0 - designated number) (Word type)

Condition code

Non message	7	E_MSG

C format

```
cc = POL_MSG(mailbox_no);
short          mailbox_no ;
```

Explanation

In case of message to exist, one message which came into mailbox at earliest shall be selected and the segment value of its message area shall be returned as function value.

In case of message not to exist, no transition onto WAIT state and returning error code E_MSG.

4.2.6 SND_MSG (SEND MESSAGE)

System call number 10

Function

Message sending to mailbox

Parameter

Mailbox number (0 - designated number) (Word type)
Sending message area segment (Word type)

Condition code

Normal termination	0	E_OK

C format

```
cc = SND_MSG(mailbox_no, msg_seg);
short           mailbox_no ;
short           msg_seg ;
```

Explanation

In case of waiting task to exist, sending message area segment value which was given to task by parameter shall be returned after release of WAIT state.
In case of waiting task not to exist, message queuing shall be done into mailbox.
Message queuing by FIFO is similar as task case.

4.2.7 RCV_DIR (DIRECT RECEIVE MESSAGE)

System call number 11

Function

Receiving message to be sent to own task

Parameter

Nil

Condition code

Nil

C format

```
cc = RCV_DIR();
```

Explanation

In case of message not to exist yet, transition shall be done onto WAIT state.

In case of message to exist, its message area segment value shall be returned as function value.

4.2.8 POL_DIR (POLL DIRECT MESSAGE)

System call number 12

Function

Receiving message to be sent to own task

Parameter

Nil

Condition code

Non message	7	E_MSG

C format

```
cc = POL_DIR();
```

Explanation

In case of message not to arrive yet, no transition onto WAIT state and information to be given by error code E_MSG.

In case of message to exist, message area segment value shall be returned.

4.2.9 SND_DIR (SEND DIRECT MESSAGE)

System call number 13

Function

Message sending to designated task

Parameter

Task number (0 - 63) (Word type)
Sending message area segment (Word type)

Condition code

Normal termination	0	E_OK

C format

```
cc = SND_DIR(task_no, msg_seg);
short          task_no ;
short          msg_seg ;
```

Explanation

In case of designated task to be in direct message waiting, message area segment value shall be returned to the task after release of WAIT state.

In case of task not to be in direct message waiting, message queuing shall be done to the designated task by FIFO.

4.3 Memory control

System calls relating to memory management are as follows :

(1) **GET_MEM**
(2) **REL_MEM**

4.3.1 GET_MEM (GET MEMORY)

System call number 14

Function

Getting memory block

Parameter

Nil

Condition code

Non memory block	3 E_BLK

C format

```
cc = GET_MEM();
```

Explanation

Obtained memory block shall be usable for message area for inter-task communication.
Memory block size is determined as fixed at system configuration time.
In case of memory block to exist, segment value of memory block shall be returned, and E_BLK return in case of memory not to exist, but as function value in both cases, no transition onto WAIT state shall be done.

4.3.2 REL_MEM (RELEASE MEMORY)

System call number 15

Function

Release memory block

Parameter

Memory block segment to be released (Word type)

Condition code

Normal termination	0	E_OK

C format

```
cc = REL_MEM(mem_blk);
short            mem_blk ;
```

Explanation

Impossible to release in case of memory block queuing in mailbox, and task waiting queue as memory block.
Always necessary to release after receiving.

4.4 Time control

System calls relating to time management are follows :

(1) **GET_TIM**
(2) **SET_TIM**

4.4.1 GET_TIM (GET TIME)

System call number 16

Function

Reading system time

Parameter

Pointer for system time housing area (Pointer type)

Condition code

Normal termination	0 E_OK

C format

```
cc = GET_TIM(time_ptr);
pointer          time_ptr ;
```

Explanation

System time consisting of 3 words data, and low data is located at the lower part of address and high data at higher part of address, as follows :

4.4.2 SET_TIM (SET TIME)

System call number 17

Function

Setting system time

Parameter

Pointer for housing time (Pointer type)

Condition code

Normal termination	0	E_OK

C format

```
cc = SET_TIM(time_ptr);
pointer          time_ptr ;
```

Explanation

Timer to start counting action immediately after system configuration because of interrupt source of system time be adopting time base counter.

Necessary to take it into consideration on handling time data that time to be taken from system call issuing until next timer interrupt should involve time aberration exactly corresponding to the interval time designated at configuration time.

4.5 Interrupt control

System calls relating to interrupt management are as follows :

(1) **DEF_INT**
(2) **SIG_INT**
(3) **WAI_INT**
(4) **CAN_INT**
(5) **DIS_INT**
(6) **ENA_INT**
(7) **RES_INT**

4.5.1 DEF_INT (DEFINE INTERRUPT HANDLER)

System call number 18

Function

Setting start address of handler

Parameter

Device number (Interrupt level or vector type)
(Word type)
Start address of handler (Pointer type)

Condition code

Normal termination	0	E OK
device number error	4	E DVN
No definition of port address	5	E SYS

C format

```
cc = DEF_INT(device no, start adr);
short          device no ;
pointer        start adr ;
```

Explanation

This system call is used to set the start address of the interrupt handler regarding two cases:

Case A: Interrupt level (Interrupt controller uPD71059)
Case B: Vector type interrupt response of the V25

Setting the start address to 0 means a release of setting the address.

Regarding case A (uPD71059) the RTOS performs the following two items:

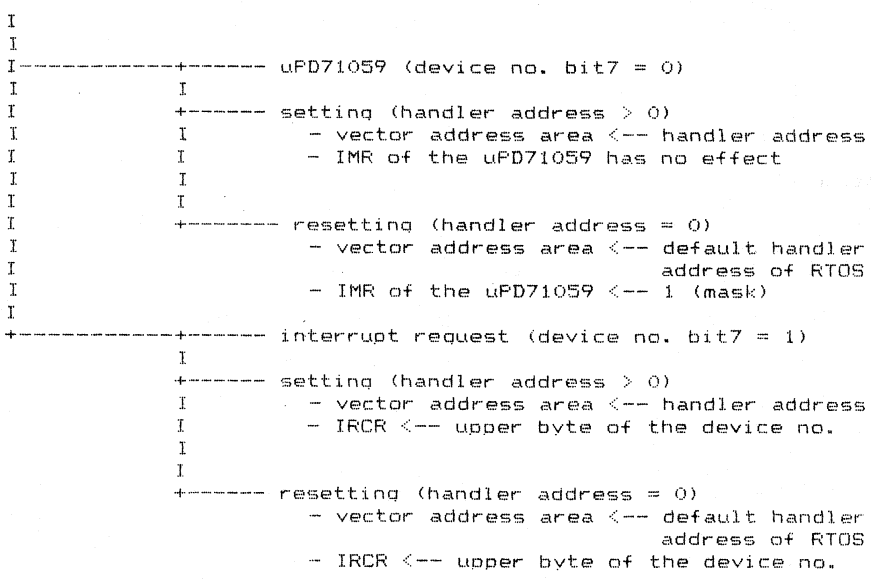
- the bit of IMR (Interrupt Mask Register of uPD71059) corresponding to the level is set in order to mask the interrupt.

- Release of address setting.

For case B it is possible to operate with bit 6 (mask bit of the IRCR=interrupt request control register) by setting the value of the IRCR simultaneously with setting address release.

In case except for start address of handler to be at 0, it means address setting. In case of address setting, to set address without usage of IMR operation under interrupt level of uPD71059 to be designated.

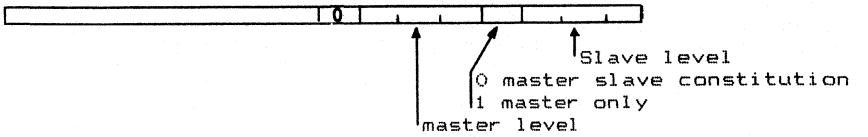
DEF_INT structure:



In case of vector type of interrupt request to be designated, as same as case of release, possible to operate interrupt mask by bit 6 (mask bit) of interrupt request control register for the value of interrupt request control register simultaneously in address setting.

Constitution of device number is as follows :
(1) Format for designation of interrupt level of uPD71059
(2) Format for designation of interrupt request

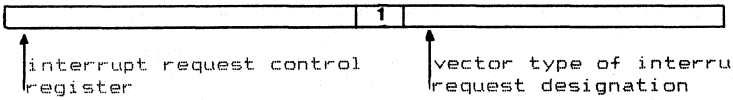
(1)



Interrupt level of uPD71059 to be designated by designation of bit 7 to be at 0. Necessary to let upper byte be 0.
Under designation of interrupt at lower byte, bit 0 - 2 means slave level at master slave constitution, bit 3 means designation weather master only or master slave constitution and bit 4 - 6 means master level respectively.

Correspondence between interrupt level of uPD71059 and vector type shall be fixed. Corresponding 72 level of interrupt level to 72 types from 56 through to 127 of vector type.
For instance, in case of master only application, master level 0 corresponds to vector type 56, and in case of master slave constitution, slave level 7 connecting with master level 7 corresponds to vector type 127.

(2)



Under designation of bit 7 to be 1, which means interrupt request to be designated, and interrupt request control register shall be set up by upper byte.

To set vector type numbers (12 - 31) assigned to each interrupt request on lower byte.

However, as OS adopts time base counter as system timer, vector type 31 corresponding to time base counter can not be used.

4.5.2 SIG_INT (SIGNAL INTERRUPT)

System call number 19

Function

Invoking interrupt waiting task and completing interrupt handler

Parameter

Task numbers necessary to invoke (0 - 63) (Word type)

Condition code

Nil

C format

```
cc = SIG_INT(task_no);
short           task_no ;
```

Explanation

Only possible from interrupt handler inside, and issuing on hand-over of control to the task after interrupt handler so that no return of control onto the next address to this system call.

Condition code, therefore, shall not return on error.

On error, control shall be returned to interrupted point immediately without invocation of designated task as well as error code return.

Also on this system call, as no multiple interrupt control in both cases of outer interrupt as well as interrupt request, nesting control of interrupt handler and instruction performance of EOI (end of interrupt) and FINT (finish interrupt) should be done from handler inside.

Sequence of this system call issuing differs from the other system calls i. e.

The following program on Assembler :

```
PUSH     task_no  : Setting task number to be invoked
JMPF   SIG-INT-ENTRY  : FAR JUMP onto absolute address
                        F000:C00E
```

4.5.3 WAI_INT (WAIT FOR INTERRUPT)

System call number 20

Function

Transition onto interrupt WAIT state

Parameter

Nil

Condition code

Normal termination	0 E_OK
Release of interrupt WAIT	8 E_INT

C format

```
cc = WAI_INT;
```

Explanation

Task issuing this system call shall come into interrupt waiting state and be released from interrupt waiting by system call SIG_INT from interrupt handler.

In case to expect to release interrupt waiting state by task itself, not from interrupt handler, system call CAN_INT shall be usable.

E_OK : released by SIG_INT
E_INT : released by CAN_INT

4.5.4 CAN_INT (CANCEL INTERRUPT)

System call number 21

Function

Releasing task of interrupt waiting from WAIT state

Parameter

Task number (0 - 63) (Word type)

Condition code

Normal termination	0	E_OK
No waiting state of corresponding task	8	E_INT

C format

```
cc = CAN_INT(task_no);
short          task_no ;
```

Explanation

Issuing for task which is under interrupt waiting state by system call WAI_INT and release WAIT state. In case of task not to be under interrupt waiting, E_INT shall be returned as error.

4.5.5 DIS_INT (DISABLE INTERRUPT)

System call number 22

Function

Prohibiting interrupt at unit basis of device number (Interrupt level or vector type)

Parameter

Device number (Word type)

Condition code

Normal termination	0	E_OK
Device number error	4	E_DVN

C format

```
cc = DIS_INT(device_no);
short           device_no ;
```

Explanation

Possible to issue during task operation as well as interrupt handler operation.

In case of designation of interrupt level of uPD71059, setting designated bit of IMR, and in case of interrupt request designation, setting 6 bit of interrupt require control register.

4.5.6 ENA_INT (ENABLE INTERRUPT)

System call number 23

Function

Permitting interrupt at unit basis of device number (interrupt level or vector type)

Parameter

Device number (Word type)

Condition code

Normal termination	0	E_OK
Device number error	4	E_DVN

C format

```
cc = ENA_INT(device_no);
short           device_no ;
```

Explanation

Possible to issue during task operation as well as interrupt handler operation.
In case of interrupt level designation of uPD71059, resetting designated bit of IMR, and in case of interrupt request designation, resetting bit 6 of interrupt request control register.

4.5.6 RES_INT (RESTART AND RETURN FROM INTERRUPT)

System call number 24

Function

Completing interrupt handler and returning control to restart address

Parameter

Nil

Condition code

Nil

C format

```
cc = RES_INT();
```

Explanation

This system call shall be usable incorporated with SET_ADR. In case that a task which is interrupted by the handler issuing RES_INT has already issued a system call SET_ADR, transition of control onto designated restart address.

In case of SET_ADR not to be issued, returning control to the position of interrupt.

On this system call, no multiple interrupt control shall not be performed in both cases of outer interrupt and interrupt request so that nesting control of interrupt handler and performance of EOI (end of interrupt) and FINT (finish interrupt) instructions shall be done in each interrupt hander.

Program of this system call differs from ones of the other like as SIG_INT i. e.

```
JMPF    RES-INT-ENTRY   : FAR JUMP onto absolute address
                          F000:C020
```

4.6 RESERVED INTERRUPT VECTOR TABLE

vector	
31	system time (time base counter)
48	RTOS data
56 - 63	master uPD71059
64 - 71	slave 0 uPD71059 connected with master IR0
72 - 79	slave 1 uPD71059 connected with master IR1
80 - 87	slave 2 uPD71059 connected with master IR2
88 - 95	slave 3 uPD71059 connected with master IR3
96 - 103	slave 4 uPD71059 connected with master IR4
104 - 111	slave 5 uPD71059 connected with master IR5
112 - 119	slave 6 uPD71059 connected with master IR6
120 - 127	slave 7 uPD71059 connected with master IR7

4.7 CONDITION CODE TABLE

condition code		
0	E_OK	normal termination
1	E_DMT	in case of STA_TSK, not DORMANT in case of SUS_TSK, RSM_TSK, DORMANT
2	E_SUS	in case of SUS_TSK, SUSPEND in case of RSM_TSK, not SUSPEND
3	E_BLK	no memory blocks
4	E_DVN	incorrect device number
5	E_SYS	undefined PIC port address
6	E_RSC	0 resource
7	E_MSG	no messages
8	E_INT	not interrupt waiting

μPD79011

SINGLE CHIP MICROCOMPUTER (V25)

WITH

REAL TIME OPERATING SYSTEM

TECHNICAL REFERENCE MANUAL

FOREWORD

The uPD79011 is a 16-bit, single-chip microcomputer that has an internal nucleus of a real-time operating system (OS). The OS incorporated in this microcomputer is simple in its specifications and capable of executing system call processing at high speeds by means of task switching using the internal register bank.
This document describes how to use the uPD79011 and the internal information of the system, so that you can efficiently organize a flexible system.

CHAPTER 1 uPD79011 SYSTEM DEVELOPMENT ENVIRONMENT AND DEVELOPMENT PROCEDURES

This chapter describes the environment and procedures necessary for developing a system that uses the uPD79011 (hereafter referred to as " Real Time Operating System or short, RTOS ").

1.1 Environment

1.1.1 Language

The RTOS is designed so that its application programs are developed in assembler and C language. For these languages, the following standard software is readily available from NEC.

. Assembler

Relocatable Assembler Package for V Series (NEC)

RA70320		RA70116 (Assembler)
LK70320	or	LK70116 (Linker)
OC70320		OC70116 (Object Converter)
LB70320		LB70116 (Librarian)

. C compiler

CC70116 (NEC, small or large model)

Assemblers and C compilers other than above can also be used if they are of the 8080 family. When using them, however, make sure that their specifications match those of the RTOS. (For details, refer to Chapter 2.)

When the C language is used, the large or big memory model of the C compiler is best for interfacing with the RTOS. When other memory model (such as small, compact, and medium) is to be used, note that the RTOS always requires 32-bit pointers (SEGMENT:OFFSET). This means that, if a memory model that has only 16-bit pointers (OFFSET) is to be used, the sequence in which parameters are saved to the stack must be changed when a pointer is used as the parameter of some system calls. Nevertheless, the small memory model has advantages over the large or bit model in terms of program size and speed. It is therefore a good practice to select the best memory model as your application requires.

1.1.2 Devices

Use the following development devices to organize your system with the uPD79011.

1 IBM - PC Series
2) VAX-TM-11 Series
3) uPD70322 (V25) In-circuit Emulator

1) and 2) above are used to develop application programs in assembler or C. 3) is used to download the application programs to the target system or to debug software.

Note that neither the development languages and development devices introduced above are not absolutely necessary, and the user can use other devices of his own choice depending on the language or form of development.

VAX™ registered trademark of DIGITAL EQUIPMENT Corp.
IBM - PC registered trademark of International Business Machine Corporation.

1.2 Procedures

Let's discuss the development procedures of the RTOS system.

(1) Developing user application programs

. Creating source program

Create the source file of your application program by describing tasks or interrupt handlers in assembler or C. Then assemble and compile the source file to create an object file.

. Creating interface routine

If tasks are described in C, create an interface routine that corresponds to each system call in an assembler, to directly call the RTOS in C. The created interface routine must be accessed in C as an external function. Moreover, it is generally convenient to create an interface routine by describing tasks in assembler, so that system calls can be consistently processed. Assemble the interface routine by assembler to create an object file.

For details on interface routines, refer to Chapter 2.

. Linking and downloading

Create a load module file by linking and locating the object file of the application program you have created, the object file of an interface library, and, as necessary, a library. At this time, the memory location of the application program is determined. Then download the load module file to the target system.

(2) Creating configuration table

. Creating source program

Create the source file of a configuration table which is the system initialization information of the RTOS. The configuration table differs in its contents and organization depending on the user system; so, describe each piece of information according to your system.

After creating the source file of a configuration table, assemble it to create an object file.

. Linking and downloading
To create a load module file, singly link and locate the object file of the configuration table. Since the load address of a configuration table is fixed to FB80:0000H, specify this address as the location address when linking and locating the object file. Then download the load module to the target system.

For details on the configuration table, refer to Chapter 3.
The above discussion on system development process is illustrated in Figure 1-2-1.

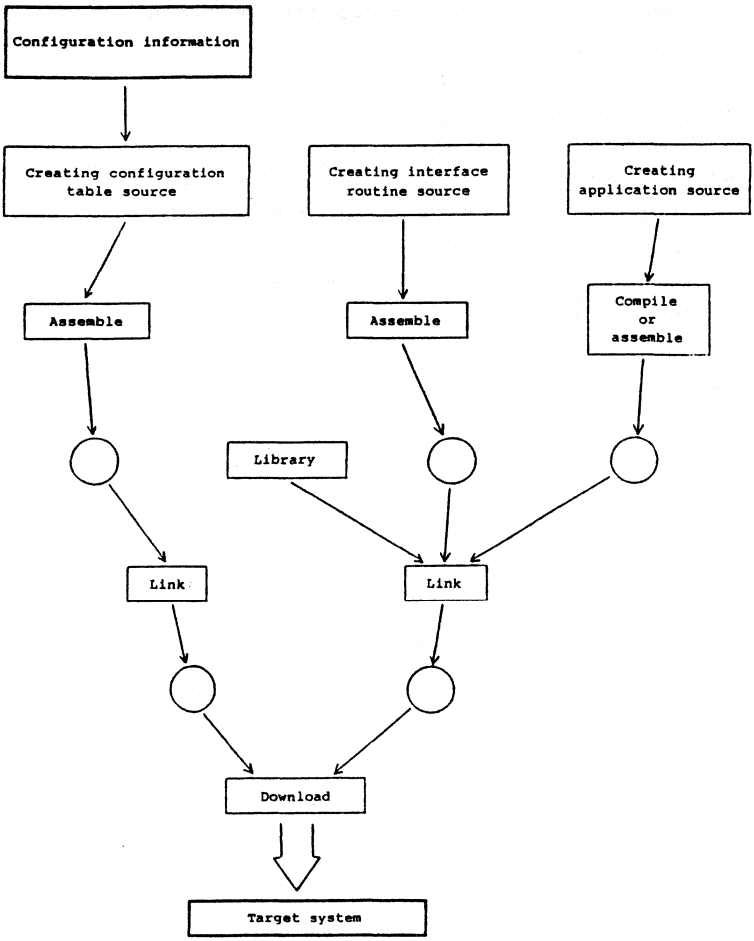


Fig. 1-2-1 Development Process

CHAPTER 2 INTERFACE ROUTINE

As stated in Chapter 1, the RTOS is designed so that its application program is created in assembler or C. If the application program is described in C, however, an interface routine described in assembler is necessary for interfacing between the application program and the RTOS. This chapter describes the procedures for issuing the RTOS's system calls and creating an interface routine.

2.1 Issuing System Call

2.1.1 General procedure

Generally, issue the system calls of the RTOS, following these steps.

1) Set the parameter of a system call on the stack.
2) Set a pointer for the parameter on the stack.
3) Set the number of the system call in register AW. 4) Call the entry address of the RTOS (F000:C000H) by executing the CALLF instruction.

The above procedure executes the system calls of the RTOS except RES_INT and SIG_INT, which are called differently. For details on these system calls, refer to 2.1.2.

1) Receiving/passing over parameter
   Since the RTOS is designed so that its application programs are described in C, parameters are passed over to the RTOS on the stack. Therefore, the interface routine is called as an external function when a system call is issued in C. At this time, all the parameters to be passed over to the interface routine are saved to the stack.
   Figure 2-1-1 illustrates the status of the stack when a function is called in C. Note that the parameters of the function are saved to the stack from the right to the left. Also note that the high word of a parameter consisting of 32 bits or more is saved to a high address.

(With the RTOS, possible 32-bit parameters are pointers, and the segment of a pointer is saved to a high address.) Parameters are saved to the stack in this way with an 86-family C compiler also. Therefore, as long as the application program is created using this type of C compiler, there is no problem. However, if other C compiler is used, an interface routine is needed to change the sequence in which parameters are saved to the stack must be changed; otherwise, the RTOS cannot restore them.

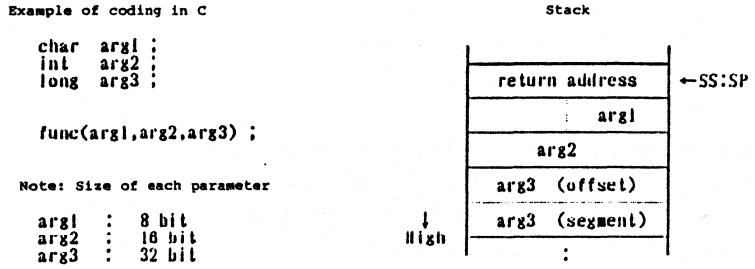


Fig. 2-1-1 Stack Status When Function is Called in C

2) Setting parameter pointer

As discussed, the RTOS receives parameters on the stack. But this does not mean that it directly references the parameters. The RTOS indirectly references the parameters, when called by CALLF, using a parameter pointer saved to the stack immediately before.

The value of the parameter pointer is the offset value of a parameter that has been saved to the stack last, that is, a parameter that is at the lowest address of the stack. The RTOS references the parameters by using this value and the value of SS. Figure 2-1-2 shows the status of the stack immediately after the RTOS has been called. The parameter pointer needs not be saved when executing a system call that does not have any no parameter.

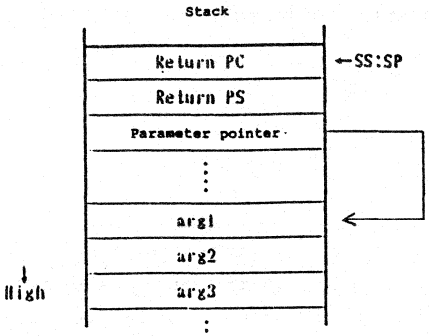


Fig. 2-1-2 Stack Status Immediately after Calling RTOS

3) Setting system call number

To issue a system call, not only its parameter but also the type of the system call must be informed to the OS. The RTOS sets the number of a system call number in the AW register immediately before the system call is issued.

4) Calling RTOS

To call the RTOS, call the following entry address of the HSS from outside a segment.

RTOS entry address - F000:C000

Even if a program exists in the same segment as the entry address of the RTOS, the RTOS must be always called from outside a segment.

2.1.2 Issuing system calls SIG_INT and RES_INT

System calls SIG_INT and RES_INT are used to return from an interrupt handler. Therefore, they are called differently from other system calls. The procedure to call these system calls are as follows:

1) Using an interrupt handler, restore or remove the registers that have been saved to the stack.
2) Save the parameters to the stack (when SIG_INT is issued. RES_INT does not have any parameter).

3) Jump to the specified entry address from outside a segment (BRF).

. Entry address
SIG_INT - F000:C00E
RES_INT - F000:C020

Note: If the code segment of the interrupt handler is at the same address (F000H) as the entry address of the RTOS, entry to the RTOS can be made even by jumping from inside a segment.

System calls SIG_INT and RES_INT do not set a system call number or a parameter pointer. Moreover, entry to the RTOS can be made executing a jump instruction instead of a call instruction.
The status of the stack immediately after SIG_INT and RES_INT have been called is illustrated in Figure 2-1-3.

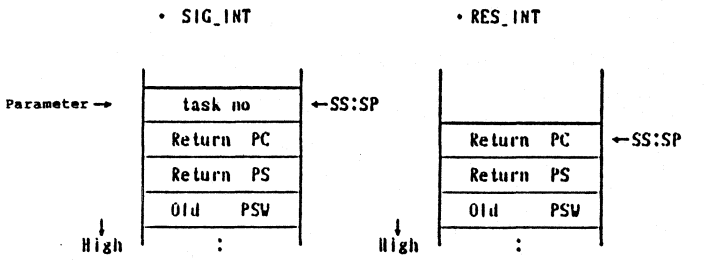


Fig. 2-1-3 Stack Status Immediately after Calling SIG_INT and RES_INT

2.1.3 Return parameter

The RTOS returns parameters, including condition codes, that indicate whether a system call has been normally completed or an error has occurred. These parameters are returned by using the AW register. When control has been returned from the RTOS after a system call was issued, a return parameter has been set in the AW register.
Although the RTOS retains the values of registers immediately before it is called (i.e., immediately before the CALLF

instruction is executed), the contents of the AW register are not retained because this register is used to return parameters. However, since system calls SIG_INT and RES_INT do not return any condition code, all the register values are retained when these system calls are issued.

2.2 Creating Interface Routine

2.2.1 Description format in C

An interface routine can be created in various ways as long as system calls are issued in the correct procedure. This section demonstrates an example of creating an interface routine. In the following example, the CC70116 C Compiler (small model) and the RA70320 Assembler are used. It is assumed that the interface routine in this example is called with the following description in C.

Description in C

cc = <name>([<parameter>]..);

cc: return value of function (return parameter from RTOS)

<name>: name of system call

<parameter>: input parameter

Example: In the case of STA TSK (Start Task)

```
unsigned int task no = 1;
unsigned int cc;

cc = STA_TSK (task no); /* starts task no. 1 */
```

Note: Since system calls SIG_INT and RES_INT do not have a return parameter, 'cc =' needs not to be described to issue these system calls. Also, although the interface routine is described in C, be sure to check the C Compiler and Assembler when linking the routine in C because there may be some restrictions in the number of characters that can be used for function names, etc.

By describing as shown above, parameters will be saved to the stack and the interface routine will be called.

The RTOS allows either a 16-bit integer or 32-bit pointer to be selected as the data type and the size of the parameter of a system call. As variable declaration in C that corresponds to the integer or pointer, the following is generally used.

- 16-bit integer: int, short
- 32-bit pointer: All variables that are declared to be pointers by * appended.

Note: Data to be handled by the RTOS are all unsigned. int and short should therefore be declared with unsigned appended. Moreover, the data size of a pointer may be 16 bits (OFFSET) depending on the memory model of the C compiler to be used (for details, refer to Chapter 6).

2.2.2 Description example

The following shows an example of describing an interface routine for system call STA_TSK by the RA70320 Assembler.

STA_TSK description example:

```
CGROUP  GROUP   CODE

CODE    SEGMENT PARA    PUBLIC  CODE
        ASSUME  PS:CGROUP

PUBLIC  _STA_TSK

_STA_TSK        PROC    NEAR            ;1

        MOV     AW,SP                   ;2
        ADD     AW,2
        PUSH    AW
        MOV     AW,0                    ;3

        CALL    DWORD PTR PS:RTOS_ENTRY ;4

        ADD     SP,2                    ;5
        RET                             ;6

_STA_TSK        ENDP

RTOS_ENTRY      DW      0C000H          ;7
                DW      0F000H

CODE    ENDS
        END
```

(1) CC70116, whose memory model is small, calls functions from inside a segment. Therefore, interface routine STA TSK is declared a NEAR type routine. When using a large-model C compiler, functions are called from outside a segment and thus, the interface routine must be declared to be FAR type.
(2) A parameter pointer is saved to the stack. In this example, the parameter pointer is called in C from inside a segment; so, SP+2 is treated as a parameter pointer. A parameter pointer is called from outside a segment when a large-model C compiler is used. Consequently, the parameter pointer is SP+4.
(3) A system call number is set in the AW register.
(4) The RTOS is called from outside a segment. In the example, it is indirectly called via memory by using the entry address of the RTOS that has been defined data in a code segment.
(5) The parameter pointer is removed from the stack after control has been returned from the RTOS.
(6) Execution returns to the point where the RTOS was called. At this time, the return parameter of the RTOS has been set in the AW register. Normally, in C, the return parameter of a function is returned by the AW register. Therefore, no special process is necessary.

When using a large-model C compiler, the segment definitions made before and after the above example must be accordingly changed. For details, refer to CC70116 Manual.

Note: Only small and large CC70116 memory models are available. With other memory models, the type that calls functions are generally as shown below:

Compact : inside segment
Medium : outside segment
Big : outside segment

The above description example is a procedure to issue ordinary system calls and, as mentioned before, system calls SIG_INT and RES_INT are issued differently from the others. The description of an interface routine for these system calls accordingly differs. An example of describing an interface routine for SIG_INT is shown below (an interface routine for RES_INT is described in the same manner except the entry address will be F000:C020H).

SIG_INT description example

```
CGROUP  GROUP   CODE

CODE    SEGMENT PARA    PUBLIC  'CODE'
        ASSUME  PS:CGROUP

PUBLIC  _SIG_INT

_SIG_INT        PROC    NEAR

        ADD     SP,2                    ;1

        BR      DWORD PTR PS:SIG_ENTRY  ;2

_SIG_INT        ENDP                    ;3

SIG_ENTRY       DW      0C00EH          ;4
                DW      0F000H

CODE    ENDS
        END
```

(1) The return address on the stack is removed. The parameter pointer is SP+4 instead of SP+2 when a large-memory model is used.
(2) Execution jumps to the entry address. In this example, execution indirectly jumps via memory, using the entry address of SIG_INT that is defined data in a code segment.
(3) Since SIG_INT and RES_INT directly returns from the RTOS, no return instruction is needed.

The interface routine created as explained above is assembled and then linked with a user application program. As necessary, put the object of each interface routine in a library, using the LB70320 Librarian.

CHAPTER 3 CONFIGURATION TABLE

When the system is started, the RTOS first executes a reset routine that initializes the system, to initialize the memory, etc. At this time, the reset routine references a configuration table for information on system initialization. The configuration table is loaded from address FB80:0000H, which is directly referenced by the reset routine. This chapter discusses the organization and how to create the configuration table.

3.1 Organization

An example of the assembler source file of a configuration table is shown below. This source file is described for the RA70320 Assembler. As can be seen, the configuration table is a data table that is defined by 'DB' and 'DW'.

Assembler source example of configuration table:

```
conf_tbl        segment         para    public
                org 00

ptr0            dw      internal_ram_base  ; (1)
ptr1            dw      task_cnt
ptr2            dw      sma_cnt
ptr3            dw      mbox_cnt

internal_ram_base
                db      07fh            ;IDB = 7Fh (2)
prc_info        db      04ch            ;PRC = 4Ch

low_ds          dw      40h             ;free RAM area from
high_ds         dw      800h            ;00400h to 07FFFh
blk_size        dw      50              ; (3)

port0           dw      01000h          ; (4)
port1           dw      02000h          ;
port2           dw      0ffffh
port3           dw      0ffffh
port4           dw      0ffffh
port5           dw      0ffffh
port6           dw      0ffffh
port7           dw      0ffffh
port8           dw      0ffffh

task_cnt        db      10              ;max number of task:
                                        ;task0 to task8, and task9
                                        ;for init-task (5)
min_task_no     db      0               ;first task
init_task       db      9               ;number of init-task

idle_sp         dw      0fffh           ;stack pointer and
idle_ss         dw      0f00h           ;stack segment for
                                        ;idle task (6)

init_pc0        dw      00000h          ;program counter for task0
init_ps0        dw      0f000h          ;program segment for task0
init_sp0        dw      00effh          ;stack pointer for task0
init_ss0        dw      00f00h          ;stack segment for task0
init_ds0        dw      00800h          ;data segment for task0
                                        ; (7)
```

```
init_pc1        dw      00200h          ;dito for task1
init_ps1        dw      0f000h
init_sp1        dw      00dffh
init_ss1        dw      00f00h
init_ds1        dw      00810h

                *
                *
                *

init_pc9        dw      00c00h          ;dito for task9
init_ps9        dw      0f000h
init_sp9        dw      001ffh
init_ss9        dw      00f00h
init_ds9        dw      00a20h

sma_cnt         dw      2               ;number of semaphores (8)
init_rsc0       dw      10              ;resource of semaphore0
init_rsc1       dw      1               ;resource of semaphore1

mbox_cnt        dw      5               ;number of mailboxes

reserve         dw      0               ;reserve value used by
                                        ;V25 RTOS (9)
conf_tbl        ends
                end
```

Next, let's discuss each constituent of the configuration table.

(1) Pointers

The organization of the configuration table changes depending on the status of the user system. Therefore, set pointers (i.e., offset value which is obtained taking a segment as FB80H) for to delimiter information at the first part of the table so that the reset routine can easily reference the contents of the table.

PRT0 read
Pointer for INTERNAL RAM BASE

PTR1 read
Pointer for TASK CNT

PTR2 read
Pointer for SMA CNT

PTR3 read
Pointer for MBOX CNT

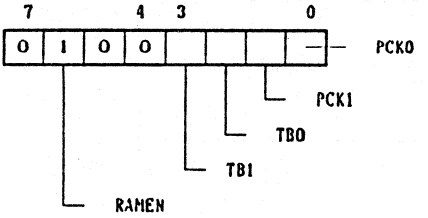
(2) System information

INTERNAL RAM BASE byte

This is information that sets the internal RAM base segment of the uPD79011, and is set in special register IDB (address 0FFFFFH). The internal RAM base segment is 0XX00H where the value of the IDB register is 0XXH, and register banks and special registers are mapped to 512 bytes starting from address 0XX00H:0E00H.

PRC INFO byte

This is a value set in special register PRC of the uPD79011. The PRC register specifies a CPU operation clock, time base interrupt cycle, and whether the internal RAM memory reference is enabled or disabled. The meaning of each of the eight bits is illustrated below. Reference to the internal RAM memory must be enabled.



PCK1-PCK0: Specifies frequency division of system clock

0 0	Divides oscillation frequency by 2
0 1	Divides oscillation frequency by 4
1 0	Divides oscillation frequency by 8
1 1	Reserved

TB1-TB0: Specifies time interval (specifies output tap of frequency divider)

0 0	Specifies TBC9 output of time base counter
0 1	Specifies TBC12 output of time base counter
1 0	Specifies TBC15 output of time base counter
1 1	Specifies TBC19 output of time base counter

RAMEN: Enables/disables internal RAM reference

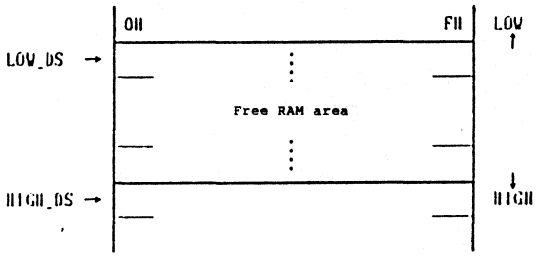
1	Enables

(3) RAM information

LOW DS, HIGH DS word

This specifies a user free RAM area. As a free RAM area, one consecutive memory area which is defined by an upper-limit and lower-limit segment addresses (OFFSET is 0) must be specified. The initialize routine reserves a system table and various control blocks on this area. The rest of the area serves as a memory block and is queued to the system table as a memory table (refer to Chapter 4). Therefore, the size of this RAM area must be large enough to accommodate the control blocks. Note that, when 0 is specified for BLK SIZE, the memory block is not generated.

Note: The paragraph indicated by HIGH DS is not included in the free RAM area as shown below.



BLK SIZE read

This specifies the size of a memory block that is to be placed under the management of the RTOS. The specification is made in units of 16 bytes.

(4) Interrupt controller information

PORT0 to PORTB 9 words

Specifies a port address for the uPD71059 Interrupt Controller to be used. PORT0 indicates the port address of the uPD71059, the master, while PORT1 through PORT8 respectively indicate the port addresses of the uPD71059s, the slaves. The port

address of a uPD71059 that is not used must be undefined and specified as 0FFFFH.

Note: This information is copied to the system table of the RTOS, and the RTOS itself assumes that no uPD71059 that corresponds to the set port address does not exist. The RTOS checks whether a uPD71059 that specifies the numbers of devices to which the parameters of system calls DEF_INT, DIS_INT, and ENA_INT are passed over. When the device numbers of both the master and slaves are specified, the RTOS only checks the port addresses of the slaves. It checks the port address of the master only when the device number of the master is specified. Therefore, when address 0FFFFH is specified for PORT0, be sure to specify 0FFFFH also for PORT1 through PORT8. By contrast, if the level of the master to which slaves are connected is specified as the master only by the device number on issuance of DEF_INT, DIS_INT, and ENA_INT, processing is performed normally.
In this case, DIS_INT and ENA_INT disables or enables interrupt to the specified level of the master. However, DEF_INT do not have any meaning even when an interrupt handler is defined for interrupt vectors (56 to 63) that are reserved for the master because it is not started.

Normally, the uPD71059 requires two ports to turn ON and OFF the A0 pin (which detects a command and data). The port address set above is the one that corresponds to the OFF status of the A0 pin. The RTOS automatically assumes the specified port address plus 2 as the port address of the ON status of the A0 pin. The user therefore must design the circuit so that it satisfies this condition.

(5) User task information

TASK CNT byte
Specifies the total number of user tasks (excluding idle tasks). Up to 63 tasks can be specified.

MIN TASK NO byte
Specifies the minimum task number of the user tasks. The user task numbers are assigned in ascending order starting from the specified number. The task having a number less than that specified is not generated. Consequently, of register banks 0 to 5 that are respectively occupied by tasks 0 to 5, register banks having a number less than the specified user task can be used for other purposes by specifying a value of 0 as the user task number. However, when a value of 7 or greater is specified, bank 0 is always used by a task.

INIT TASK byte
Specifies the number of the task that should be executed first. Tasks other than that specified are dormant when the system is started.

(6) Stack information on idle task

IDLE SP read
Specifies the SP value of an idle task.

IDLE SS read
Specifies the SS value of an idle task.

The stack can be set in any range of addresses. However, the stack size must be at least 32 bytes.
For details on idle tasks, refer to 5.1.5, System execution.

(7) User task register information

INIT PC0 word
Specifies the initial PC value for the minimum user task number that is specified by MIN TASK NO described above.
INIT PS0 word
Specifies the initial PS value of the same user task.
INIT SP0 word
Specifies the initial SP value of the same user task.

INIT SS0 word
Specifies the initial SS value of the same user task.
INIT DS0 word
Specifies the initial DS0 value of the same user task.

Specifies this information for all the user tasks starting from the one having the least significant number in the order in which the above register descriptions are presented.

(8) Information on semaphore and mail box

SMA_CNT word
Specifies the number of semaphores to be used. Up to 256 semaphores can be specified.
INIT RSC0 word
Specifies the number of initial resources for semaphore numbered 0.

Specify this information for all semaphores starting from semaphore numbered 1. Note, however, that this information needs not be specified at all when 0 is specified for SMA_CNT.

MBOX_CNT word
This specifies the number of mail boxes to be used. Up to 256 mail boxes can be specified.

(9) Reserved word area

RESERVE read
This specifies a reserved word area. Be sure to specify 0.

3.2 Creating Configuration Table

3.2.1 Notes on coding configuration table

An assembler source file for the configuration table should be directly created by the user in accordance with the organization of the table described in the preceding section. In doing so, however, pay attention to the following points:

referring to 3.1. However, data that indicates the pointer of the first 4 words of the configuration table and data immediately before that data need not to be consecutive.

2) Specify FB80:0000H for the start address of the final load module of the configuration table. Therefore, the segment value of the configuration table that is to be specified when the object files of the configuration table are linked must be FB800H. Moreover, specify the pointer for the first 4 words of the configuration table so that it is successively set starting from offset address 0.
   In the example shown in the preceding section, directive 'ORG 00H' is used to satisfy the above conditions.
3) When semaphores and mail boxes do not need to be generated, set the following information of the configuration table to 0.

   . Semaphore: SMA_CNT
   . Mail box: MBX_CNT
   . Memory block: BLK_SIZE

   Note, however, that at least one task must be generated; so, do not set TSK_CNT to 0.
4) Be sure to set RESERVE, which is a keyword, to 0.

3.2.2 Creating load module file

This section describes the procedure for creating a load module file from the assembler source file of the configuration table created in the preceding section. In the following discussion, it is assumed that the RA70320Assembler is used, and, as the source file, 'CONF_FIL' mentioned in the preceding section is used.

(1) Assembly
   Assemble the source file by executing command 'RA70320'. Suppose that RA70320 is now in current drive A, and that the configuration file is in drive B. Then the procedure for assembling the source file will be:

```
D:\> RA70320 B:CONF.FIL

UPD70320 ASSEMBLER V1.0  [14 Sep 85]
   Copyright (C) 1985 NEC Corporation

ASSEMBLY COMPLETE,     0 ERROR(S) FOUND
```

The above procedure creates an object file (CONF.REL) and list file.

(2) Linking

Module files are linked by executing command 'LK70320'. To link module files so that the configuration table is loaded from address FB800H, the following command file must be separately created for linking purpose.

LINK.CMD:

```
B:CONF TO B:CONF
ORDER(SEGMENTS(CONF_TBL))
ADDRESSES(SEGMENTS(CONF_TBL(0FB800H)))
```

(A) Specifies the REL file to be linked. In this example, file CONF.REL in drive B is specified ('.REL' can be omitted).

(B) Specifies the name of the file to which the result of the linking is to be output. In this example, file CONF.LNK is created in drive B ('.LNK' is assumed if the attribute is omitted).

(C) Specifies the location address of the segment CONF_TBL that is defined in the source file as FB800H. The segment address of CONF_TBL is FB80H.

After command file 'LINK.CMD' has been prepared, link the module files according to the following procedure:

```
D:\> LK70320 ! B:LINK.CMD

UPD70320 LINKER V1.0  [17 Sep 85]
   Copyright (C) 1985 NEC Corporation


LINK COMPLETE
```

In the above example, it is assumed that file LINK.CMD is in drive B. When the module files have been linked, a link file (CONF.LNK) and map file (CONF.MAP) are created in drive B. The link file CONF.LNK is itself a load module file. Generally, however, it is converted into a HEX file. The procedure for this conversion is discussed next.

. Creating HEX file

The link file is converted into a HEX file by executing command 'OC70320' as follows:

```
D:\> OC70320 B:CONF TO B:CONF

UPD70320 OBJECT CONVERTER V1.0  [28 Aug 85]
   Copyright (C) 1985 NEC Corporation


OBJECT CONVERSION COMPLETE
```

(D) Specifies that CONF.LNK in drive B is converted into so that a HEX file CONF.H8G is created in drive B. Although no attribute is specified in this example, '.LNK' and '.H86' are automatically employed.

The load module file CONF.LNK or CONF.H86 will then be downloaded to the target system.

CHAPTER 4 RTOS MEMORY CONFIGURATION AND CONTROL BLOCKS

4.1 RTOS Memory Configuration

The memory of the RTOS consists of the following system blocks or user blocks.

1) OS nucleus
2) Reset routine for system initialization
3) Configuration table
4) OS control blocks such as system table
5) Interrupt vectors
6) User application program

The OS nucleus and the reset routine are set in the internal ROM area that is the higher 16K bytes of the 1M-byte memory space of the uPD79011. The user should set the configuration table in an area made up of FB800H through FBFFFH, which is reserved for this table. The OS control blocks are automatically generated by the reset routine in the free RAM area that is specified in the configuration table. Therefore, they can be set in any addresses by the user (however, exercise care that the control blocks do not overlap 1), 2), 3), and 5)). The interrupt vectors take up an area consisting of 00000H through 003FFH. Note, however, that interrupt vector 48 is reserved for the system and thus, cannot be used by the user.

Note: Interrupt vectors 31, and 56 through 127 are also reserved for the system. However, vector 31 can be used when system calls related to time management are not used. Moreover, some or all interrupt vectors 56 to 127 can be used by the user depending on whether uPD71059s corresponding to the respective vectors are used or not (refer to List of reserved vectors in Appendix).

Consequently, the application program can use any memory space except the areas occupied by 1), 2), 3), 4), and 5). Figure 4-1-1 shows the memory mapping of the HSS system.

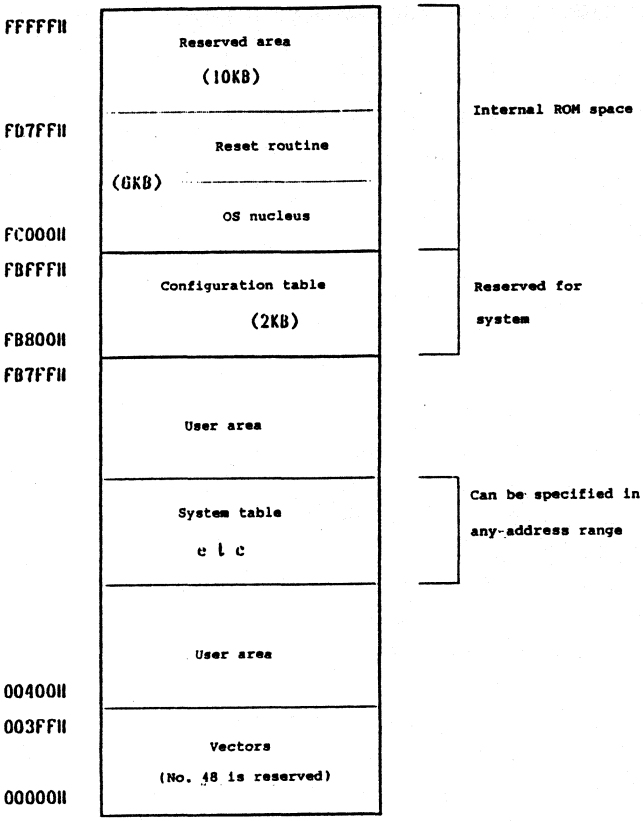


Figure 4-1-1 Memory Mapping of RTOS System

4.2 RTOS Control Blocks

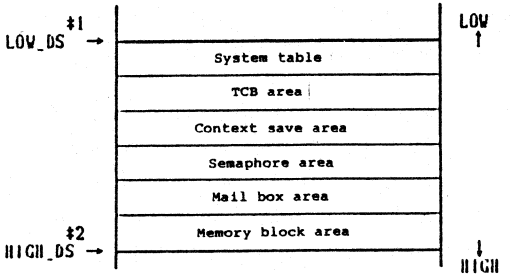
4.2.1 Memory area of control blocks

The RTOS is internally provided with various control blocks on the memory each of which performs its own process such as task execution and synchronous communication management. All these control blocks are under management of the RTOS and are invisible from the user task because they are intended for internal processing. Here are the control blocks and their size.

1) System table : 30H bytes
2) TCBs (Task Control Blocks): 1FH bytes
3) Context save areas : 20H bytes
4) Semaphore : 6H bytes
5) Mail box : 8H bytes
6) Memory block : Specified by user (in units of 16 bytes)

Successive memory areas for these control blocks are secured in the free RAM, which is specified by the user in the configuration address when the system is started, starting from the lowest address and in paragraph units. The memory areas are secured in the order in which the control blocks are presented above and queued.

Figure 4-2-1 shows the assignment of these memory areas.



*1 & *2: Refer to Chapter 3.

Fig. 4-2-1 Memory Areas for Control Blocks

. Configuration of TCB area

Figure 4-2-2 shows the configuration of the TCB area. TCBs are set in this memory area in the ascending order of task numbers and in paragraph units from the TCB for a task having the least significant number. The number of TCBs set in this area is the number of user tasks specified in the configuration table plus one of idle tasks, which are system tasks.

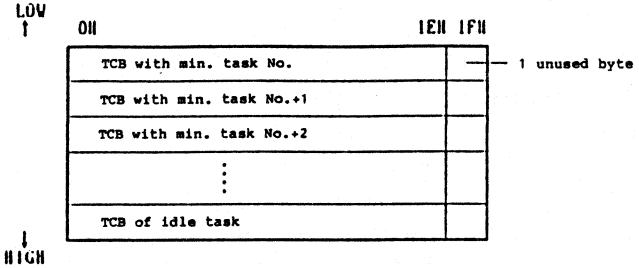


Fig. 4-2-2 Configuration of TCB Area

. Configuration of context save area

Figure 4-2-3 shows the configuration of the context save area. This area is actually a set of small subdivisions each of which corresponds to tasks. The subdivisions are set in units of two paragraphs and in the ascending order of task number. The number of subdivisions to be set equals the number of tasks that use register bank 6, in other words, the number of tasks that are numbered 6 or more.

Note: If a task numbered 6 or less does not exist, i.e., if 7 or greater number is specified as the minimum task number, the subdivisions are set starting from the minimum task number available. If only one or no task numbered 6 or more exists, this area is not secured.

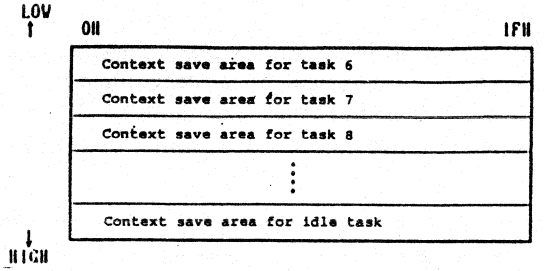


Fig. 4-2-3 Configuration of Context Save Area

. Configuration of semaphore area

Figure 4-2-4 shows the configuration of the semaphore area. Two semaphore are set in one paragraph in ascending order, starting from semaphore 0. The number of semaphore specified in the configuration table are set in this area.

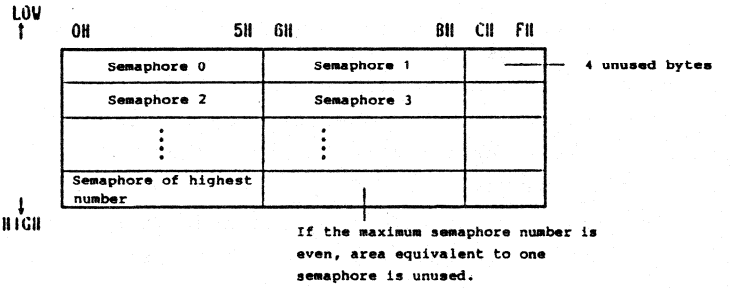


Fig. 4-2-4 Configuration of Semaphore Area

. Configuration of mail box area

Figure 4-2-5 shows the configuration of the mail box area. Two mail boxes are set in one paragraph in ascending order, starting from mail box 0. Like semaphore, the number of mail boxes specified in the configuration table are set in this area.

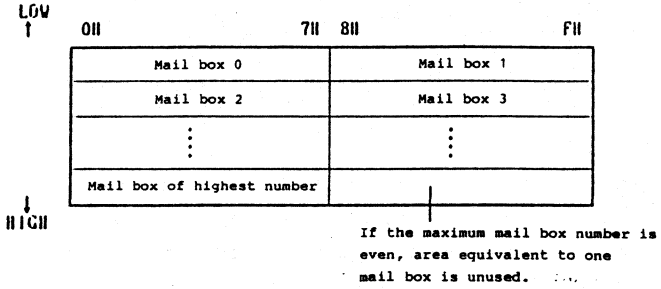


Fig. 4-2-4 Configuration of Mail box Area

. Configuration of memory block

Figure 4-2-6 shows the configuration of the memory block. The memory block is generated in units specified by the user in the configuration table after the other control blocks have been assigned to the memory.

Note: If memory block size 0 is specified by the user, the memory block is not generated. Therefore, this area remains unused.

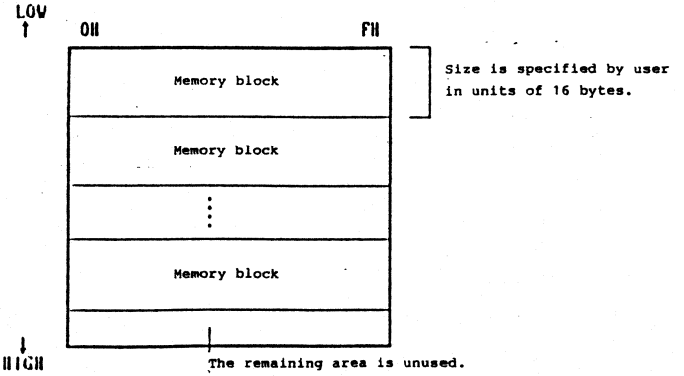


Fig. 4-2-6 Configuration of Memory Block

4.2.2 Explanations of control blocks

1) System table

The RTOS uses the system table in order to manage the entire system. The base address of the system table is stored in one word of interrupt vector 48 at the lower side. The RTOS obtains the addresses of the system table by referencing this vector.

Note: The RTOS represents the address for each control block as only a segment with offset being 0. Unless otherwise stated, addresses refer to this segment value.

Figure 4-2-1 shows the configuration of the system table. The numerals on the left mean offset addresses from the head of the block.

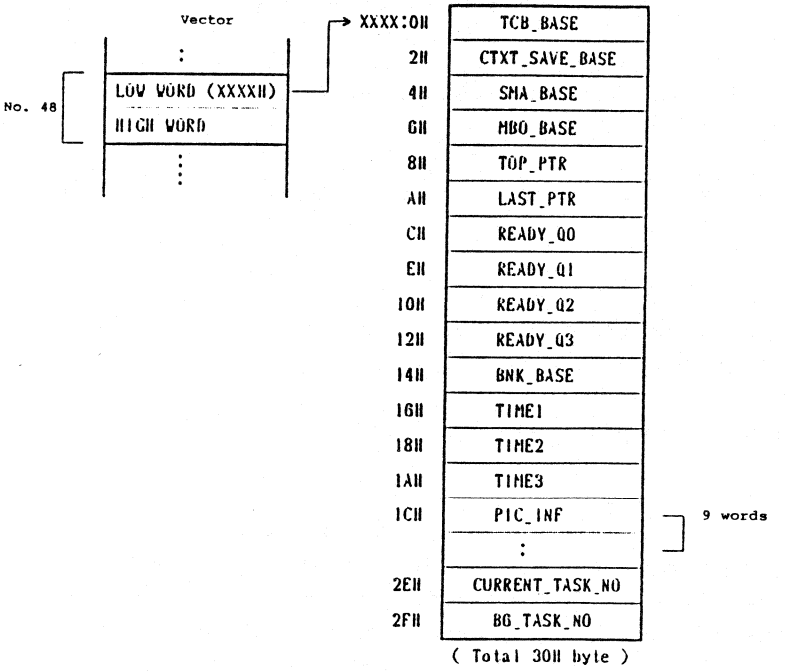


Fig. 4-2-1 Configuration of System Table

TCB_BASE word
This is the base address of the TCB area. The base address is set in TCB_BASE on the assumption that the area starts with TCB of task 0. If the minimum task number is greater than 0, therefore, the base address is obtained by the following formula:

Actual base address - Minimum task number = Base address set to TCB_BASE

CTXT_SAVE_BASE word
This is the base address of the context save area. However, like TCB_BASE, the base address is set on the assumption that the area starts with the context save area of task 6. Therefore, if the minimum task number is greater than 6, the base address is obtained by the following formula:

Actual base address - (Minimum task number - 6) * 2
= Base address set to CTXT_SAVE_BASE

If the context save area is not secured, 0 is set to CTXT_SAVE_BASE.

SMA_BASE word
This is the base address of the semaphore area.

MB0_BASE word
This is the base address of the mail box.

TOP_PTR, LAST_PTR 2 words
These two words form an area that stores information on management of memory block queuing. Unused memory blocks are queue in this area. The base addresses for the memory blocks at the beginning of the queue (TOP_PTR) and at the last of the queue (LAST_PTR) are set in this area. When a task issues a request (GET_MEM) to get a memory block, the memory block at the head of the queue is released

from the queue and is passed over to the task.
Conversely, when a memory block is returned from a task (REL_MEM), it is queued at the end of the queue again.
For queuing, refer to 6) Memory block.
If no memory block is queued, 0 is set for TOP_PTR (LAST_PTR is undefined).

READY_00 - READY_03 4 words
These are ready queues. Each bit of READY_00 to READY_03 corresponds to task numbers 0 to 63 (i.e., each priority level) and indicates whether it is ready or not. When the bit is 1, it is ready; when it is 0, the bit is not ready. Figure 4-2-2 shows the configuration of each ready queue.

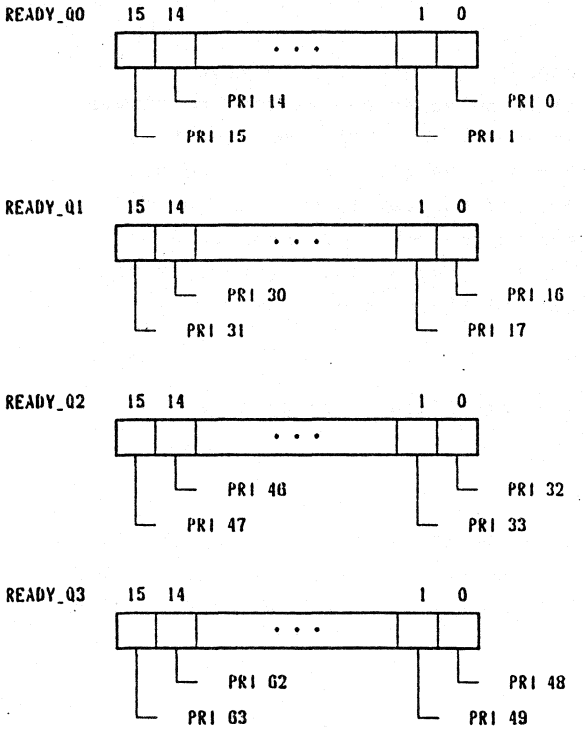


Fig. 4-2-2 Configuration of Ready Queues

BNK_BASE word
This is the base address of the register bank. When the value of INTERNAL RAM BASE which is specified as the set value for special register IDB in the configuration table (refer to Chapter 3) is set to XXH, a value of XXE0H is set to BNK_BASE.

TIME1 - TIME3 3 words
This is system time. These three words consist of 48 bits with TIME1 being lower and TIME3 being higher, and are incremented by 1 each time a timer interrupt occurs. The initial value of the system time is 0, and is incremented immediately after the system has been reset. Moreover, it can be read and set by system calls GET_TIM and SET_TIME.

PIC_INF 9 words
Port addresses corresponding to uPD71059 Interrupt Controllers uPD71059 that serve as the master and slaves 0 through 7 are set as specified in the configuration table (refer to Chapter 3). The first word is the port address for the master, and the remaining eight words respectively correspond to the port addresses for the slaves. When FFFFH is set in any of these words, it means that the corresponding uPD71059 does not exist.

CURRENT_TASK_NO
This word retains the number of the task that is currently running.

BG_TASK_NO
This word retains the number of the task that currently uses register bank 6. If no task is using register bank 6 (i.e., if only tasks having number 5 or less exist), 0 is set in this word.

2) Task control block (TCB)

The RTOS uses TCBs to manage tasks, and one TCB is secured for one task. Figure 4-2-3 shows the configuration of TCB.

0H	WAIT_Q
2H	DMSG_Q_TOP
4H	DMSG_Q_LAST
6H	INITIAL_PC
8H	INITIAL_PS
AH	RES_ADR_PC
CH	RES_ADR_PS
EH	STATUS

(Total FH byte)

Fig. 4-2-3 TCB Configuration

WAIT_Q word

This is a link pointer that is used when the TCB is queued in a semaphore or a mail box while waiting for resources or messages. In this pointer, the base address of a TCB that is to be queued next is set. For details on queuing, refer to 5) Mail box.

DMSG_Q_TOP, DMSG_Q_LAST 2 words

These two words form a management information storage area in which messages sent to tasks by send message direct (SMD_DIR) are queued. The page address of the head of the queue is set in DMSG_Q_TOP, while the page address of the end of the queue is set in DMSG_Q_LAST. Messages are passed over to the task that has been released from the head of the queue by receive message direct (RCV_DIR). If no message is queued, 0 is set in DMSG_Q_TOP (DMSG_Q_LAST is undefined). For details on queuing, refer to 6) Memory block.

INITIAL PC, INITIAL PS 2 words

These two words are the PS and PC values that are used as task start addresses when the RTOS starts tasks (STA_TSK).

The PC and PS values to be set in these words are the initial value of each task that is specified in the configuration table.

RES_ADR_PC, RES_ADR_PS 2 words
These are the PC and PS values that are used as restart addresses when execution is returned from an interrupt destination by system call RES_INT. They are set or released by system call SET_ADR. when they are reset, both RES_ADRE_PC and RES_ADR_PS become logical 0.

STATUS 1 byte
This byte indicates the current statuses of a task. Each bit of STATUS has the following meaning:

bit 0 : indicates whether the task is dormant or not. When this bit is 0, the task is dormant. So, it is normally 1.
bit 1 : indicates whether the task is waiting (for semaphore and mail boxes). When this bit is 1, the task is waiting.
bit 2 : indicates whether the task is suspended. When this bit is 1, the task is suspended.
bit 3 : indicates whether the task is waiting for direct communication. When this bit is 1, the task is waiting.
bit 4 : indicates whether the task is waiting for an interrupt. When this bit is 1, the task is waiting.
bit 5 : indicates whether the current task is occupying the register bank. When this bit is 1, the task is occupying the register bank.
bits 6 & 7: Unused

Note: When the task is ready (or running), bit 0 is 1 and all the other bits except 5 are 0.
Bits 1, 3, and 4 respectively indicate waiting statuses. These bits, however, cannot become 1 at

the same time because a task cannot simultaneously enter more than one waiting status.
Because the semaphore and mail box waiting statuses are indicated by a single bit, for which the task is waiting cannot be determined by checking STATUS alone. To do so, it is necessary to check the status of the queued semaphore and mail boxes.

3) Context save areas

With the RTOS, tasks 0 to 5 always occupy register banks 0 to 5. Tasks numbered 6 or higher are executed, alternately occupying register bank 6. Tasks not occupying register bank 6 are saved to the context save areas. One context save area is secured for each task that is executed on bank 6. The context save area is configured in exactly the same manner as a register bank as Figure 4-2-4 shows.
If tasks numbered 6 or higher do not exist, or if only one such task exists, no context save area is secured because register bank 6 is not alternately used.

0H	RESERVE0
2H	RESERVE1
4H	SAVE_PSW
6H	SAVE_PC
8H	SAVE_DS0
AH	SAVE_SS
CH	SAVE_PS
EH	SAVE_DS1
10H	SAVE_IY
12H	SAVE_IX
14H	SAVE_BP
16H	SAVE_SP
18H	SAVE_BW
1AH	SAVE_DW
1CH	SAVE_CW
1EH	SAVE_AW

(Total 20H byte)

Fig. 4-2-4 Configuration of Context Save Area

RESERVE0, RESERVE1 4 words
These words are not used.

SAVE_PSW - SAVE_AW 12 words
These words form an area to which register values are saved.

4) Semaphore
A semaphore is a resource management block which is used to synchronize or mutually reject tasks. Figure 4-2-5 shows the configuration of a semaphore.

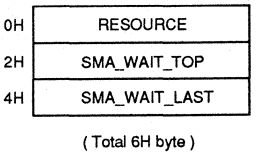
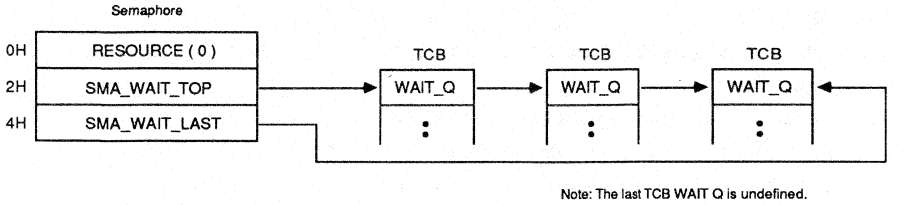


Fig. 4-2-5 Configuration of Semaphore

RESOURCE 1 word
This indicates the number of resources the current semaphore is retaining.

SMA_WAIT_TOP, SMA_WAIT_LAST 2 words
These two words form a management information storage area that is used to queue tasks (TCB) waiting for resources from a semaphore (when RESORCE is 0). The page address of TCB at the head of the queue is set in SMA_WAIT_TOP, while the page address of TCB at the end of the queue is set in SMA_WAIT_LAST.
Figure 4-2-6 shows the queuing status.
Queued tasks are sequentially released from the head of the task each time the resources are released from other tasks (REL_RSC). When a task requests resources (REQ_RSC), and when RESOURCE is 0, it is queue at the end of the queue. If there is no queued task, 0 is set in SMA_WAIT_TOP (SMA_WAIT_LAST is undefined).



5) Mail box

A mail box is a block that manages and controls message communication between tasks. Figure 4-2-7 shows its configuration.

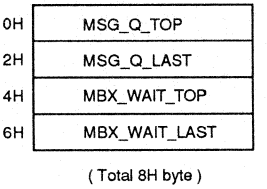


Fig. 4-2-7 Mail Box Configuration

MSG_Q_TOP, MSG_Q_LAST 2 words

These words form a management information storage area that is used to queue messages sent to the mail box by send message (SND_MSG). The page address of the message at the head of the queue is set in MSG_Q_TOP, while that of the message at the end of the queue is set in MSG_Q_LAST. Messages are released from the head of the queue and passed over to tasks by receive message direct (RCV_MSG). If no message is queue, 0 is set in MSG_Q_TOP (MSG_Q_LAST is undefined). For details on queuing, refer to 6) Memory block.

MBX_WAIT_TOP, MBX_WAIT_LAST

This is a management information storage area that is used to queue tasks (TCB) waiting for messages from a mail box (i.e., when there is no message). The page address of the TCB at the head of the queue is set in MBX_

WAIT_TOP, while that o the TCB at the end of the queue is set in MBX_WAIT_LAST. Figure 4-2-8 shows the status of the queue.

Queued tasks are sequentially released from the head of the queue each time messages are sent by other tasks (SND_MSG). If there is no message when a task should receive a message (RCV_MSG), the task is queue at the end of the queue. If there is no waiting task, 0 is set in MBX_WAIT_TOP (MBX_WAIT_LAST is undefined).

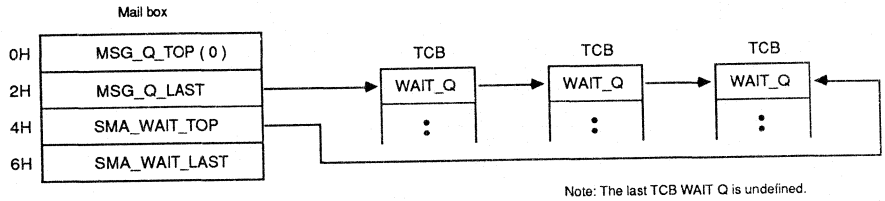


Fig. 4-2-8 Queue Waiting for Mail Box

6) Memory Block

The memory block is a dynamic memory area that can be secured (GET_MEM) or released (REL_MEM) by a task any time. However, the size of the memory block is fixed to the size specified by the user in the configuration table. Normally, the memory block is used to store messages when they are communicated, but it is also used as a temporary work area of tasks. Figure 4-2-9 shows the configuration of the memory block.

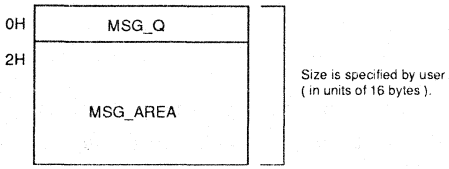


Fig. 4-2-9 Memory Block Configuration

MSG_Q 1 word
This is a link pointer that is used to queue memory blocks to a mail box (SND_MSG) or TCB (SND_DIK) when messages are sent. It is also used to queue unused memory blocks to the system table. Therefore, the first one word of the memory block cannot be used as a message area. Figures 4-2-10, 4-2-11, and F-2-12 show the respective queuing statuses.

Note: When a memory block is used as the work area of a task, MSG_Q is not necessary because the entire memory block can be used as a work area.

MSG AREA user-specified size - 1 word
This is an area tasks use freely as a message area.

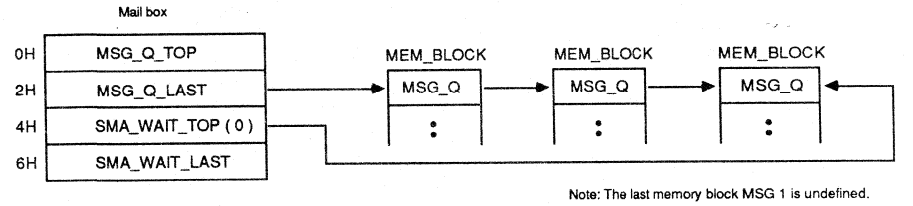


Fig. 4-2-10 Mail Box Message Queue

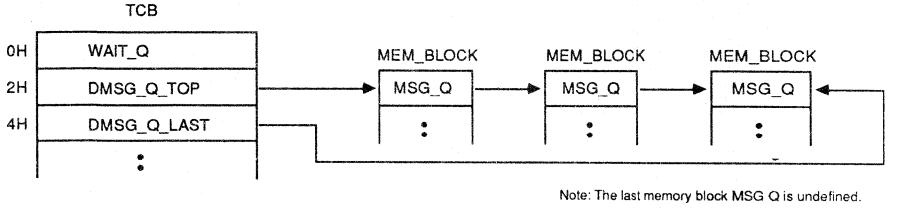


Fig. 4-2-11 TCB Message Queue

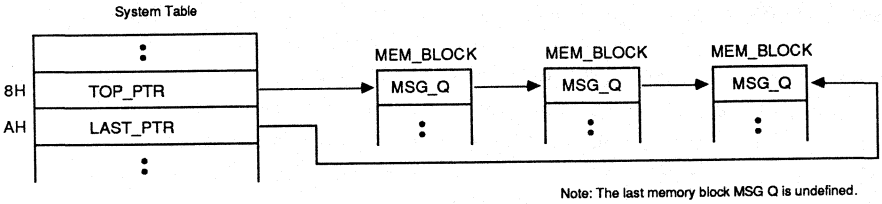


Fig. 4-2-12 System Table Vacant Message Queue

CHAPTER 5 INITIALIZATION PROCESSING BY RTOS

The RTOS system is initialized by the reset routine stored in the internal ROM of the uPD79011 together with the OS nucleus. Since a jump instruction which causes execution jumps to the reset routine is set in address FFFF0H, the reset address of the uPD79011, execution of the reset routine is started as soon as the system has been reset and the system is initialized according to the information of the configuration table stored in memory from address FB800H.
This chapter discusses the initialization processing by the RTOS.

5.1 Reset Routine Processing

The reset routine performs the following processes:

- Initialization of the uPD79011 special registers
- Initialization of register banks
- Initialization of interrupt vector table
- Generation and initialization of system table, etc.
- Execution of system

5.1.1 Initializing special registers

The reset routine initializes the following special registers:

1) IDB (address FFFF:FH)
   A value specified in the configuration table is set in this register (refer to INTERNAL RAM BASE in Chapter 3).
2) PRC (address X00H:FEBH)
   A value specified in the configuration table is set in this register (refer to PRC INF0 in Chapter 3).
3) TBIC (address XX00H:FECH)
   Bit 0 is reset to 0 to enable a timer interrupt (which is used to increment the system clock). Actually, however, the timer interrupt takes place after control has been shifted to the user task because the reset routine itself is executed with the interrupt disabled.
   Note: XX in address is the value set in IDB.

5.1.2 Initializing register banks

The reset routine initializes register banks 0 through 6 as follows:

1) When register banks are used by user task
   Registers PS, PC, SS, SP, and DS0 are initialized to the values specified in the configuration table. In the PSW register, X202H is set (X can be 8 to E depending on the bank number). The other registers become undefined because no values are set in them.
2) When register banks are used by idle task
   The start address of the idle task is set in PS and PC and 0 is set in DS0. The values specified in the configuration table are set in SS and SP. In PSW and other registers, the same values as when the register banks are used by a user task are set.

Note: Register banks 0 to 5 are used by tasks 0 to 5 and register bank 6 are used by other tasks. In the register bank to which no task corresponds, nothing is set. Moreover, in register bank 6, the task having the least significant number of the tasks that use register bank 6 is set. Registers for the other tasks are saved to context save areas in the same manner as above.

Moreover, the SS, SP, and PSW registers of the register bank used by the reset routine and RTOS are initialized as follows:

. SS:SP

The SS and SP values of an idle task are set in these registers when the reset routine is executed. Afterward, the SS and SP values of a user task are copied to them each time a system call is issued.

. PSW

Since interrupts are disabled while the reset routine and RTOS are being executed, F002H is set in this register as an initial value.

5.1.3 Initializing interrupt vector table

The interrupt vector table is initialized as follows:

1) The address of RTOS timer interrupt process is set in vector 31.
2) The base address of the system table is set in one low word of vector 48.
3) The address of the default interrupt process (handler for IRET only) is set for all the interrupt types other than above.

5.1.4 Generating and initializing system table, etc.

The reset routine secures areas for the system table and TCBs as discussed in Chapter 4 and initializes each block.

5.1.5 Executing system

The reset routine shifts control to the RTOS and executes the system after all the above initialization processes have been completed. At this time, only the following two tasks can be executed on the system. The other tasks are all dormant.

. Initial task

This is the user task having the number specified as an initial task in the configuration table. It is executed first in the system and then the other tasks will enter the executable status.

. Idle task

This is a task reserved by the system. It is generated, being provided with the highest task number but with the lowest priority. Consequently, the idle task is executed only when there is no other executable task.
The source code of the idle task is shown below.

Idle task source code

```
IDLE:
        EL          : enables interrupt
        HALT        : places uPD79011 in HALT status
        BR  IDLE    : returns to beginning
```

As can be seen from the source code, the idle task, when executed, enables interrupts and executes the HALT instruction. At this point, the user tasks are all waiting, suspended, or dormant. To return control user tasks again, therefore, one of the user tasks must be started by an interrupt. Note, however, that only system call SIG_INT can release the task waiting status in an interrupt handler. Therefore, there is at least one user task that is waiting for an interrupt (refer to 6.4.2 System call restrictions in interrupt handler).

5.2 Software Reset

Normally, the reset routine is started when the hardware is reset and performs the initialization processes discussed in 5.1. However, it can also restart the system when the software is reset by a user task (by jumping execution to address FFFF0H). In doing so, however, pay attention to the following points.

When hardware reset is performed, most special registers of the uPD79011 are initialized to a specific value. However, when software reset is executed, registers other than IDB, FR, and TBIC which are set by the reset routine retain the values before reset is executed.
To perform software reset, therefore, the special registers to be used must always set to their specific values in advance, or they must be initialized before reset is executed.
Especially register ISPR calls for your attention. Each bit of this register indicates the servicing status of an interrupt whose level corresponds to that bit (servicing is in progress when the bit is ON). Interrupts having a level

lower than the level indicated by a bit that turns ON cannot be accepted. For this reason, if software reset is executed while a bit of ISPR is ON, normal interrupt operation will not be performed, unless ISPR is cleared by the initial task (by executing the FINT instruction). Other than ISPR, any register related to interrupt must be carefully used.

CHAPTER 6 APPLICATION PROGRAM DESCRIPTION

This chapter discusses the method and points to be noted when describing application programs.

6.1 Task Configuration in Load Module

Application tasks for the RTOS are described in C and assembler and can be either of the two program format:

1) One load module per task
2) One load module for several tasks

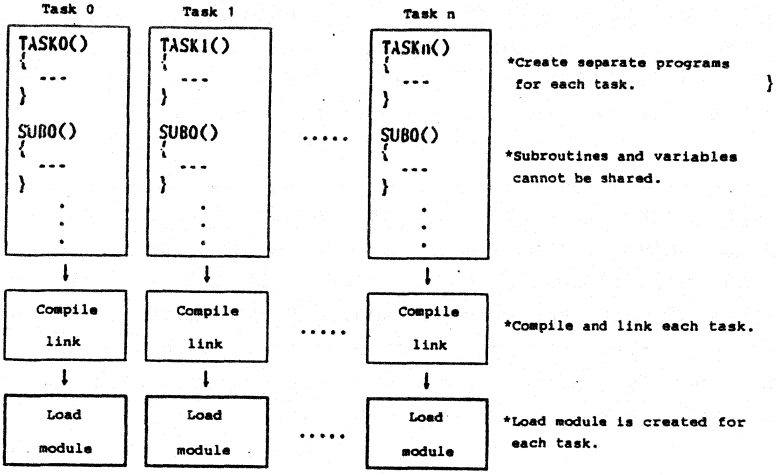
If one load module is to be created per task, application programs can be described in the same manner as ordinary programs. If one load module is to be created for several tasks, however, application programs must be described as independent functions (in C) or subroutines (in assembler) that will not be called. Normally, such functions or subroutines will not be executed. But the RTOS can execute them by registering the entry addresses of the functions or subroutines in the configuration table as task start addresses (PS:PC).

When one load module is created for several tasks, subroutines and variables can be shared by several tasks. In this case, however, it is necessary to synchronize the tasks when the shared variables are manipulated.

Moreover, when the application program is described in C, function main() is usually used as the main routine. However, this is not always necessary regardless of whether the load module is created for each task or for several tasks (except when a problem such as that linking cannot be performed occurs). Rather, if main() is used, programs and data unnecessary for executing the application program on the HSS such as a startup routine are appended by the compiler, significantly increasing the program size.

Figure 6-1-1 shows examples for task creating procedures in both the above cases 1) and 2).

1) To create one load module for one task



2) To create load module for several tasks

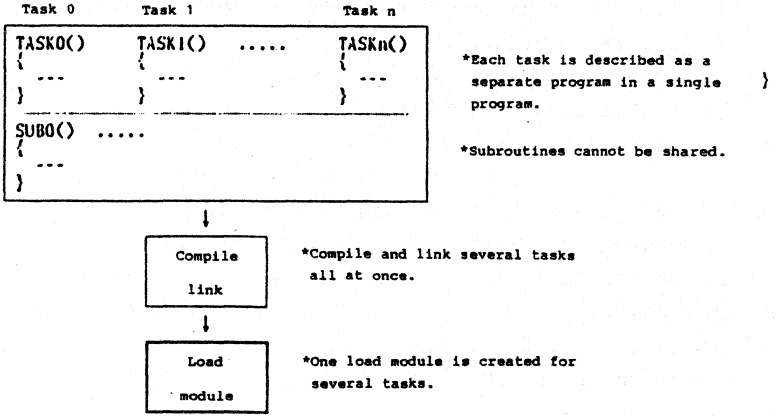


Fig. 6-1-1 Task Creating Procedure

6.2 Task Information

In addition to the program of each task, a configuration table must be created when organizing a system on the RTOS. At this time, DS0 and SS:SP must be specified as the context information of each task. The value of these are determined as follows:

1) PS:PC
   Of the start addresses of the function indicated in the map file that has been created when load modules are linked, specify the start address of the function that serves as the main routine of each task when the application program is described in C. When the application program is described in assembler, specify the label address of the subroutine that serves as the main routine of each task.
2) DS0
   In the same way as PS:PC, specify the data segment value indicated in the map file. If there is more than one data segment, that is, when C having large memory model is used (the value of DS0 differs depending on the function), or if several data segments are defined in assembler, specify the data segment value that is used by the function or subroutine that serves as the main routine of each task.
3) SS:SP
   The stack area pointed to by SS:SP can be basically any vacant memory area. Secure sufficiently large stack area, giving consideration to the function in each task and the nesting status of subroutines. If C having small or medium memory model is used, DS0 may be used to reference variables on the stack; so, set the same value as DS0 in SS.

6.3 Notes on Pointers of Application Programs in C

When the application program is described in C, the pointer size varies as shown in Table 1 depending on the memory model to be used for compiling, resulting mismatching with the RTOS or making operation difficult.

Table 1 Pointer Size of Each Memory Model

Memory model	Pointer size	
	Pointer to functions	Pointer to variables
Small	16 bits (CALL)	16 bits
Compact	16 bits (CALL)	32 bits
Medium	32 bits (CALLF)	16 bits
Big	32 bits (CALLF)	32 bits
Large	32 bits (CALLF)	32 bits

As can be seen from Table 1, the small and compact memory models provide the pointer to functions which is made up of only 16 bits (OFFSET), and the small and medium memory models offer only 16-bit pointer to variables. However, the RTOS is designed to process 32-bit pointers. Therefore, if the small, compact, and medium memory models are used, there are the following restrictions:

1) The memory block that has been obtained by such system calls as GET_MEM and RCV_MSG cannot be directly manipulated with pointers as it is (small and medium).

   If a pointer (SEGMENT:OFFSET) is generated from the segment value of the memory block that can be obtained by such system calls as GET_MEM (OFFSET should be 0) with a memory model other than the small and medium, data can be directly written to or read from the obtained memory block by the pointer. Moreover, the memory block can be treated as a structure if the generated pointer is used as the one to a structure.
   However, with the small or medium memory model, pointers cannot manipulate areas other than the data area of their own tasks because the pointers consist of offset only (DS0 is used as SEGMENT). To manipulate a memory block by a pointer, therefore, an additional process such as exchanging DS0 with the segment value of the obtained memory block is necessary.

2) When a parameter (32-bit pointer) is passed over to system calls SET_TIM or GET_TIM, data segment value or stack segment value must be also passed in addition to the pointer to variables (small and medium).

SET_TIM and GET_TIM requires 32-bit pointer (SEGMENT:OFFSET) that points to an area to in which the system time to be set is written or in which obtained system time should be written. With the small and medium memory models, therefore, a segment value must be passed over to the system calls in addition to a pointer (OFFSET).
The segment value to be passed is a stack segment value if the abovementioned area is an internal variable; if it is an external variable or static variable, a data segment value must be passed over to the system calls. These segment values must be either be obtained from the SS and DS0 register in assembler, or passed to GET_TIM and SET_TIME by directly specifying an immediate value.
Figure 8-9-1 shows an example of using SET_TIM with the compact (32-bit pointer), small, and medium models.

```
struct time {
            int time_l ;
            int time_m ;
            int time_h ;
          } sys_time ;
```

```
/* COMPACT */

c_sub()
{
  int cc ;

  sys_time.time_h = 0x1000 ;
  sys_time.time_m = 0x2000 ;
  sys_time.time_l = 0x3000 ;

  cc = SET_TIM( &sys_time ) ;

  }
```

```
/* SMALL or MEDIUM */

s_sub()
{
  int cc , ds0_val ;

  sys_time.time_h = 0x1000 ;
  sys_time.time_m = 0x2000 ;
  sys_time.time_l = 0x3000 ;

  ds0_val = GET_DS0() ;
  cc = SET_TIM( &sys_time ,
                ds0_val ) ;
  }
```

Note

GET_DS0() : Function described in assembler that returns the value of DS0

Fig. 6-3-1 Example of Issuing SET TIM

3) When a parameter (32-bit pointer) is passed to system calls DEF_INT and SET_ADR, a code segment value must be also passed in addition to an interrupt handler and a pointer to the restart position (small and compact).

In the same manner as 2), DEF_INT and SET_ADR requires 32 bits (SEGMENT:OFFSET) for the pointer to the interrupt handler and restart address to be defined. With the small and compact memory models, therefore, a code segment value must be passed to the system calls in addition to the pointer (OFFSET). Figure 6-3-1 shows an example of using DEF_INT with the small and compact models.

```
/* BIG */

b_sub()
{
  int cc ;

  cc =  DEF_INT( 0x10 ,
                 INT_HNDR ) ;
}
```

```
/* SMALL or COMPACT */

s_sub()
{
  int cc , cs_val ;

  cs_val = GET_CS() ;
  cc = DEF_INT( 0x10 ,
          INT_HNDR , cs_val ) ;
}
```

Note

GET_CS() : Function described in assembler that returns the value of CS

INT_HNDR() : Interrupt handler

Fig. 6-3-2 Example of Issuing DEF_INT

6.4 Notes on Describing Interrupt Processes

The RTOS supports 15 interrupt sources such as the interrupt requests of the Interrupt Controllers uPD71059 and uPD79011. The interrupt level of the uPD71059 can be expanded up to 64 when the system is configured of a master and slaves. This section discusses the points to be noted when describing interrupt handlers that are started by these interrupt sources.

6.4.1 Nesting management of interrupt handler

To describe several interrupt processes, nesting must be managed when execution is returned from the interrupt handler. However, the system calls SIG_INT and RES_INT of the RTOS, which returns execution from an interrupt handler, does not judge or manipulate the ISR (in service register) of uPD71059 Interrupt Controller and the special register ISPR (in service priority register) of the uPD79011 that controls interrupt priority. Therefore, nesting of the interrupt handler must be all managed by an interrupt handler that is described by the user.

Note: Nesting of a system call does not take place because all interrupts are disabled while the system calls of the HSS are being processed.

Figure 6-4-1 shows an example of a return process by an interrupt handler when the uPD71059 is the interrupt source (when the system is configured of a master and slaves). Figure 6-4-2 shows another example when the uPD79011 is the interrupt source.

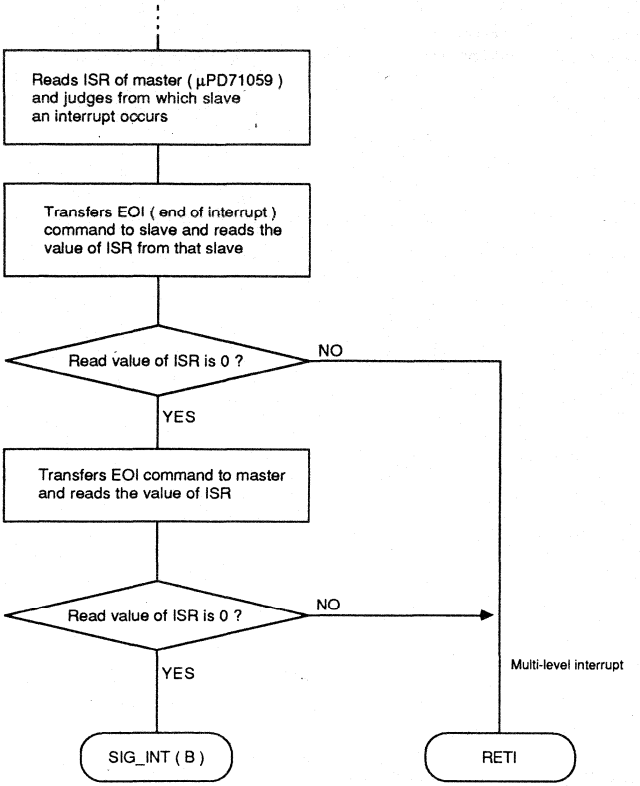


Fig. 6-4-1 (a) Example of Returning Process of Interrupt Handler

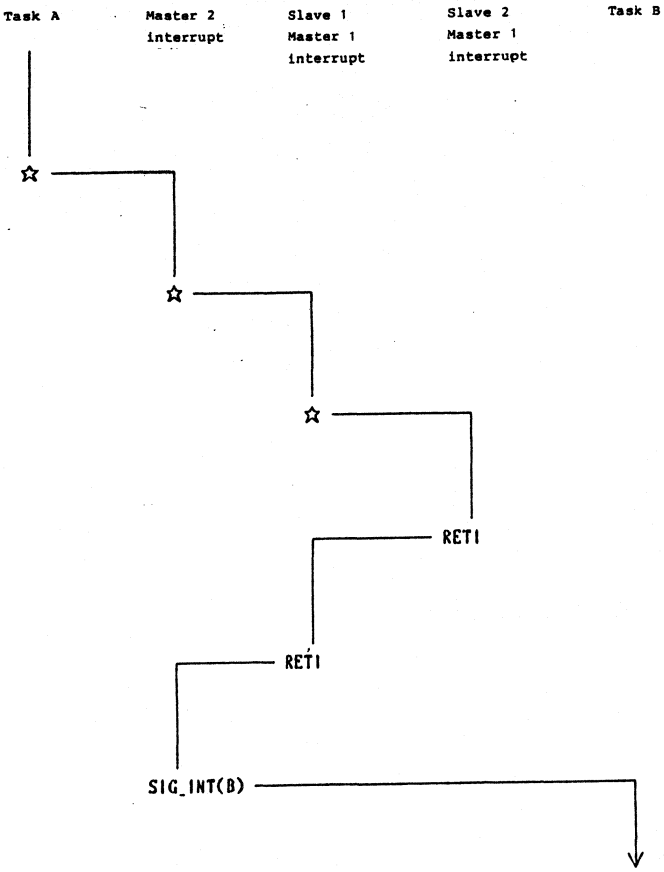


Fig. 6-4-1 (b) Flow of Control

Figure 6-4-1 (a) is an example of returning process by an interrupt handler whose interrupt source is the uPD71059 (with an interrupt controller configured of a master and slaves, it is assumed that the master is initialized in special free nested mode and slaves are all in free nested mode). In this case, there is not nested interrupt when the ISRs of all the slaves are 0. In the figure, control is shifted to a task when there is not nested interrupt and, while an interrupt is nested, the interrupt process is aborted by transferring control to the point where the interrupt has occurred.

Figure 6-4-1 (b) shows the flow of control in the above process. During a multi-level interrupt, control is returned to the point where the interrupt has occurred as soon as the RETI instruction is executed. Control will then be shifted to task B by system call SIG_INT when there is no nesting of the interrupt handler.

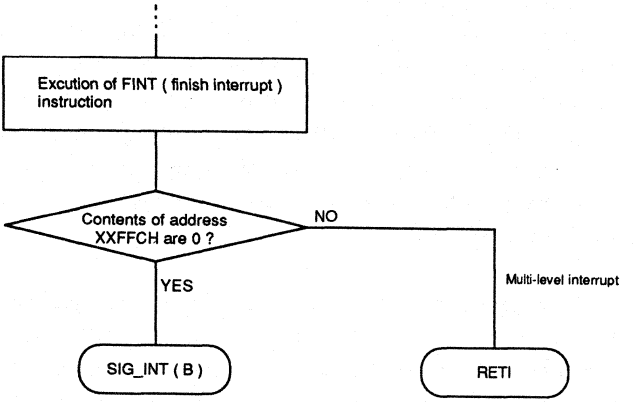


Fig. 6-4-2 (a) Example of Returning Process of Interrupt Handler

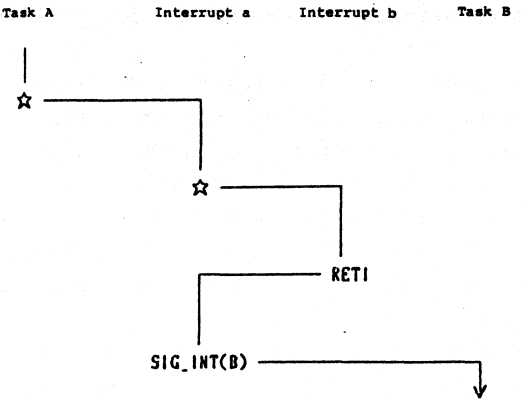


Figure 6-4-2 (b) Flow of Control

Multi-level interrupt must be processed in the interrupt handler when an interrupt request is generated from the uPD79011. Figure 6-4-2 (a) shows an example of returning process that must be performed in that occasion. After the FINT instruction has been executed, the contents of address XXFFCH are read. If the contents are 0, there is no nesting in the handler. If the contents are not 0, it is judged that nesting has taken place, and control is returned by RETI to the point where the interrupt has occurred so that the handler is not aborted.

Note: Address XXFFCH is special register ISPR that manages the priority (levels 0 to 7. 0 indicates the top priority) of the interrupt request of the uPD79011. Each bit indicates the level of the interrupt currently being serviced. When the bit is ON, it indicates that the interrupt is being serviced. These bits are assigned to the priority levels in ascending order with bit 0 corresponding to level 0. XX of the 8 bits of the higher address of ISPR is the value specified by IDB.

When the above process is illustrated by Figure 6-4-2 (b). Interrupt a occurs while task A is running. If interrupt b, which

is assigned a higher priority, occurs while interrupt a is being processed by a handler, control is immediately returned by RETI to the position where the interrupt has occurred at the end of interrupt b, and control is shifted to task B by SIG_INT at the end of interrupt a.
The above process needs not be performed in a system in which consideration does not have to be given to multi-level interrupt, and only system call SIG_INT should be described at the end of an interrupt handler.

6.4.2 System call control in interrupt handler
The HSS limits the number of system calls that can be issued in an interrupt handler to the following four. All the other system calls can be issued from tasks only and thus must not be used in an interrupt handler.

SIG_INT	Wakes up tasks waiting for an interrupt
RES_INT	Completes the interrupt handler and returns control to the restart address
DIS_INT	Disables interrupts
ENA_INT	Enables interrupts

Of the above system calls, SIG_IN and RES_INT can be issued in an interrupt handler only. DIS_INT and ENA_INT can be issued from both the handler and tasks.

6.5 Creating Load Module

This section shows examples of creating tasks using CC70116 C Compiler (small) and RA70116 Assembler.

1) Compile and assemble

First, compile and assemble all the necessary C programs and assembler programs as follows (TSK_C.C is a C source file and TSK_A.ASM is an assembler source file):

```
A>CC70116 B:TSK_C.C          ← Compiles TSK_C.C in drive B
CC70116 COMPILER V1.0 [17 Sep 84]
  Copyright (C) 1984 NEC Corporation

A>RA70116 B:TSK_A.ASM        ← Assembles TSK_A.ASM in drive B

uPD70108/70116 ASSEMBLER V1.1 [17 Apr 85]
  Copyright (C) 1984 NEC Corporation

ASSEMBLER COMPLETE,    0 ERROR(S) FOUND

A>
```

In this example, REL files such as TSK_C.REL and TSK_A.REL are created in drive B.

2) Link

The REL files created in step 1) above are linked using LK70320.

```
A>LK70116 CC70116X,B:TSK_C,B:CC70116.LIB TO B:TSK_C

uPD70108/70116 LINKER V1.0 [20 Feb 85]
  Copyright (C) 1984 NEC Corporation

LINK COMPLETE

A>LK70116 B:TSK_A TO B:TSK_A

uPD70108/70116 LINKER V1.0 [20 Feb 85]
  Copyright (C) 1984 NEC Corporation

LINK COMPLETE

A>
```

In the above example, load module files TSK_C.LNK and TSK_A.LNK and map files TSK_C.MAP and TSK_A.MAP are created from TSK_C.REL and TSK_A.REL (CC70116X and CC70116.LIB are a startup module and library, respectively).

If load module files and map files are created in this way, however, location addresses for code segment, etc., are

determined by the linker. To locate the files in desired addresses, link them by using the following command file.

```
A>LK70116 | LINK.CMD    ← Command file

uPD70108/70116 LINKER V1.0 [20 Feb 85]
  Copyright (C) 1984 NEC Corporation

LINK COMPLETE

A>
```

* LINK.CMD

```
CC70116X,B:TSK_C,CC70116.LIB
TO B:TSK_C
ORDER(SEGMENTS(CODE,DATA))
ADDRESSES(SEGMENTS(CODE(10000H),DATA(20000H)))
```

Command file LINK.CMD specifies the location address of segments 'CODE' and 'DATA'.

Four segments are included in the REL files generated by CC70116 (small): CODE, DATA, CONST, and STACK. The CODE segment belongs to group CGROUP, while the others belong to DGROUP. In the above example, the sequence of CODE and DATA is specified by ORDER and the location address is specified by ADDRESSES.

When the assembler is used, specify the segment names set by the user in the same manner as above to locate the files at desired addresses.
The following shows how the context information of a task specified in the configuration table is obtained from the map files that are created when load module files are linked.

```
**  GROUP MAP  **

  GROUP          BASE  SEGMENTS

  CGROUP         1000  CODE
  DGROUP         2000  DATA          CONST         STACK
```

The value of PS and DS0 can be obtained from GROUP MAP in the above map files. In the above example, PS is 1000H and DS0 is 2000H because PS and DS0 only have to indicate CGROUP and DGROUP, respectively.

The value of PC is obtained from PUBLIC SYMBOL LIST in the map files.

```
##  PUBLIC SYMBOL LIST  ##

  SYMBOL          MOD   ADDRESS        SYMBOL          MOD  ...

  START             1   1000:0000      _CAN_INT          3  ...
  _TASK1            3   1000:0100      _DIS_INT          3  ...
  _TASK2            3   1000:0350      _CC               3  ...
  _TASK3            3   1000:0500          .                .
    .                       .             .                .
    .                       .             .                .
```

In PUBLIC SYMBOL LIST shown above, the address of each symbol is indicated as SEGMENT:OFFSET. Therefore, the offset value of the symbol of the main routine name of the task is the PC value of that task. Of course, the value of PS can also be obtained from a segment value.

This example shows how load module files are linked where several tasks are described in one load module file. TASK1, TASK2, and TASK3 serve as the main routine of each task. Therefore, The PC value of each task is 100H for TASK1, 350H for TASK2, and 500H for TASK3. When one task is described in one load module, the PC value can be obtained in the same manner except that only one symbol is referenced.

3) Creating HEX file

The LNK file created in step 2) above is a load module file. This file is converted into a HEX file, which is more generally used, by using OC70320.

```
A>OC70116 B:TSK_C TO B:TSK_C

uPD70108/70116 OBJECT CONERTER V1.1 [22 Apr 85]
  Copyright (C) 1984 NEC Corporation

OBJECT CONVERSION COMPLETE

A>
```

The above process creates load module file TSK C.H86 from TSK_C.LNK. In the same way, TSK_A.H86 can be obtained from TSK_A.LNK.

The load module files created above are downloaded to the target system.

APPENDIX 4A

List of RTOS Error Codes

Error code	Meaning
0 E OK	Normal completion
1 E DMT	The task is not dormant when system call STA_TSK is executed. It is dormant when system call SUS_TSK or RSM_TSK is executed.
2 E SUS	The task is suspended when system call SUS_TSK is executed. It is not suspended when system call RSM_TSK is executed.
3 E BLK	No memory block
4 E DVN	Device number error
5 E SYS	The big port address is undefined.
6 E RSC	The number of resources is 0.
7 E MSG	No message
8 E INT	The task is not waiting for an interrupt.

List of RTOS System Call Names and System Call Numbers

System call name	System call number	System call name	System call number
STA_TSK	0	DIR SND	13
EXI_TSK	1	GET MEM	14
SUS_TSK	2	REL MEM	15
RSM_TSK	3	GET TIM	16
SET_ADR	4	SET TIM	17
REQ_RSC	5	DEF INT	18
POL_RSC	6	SIG INT	19
REL_RSC	7	WAI INT	20
RCV_MSG	8	CAN INT	21
POL_MSG	9	DIS INT	22
SND_MSG	10	ENA INT	23
DIR_RCV	11	RES INT	24
POL_DIR	12		

List of RTOS Reserved Interrupt Vectors

The interrupt vectors the RTOS uses to operate are as follows. Do not use the vectors reserved for the RTOS.

Vector number	
31	For system clock (time base count)
48	For RTOS data
56 to 63	For master uPD71059
64 to 71	For slave uPD71059 connected to master IR0
72 to 79	For slave uPD71059 connected to master IR1
80 to 87	For slave uPD71059 connected to master IR2
88 to 95	For slave uPD71059 connected to master IR3
96 to 103	For slave uPD71059 connected to master IR4
104 to 111	For slave uPD71059 connected to master IR5
112 to 119	For slave uPD71059 connected to master IR6
120 to 127	For slave uPD71059 connected to master IR7

List of RTOS Interface Library

The following pages show the RTOS interface library which can be linked together with RTOS-tasks compiled with C-compiler (CC70116) in small memory model.

For RTOS-tasks compiled in large memory model, please take care regarding the additional '_' sign for the procedure labels and the different pointers.
See following example for the system call EXI_TSK:

small memory model:

```
        EXI_TSK         PROC    NEAR
_EXI_TSK:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,1
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        EXI_TSK         ENDP
```

large memory model:

```
        EXI_TSK         PROC    FAR
__EXI_TSK:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,4
        PUSH    BP
        MOV     BP,AW
        MOV     AW,1
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        EXI_TSK         ENDP
```

```
;*****************************************************************;
;*              RTOS   INTERFACE   LIBRARY                       *;
;*****************************************************************;
;-----------------------------------------------------------------;
                PUBLIC          _STA_TSK
;-----------------------------------------------------------------;
        STA_TSK         PROC    NEAR
_STA_TSK:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,0                    ; CODE NO.
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
        ADD     SP,2
        RET
;
        STA_TSK         ENDP
;-----------------------------------------------------------------;
                PUBLIC          _EXI_TSK
;-----------------------------------------------------------------;
        EXI_TSK         PROC    NEAR
_EXI_TSK:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,1
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        EXI_TSK         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _SUS_TSK
;-------------------------------------------------------------------------;
        SUS_TSK         PROC    NEAR
_SUS_TSK:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,2
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        SUS_TSK         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _RSM_TSK
;-------------------------------------------------------------------------;
        RSM_TSK         PROC    NEAR
_RSM_TSK:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,3
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        RSM_TSK         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _SET_ADR
;-------------------------------------------------------------------------;
        SET_ADR         PROC    NEAR
_SET_ADR:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,4
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        SET_ADR         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _REQ_RSC
;-------------------------------------------------------------------------;
        REQ_RSC         PROC    NEAR
_REQ_RSC:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,5
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        REQ_RSC         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _POL_RSC
;-------------------------------------------------------------------------;
        POL_RSC         PROC    NEAR
_POL_RSC:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,6
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        POL_RSC         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _REL_RSC
;-------------------------------------------------------------------------;
        REL_RSC         PROC    NEAR
_REL_RSC:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,7
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        REL_RSC         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _RCV_MSG
;-------------------------------------------------------------------------;
        RCV_MSG         PROC    NEAR
_RCV_MSG:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,8
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        RCV_MSG         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _POL_MSG
;-------------------------------------------------------------------------;
        POL_MSG         PROC    NEAR
_POL_MSG:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,9
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        POL_MSG         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _SND_MSG
;-------------------------------------------------------------------------;
        SND_MSG         PROC    NEAR
_SND_MSG:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,10
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        SND_MSG         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _RCV_DIR
;-------------------------------------------------------------------------;
        RCV_DIR         PROC    NEAR
_RCV_DIR:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,11
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        RCV_DIR         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _POL_DIR
;-------------------------------------------------------------------------;
        POL_DIR         PROC    NEAR
_POL_DIR:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,12
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        POL_DIR         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _SND_DIR
;-------------------------------------------------------------------------;
        SND_DIR         PROC    NEAR
_SND_DIR:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,13
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        SND_DIR         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _GET_MEM
;-------------------------------------------------------------------------;
        GET_MEM         PROC    NEAR
_GET_MEM:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,14
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        GET_MEM         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _REL_MEM
;-------------------------------------------------------------------------;
        REL_MEM         PROC    NEAR
_REL_MEM:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,15
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        REL_MEM         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _GET_TIM
;-------------------------------------------------------------------------;
        GET_TIM         PROC    NEAR
_GET_TIM:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,16
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        GET_TIM         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _SET_TIM
;-------------------------------------------------------------------------;
        SET_TIM         PROC    NEAR
_SET_TIM:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,17
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        SET_TIM         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _DEF_INT
;-------------------------------------------------------------------------;
        DEF_INT         PROC    NEAR
_DEF_INT:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,18
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        DEF_INT         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _SIG_INT
;-------------------------------------------------------------------------;
        SIG_INT         PROC    NEAR
_SIG_INT:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,19
        ; CALLF RTOS_ENTRY
        DB      0EAH
        DW      0C00EH
        DW      0F000H
        ADD     SP,2
        RET
;
        SIG_INT         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _WAI_INT
;-------------------------------------------------------------------------;
        WAI_INT         PROC    NEAR
_WAI_INT:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,20
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        WAI_INT         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _CAN_INT
;-------------------------------------------------------------------------;
        CAN_INT         PROC    NEAR
_CAN_INT:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,21
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        CAN_INT         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _DIS_INT
;-------------------------------------------------------------------------;
        DIS_INT         PROC    NEAR
_DIS_INT:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,22
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        DIS_INT         ENDP
;-------------------------------------------------------------------------;
                PUBLIC          _ENA_INT
;-------------------------------------------------------------------------;
        ENA_INT         PROC    NEAR
_ENA_INT:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,23
        ; CALLF RTOS_ENTRY
        DB      9AH
        DW      0C000H
        DW      0F000H
;
        ADD     SP,2
        RET
;
        ENA_INT         ENDP
```

```
;-------------------------------------------------------------------------;
                PUBLIC          _RES_INT
;-------------------------------------------------------------------------;
        RES_INT         PROC    NEAR
_RES_INT:
        MOV     AW,BP
        MOV     BP,SP
        ADD     BP,2
        PUSH    BP
        MOV     BP,AW
        MOV     AW,24
        ; CALLF RTOS_ENTRY
        DB      OEAH
        DW      OCO2OH
        DW      OFOOOH
        ADD     SP,2
        RET
;
        RES_INT         ENDP
```

Part 5.

μPD 70320/70322
μPD 70330/70332
μPD79011

INSTRUCTION SET

User´s Manual

The following sections include instruction formats, descriptions, and examples

Table 1. ***Operand Types***

Identifier	Description
reg	8- or 16-bit general-purpose register
reg8	8-bit general-purpose register
reg16	16-bit general-purpose register
mem	8- or 16-bit memory location
mem8	8-bit memory location
mem16	16-bit memory location
mem32	32-bit memory location
dmem	16-bit direct memory address
imm	8- or 16-bit immediate data
imm3	3-bit immediate data
imm4	4-bit immediate data
imm8	8-bit immediate data
imm16	16-bit immediate data
acc	AW or AL accumulator
sreg	Segment register
src-table	Name of 256-byte translation table
src-block	Name of source block addressed by IX register
dst-block	Name of destination block addressed by IY register
near-proc	Procedure within the current program segment
far-proc	Procedure located in another program segment
near-label	Label in current program segment
short-label	Label within range of −128 or +127 bytes from end of instruction
far-label	Label in another program segment
regptr16	16-bit general-purpose register containing an offset within the current program segment
memptr16	16-bit memory address containing an offset within the current program segment
memptr32	32-bit memory address containing the offset and segment data of another program segment
pop-value	Number of bytes of the stack to be discarded (0-64K, usually even addresses)
fp-op	Immediate value to identify instruction code of the external floating point processor chip
R	Register set (AW, BW, CW, DW, SP, BP, IX, IY)
DS1-spec	(1) DS_1 (2) Segment of group name assumed to DS_1
Seg-spec	(1) Any name or segment register (2) Segment or group name assumed to segment register
[ ]	Optional, may be omitted

Table 2. Instruction Words

Identifier	Description
W	Word/Byte specification bit (1 = word, 0 = byte)
reg	8/16-bit general register specification bit (000-111)
mod,mem	Memory addressing specification bits (mod = 00-10, mem = 000-111)
(disp-low)	Optional 16-bit displacement lower byte
(disp-high)	Optional 16-bit displacement higher byte
disp-low	16-bit displacement lower byte for PC relative addition
disp-high	16-bit displacement higher byte for PC relative addition
imm3	3-bit immediate data
imm4	4-bit immediate data
imm8	8-bit immediate data
imm16-low	16-bit immediate data lower byte
imm16-high	16-bit immediate data higher byte
addr-low	16-bit direct address lower byte
addr-high	16-bit direct address higher byte
sreg	Segment register specification bit
s	Sign-extension specification bit (1 = sign extension, 0 = no sign extension)
offset-low	Low byte of 16-bit offset data loaded to PC
offset-high	High byte of 16-bit offset data loaded to PC
seg-low	Low byte of 16-bit segment data loaded to PS
pop-value-low	Low byte of 16-bit data which specifies number of bytes of stack to be discarded
pop-value-high	High byte of 16-bit data which specifies number of bytes of stack to be discarded
disp8	8-bit displacement added to PC
X XXX YYY ZZZ	Operation codes for external floating point processor chip

Table 3 Operation Description

Identifier	Description
AW	Accumulator (16 bits)
AH	Accumulator (high byte)
AL	Accumulator (low byte)
BW	BW register (16 bits)
CW	CW register (16 bits)
CL	CL register (low byte)
DW	DW register (16 bits)
SP	Stack pointer (16 bits)
PC	Program counter (16 bits)
PSW	Program status word (16 bits)
IX	Index register (source) (16 bits)
PS	Program segment register (16 bits)
DS1	Data segment 1 register (16 bits)
DS0	Data segment 0 register (16 bits)
SS	Stack segment register (16 bits)
AC	Auxiliary carry flag
CY	Carry flag
P	Parity flag
S	Sign flag
Z	Zero flag
DIR	Direction flag
IE	Interrupt enable flag
V	Overflow flag
BRK	Break flag
(...)	Values in parentheses are memory contents
disp	Displacement (8 or 16 bits)
temp	Temporary register (8, 16, or 32 bits)
seg	Immediate segment data (16 bits)
offset	Immediate offset data (16 bits)
←	Transfer direction
+	Addition
−	Subtraction
×	Multiplication
÷	Division
%	Modulo
AND	Logical and
OR	Logical or
XOR	Exclusive or
XXH	2-digit Hexadecimal data
XXXXH	4-digit Hexadecimal data

Table 4. Flag Operations

Identifier	Description
(blank)	No change
0	Cleared to 0
1	Set to 1
X	Set or cleared according to the result
U	Undefined
R	Value saved earlier is restored

Table 5. Memory Addressing

	mod		
mem	00	01	10
000	BW + IX	BW + IX + disp8	BW + IX + disp16
001	BW + IY	BW + IY + disp8	BW + IY + disp16
010	BP + IX	BP + IX + disp8	BP + IX + disp16
011	BP + IY	BP + IY + disp8	BP + IY + disp16
100	IX	IX + disp8	IX + disp16
101	IY	IY + disp8	IY + disp16
110	Direct Address	BP + disp8	BP + disp16
111	BW	BW + disp8	BW + disp16

Table 6. Selection of 8- and 16-Bit Registers

reg	W=0	W=1
000	AL	AW
001	CL	CW
010	DL	DW
011	BL	BW
100	AH	SP
101	CH	BP
110	DH	IX
111	BH	IY

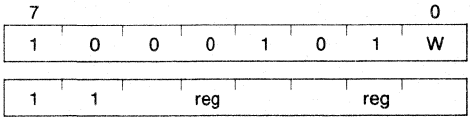
Table 7. Selection of Segment Registers

sreg	
00	DS1
01	PS
10	SS
11	DS0

DATA TRANSFER

MOV reg,reg

Move register to register



reg ← reg

Transfers the contents of the 8- or 16-bit register specified by the second operand to the 8- or 16-bit register specified by the first operand.

Bytes: 2

Transfers: None

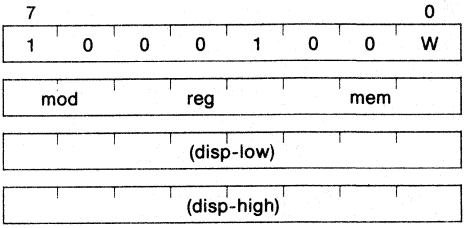
Flag operation: None

Example:

```
MOV     BP,SP
MOV     AL,CH
```

MOV mem,reg

Move register to memory



(mem) ← reg

Transfers the contents of the 8- or 16-bit register specified by the second operand to the 8- or 16-bit memory location specified by the first operand.

Bytes: 2/3/4

Transfers: 1

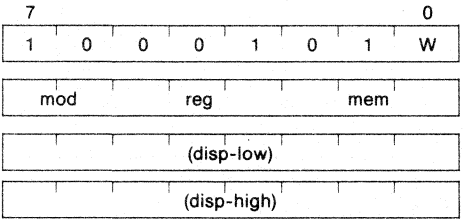
Flag operation: None

Example:

```
MOV     [BP][IX],AW
MOV     BYTE_VAR,BL
```

MOV reg,mem

Memory to register



reg ← (mem)

Transfers the 8- or 16-bit memory contents specified by the second operand to the 8- or 16-bit register specified by the first operand.

Bytes: 2/3/4

Transfers: 1

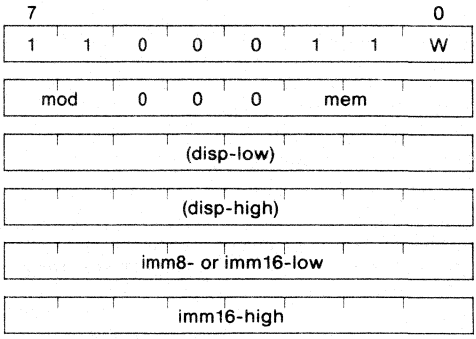
Flag operation: None

Example:

```
MOV     AW,[BW][IY]
MOV     CL,BYTE_VAR
```

MOV mem,imm

Immediate data to memory



(mem) ← imm

Transfers the 8- or 16-bit immediate data specified by the second operand to the 8- or 16-bit memory location addressed by the first operand.

Bytes: 3/4/5/6

Transfers: 1

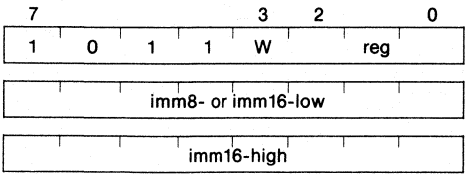
Flag operation: None

Example:

```
MOV     BYTE PTR [BP][IX],0
MOV     WORD PTR [BW],12
MOV     [BP][IX],5    ;Note: assembler assumes
                      ;WORD PTR as default.
MOV     BYTE_VAR,123
MOV     WORD_VAR,1000H
```

MOV reg,imm

Immediate data to register



reg ← imm

Transfers the 8- or 16-bit immediate data specified by the second operand to the 8- or 16-bit register specified by the first operand.

Bytes: 2/3

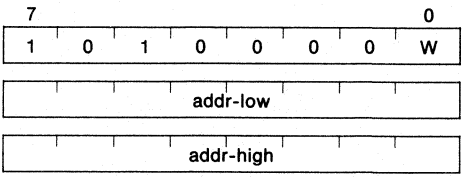
Transfers: None

Flag operation: None

Example: MOV BP,8000H

MOV acc,dmem

Memory to accumulator



When W = 0 AL ← (dmem)
When W = 1 AH ← (dmem + 1), AL ← (dmem)

Transfers the memory contents addressed by the second operand to the accumulator (AL or AW) specified by the first operand.

Bytes: 3

Transfers: 1

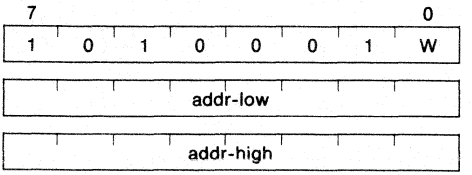
Flag operation: None

Example:

```
MOV     AW,WORD_VAR
MOV     AL,BYTE_VAR
```

MOV dmem,acc

Accumulator to memory



When W = 0, (dmem) ← AL
When W = 1, (dmem + 1) ← AH, (dmem) ← AL

Transfers the contents of the accumulator (AL or AW) specified by the second operand to the 8- or 16-bit memory location addressed by the first operand.

Bytes: 3

Transfers: 1

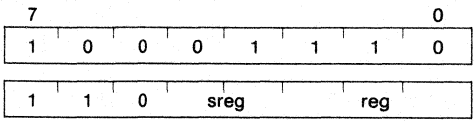
Flag operation: None

Example:

```
MOV     WORD_VAR,AW
MOV     BYTE_VAR,AL
```

MOV sreg,reg16

Register to segment register



sreg ← reg16 sreg: SS,DS_0,DS_1

Transfers the contents of the 16-bit register specified by the second operand to the segment register (except PS) specified by the first operand. External interrupts (NMI, INT) or a single-step break is not accepted between this instruction and the next.

Bytes: 2

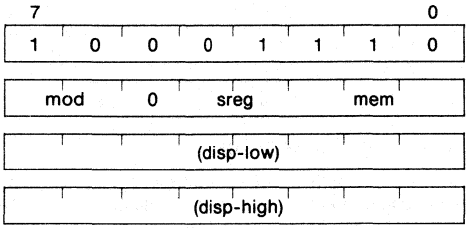
Transfers: None

Flag operation: None

Example: MOV SS,AW

MOV sreg,mem16

Memory to segment register



sreg ← (mem16) sreg: SS,DS_0,DS_1

Transfers the 16-bit memory contents addressed by the second operand to the segment register (except PS) specified by the first operand. However, external interrupts (NMI, INT) or a single-step break is not accepted during the period between this instruction and the next.

Bytes: 2/3/4

Transfers: 1

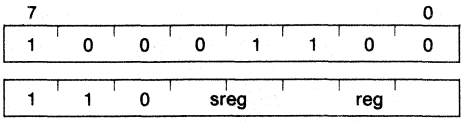
Flag operation: None

Example:

```
MOV     DS0,[BW][IX]
MOV     SS,WORD_VAR
```

MOV reg16,sreg

Segment register to register



reg 16 ← sreg

Transfers the contents of the segment register specified by the second operand to the 16-bit register specified by the first operand.

Bytes: 2

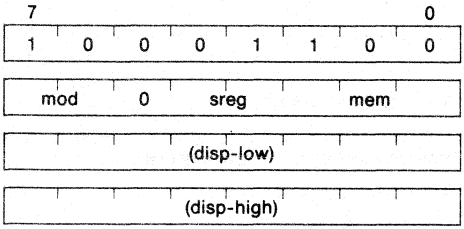
Transfers: None

Flag operation: None

Example: MOV AW,DS1

MOV mem16,sreg

Segment register to memory



(mem16) ← sreg

Transfers the contents of the segment register specified by the second operand to the 16-bit memory location addressed by the first operand.

Bytes: 2/3/4

Transfers: 1

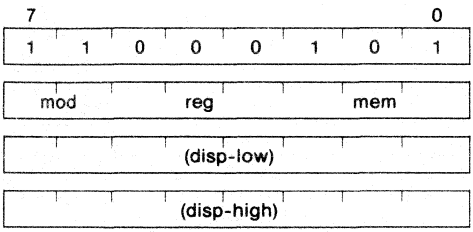
Flag operation: None

Example:

```
MOV     [IX],PS
```

MOV DS0,reg16,mem32

32-bit memory to 16-bit register and DS0



reg 16 ← (mem32)
DS_0 ← (mem32 + 2)

Transfers the lower 16 bits (offset word of a 32-bit pointer variable) addressed by the third operand to the 16-bit register specified by the second operand, and the higher 16 bits (segment word) to the DS_0 segment register.

Bytes: 2/3/4

Transfers: 2

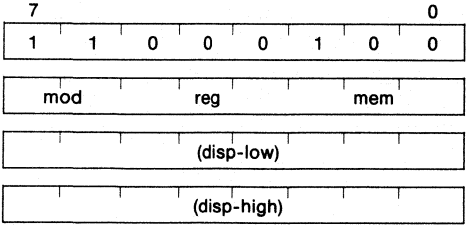
Flag operation: None

Example:

```
MOV     DS0,BW,DWORD_VAR
```

MOV DS1,reg16,mem32

32-bit memory to 16-bit register and DS_1



reg16 ← (mem32)
DS1 ← (mem32 + 2)

Transfers the lower 16 bits (offset word of a 32-bit pointer variable) addressed by the third operand to the 16-bit register specified by the second operand, and the higher 16 bits (segment word) to the DS_1 segment register.

Bytes: 2/3/4

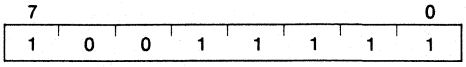
Transfers: 2

Flag operation: None

Example: MOV DS1,IY,DWORD_VAR

MOV AH,PSW

PSW to AH



AH ← S,Z,X,AC,X,P,X,CY

Transfers flags S, Z, AC, P, and CY of PSW to the AH register. Bits 5, 3, and 1 are undefined.

Bytes: 1

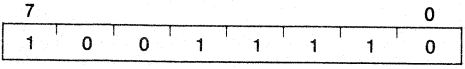
Transfers: None

Flag operation: None

Example: MOV AH,PSW

MOV PSW,AH

AH to PSW



S,Z,X,AC,X,P,X,CY ← AH

Transfers bits 7, 6, 4, 2, 0 of the AH register to flags S, Z, AC, P, and CY of PSW.

Bytes: 1

Transfers: None

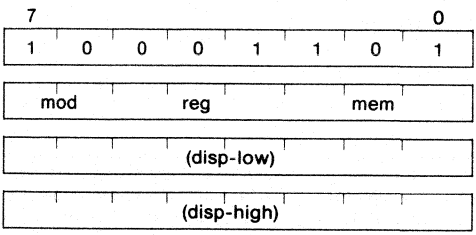
Flag operation:

V	S	Z	AC	P	CY
	X	X	X	X	X

Example: MOV PSW,AH

LDEA reg16, mem16

Load effective address to register



reg16 ← mem16

Loads the effective address (offset) generated by the second operand to the 16-bit general-purpose register specified by the first operand. Used to set starting address values to the registers that automatically specify the operand for TRANS or block instructions.

Bytes: 2/3/4

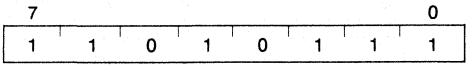
Transfers: None

Flag operation: None

Example: LDEA BW,TABLE[IX]

TRANS no operand
TRANS src-table
TRANSB no operand

Translate byte



AL ← (BW + AL)

Transfers to the AL register one byte specified by the BW and AL registers from the 256-byte conversion table. This time, the BW register specifies the starting (base) address of the table, while the AL register specifies the offset value within 256 bytes of the starting address.

Bytes: 1

Transfers: 1

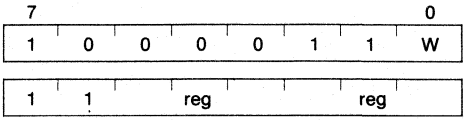
Flag operation: None

Example:

```
TRANS    TABLE
TRANS
TRANSB
```

XCH reg,reg

Exchange register with register



reg ↔ reg

Exchanges the contents of the 8- or 16-bit register specified by the first operand with the contents of the 8- or 16-bit register specified by the second operand.

Bytes: 2

Transfers: None

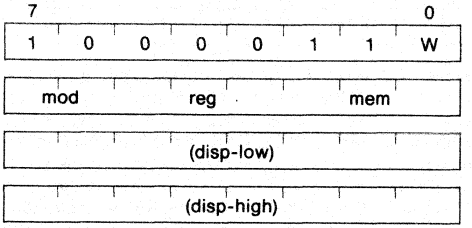
Flag operation: None

Example:

```
XCH    CW,BW
XCH    AH,AL
```

XCH mem,reg
XCH reg,mem

Exchange memory with register



(mem) ↔ reg

Exchanges the 8- or 16-bit memory contents addressed by the first operand with the contents of the 8- or 16-bit register specified by the second operand.

Bytes: 2/3/4

Transfers: 2

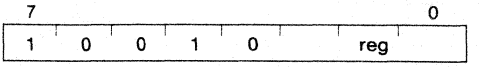
Flag operation: None

Example:

```
XCH WORD_VAR,CW
XCH AL,TABLE[BW]
```

XCH AW,reg16
XCH reg16,AW

Exchange accumulator with register



AW ↔ reg16

Exchanges the contents of the accumulator (AW only) specified by the first operand with the contents of the 16-bit register specified by the second operand.

Bytes: 1

Transfers: None

Flag operation: None

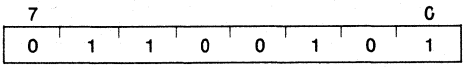
Example:

```
XCH AW,DW
XCH CW,AW
```

REPEAT PREFIXES

REPC (no operand)

Repeat while carry



While CW ≠ 0, the block comparison instruction (CMPBK or CMPM) placed in the following byte is executed and CW is decremented (−1). If the result of the block comparison instruction is CY ≠ 1, the instruction terminates. CW is checked against the condition immediately before the execution of the block comparison instruction. Therefore, if CW = 0 the first time the REPC instruction is executed, the program will proceed immediately to the instruction following the block comparison instruction and the block comparison instruction will not be executed at all. The contents of CY immediately before the first execution of the REPC instruction are "don't care."

Bytes: 1

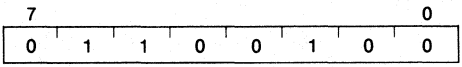
Transfers: None

Flag operation: None

Example: REPC CMPBKW

REPNC (no operand)

Repeat while no carry



While CW ≠ 0, the block comparison instruction (CMPBK or CMPM) placed in the following byte is executed and CW is decremented (−1). If the result of the comparison instruction is CY = 1, the instruction terminates. CW is checked against the condition immediately before the execution of the block comparison instruction. Therefore, if CW = 0 the first time the REPNC instruction is executed, the program will proceed immediately to the instruction following the block comparison instruction and the block comparison instruction will not be executed at all. The contents of CY immediately before the first execution of the REPNC instruction are "don't care."

Bytes: 1

Transfers: None

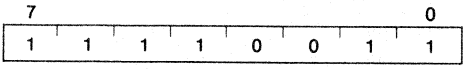
Flag operation: None

Example: REPNC CMPMB

REP/REPE/REPZ

Repeat/repeat while equal/repeat while zero

REP (no operand)
REPE/REPZ (no operand)



While CW ≠ 0, the following instruction is executed and CW is decremented (−1).

REP is used with MOVBK, LDM, STM, OUTM, or INM instructions and performs repeat operations while CW ≠ 0. The Z flag is disregarded.

REPZ or REPE is used with the CMPBK or CMPM instruction. A program will exit the loop if the comparison result by each block instruction is Z ≠ 1 or when CW becomes 0.

CW is checked against the condition immediately before the execution of REP/REPE/REPZ instruction. Consequently, if CW=0 the first time the REP/REPE/REPZ instruction is executed, the program will move to the instruction following the block instruction and the block instruction will not be executed at all.

A zero flag check is performed against the result of the block instruction. The contents immediately before the first execution of the REPE/REPZ instruction are "don't care."

Bytes: 1

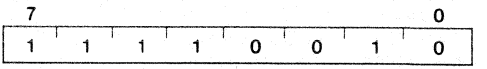
Transfers: None

Flag operation: None

Example:
REP MOVBKW
REPZ CMPBKW
REPE CMPMB

REPNE/REPNZ (no operand)

Repeat while not equal/repeat while not zero



While CW ≠ 0, the block comparison instruction (CMPBK, CMPM) is executed and CW is decremented (−1). If the result of the block comparison instruction is Z ≠ 0 or CW becomes 0, the instruction terminates. CW is checked against the condition immediately before the execution of the block comparison instruction. Consequently, if CW = 0 the first time the REPNE/REPNZ instruction is executed, the program will proceed immediately to the instruction following the block comparison instruction, and the block comparison instruction will not be executed at all.

A zero flag check is performed to test the result of the block comparison instruction. The contents of Z immediately before the first execution of the REPNE/REPNZ instruction are "don't care."

Bytes: 1

Transfers: None

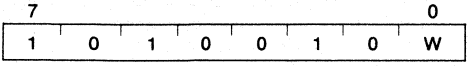
Flag operation: None

Example:
REPNE CMPMB
REPNZ CMPBKW

PRIMITIVE BLOCK TRANSFER

MOVBK/MOVBKB/MOVBKW
(repeat) MOVBK [DS1-spec:]dst-block,[Seg-spec:] src-block
(repeat) MOVBKB (no operand)
(repeat) MOVBKW (no operand)

Move block/move block byte/move block word



When W = 0, (IY) ← (IX)
DIR = 0: IX ← IX + 1, IY ← IY + 1
DIR = 1: IX ← IX − 1, IY ← IY − 1
When W = 1, (IY + 1, IY) ← (IX + 1, IX)
DIR = 0: IX ← IX + 2, IY ← IY + 2
DIR = 1: IX ← IX − 2, IY ← IY − 2

Transfers the block addressed by the IX register to the block addressed by the IY register by repeating the data word byte. In order to transfer the next byte/word, the IX or IY register is automatically incremented (+1 or +2) or decremented (−1 or −2) each time a byte/word is transferred. The direction of the block is determined by the direction flag (DIR).

Byte or word specification is made by the attribute of the operand when the MOVBK is used. If the MOVBKB or MOVBKW is used, the type is specified by the instruction.

The destination block must always be located within the segment specified by the DS_1 segment register. The default segment for the source block register is DS_0, and a segment override is permitted. The source block may be located in a segment specified by any of the segment registers.

Bytes: 1

Transfers:
Repeat: 2/rep
Single operation: 2

Flag operation: None

Examples:

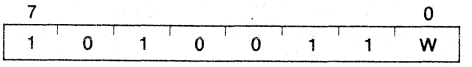
```
1.  MOV     AW,SEG SRC_BLOCK
                ;point to source
    MOV     DS0,AW
                ;segment and offset
    MOV     IX,OFFSET SRC_BLOCK
    MOV     AW,SEG DST_BLOCK
                ;point to destination
    MOV     DS1,AW
    MOV     IY,OFFSET DST_BLOCK
    MOV     CW,22
                ;set count
    REP     MOVBKW
                ;move 22 words

2.  MOV     IX,SP
                ;source will be stack
    MOV     DS1,IY,DST_DWPTR
                ;fetch pointer to destination
    MOV     CW,5
                ;set count
    REP     MOVBK DS1:DST_BLOCK,SS:[IX]
    :           ;move from stack (override prefix)
    :           ;to destination

DATA0 SEGMENT AT 0
SRC_BLOCK       DW 22 DUP (?)
SRC_DWPTR       DD SRC_BLOCK
DST_DWPTR       DD DST_BLOCK
DATA0 ENDS
DATA1 SEGMENT AT 1000H
DST_BLOCK       DW 22 DUP (?)
DATA1 ENDS
```

CMPBK/CMPBKB/CMPBKW
(repeat) CMPBK [Seg-spec:]src-block,[DS1-spec:]dst-block
(repeat) CMPBKB (no operand)
(repeat) CMPBKW (no operand)

Compare block/compare block byte/compare block word



When W=0: (IX) − (IY)
DIR=0: IX ← IX+1, IY ← IY+1
DIR=1: IX ← IX−1, IY ← IY−1
When W=1: (IX+1, IX) − (IY+1, IY)
DIR=0: IX ← IX+2, IY ← IY+2
DIR=1: IX ← IX−2, IY ← IY−2

Repeatedly compares the block addressed by the IY register with the block addressed by the IX register, byte by byte or word by word. The result of the comparison is shown by the flag. In order to process the next byte or word, IX and IY are automatically incremented (+1 or +2) or decremented (−1 or −2) each time one byte or word is processed. The direction of the block is determined by the direction flag (DIR).

The byte or word specification is made by the attribute of the operand when CMPBK is used. If CMPBKB or CMPBKW is used, it is specified directly to be the byte or word type.

The destination block must always be located within the segment specified by the DS_1 register. The default segment register for the source block is DS_0 and a segment override prefix is permitted.

Bytes: 1

Transfers:
Repeat: 1/rep
Single operation: 2

Flag operation

V	S	Z	AC	P	CY
X	X	X	X	X	X

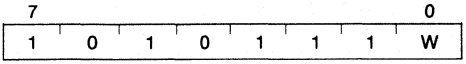
Example:

```
        MOV     DS0,IX,SRC_DWPTR
                        ;point to areas to compare
        MOV     DS1,IY,DST_DWPTR
        MOV     CW,16
                        ;set count
        REPNC   CMPBKB
                        ;compare 16 pairs of bytes
        BCWZ    GREATER
                        ;if CW = 0, then SRC ≥ DST
LESS:   ———
```

CMPM/CMPMB/CMPMW

(repeat) CMPM [DS1-spec:]dst-block
(repeat) CMPMB (no operand)
(repeat) CMPMW (no operand)

Compare multiple/compare multiple byte/compare multiple word



When W=0: (AL) − (IY)
DIR=0: IY ← IY+1,
DIR=1: IY ← IY − 1
When W=1: AW − (IY+1, IY)
DIR=0: IY ← IY+2
DIR=1: IY ← IY−2

Repeatedly compares the block addressed by the IY with the accumulator (AL or AW). To process the next byte or word, the IY is automatically incremented (+1 or +2) or decremented (−1 or −2) each time one byte or word is processed. The direction of the block is determined by the direction flag (DIR). Byte or word specification is made by the attribute of the operand when CMPM is used. If CMPMB or CMPMW is used, it is specified directly by the instruction.

The destination block must always be located within the segment specified by the DS_1 segment register.

Bytes: 1

Transfers:
Repeat: 1/rep
Single operation: 1

Flag operation

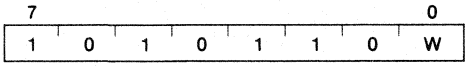
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
MOV     DS1,IY,DST_DWPTR
                ;point to destination block
MOV     AL,'A'
MOV     CW,20
                ;search for first 'A'
REPNZ   CMPMB
```

LDM/LDMB/LDMW
(repeat) LDM [Seg-spec:]src-block
(repeat) LDMB (no operand)
(repeat) LDMW (no operand)

Load multiple/load multiple byte/load multiple word



When W=0: AL ← (IX)
DIR=0: IX ← IX+1
DIR=1: IX ← IX−1
When W=1: AW ← (IX+1, IX)
DIR=0: IX ← IX+2
DIR=1: IX ← IX−2

Transfers the block addressed by the IX register to the accumulator (AL or AW). To process the next byte or word the IX register is automatically incremented (+1 or +2) or decremented (−1 or −2) each time one byte or word is processed. The direction of the block is determined by the direction flag (DIR). Byte or word specification is made by the attribute of the operand when LDM is used. If LDMB or LDMW is used, it is specified directly to be the byte or word type. The instruction may have a repeat prefix, but is usually used without one.

The default segment register for the source block is DS_0, and therefore segment override is possible. The source block may be located within the segment specified by any (optional) segment register.

Bytes: 1

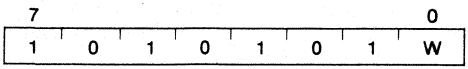
Flag operation: None

Example:

```
                ;Add a constant to a string
        MOV     DS1,IY,DST_DWPTR
                ;point DS1:IY to string
        MOV     IX,IY
                ;point DS1:IX to same area
        MOV     CW,10
                ;length of string
HERE:   LDM     BYTE PTR DS1:[IX]
                ;fetch byte (from DS1, with
                segment override prefix),
                increment IX
        ADD     AL,20H
                ;add constant
        STMB    ;replace modified value at
                DS1:IY,
                ;increment IY
        DBNZ    HERE
                ;loop until CW = 0
```

STM/STMB/STMW
(repeat) STM [DS1-spec:]dst-block
(repeat) STMB (no operand)
(repeat) STMW (no operand)

Store multiple/store multiple byte/store multiple word



When W=0: (IY) ← AL
DIR=0: IY ← IY+1
DIR=1: IY ← IY−1
When W=1: (IY+1, IY) ← AW
DIR=0: IY ← IY+2
DIR=1: IY ← IY−2

Transfers the contents of AL or AW to the block addressed by IY.

To process the next byte or word, IY is automatically incremented (+1 or +2) or decremented (−1 or −2) each time one byte or word is processed. The direction of the block is determined by the direction flag (DIR).

Byte or word specification is made by the attribute of the operand when STM is used. If STMB or STMW is used, it is specified directly to be the byte or word type.

The destination block must always be located within the segment specified by the DS_1 segment register.

Bytes: 1

Transfers:

Repeat: 1/rep

Single operation: 1

Flag operation: None

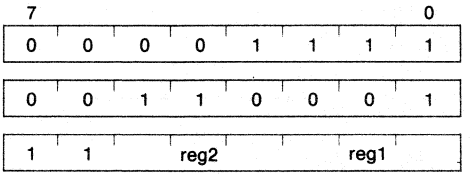
Example:

```
        ;Fill memory area with a constant
MOV     DS1,IY,DST_DWPTR
        ;point to block
XOR     AW,AW
        ;zero the accumulator
MOV     CW,10
        ;count = 10
REP     STMW
        ;fill 10 words with zero
```

BIT FIELD MANIPULATION INSTRUCTIONS

INS reg 8,reg 8

Insert bit field (register)



16-bit field ← AW

Transfers the lower data bits of the 16-bit AW register (bit length is specified by the 8-bit register of the second operand) to the memory location determined by the byte offset (addressed by the DS_1 segment register and the IY index register) and bit offset (specified by the 8-bit register of the first operand).

After the transfer, the IY register and the 8-bit register specified by the first operand are automatically updated to point to the next bit field.

Only the lower 4 bits (0-15) will be valid for the 8-bit register of the first operand that specifies the bit offset (maximum length: 15 bits). Also, only the lower 4 bits (0-15) will be valid for the 8-bit register of the second operand that specifies the bit length (maximum length: 16 bits). 0 specifies a 1-bit length, and 15 specifies a 16-bit length.

Bit field data may overlap the byte boundary of memory.

Note: For correct operation the upper four bits of the 8-bit registers used as first and second operands must be set to 0.

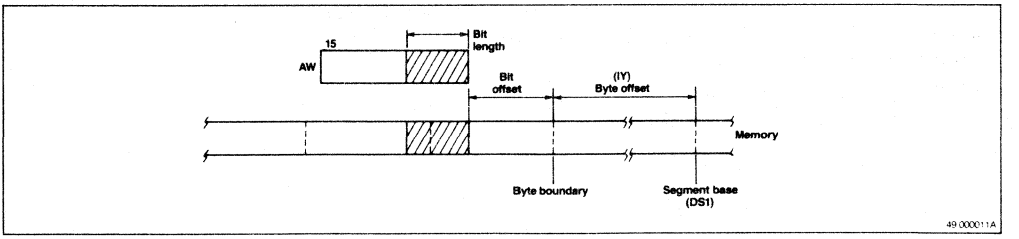
Bytes: 3

Transfers: 2 or 4

Flag operation:

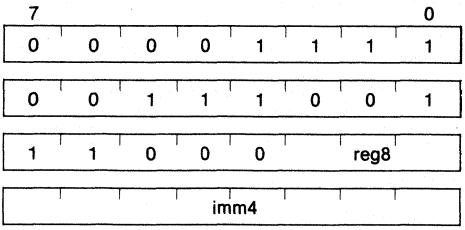
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example: INS DL,CL (See below for detailed example)



INS reg8,imm4

Insert bit field (immediate data)



16-bit field ← AW

Transfers the lower data bits of the 16-bit AW register (bit length specified by the 4-bit immediate data of the second operand) to the memory location determined by the byte offset (addressed by the DS_1 segment register and the IY register) and bit offset (specified by the 8-bit register of the first operand). After the transfer, the IY register and the 8-bit register specified by the first operand are updated to point to the next bit field.

Only the lower 4 bits (0-15) for the 8-bit register of the first operand (15 bits maximum length) are valid. The immediate data value of the second operand (16 bits maximum length) is valid only from 0-15.

0 specifies a 1-bit length, and 15 specifies a 16-bit length. The bit field data may overlap the byte boundary of memory.

Note: For correct operation, set the upper four bits of the 8-bit register used as the first operand to 0.

Bytes: 4

Transfers: 2 or 4

Flag operation:

V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
      MOV   DS1,IY,DST_DWPTR
                ;Point to destination
      MOV   CL,3
                ;Start at bit 3
      MOV   DL,4
                ;Insert 5 bits
(A)   MOV   AW,5555H
                ;Pattern to insert (A)
(B)   INS   CL,DL
                ;Insert 5 bits at bit 3 (B)
(C)   INS   CL,12
                ;Insert 13 bits at bit 8 (C)
```

at (A) memory =

MSB LSB | MSB LSB

XXXX XXXX XXXX XXXX | XXXX XXXX XXXX XXXX

CL = 3, IY = base

at (B) memory =

XXXX XXXX XXXX XXXX | XXXX XXXX 1010 1XXX

CL = 8, IY = base

at (C) memory =

XXXX XXXX XXX1 0101 | 0101 0101 1010 1XXX

CL = 5, IY = base + 2

EXT reg 8,reg 8

Extract bit field (register)

7							0
0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
1	1	reg1				reg2	

AW ← 16-bit field

Loads the bit field data (bit length specified by the 8-bit register of the second operand) into the AW register. The segment base of the memory location of the bit field is specified by the DS_0 register, the byte offset by the IX index register, and the bit offset by the 8-bit register of the first operand. At the same time zeros are loaded to the remaining upper bits of the AW register.

After the transfer, the IX register and the 8-bit register specified by the first operand are updated to point to the next bit field. Only the lower 4 bits (0-15) of the 8-bit register of the first operand (maximum length: 15 bits) are valid. Only the lower 4 bits of the 8-bit register of the second operand (maximum length: 16 bits) are valid.

0 specifies a 1-bit length, and 15 specifies a 16-bit length. Bit field data may overlap the byte boundary of memory.

Note: For correct operation, the upper 4 bits of the 8-bit registers used as first and second operands must be set to 0.

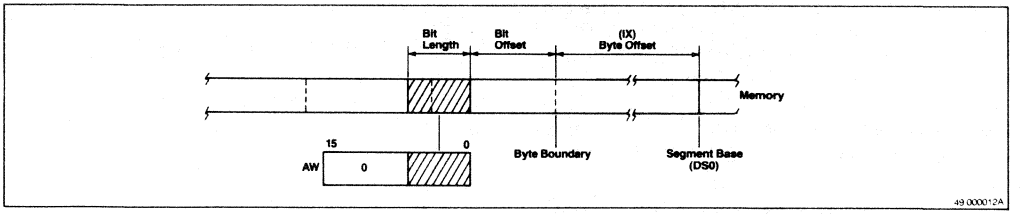
Bytes: 3

Transfers: 1 or 2

Flag operation:

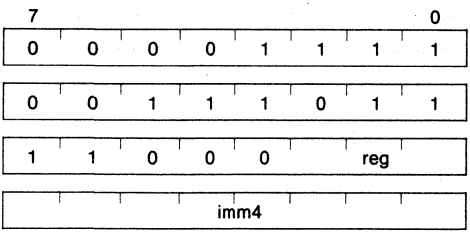
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example: EXT CL,DL (See below for detailed example)



EXT reg8,imm4

Extract bit field (immediate data)



AW ← 16-bit field

Loads bit field data from the memory location specified by the byte offset to the AW register (addressed by the DS_0 segment register and the IX index register) and the bit offset (specified by the 8-bit register of the first operand).

The bit length is specified by the 4-bit immediate data of the second operand.

After the transfer, the IX register and the 8-bit register specified by the first operand are updated to point to the next bit field. Only the lower 4 bits (0-15) of the 8-bit register of the first operand (maximum length: 15 bits) will be valid. The immediate data value of the second operand (maximum length: 16 bits) will be valid only from 0-15.

Zero specifies a 1-bit length, and 15 specifies a 16-bit length. Bit field data may overlap the byte boundary of memory.

Note: For correct operation, set the upper 4 bits of the 8-bit register used as the first operand to 0.

Bytes: 4

Transfers: 1 or 2

Flag operation:

V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
        MOV     DS0,IX,SRC_DWPTR
                ;Point to area to extract
        MOV     [IX],5555H
                ;Fill in sample patterns
        MOV     [IX+2],3333H
        MOV     CL,3
                ;Start at bit 3
(A)     MOV     DL,4
                ;(A)
(B)     EXT     CL,DL
                ;Extract 5 bits starting at 3 (B)
(C)     EXT     CL,12
                ;Extract 13 bits starting at 8 (C)
```

at (A) memory =

MSB			LSB	MSB			LSB
0011	0011	0011	0011	0101	0101	0101	0101

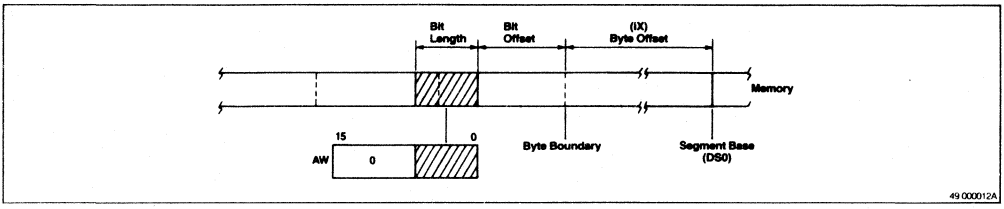
CL = 3, IX = base, AW = unknown

at (B)

CL = 8, IX = base, AW = (0000 0000 000)01010

at (C)

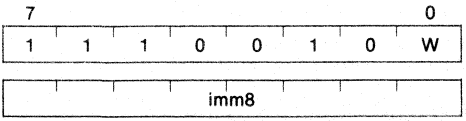
CL = 5, IX = base + 2, AW = (000)1 0011 0101 0101



INPUT/OUTPUT

IN acc,imm8

Input specified I/O device



When W=0 AL ← (imm8)
When W=1 AH ← (imm8+1), AL ← (imm8)

Inputs the contents of the I/O device specified by the second operand to the accumulator (AL or AH) specified by the first operand.

Bytes: 2

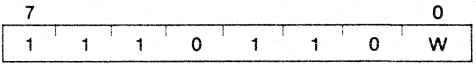
Transfers: 1

Flag operation: None

Example:
IN AL,20H
IN AW,48H

IN acc,DW

Input to device indirectly specified by DW



When W=0: AL ← (DW)
When W=1: AH ← (DW+1),AL ← (DW)

Inputs the contents of the I/O device specified by the DW register to the accumulator (AL or AW) specified by the first operand.

Bytes: 1

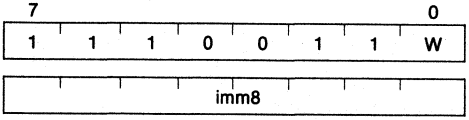
Transfers: 1

Flag Operation: None

Example: IN AL,DW

OUT imm8,acc

Output to directly specified I/O device



When W=0: (imm8) ← AL
When W=1: (imm8+1) ← AH, (imm8) ← AL

Outputs the contents of the accumulator (AL or AH) specified by the second operand to the I/O device specified by the first operand.

Bytes: 2

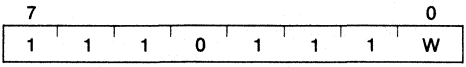
Transfers: 1

Flag operation: None

Example: OUT 30H,AW

OUT DW,acc

Output to indirectly specified (by DW) I/O device



When W=0: (DW) ← AL
When W=1: (DW+1) ← AH, (DW) ← AL

Outputs the contents of the accumulator (AL or AW) specified by the second operand to the I/O device specified by the first operand.

Bytes: 1

Transfers: 1

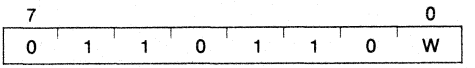
Flag operation: None

Example: OUT DW,AW

PRIMITIVE INPUT/OUTPUT

(repeat) INM [DS1-spec:]dst-block,DW

Input multiple



When W=0: (IY) ← (DW)
Dir=0: IY ← IY+1
Dir=1: IY ← IY−1
When W=1: (IY+1, IY) ← (DW+1,DW)
Dir=0: IY ← IY+2
Dir=1: IY ← IY−2

Transfers the contents of the I/O device addressed by the DW register to the memory location addressed by the IY index register.

When this instruction is paired with a repeat prefix (REP), the REP prefix controls the number of times the transfer will be repeated. When transfers are repeated, the contents (address of the I/O device) of the DW register are fixed. However, to transfer the next byte or word, the IX index register is automatically incremented (+1 or +2) or decremented (−1 or−2) each time one byte or word is transferred. The direction of the block is determined by the direction flag (DIR).

Byte or word specification is performed according to the attribute of the operand. The destination block must always be located within the segment specified by the DS_1 segment register, and a segment override prefix is prohibited.

Bytes : 1

Transfers:
Repeat: 2/rep
Single operation: 2

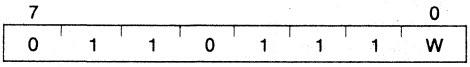
Flag operation: None

Example:

```
MOV     CW,30
MOV     IY,OFFSET BYTE_VAR
REP     INM BYTE_VAR,DW
            ;Input 30 bytes
```

OUTM DW,[seg-spec:]src-block

Output multiple



When W=0: (DW) ← (IX)
DIR=0: IX ← IX+1
DIR=1: IX ← IX−1
When W=1: (DW+1, DW) ← (IX+1,IX)
DIR=0: IX ← IX+2
DIR=1: IX ← IX−2

Transfers the memory contents addressed by the IX index register to the I/O device addressed by the DW register. When this instruction is paired with a repeat prefix (REP), REP controls the number of times the transfer will be repeated. When transfers are repeated, the contents (address of the I/O device) of the DW register are fixed. However, to transfer the next byte or word, the IX index register is automatically incremented (+1 or +2) or decremented (−1 or −2) each time one byte or word is transferred. The direction or the block is determined by the direction flag (DIR).

Byte or word specification is performed according to the attribute of the operand. The default segment register for the source block is DS_0, and segment override is possible. The source block may be located within the segment specified by any (optional) segment register.

Bytes: 1

Transfers:
Repeat: 2/rep
Single operation: 2

Flag operation: None

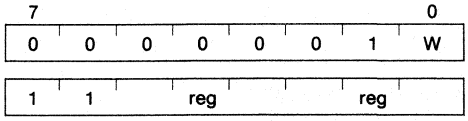
Example:

```
REP     OUTM DW,BYTE PTR DS1:[IX]
```

ADDITION/SUBTRACTION

ADD reg,reg

Add register with register to register



reg ← reg + reg

Adds the contents of the 8- or 16-bit register specified by the second operand to the contents of the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 2

Transfers: None

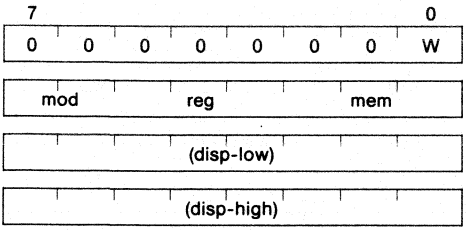
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: ADD AW,BW

ADD mem,reg

Add memory with register to memory



(mem) ← (mem) + reg

Adds the contents of the 8- or 16-bit register specified by the second operand to the 8- or 16-bit memory contents addressed by the first operand. Stores the result in the memory location addressed by the first operand.

Bytes: 2/3/4

Transfers: 2

Flag operation:

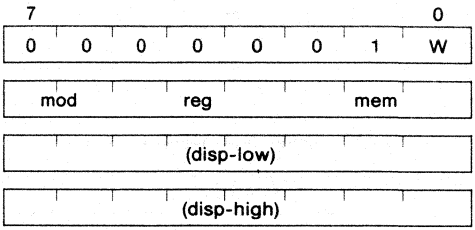
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
ADD     WORD_VAR,AW
ADD     [IX],CW
```

ADD reg,mem

Add register with memory to register



reg ← reg + (mem)

Adds the 8- or 16-bit memory contents addressed by the second operand to the contents of the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 2/3/4

Transfers: 1

Flag operation:

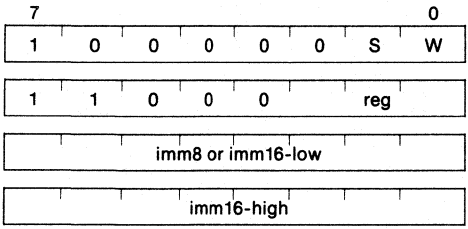
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
ADD     AW,WORD_VAR
ADD     BW,[BP][IX]
```

ADD reg,imm

Add register with immediate data to register



reg ← reg + imm

Adds the 8- or 16-bit immediate data specified by the second operand to the contents of the 8- or 16-bit register specified by the first operand, and stores the result in the register specified by the first operand.

Bytes: 2/3/4

Transfers: None

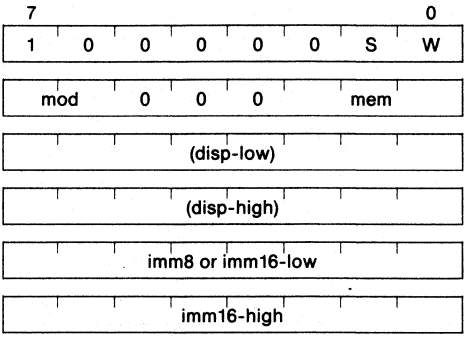
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: ADD DL,10

ADD mem,imm

Add memory with immediate data to memory



(mem) ← (mem) + imm

Adds the 8- or 16-bit immediate data specified by the second operand to the 8- or 16-bit memory contents addressed by the first operand. Stores the result in the memory location addressed by the first operand.

Bytes: 3/4/5/6

Transfers: 2

Flag operation:

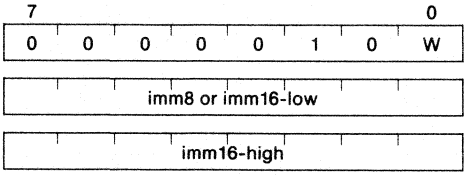
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
ADD     BYTE_VAR[BP],100
ADD     WORD_VAR[BW][IX],1234H
```

ADD acc,imm

Add accumulator with immediate data to accumulator



When W=0: AL ← AL imm
When W=1: AW ← AW imm

Adds the 8- or 16-bit immediate data specified by the second operand to the contents of the accumulator (AL or AW) specified by the first operand. Stores the result in the accumulator specified by the first operand.

Bytes: 2/3

Transfers: None

Flag operation:

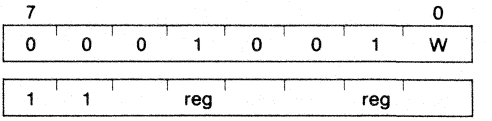
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
ADD     AL,3
ADD     AW,2000H
```

ADDC reg,reg

Add with carry, register with register to register



reg ← reg + reg + CY

Adds the contents of the 8- or 16-bit register specified by the second operand and the contents of the carry flag to the contents of the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 2

Transfers: None

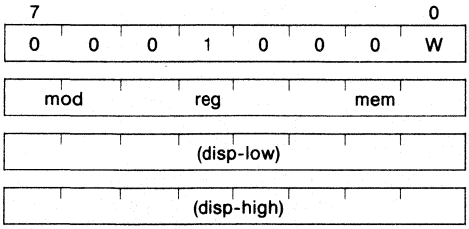
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: ADDC BW,DW

ADDC mem,reg

Add with carry, memory with register to memory



(mem) ← (mem) + reg + CY

Adds the contents of the 8- or 16-bit register specified by the second operand and the contents of the carry flag to the 8- or 16-bit memory contents addressed by the first operand. Stores the result in the memory location addressed by the first operand.

Bytes: 2/3/4

Transfers: 2

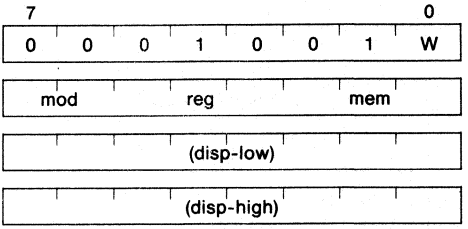
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: ADDC WORD_VAR,CW

ADDC reg,mem

Add with carry, register with memory to register



reg ← reg + (mem) + CY

Adds the 8- or 16-bit memory contents addressed by the second operand and the contents of the carry flag to the contents of the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Byte: 2/3/4

Transfers: 1

Flag operation:

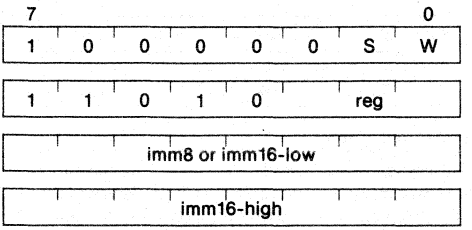
V	S	Z	AC	P	CY
X	X	X	X	X	X

Examples:

```
ADDC    AW,WORD_VAR
ADDC    BW,[BP][IX]
```

ADDC reg,imm

Add with carry, register with immediate data to register



reg ← reg + imm + CY

Adds the 8- or 16-bit immediate data specified by the second operand and the contents of the carry flag to the contents of the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 3/4

Transfers: None

Flag operation:

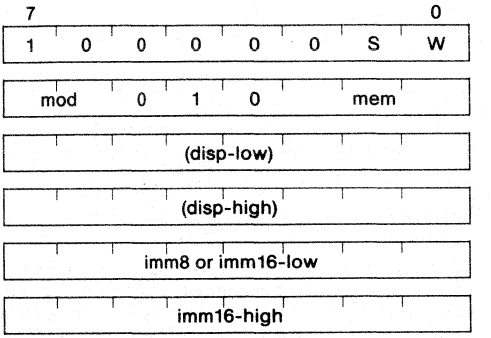
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
ADDC    CW,404H
ADDC    DL,3
```

ADDC mem,imm

Add with carry, memory with immediate data to memory



(mem) ← (mem) + imm + CY

Adds the 8- or 16-bit immediate data specified by the second operand and the contents of the carry flag to the 8- or 16-bit memory contents addressed by the first operand. Stores the result in the memory location addressed by the first operand.

Bytes: 3/4/5/6

Transfers: 2

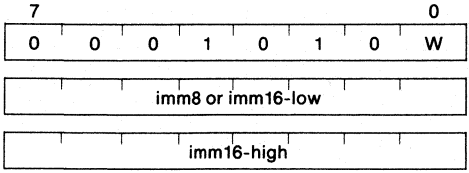
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: ADDC WORD_VAR,2000H

ADDC acc,imm

Add with carry, accumulator with immediate data to accumulator



When W=0: AL ← AL + imm8 + CY
When W=1: AW ← AW + imm16 + CY

Adds the 8- or 16-bit immediate data specified by the second operand and the contents of the carry flag to the accumulator (AL or AW) specified by the first operand. Stores the result in the accumulator specified by the first operand.

Bytes: 2/3

Transfers: None

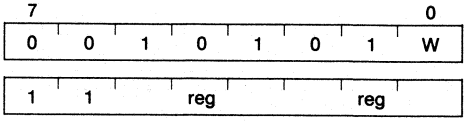
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: ADDC AL,7

SUB reg,reg

Subtract register from register to register



reg ← reg − reg

Subtracts the contents of the 8- or 16-bit register specified by the second operand from the contents of the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 2

Transfers: None

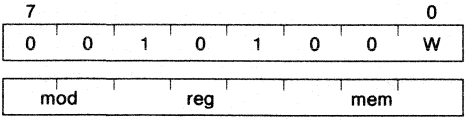
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUB BW,CW

SUB mem,reg

Subtract register from memory to memory



(mem) ← (mem) − reg

Subtracts the contents of the 8- or 16-bit register specified by the second operand from the 8- or 16-bit memory contents addressed by the first operand. Stores the result in the memory location addressed by the first operand.

Bytes: 2/3/4

Transfers: 2

Flag operation:

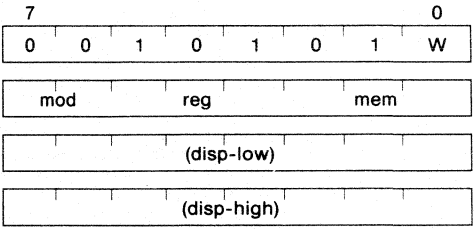
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
SUB     WORD_VAR,BW
SUB     [IX],AL
```

SUB reg,mem

Subtract memory from register to register



reg ← reg − (mem)

Subtracts the 8- or 16-bit memory contents addressed by the second operand from the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 2/3/4

Transfers: 1

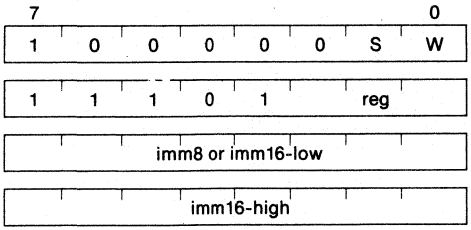
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUB CW,WORD_VAR

SUB reg,imm

Subtract immediate from register to register



reg ← reg − imm

Subtracts the 8- or 16-bit immediate data specified by the second operand from the contents of the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 3/4

Transfers: None

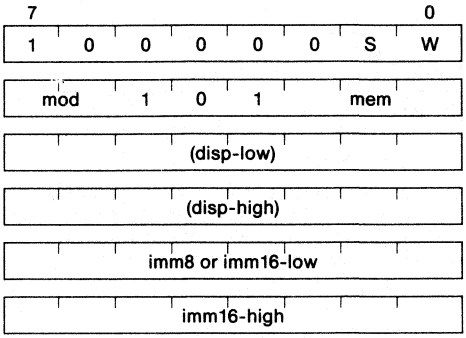
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUB IX,4

SUB mem,imm

Subtract immediate data from memory to memory



(mem) ← (mem) − imm

Subtracts the 8- or 16-bit immediate data specified by the second operand from the 8- or 16-bit memory contents addressed by the first operand. Stores the result in the memory location addressed by the first operand.

Bytes: 3/4/5/6

Transfers: 2

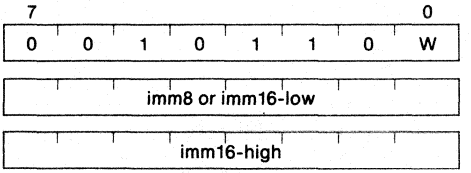
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUB WORD_VAR,10

SUB acc,imm

Subtract immediate data from accumulator to accumulator



When W=0: AL ← AL − imm8
When W=1: AW ← AW − imm16

Subtracts the 8- or 16-bit immediate data specified by the second operand from the accumulator (AL or AW) specified by the first operand. Stores the result in the accumulator specified by the first operand.

Bytes: 2/3

Transfers: None

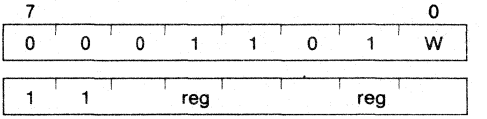
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUB AL,8

SUBC reg,reg

Subtract with carry, register from register to register



reg ← reg − reg − CY

Subtracts the contents of the 8- or 16-bit register specified by the second operand and the contents of the carry flag from the 8- or 16-bit register specified by the first operand.

Bytes: 2

Transfers: None

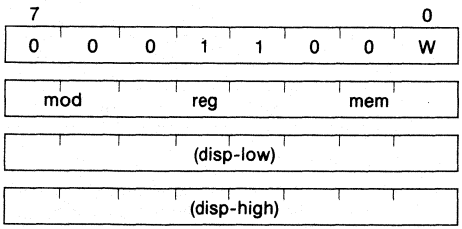
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUBC BW,DW

SUBC mem,reg

Subtract with carry, register from memory to memory



(mem) ← (mem) − reg − CY

Subtracts the contents of the 8- or 16-bit register specified by the second operand and the contents of the carry flag from the 8- or 16-bit memory contents specified by the first operand. Stores the result in the memory location addressed by the first operand.

Bytes: 2/3/4

Transfers: 2

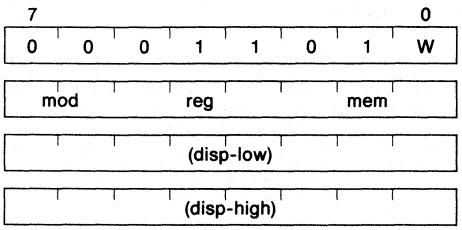
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUBC BYTE_VAR,AL

SUBC reg,mem

Subtract with carry, memory from register to register



reg ← reg − (mem) − CY

Subtracts the contents of the 8- or 16-bit memory addressed by the second operand and the contents of the carry flag from the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 2/3/4

Transfers: 1

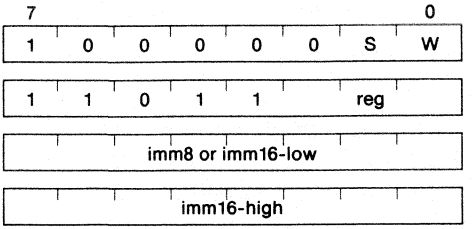
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUBC AW,WORD_VAR

SUBC reg,imm

Subtract with carry, immediate data from register to register



reg ← reg − imm − CY

Subtracts the contents of the 8- or 16-bit immediate data specified by the second operand and the contents of the carry flag from the 8- or 16-bit register specified by the first operand. Stores the result in the register specified by the first operand.

Bytes: 3/4

Transfers: None

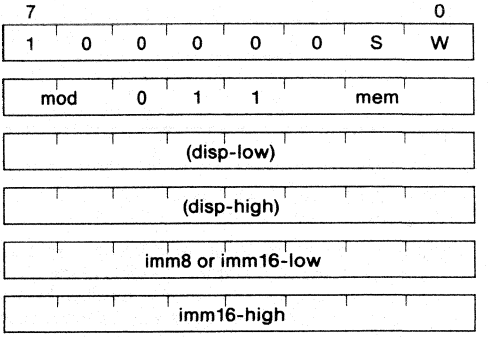
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUBC DL,10

SUBC mem,imm

Subtract with carry, immediate data from memory to memory



(mem) ← (mem) − imm − CY

Subtracts the contents of the 8- or 16-bit immediate data specified by the second operand and the contents of the carry flag from the 8- or 16-bit memory contents addressed by the first operand. Stores the result in the memory location addressed by the first operand.

Bytes: 3/4/5/6

Transfers: 2

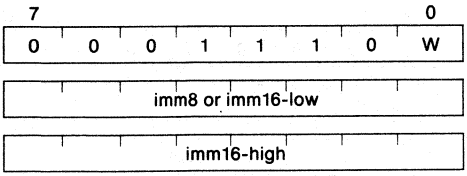
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUBC WORD_VAR,25

SUBC acc,imm

Subtract with carry, immediate data from accumulator to accumulator



When W=0: AL ← AL − imm8 − CY
When W=1: AW ← AW − imm16 − CY

Subtracts the 8- or 16-bit immediate data specified by the second operand and the contents of the carry flag from the accumulator (AL or AW) specified by the first operand. Stores the result in the accumulator specified by the first operand.

Bytes: 2/3

Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: SUBC AL,8

BCD ARITHMETIC

ADD4S [DS1-spec:]dst-string,[seg-spec:]src-string
ADD4S (no operand)

Add nibble string

7							0
0	0	0	0	1	1	1	1

0	0	1	0	0	0	0	0

BCD string (IY,CL) ← BCD string (IY,CL) + BCD string (IX,CL)

Adds the packed BCD string addressed by the IX index register to the packed BCD string addressed by the IY index register. Stores the result in the string addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register and can vary from 1 to 254 digits.

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

The destination string must always be located within the segment specified by the DS_1 segment register. Segment override is prohibited.

The default segment register for the source string is DS_0 and segment override is possible. The source string may be located within the segment specified by any (optional) segment register.

The format for the packed BCD string follows.

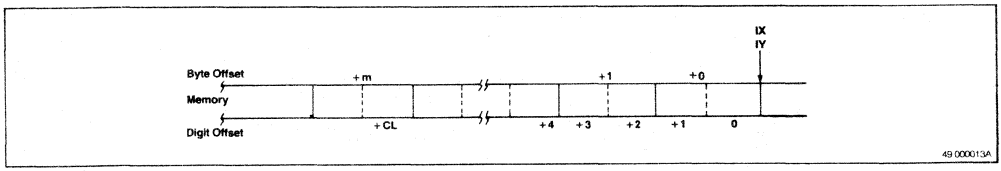
Bytes: 2

Transfers: 3n

Flag operation:

V	S	Z	AC	P	CY
U	U	X	U	U	X

Example: See example for CMP4S



SUB4S [DS1-spec:]dst-string,[seg-spec:]src-string
SUB4S (no operand)

Subtract nibble string

7							0
0	0	0	0	1	1	1	1

0	0	1	0	0	1	0	

BCD string (IY,CL) ← BCD string (IY,CL) − BCD string (IX,CL)

Subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY index register. Stores the result in the string addressed by the IY register.

The length of the string (number of BCD digits) is specified by the CL register and can vary from 1 to 254 digits.

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

The destination string must always be located within the segment specified by the DS_1 segment register. Segment override is prohibited.

The default segment register for the source string is DS_0, and segment override is possible. The source string may be located within the segment specified by any (optional) segment register.

The format for the packed BCD string is shown as follows.

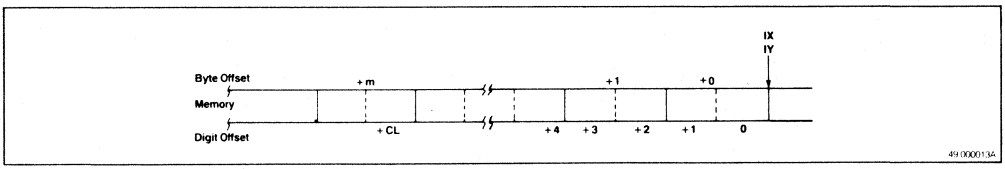
Bytes: 2

Transfers: 3n

Flag operation:

V	S	Z	AC	P	CY
U	U	X	U	U	X

Example: See example for CMP4S



CMP4S

[DS1-spec:]dst-string,[seg-spec:]src-string
CMP4S (no operand)
Compare nibble string

7							0
0	0	0	0	1	1	1	1
0	0	1	0	0	1	1	0

BCD string (IY,CL) − BCD string (IX,CL)

Subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY index register. The result is not stored and only the flags are affected. The length of the string (number of BCD digits) is specified by the CL register and can vary from 1 to 254 digits.

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

The default segment register for the source string is DS_0 and segment override is possible.

The source string may be located within the segment specified by any (optional) segment register. The format for the packed BCD string is shown below.

Bytes: 2

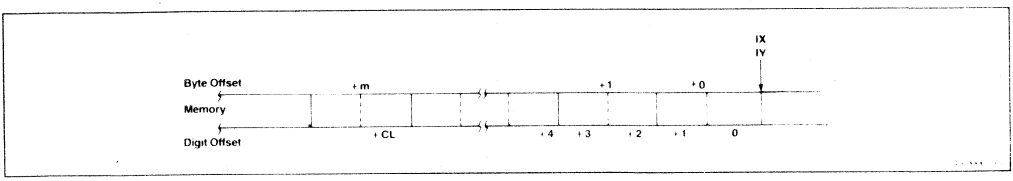
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U	U	X	U	U	X

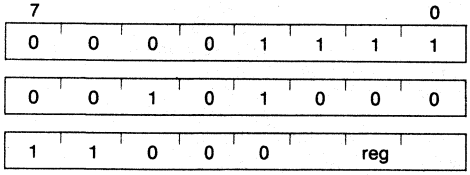
Example:

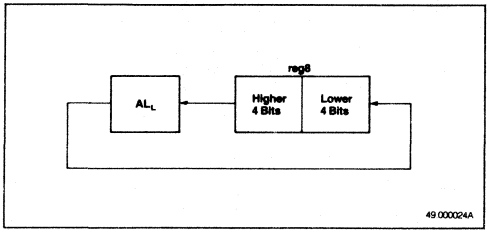
```
;μPD70116 BCD string operation
  MOV     AW,PS       ;Set both data
                      ;segments to
  MOV     DS0,AW      ;same as program
  MOV     DS1,AW      ;segment
  MOV     IX,OFFSET STR0
                      ;Point to BCD strings
  MOV     IY,OFFSET STR1
  MOV     CL,8        ;Eight digits
                      ;in strings (A)
  CMP4S               ;Compare (B)
  ADD4S               ;Add string0
                      ;to string1 (C)
  CMP4S               ;Compare again (D)
  SUB4S               ;Subtract string0
                      ;from string1 (E)
  SUB4S               ;again (result is
                      zero) (F)
  SUB4S               ;and again
                      ;(underflow) (G)
  HALT
;
STR0  DW 4321H,0765H
            ;BCD# 07654321
STR1  DW 4321H,0765H
            ;BCD# 07654321
;
; at (A), STR0 = 7654321,
;         STR1 = 7654321, Z = ?, CY = ?
; at (B), STR0 = 7654321,
;         STR1 = 7654321, Z = 1, CY = 0
; at (C), STR0 = 7654321,
;         STR1 = 15308642, Z = 0, CY = 0
; at (D), STR0 = 7654321,
;         STR1 = 15308642, Z = 0, CY = 0
; at (E), STR0 = 7654321,
;         STR1 = 7654321, Z = 0, CY = 0
; at (F), STR0 = 7654321,
;         STR1 = 00000000, Z = 1, CY = 0
; at (G), STR0 = 7654321,
;         STR1 = 92345679, Z = 0, CY = 1
;
```



ROL4 reg8

Rotate left nibble, 8-bit register





Treats the byte data of the 8-bit register specified by the operand as a two-digit BCD and uses the lower 4 bits of the AL register (AL_L) to rotate that data one digit to the left.

Due to the result of this instruction, the contents of the upper 4 bits of the AL register are not assured.

Bytes: 3
Transfers: None

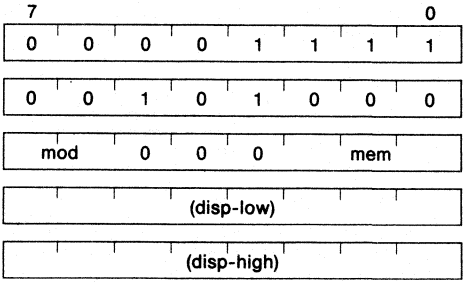
Flag operation: None

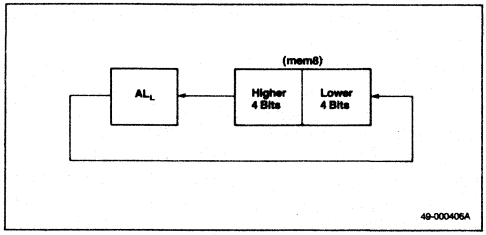
Example:

```
MOV    BL,95H
MOV    AL,03H
ROL4   BL   ;BL = 53H, AL = X9H
```

ROL4 mem8

Rotate left nibble, 8-bit memory





Treats the byte data of the 8-bit memory location addressed by the operand as a two-digit BCD and uses the lower 4 bits of the AL register (AL_L) to rotate that data one digit to the left.

Due to the result of this instruction, the contents of the upper 4 bits of the AL register are not assured.

Bytes: 3/4/5

Transfers: 2

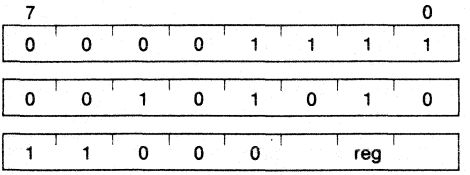
Flag operation: None

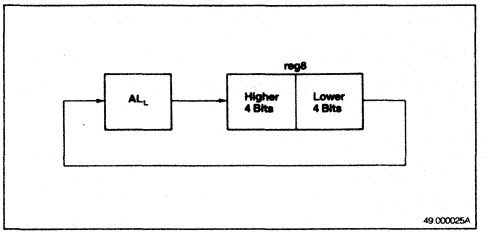
Example:

```
MOV    BYTE PTR [IX],12H
MOV    AL,03H
ROL4   [IX]   ;[IX] = 23H, AL = X1H
```

ROR4 reg8

Rotate right nibble, 8-bit register





Treats the byte data of the 8-bit register specified by the operand as two-digit BCD and uses the lower 4 bits of the AL register (AL_L) to rotate the data one digit to the right.

Due to the result of this instruction, the contents of the upper 4 bits of the AL register are not assured.

Bytes: 3

Transfers: None

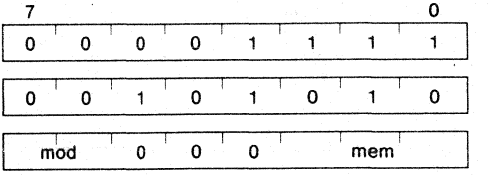
Flag operation: None

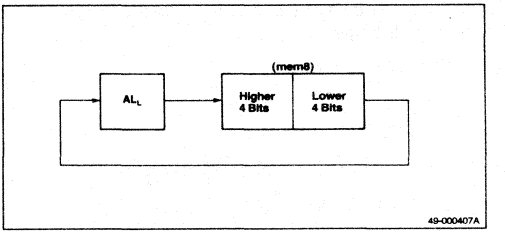
Example:

```
MOV    CL,95H
MOV    AL,03H
ROR4   CL   ;CL = 39H, AL = X5H
```

ROR4 mem8

Rotate right nibble, 8-bit memory





Treats the byte data of the 8-bit memory location addressed by the operand as a two-digit BCD and uses the lower 4 bits of the AL register (AL_L) to rotate that data one digit to the right. Due to the result of this instruction, the contents of the upper 4 bits of the AL register are not assured.

Bytes: 3/4/5

Transfers: 2

Flag operation: None

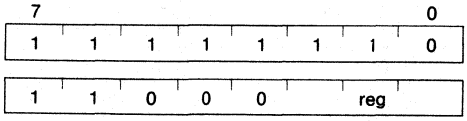
Example:

```
MOV    BYTE PTR [IX],12H
MOV    AL,03H
ROR4   [IX]   ;[IX] = 31H, AL = X2H
```

INCREMENT/DECREMENT

INC reg8

Increment 8-bit register



reg8 ← reg + 1

Increments by 1 the contents of the 8-bit register specified by the operand.

Bytes: 2

Transfers: None

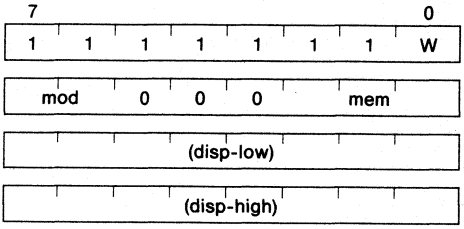
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	

Example: INC BL

INC mem

Increment memory



(mem) ← (mem) + 1

Increments by 1 the contents of the 8- or 16-bit memory location specified by the operand.

Bytes: 2/3/4

Transfers: 2

Flag operation:

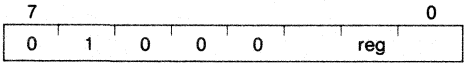
V	S	Z	AC	P	CY
X	X	X	X	X	

Example:

```
INC     WORD_VAR
INC     BYTE PTR [BW]
```

INC reg16

Increment 16-bit register



reg16 ← reg16 + 1

Increments by 1 the contents of the 16-bit register specified by the operand.

Bytes :1

Transcribes: None

Flag operation:

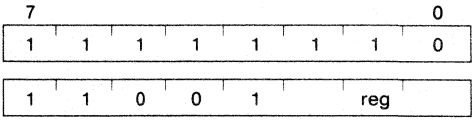
V	S	Z	AC	P	CY
X	X	X	X	X	

Example:

```
INC   BW
INC   IX
```

DEC reg8

Decrement 8-bit register



reg8 ← reg8 − 1

Decrements by 1 the contents of the 8-bit register specified by the operand.

Bytes: 2

Transfers: None

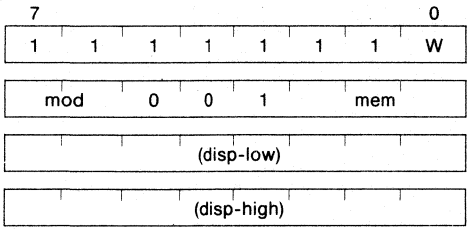
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	

Example: DEC DH

DEC mem

Decrement memory



(mem) ← (mem) − 1

Decrements by 1 the 8- or 16-bit memory contents addressed by the operand.

Bytes: 2/3/4
Transfers: 2

Flag operation:

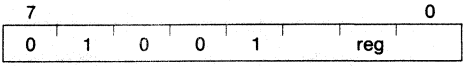
V	S	Z	AC	P	CY
X	X	X	X	X	

Example:

```
DEC     BYTE_VAR
DEC     WORD_VAR[BW][IX]
```

DEC reg16

Decrement 16-bit register



reg16 ← reg16 − 1

Decrements by 1 the contents of the 16-bit register specified by the operand.

Bytes: 1
Transfers: None

Flag operation:

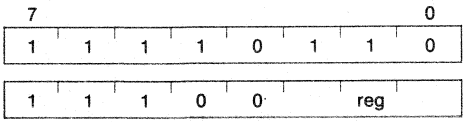
V	S	Z	AC	P	CY
X	X	X	X	X	

Example: DEC BP

MULTIPLICATION

MULU reg8

Multiply unsigned, 8-bit register



AW ← AL × reg8
When AH=0: CY ← 0, V ← 0
When AH≠0: CY ← 1, V ← 1

Performs unsigned multiplication of the contents of the AL register and the contents of the 8-bit register specified by the operand. Stores the word result in the AL and AH registers. When the upper half (AH) of the result is not 0, the carry and overflow flags are set.

Bytes: 2

Transfers: None

Flag operation:

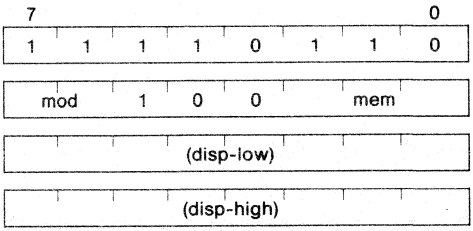
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MOV     AL,13   ;AW = XX0DH
MOV     CL,5
MULU    CL      ;AW = 0041H = 65, C = V = 0
```

MULU mem8

Multiply unsigned, 8-bit memory



AW ← AL × (mem8)
When AH=0: CY ← 0, V ← 0
When AH≠0: CY ← 1, V ← 1

Performs unsigned multiplication of the contents of the AL register and the 8-bit memory location addressed by the operand. Stores the word result in the AL and AH registers. When the upper half (AH) of the result is not 0, the carry and overflow flags are set.

Bytes: 2/3/4

Transfers: 1

Flag operation:

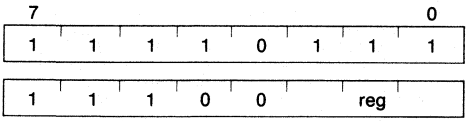
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MOV     AL,35
                ;AW = XX23H
MOV     BYTE_VAR,20
MULU    BYTE_VAR
                ;AW = 02BCH = 700, C = V = 1
  •
  •
MULU    BYTE PTR [IX]
```

MULU reg16

Multiply unsigned, 16-bit register



DW, AW ← AW × reg16
When DW=0: CY ← 0, V ← 0
When DW≠0: CY ← 1, V ← 1

Performs unsigned multiplication of the contents of the AW register and the contents of the 16-bit register specified by the operand. Stores the double-word result in the AW and DW registers. When the upper half (DW) of the result is not 0, the carry and overflow flags are set.

Bytes: 2

Transfers: None

Flag operation:

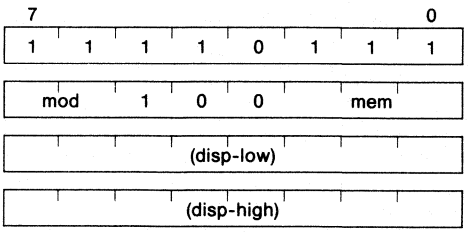
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MOV     AW,1234H
MOV     CW,3
MULU    CW
            ;DW = 0000H, AW = 369CH,
            ;C = V = 0
```

MULU mem16

Multiply unsigned, 16-bit memory



DW, AW ← AW × (mem16)
When DW=0: CY ← 0, V ← 0
When DW≠0: CY ← 1, V ← 1

Performs unsigned multiplication of the contents of the AW register and the 16-bit memory contents addressed by the operand. Stores the double-word result in the AW and DW registers. When the upper half (DW) of the result is not 0, the carry and overflow flags are set.

Bytes: 2/3/4

Transfers: 1

Flag operation:

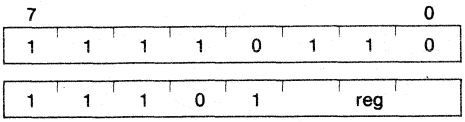
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MOV     AW,400H
MOV     WORD_VAR,9310H
MULU    WORD_VAR
            ;DW = 024CH,AW = 4000H,
            ;C = V = 1
```

MUL reg8

Multiply signed, 8-bit register



AW ← AL × reg8
When AH=sign extension of AL: CY ← 0, V ← 0
When AH≠sign extension of AH: CY ← 1, V ← 1

Performs signed multiplication of the contents of the AL register and the contents of the 8-bit register specified by the operand. Stores the double-word result in the AL and AH registers. When the upper half (AH) of the result is not the sign extension of the lower half (AL), the carry and overflow flags are set.

Bytes: 2

Transfers: None

Flag operation:

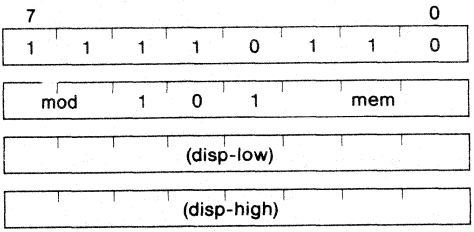
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MOV   AL,18
          ;AW = XX12H
MOV   CL,-2
          ;CL = FEH
MUL   CL
          ;AW = FFDC = -36, C = V = 0
```

MUL mem8

Multiply signed, 8-bit memory



AW ← AL × (mem8)
When AH=sign extension of AL: CY ← 0, V ← 0
When AH≠sign extension of AH: CY ← 1, V ← 1

Performs signed multiplication of the contents of the AL register and the 8-bit memory location addressed by the operand. Stores the double-word result in the AL and AH registers. When the upper half (AH) of the result is not the sign extension of the lower half (AL), the carry and overflow flags are set.

Bytes: 2/3/4

Transfers: None

Flag operation:

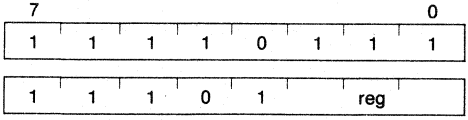
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MOV   AL,100
          ;AW = XX64H
MOV   BYTE_VAR,-4
          ; = FCH
MUL   BYTE_VAR
          ;AW = FE70H = -400, C = V = 1
```

MUL reg16

Multiply signed, 16-bit register



DW, AW ← AW × reg16
When DW=sign extension of AW: CY ← 0, V ← 0
When DW≠sign extension of AH: CY ← 1, V ← 1

Performs signed multiplication of the contents of the AW register and the contents of the 16-bit register specified by the operand. Stores the double-word result in the AW and DW registers. When the upper half (DW) of the result is not the sign extension of the lower half (AW), the carry and overflow flags are set.

Bytes: 2

Transfers: None

Flag operation:

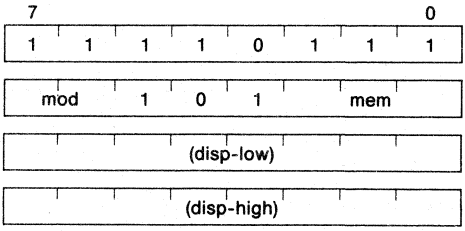
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MOV   AW,−10
          ;AW = FFF6H
MOV   BW,−10
          ;BW = FFF6H
MUL   BW
          ;DW = 0000, AW = 0064H = 100,
          ;C = V = 0
```

MUL mem16

Multiply signed, 16-bit memory



DW, AW ← AW × (mem16)
When DW=sign extension of AW: CY ← 0, V ← 0
When DW≠sign extension of AW: CY ← 1, V ← 1

Performs signed multiplication of the contents of the AW register and the 16-bit memory contents addressed by the operand. Stores the double-word result in the AW and DW registers. When the upper half (DW) of the result is not the sign extension of the lower half (AW), the carry and overflow flags are set.

Bytes: 2/3/4

Transfers: 1

Flag operation:

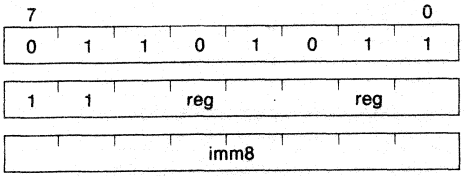
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MOV   AW,−10
          ;AW = FFF6
MOV   [IX],−20
          ; = FFEC
MUL   WORD PTR [IX]
          ;DW = 0000, AW = 00C8H = 200,
          ;C = V = 0
```

MUL reg16,reg16,imm8
MUL reg16,imm8

Multiply signed, 16-bit register × 8-bit immediate data to 16-bit register



reg16 ← reg16 × imm8
Product ≤ 16 bits: CY ← 0, V ← 0
Product > 16 bits: CY ← 1, V ← 1

Performs signed multiplication of the contents of the 16-bit register specified by the second operand. (If a two-operand description, then performs signed multiplication on the contents specified by the first operand.) Performs signed multiplication on the 8-bit immediate data specified by the third operand. (If a two-operand description then performs signed multiplication on the data specified by the second operand.)

When the source register and the destination register are the same, a two-operand description is acceptable.

Bytes: 3

Transfers: None

Flag operation:

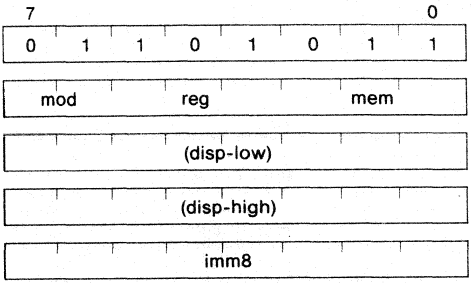
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MUL     AW,BW,10
            ;AW = BW*10
MUL     CW,25
            ;CW = CW*25
```

MUL reg16,mem16,imm8

Multiply signed, 16-bit memory × 8-bit immediate data to 16-bit register



reg16 ← (MEM16) × imm8
Product ≤ 16 bits: CY ← 0, V ← 0
Product > 16 bits: CY ← 1, V ← 1

Performs signed multiplication of the contents of the 16-bit memory contents addressed by the second operand and the 8-bit immediate data specified by the third operand. Stores the result in the 16-bit register specified by the first operand.

Bytes: 3/4/5

Transfers: 1

Flag operation:

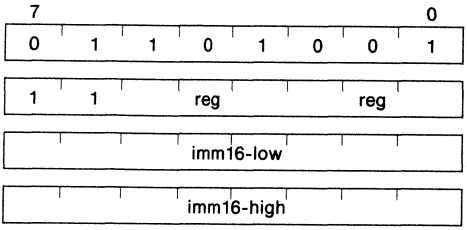
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MUL     CW,WORD_VAR,7
            ;CW = [WORD_VAR]*7
MUL     AW,[IX],22
            ;AW = [IX]*22
```

MUL reg16,reg16,imm16
MUL reg16,imm16

Multiply signed, 16-bit register × 16-bit immediate data to 16-bit register



reg16 ← reg16 × imm16
If product ≤ 16 bits: CY ← 0, V ← 0
If product > 16 bits: CY ← 1, V ← 1

Performs signed multiplication of the contents of the 16-bit register specified by the second operand — the first operand, when a two-operand description — and the 16-bit immediate data specified by the third (second) operand. Stores the result in the 16-bit register specified by the first operand.

When the source register and the destination register are the same, a two-operand description is possible.

Bytes: 4

Transfers: None

Flag operation:

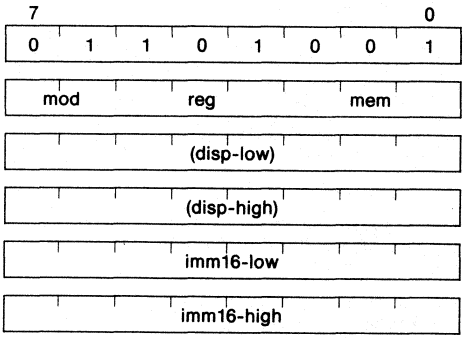
V	S	Z	AC	P	CY
X	U	U	U	U	X

Example:

```
MUL     AW,BW,200H
            ;AW = BW*200H
MUL     IX,300
            ;IX = IX*300
```

MUL reg16,mem16,imm16

Multiply signed, 16-bit memory × 16-bit immediate data to 16-bit register



reg16 ← (mem16) × imm16

If product ≤ 16 bits: CY ← 0, V ← 0
If product > 16 bits: CY ← 1, V ← 1

Performs signed multiplication of the 16-bit memory contents specified by the second operand and the 16-bit immediate data specified by the third operand. Stores the result in the 16-bit register specified by the first operand.

Bytes: 4/5/6

Transfers: 1

Flag operation:

V	S	Z	AC	P	CY
X	U	U	U	U	X

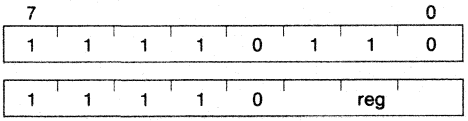
Example:

```
MUL     CW,WORD_VAR,200H
            ;CW = [WORD_VAR]*200H
MUL     AW,[IX],850
            ;AW = [IX]*850
```

DIVISION

DIVU reg8

Divide unsigned, 8-bit register



temp ← AW
When temp ÷ reg3 ≤ FFH:
  AH ← temp % reg8
  AL ← temp ÷ reg8
When temp ÷ reg8 > FFH:
  (SP−1,SP−2) ← PSW,
  (SP−3,SP−4) ← PS,
  (SP−5,SP−6) ← PC,
  SP ← SP − 6,
  IE ← 0,
  BRK ← 0,
  PS ← (003H, 002H),
  PC ← (001H, 000H)

Divides (using unsigned division) the contents of the AW 16-bit register by the contents of the 8-bit register specified by the operand. The resulting quotient is stored in the AL register. Any remainder is stored in the AH register.

When the quotient exceeds FFH (the capacity of the AL destination register) the vector 0 interrupt is generated. When this occurs, the quotient and remainder become undefined. This usually occurs when the divisor is 0. The fractional quotient is rounded off.

Bytes: 2

Transfers: None

Flag operation:

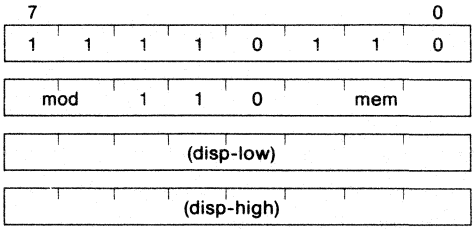
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
MOV     AW,204
MOV     CL,10
DIVU    CL
                ;AL = 20, AH = 4
```

DIVU mem8

Divide unsigned, 8-bit memory



temp ← AW
When temp ÷ (mem8) = FFH:
  AH ← temp % (mem8),
  AL ← temp ÷ (mem8).
When temp ÷ (mem8) > FFH:
  (SP−1,SP−2) ← PSW,
  (SP−3,SP−4) ← PS,
  (SP−5,SP−6) ← PC,
  SP ← SP − 6
  IE ← 0,
  BRK ← 0,
  PS ← (003H, 002H),
  PC ← (001H, 000H),

Divides (using unsigned division) the contents of the AW 16-bit register by the 8-bit memory contents specified by the operand. The quotient is stored in the AL register and the remainder, if any, is stored in the AH register.

When the quotient exceeds FFH — the capacity of the AL destination register — the vector 0 interrupt is generated. When this occurs, the quotient and remainder become undefined. This especially occurs when the divisor is 0. The fractional quotient is rounded off.

Bytes: 2/3/4

Transfers: 1

Flag operation:

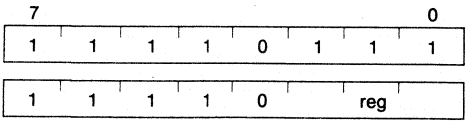
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
MOV     AW,3410
MOV     [BW],19
DIVU    [BW]
                ;AL = 179, AH = 9
```

DIVU reg16

Divide unsigned, 16-bit register



temp ← DW,AW
When temp ÷ reg16 > FFFFH:
(SP−1,SP−2) ← PSW,
(SP−3,SP−4) ← PS,
(SP−5,SP−6) ← PC,
SP ← SP − 6
IE ← 0,
BRK ← 0
PS ← (003H, 002H),
PC ← (001H, 000H)
All other times:
DW ← temp % reg16, AW ← temp ÷ reg16

Divides (using unsigned division) the contents of the DW and AW 16-bit register pair by the contents of the 16-bit register specified by the operand. The quotient is stored in the AW register. The remainder, if any, is stored in the DW register. When the quotient exceeds FFFFH (the capacity of the AW destination register) the vector 0 interrupt is generated, and the quotient and remainder become undefined. This most often occurs when the divisor is 0. The fractional quotient is rounded off.

Bytes: 2

Transfers: None

Flag operation:

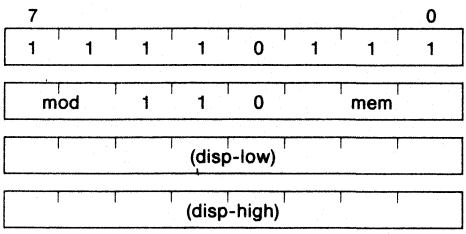
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
MOV     DW,0348H
MOV     AW,2197H
            ;DW,AW = 03482197H
MOV     BW,2000H
DIVU    BW
            ;AW = 1A41H,DW = 0197H
```

DIVU mem 16

Divide unsigned, 16-bit memory



temp ← DW,AW
When temp ÷ (mem16) > FFFFH:
(SP−1,SP−2) ← PSW,
(SP−3,SP−4) ← PS,
(SP−5,SP−6) ← PC,
SP ← SP−6
IE ← 0,
BRK ← 0,
PS ← (003H, 002H),
PC ← (001H, 000H)
All other times:
DW ← temp % (mem16), AL ← temp ÷ (mem16)

Divides (using unsigned division) the contents of the DW and AW 16-bit register pair by the 16-bit memory contents specified by the operand. The quotient is stored in the AW register. The remainder, if any, is stored in the DW register.

When the quotient exceeds FFFFH (the capacity of the AW destination register) the vector 0 interrupt is generated and the quotient and remainder become undefined. This especially occurs when the divisor is 0. The fractional quotient is rounded off.

Bytes: 2/3/4

Transfers: 1

Flag operation:

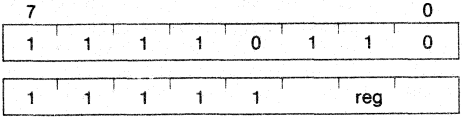
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
MOV     DW,0
MOV     AW,100
MOV     [IX][BX],5
DIVU    [IX][BX]
            ;AW = 0014H = 20,DW = 0
```

DIV reg8

Divide signed, 8-bit register



temp ← AW
When temp ÷ reg8 > 0 and temp ÷ reg8 > 7FH or temp ÷ reg8 > 0 and temp ÷ reg8 > 0-7FH-1:
(SP−1,SP−2) ← PSW,
(SP−3,SP−4) ← PS,
(SP−5,SP−6) ← PC,
SP ← SP − 6,
IE ← 0,
BRK ← 0,
PS ← (003H, 002H),
PC ← (001H, 000H)
All other times:
AH ← temp % reg8,
AL ← temp ÷ reg8

Divides (using signed division) the contents of the AW 16-bit register by the contents of the 8-bit register specified by the operand. The quotient is stored in the AL 8-bit register. The remainder, if any, is stored in the AH register. The maximum value of a positive quotient is +127 (7FH), and the minimum value of a negative quotient is −127 (81H).

When a quotient is greater than either maximum value(s) the quotient and remainder become undefined, and the vector 0 interrupt is generated. This especially occurs when the divisor is 0. A fractional quotient is rounded off. The remainder will have the same sign as the dividend.

Bytes: 2

Transfers: None

Flag operation:

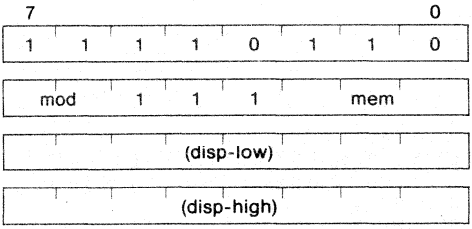
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
MOV   AW, −247
MOV   CL,3
DIV   CL
          ;AL = −82,AH = −1
```

DIV mem8

Divide signed, 8-bit memory



temp ← AW
When temp ÷ (mem8) > 0 and (mem8) > 7FH or temp ÷ (mem8) < 0 and temp ÷ (mem8) > 0-7FH-1:
(SP−1,SP−2) ← PSW,
(SP−3,SP−4) ← PS,
(SP−5,SP−6) ← PC,
SP ← SP − 6,
IE ← 0,
BRK ← 0,
PS ← (003H, 002H),
PC ← (001H, 000H),
All other times:
AH ← temp % (mem8), AL ← temp ÷ (mem8)

Divides (using signed division) the contents of the AW 16-bit register by the contents of the 8-bit memory location specified by the operand. The quotient is stored in the 8-bit AL register, while the remainder, if any, is stored in the AH register. The maximum value of a positive quotient is +127 (7FH), and the minimum value of a negative quotient is −127 (81H). When a quotient is greater than either maximum value(s), the quotient and remainder become undefined and the vector 0 interrupt is generated.

This especially occurs when the divisor is 0. A fractional quotient is rounded off. The remainder will have the same sign as the dividend.

Bytes: 2/3/4

Transfers: 1

Flag operation:

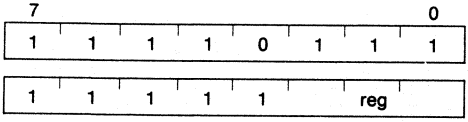
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
MOV   AW,1234
MOV   [BW], −20
DIV   [BW]
          ;AL = −61, AH = 14
```

DIV reg16

Divide signed, 16-bit register



temp ← DW,AW
When temp ÷ reg16 > 0 and temp ÷ reg16 < 7FFFH or temp ÷ reg16 < 0 and temp ÷ reg16 > 0-7FFFH-1:
(SP−1,SP−2) ← PSW,
(SP−3,SP−4) ← PS,
(SP−5,SP−6) ← PC,
SP ← SP − 6,
IE ← 0,
BRK ← 0,
PS ← (003H, 002H),
PC ← (001H, 000H)
All other times:
DW ← temp % reg16, AW ← temp ÷ reg16

Divides (using signed division) the contents of the DW and AW 16-bit register pair by the contents of the 16-bit register specified by the operand. The quotient is stored in the AW 16-bit register, while the remainder, if any, is stored in the DW register. The maximum value of a positive quotient is +32,767 (7FFFH) and the minimum value of a negative quotient is −32,767 (8001H). When the quotient is greater than either maximum value(s), the quotient and remainder become undefined, and the vector 0 interrupt is generated. This especially occurs when the divisor is 0. A fractional quotient is rounded off. The remainder will have the same sign as the dividend.

Bytes: 2

Transfers: None

Flag operation:

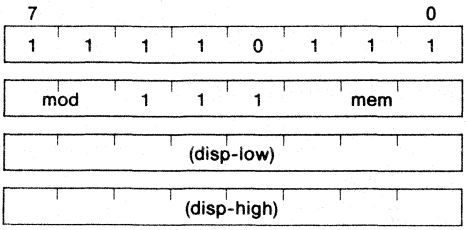
V	S	Z	AC	P	CY
U	U	U	U	U	U

Example:

```
MOV     DW,0123H
MOV     AW,4567H
MOV     CW,1000H
DIV     CW
            ;AW = 1234H, DW = 0567H
```

DIV mem16

Divide signed, 16-bit memory



temp ← DW,AW
When temp ÷ (mem16) > 0 and temp ÷ (mem16) < 7FFFH or temp ÷ (mem16) < 0 and temp ÷ (mem16) > 0-7FFFH-1:

(SP−1,SP−2) ← PSW,
(SP−3,SP−4) ← PS,
(SP−5,SP−6) ← PC,
SP ← SP − 6,
IE ← 0,
BRK ← 0,
PS ← (003H, 002H),
PC ← (001H, 000H)

All other times:

DW ← temp % (mem16), AW ← temp ÷ (mem16)

Divides (using signed division) the contents of the DW and the AW 16-bit register pair by the contents of the 16-bit memory location specified by the operand. The quotient is stored in the AW 16-bit register, while the remainder, if any, is stored in the DW register. The maximum value of a positive quotient is +32,767 (7FFFH), and the minimum value of a negative quotient is −32,767 (8001H). When the quotient is greater than either maximum value(s), the quotient and remainder become undefined and the vector 0 interrupt is generated. This especially occurs when the divisor is 0. A fractional quotient is rounded off. The remainder will have the same sign as the dividend.

Bytes: 2/3/4

Transfers: 1

Flag operation:

V	S	Z	AC	P	CY
U	U	U	U	U	U

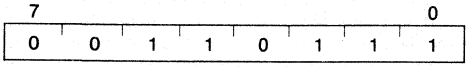
Example:

```
MOV   DW,0
MOV   AW, −34
MOV   [IY], −2
      DIV [IY]
      ;AW = 17, DW = 0
```

BCD ADJUST

ADJBA (no operand)

Adjust byte add



Adjusts the result of unpacked decimal addition stored in the AL register into a single unpacked decimal number. The higher 4 bits become zero.

When AL AND 0FH > 9 or AC=1:
AL ← AL + 6,
AH ← AH + 1,
AC ← 1,
CY ← AC,
AL ← AL AND 0FH

Bytes: 1

Transfers: None

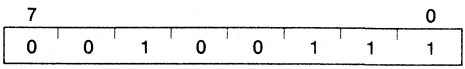
Flag operation:

V	S	Z	AC	P	CY
U	U	U	X	U	X

Example: ADJBA

ADJ4A (no operand)

Adjust Nibble Add



When AL AND 0FH < 9 or AC=1:
AL ← AL + 6,
CY ← CY OR AC,
AC ← 1
When AL > 9FH or CY=1:
AL ← AL + 60H,
CY ← 1

Adjusts the result of packed decimal addition stored in the AL register into a single packed decimal number.

Bytes: 1

Transfers: None

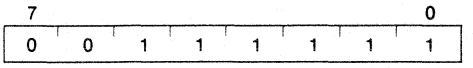
Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

Example: ADJ4A

ADJBS (no operand)

Adjust byte subtract



When AL AND 0FH > 9 or AC=1:
AL ← AL − 6,
AH ← AH − 1,
AC ← 1,
CY ← AC,
AL ← AL AND 0FH

Adjust the result of unpacked decimal subtraction stored in the AL register into a single unpacked decimal number. The higher 4 bits become zero.

Bytes: 1

Transfers: None

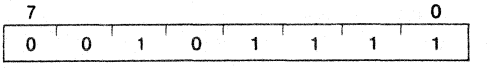
Flag operation:

V	S	Z	AC	P	CY
U	U	U	X	U	X

Example: ADJBS

ADJ4S (no operand)

Adjust nibble subtract



When AL AND 0FH > 9 or AC=1:
AL ← AL − 6,
CY ← AC OR CY,
AC ← 1,
When AL > 9FH or CY=1:
AL ← AL − 60H,
CY ← 1

Adjusts the result of packed decimal subtraction stored in the AL register into a single packed decimal number.

Bytes: 1

Transfers: None

Flag operation:

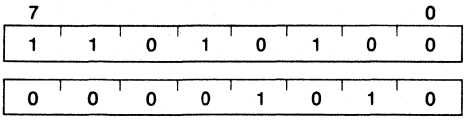
V	S	Z	AC	P	CY
U	X	X	X	X	X

Example: ADJ4S

DATA CONVERSION

CVTBD (no operand)

Convert binary to decimal



AH ← AL ÷ 0AH
AL ← AL ...% 0AH

Converts the binary 8-bit value in the AL register into a two-digit unpacked decimal number.

The quotient of AL divided by 10 is stored in the AH register. The remainder of this operation is stored in the AL register.

Bytes: 2

Transfers: None

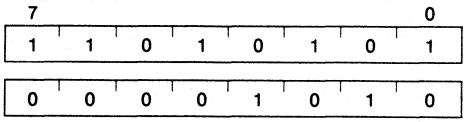
Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	U

Example: CVTBD

CVTDB (no operand)

Convert decimal to binary



AL ← AH × 0AH + AL
AH ← 0

Converts a two-digit unpacked decimal number in the AH and AL registers into a single 16-bit binary number. The value in the AH is multiplied by 10. The product is added to the contents of the AL register and the result is stored in AL. AH becomes 0.

Bytes: 2

Transfers: None

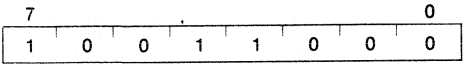
Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	U

Example: CVTDB

CVTBW (no operand)

Convert byte to word



When AL < 80H:
AH ← 0
All other times
AH ← FFH

Expands the sign of the byte in the AL register to the AH register. Use this instruction to produce a double-length (word) dividend from a byte before a byte division is performed.

Bytes: 1

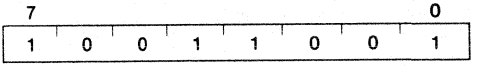
Transfers: None

Flag operation: None

Example: CVTBW

CVTWL (no operand)

Convert word to long word



When AW < 8000H:
DW ← 0

All other times :
DW ← FFFFH

Expands the sign of the word in the AW register to the DW register. Use this instruction to produce a double-length (double-word) dividend from a word before a word division is performed.

Bytes: 1

Transfers: None

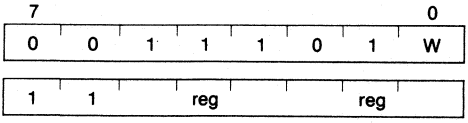
Flag operation: None

Example: CVTWL

COMPARISON

CMP reg,reg

Compare register and register



reg — reg

Subtracts the contents of the 8- or 16-bit register specified by the second operand from the contents of the 8- or 16-bit register specified by the first operand. The result is not stored and only the flags are affected.

Bytes: 2

Transfers: None

Flag operation:

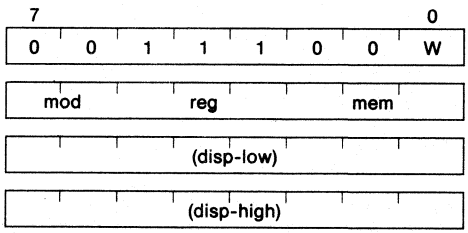
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
CMP     AW,BW
CMP     CH,DL
```

CMP mem,reg

Compare memory and register



(mem) — reg

Subtracts the contents of the 8- or 16-bit register specified by the second operand from the 8- or 16-bit memory contents addressed by the first operand. The result is not stored and only the flags are affected.

Bytes: 2/3/4

Transfers: 1

Flag operation:

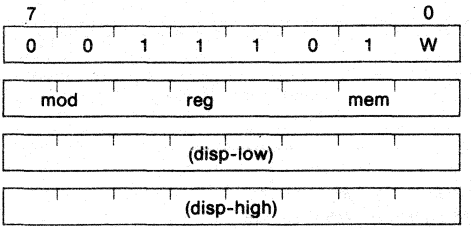
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
CMP     WORD_VAR,IX
CMP     BYTE_VAR,CL
CMP     [BW],AH
```

CMP reg,mem

Compare register and memory



Subtracts the 8- or 16-bit memory contents addressed by the second operand from the contents of the 8- or 16-bit register specified by the first operand. The result is not stored and only the flags are affected.

reg − (mem)

Bytes: 2/3/4

Transfers: 1

Flag operation:

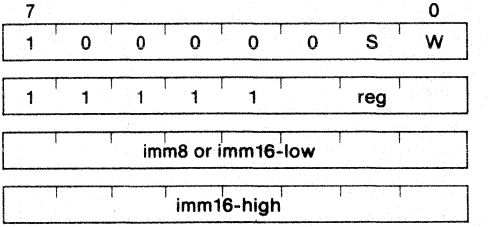
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
CMP    AW,[IX]
CMP    CH,BYTE_VAR
```

CMP reg,imm

Compare register and immediate data



reg − imm

Subtracts the 8- or 16-bit immediate data specified by the second operand from the contents of the 8- or 16-bit register specified by the first operand. The result is not stored and only the flags are affected.

Bytes: 3/4

Transfers: None

Flag operation:

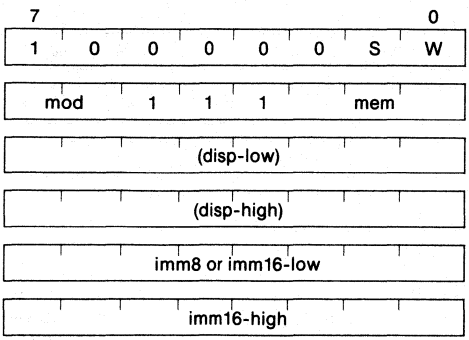
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
CMP    BL,5
CMP    DW,1200H
```

CMP mem,imm

Compare memory and immediate data



(mem) − imm

Subtracts the 8- or 16-bit immediate data specified by the second operand from the 8- or 16-bit memory contents addressed by the first operand. The result is not stored and only the flags are affected.

Bytes: 3/4/5/6

Transfers: 1

Flag operation:

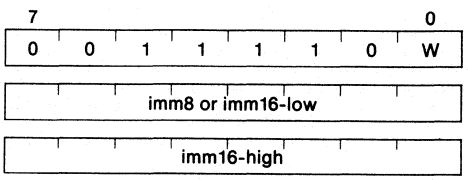
V	S	Z	AC	P	CY
X	X	X	X	X	X

Example:

```
CMP    BYTE PTR [BW],3
CMP    WORD_VAR,7000H
```

CMP acc,imm

Compare accumulator and immediate data



When W=0: AL − imm8
When W=1: AW − imm16

Subtracts the 8- or 16-bit immediate data specified by the second operand from the accumulator (AL or AW) specified by the first operand. The result is not stored and only the flags are affected.

Bytes: 2/3

Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X	X	X	X	X	X

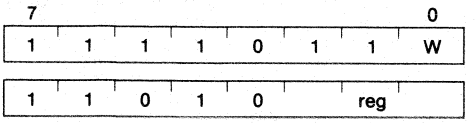
Example:

```
CMP    AL,0
CMP    AW,800H
```

COMPLEMENT OPERATION

NOT reg

Not register



reg ← $\overline{reg}$

Inverts (by performing a 1's complement) each bit of the 8- or 16-bit register specified by the operand and stores the result in the specified register.

Bytes: 2

Transfers: None

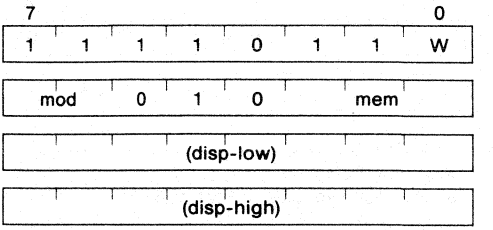
Flag operation: None

Example:

```
NOT    BW
NOT    CL
```

NOT mem

Not memory



(mem) ← ($\overline{mem}$)

Inverts (by performing a 1's complement) each bit of the 8- or 16-bit memory location addressed by the operand and stores the result in the addressed memory location.

Bytes: 2/3/4

Transfers: 2

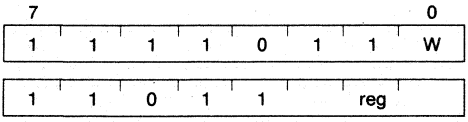
Flag operation: None

Example:

```
NOT    WORD_VAR[IX][2]
NOT    BYTE PTR [IY]
```

NEG reg

Negate register



reg ← $\overline{\text{reg}}$ + 1

Takes the 2's complement of the contents of the 8- or 16-bit register specified by the operand.

Bytes: 2

Transfers: None

Flag operation:

V	S	Z	AC	P	CY*
X	X	X	X	X	1

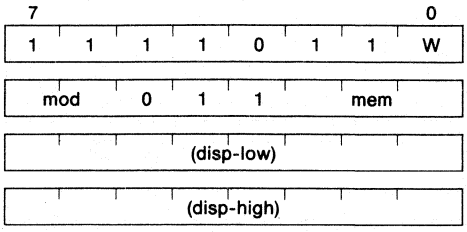
Note: * = 0 if the contents of the operand register is 0.

Example:

```
NEG     BL
NEG     AW
```

NEG mem

Negate memory



(mem) ← ($\overline{\text{mem}}$) + 1

Takes the 2's complement of the 8- or 16-bit memory contents addressed by the operand.

Bytes: 2/3/4

Transfers: 2

Flag operation:

V	S	Z	AC	P	CY*
X	X	X	X	X	1

Note: * = 0 if the contents of the memory operand is 0.

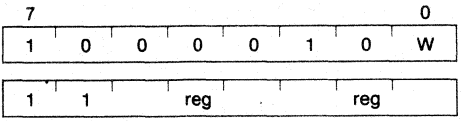
Example:

```
NEG     WORD_VAR
NEG     BYTE PTR [BW][IX]
```

LOGICAL OPERATION

TEST reg,reg

Test register and register



reg AND reg

ANDs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit register specified by the second operand. The result is not stored and only the flags are affected.

Bytes: 2

Transfers: None

Flag operation:

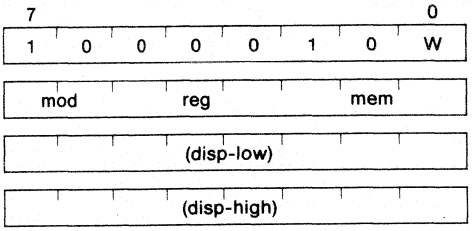
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
TEST    AW,CW
TEST    CL,AH
```

TEST mem,reg or TEST reg,mem

Test register and memory



(mem) AND reg

ANDs the contents of the 8- or 16-bit second operand and the contents of the 8- or 16-bit first operand.

The result is not stored and only the flags are affected.

Bytes: 2/3/4

Transfers: 1

Flag operation:

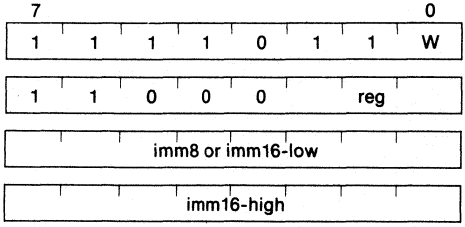
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
TEST    BYTE_VAR,DL
TEST    AH, [IX]
```

TEST reg,imm

Test immediate data and register



reg AND imm

ANDs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit immediate data specified by the second operand. The result is not stored and only the flags are affected.

Bytes: 3/4

Transfers: None

Flag operation:

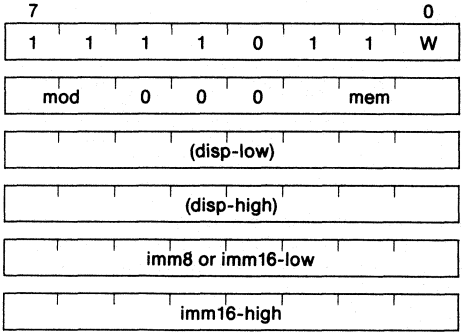
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
TEST    CW,1
TEST    AL,50H
```

TEST mem,imm

Test immediate data and memory



(mem) AND imm

ANDs the 8- or 16-bit memory contents addressed by the first operand and the 8- or 16-bit immediate data specified by the second operand. The result is not stored and only the flags are affected.

Bytes: 3/4/5/6

Transfers: 1

Flag operation:

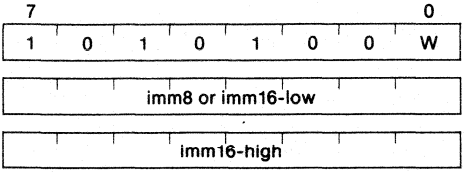
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
TEST    BYTE PTR [BW],80H
TEST    WORD_VAR,00FFH
```

TEST acc,imm

Test immediate data and accumulator



When W=0: AL AND imm8
When W=1: AW AND imm16

ANDs the contents of the accumulator (AL or AW) specified by the first operand and the 8- or 16-bit immediate data specified by the second operand. The result is not stored and only the flags are affected.

Bytes: 2/3

Transfers: None

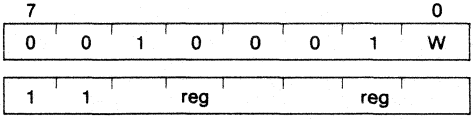
Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:
TEST AL,12H
TEST AW,8000H

AND reg,reg

AND register with register to register



reg ← reg AND reg

ANDs the contents of the 8- or 16-bit register specified by the first operand and the contents of the 8- or 16-bit register specified by the second operand. Stores the result in the register specified by the first operand.

Bytes: 2

Transfers: None

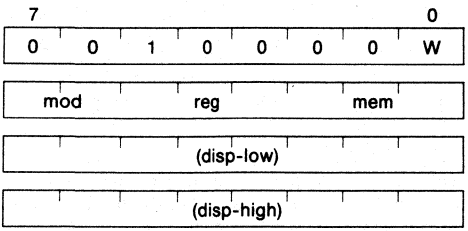
Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	0

Example: AND IX,AW

AND mem,reg

AND memory with register to memory



(mem) ← (mem) AND reg

ANDs the 8- or 16-bit memory contents addressed by the first operand and the contents of the 8- or 16-bit register specified by the second operand. Stores the result in the memory location addressed by the first operand.

Bytes: 2/3/4

Transfers: 2

Flag operation:

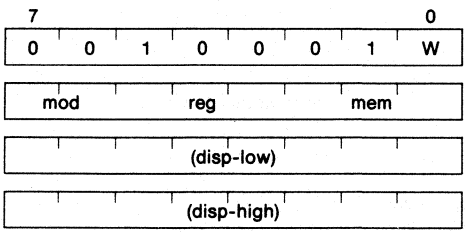
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
AND     [BW][IX]3,AL
AND     WORD_VAR,CW
```

AND reg,mem

AND register with memory to register



reg ← reg AND (mem)

ANDs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit memory contents addressed by the second operand. Stores the result in the register specified by the first operand.

Bytes: 2/3/4

Transfers: 1

Flag operation:

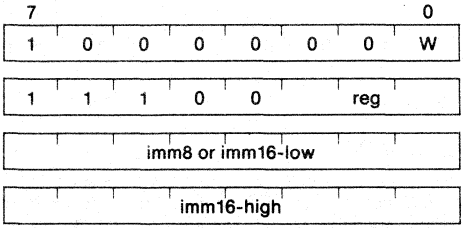
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
AND     CL,BYTE_VAR
AND     DW,[IY]
```

AND reg,imm

AND register with immediate data to register



reg ← reg AND imm

ANDs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the register specified by the first operand.

Bytes: 3/4

Transfers: None

Flag operation:

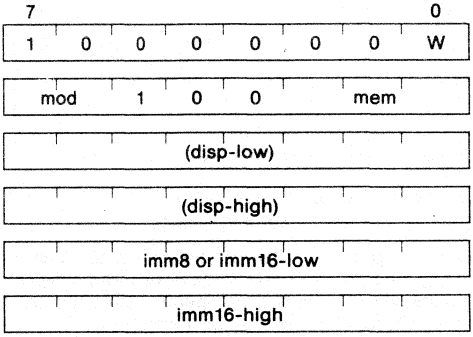
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
AND    CL,0FEH
AND    DW,14H
```

AND mem,imm

AND memory with immediate data to memory



(mem) ← (mem) AND imm

ANDs the 8- or 16-bit memory contents addressed by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the memory location addressed by the first operand.

Bytes: 3/4/5/6

Transfers: 2

Flag operation:

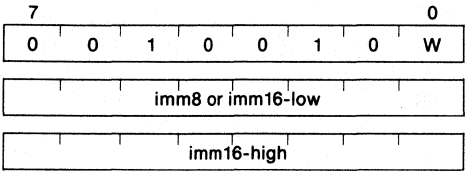
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
AND    BYTE PTR [IY],30H
AND    [IY],3000H
```

AND acc,imm

AND accumulator with immediate data to accumulator



When W=0: AL ← AL AND imm8
When W=1: AW ← AW AND imm16

ANDs the contents of the accumulator (AL or AW) specified by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the accumulator specified by the first operand.

Bytes: 2/3

Transfers: None

Flag operation:

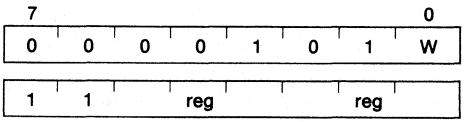
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
AND     AL,80H
AND     AW,0FH
```

OR reg,reg

OR register and register to register



reg ← reg OR reg

ORs the contents of the 8- or 16-bit register specified by the first operand and the contents of the 8- or 16-bit register specified by the second operand. Stores the result in the register specified by the first operand.

Bytes: 2

Transfers: None

Flag operation:

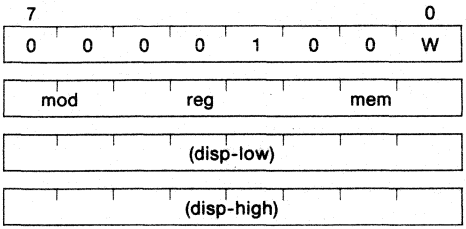
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
OR      AL,AH
OR      BW,CW
```

OR mem,reg

OR memory and register to memory



(mem) ← (mem) OR reg

ORs the 8- or 16-bit memory contents addressed by the first operand and the contents of the 8- or 16-bit register specified by the second operand. Stores the result in the memory location addressed by the first operand.

Bytes: 2/3/4

Transfers: 2

Flag operation:

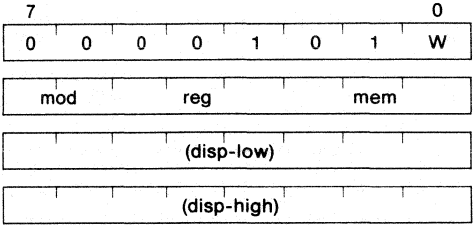
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
OR    BYTE_VAR,CL
OR    WORD_VAR [BP],AW
```

OR reg,mem

OR register and memory to register



reg ← reg OR (mem)

ORs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit memory contents addressed by the second operand. Stores the result in the register specified by the first operand.

Bytes: 2/3/4

Transfers: 1

Flag operation:

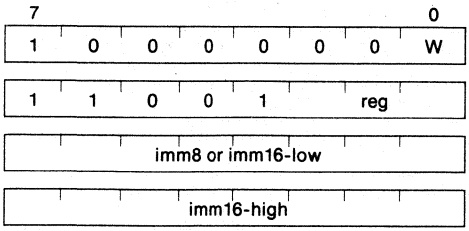
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example

```
OR    CL,[IX]
OR    CW,WORD_VAR
```

OR reg,imm

OR register with immediate data to register



reg ← reg OR imm

ORs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the register specified by the first operand.

Bytes: 3/4

Transfers: None

Flag operation:

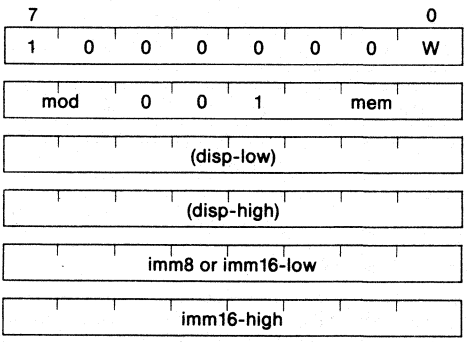
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
OR    CL,80H
OR    AW,0FH
```

OR mem,imm

OR memory with immediate data to memory



(mem) ← (mem) OR imm

ORs the 8- or 16-bit memory contents addressed by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the memory location addressed by the first operand.

Bytes: 3/4/5/6

Transfers: 2

Flag operation:

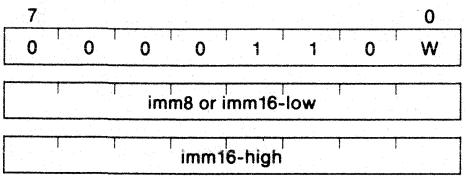
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
OR    BYTE_VAR,2
OR    WORD PTR [IX],0FH
```

OR acc,imm

OR accumulator with immediate data to accumulator



When W=0: AL ← AL OR imm8
When W=1: AW ← AW OR imm16

ORs the contents of the accumulator (AL or AW) specified by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the accumulator specified by the first operand.

Bytes: 2/3

Transfers: None

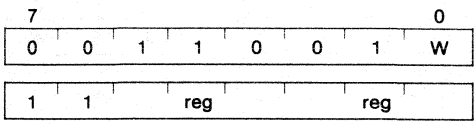
Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:
OR AL,34H
OR AW,1

XOR reg,reg

Exclusive OR, register and register to register



reg ← reg XOR reg

XORs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit register specified by the second operand. Stores the result in the register specified by the first operand.

Bytes: 2

Transfers: None

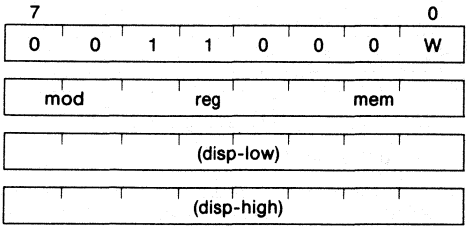
Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:
XOR AL,AH
XOR CW,BW

XOR mem,reg

Exclusive OR, memory and register to memory



(mem) ← (mem) XOR reg

XORs the 8- or 16-bit memory contents addressed by the first operand and the contents of the 8- or 16-bit register specified by the second operand. Stores the result in the memory location addressed by the first operand.

Bytes: 2/3/4

Transfers: 2

Flag operation:

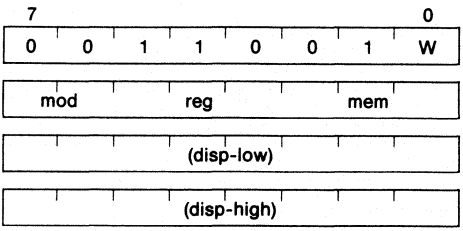
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example

```
XOR     [BW],CL
XOR     WORD_VAR,BP
```

XOR reg,mem

Exclusive OR, register and memory to register



reg ← reg XOR (mem)

XORs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit memory contents addressed by the second operand. Stores the result in the register specified by the first operand.

Bytes: 2/3/4

Transfers: 1

Flag operation:

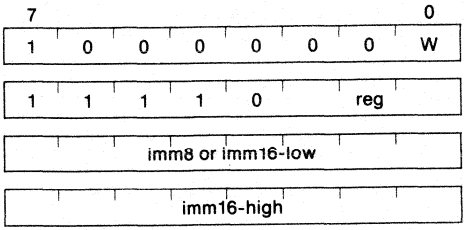
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example

```
XOR     BH,[IX]
XOR     AW,WORD_VAR
```

XOR reg,imm

Exclusive OR, register with immediate data to register



reg ← reg XOR imm

XORs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the register specified by the first operand.

Bytes: 3/4

Transfers: None

Flag operation:

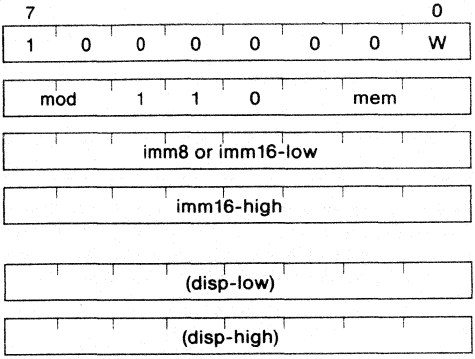
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example

```
XOR    CL,2
XOR    IX,0FF00H
```

XOR mem,imm

Exclusive OR, memory with immediate data to memory



(mem) ← (mem) XOR imm

XORs the 8- or 16-bit memory contents addressed by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the memory location addressed by the first operand.

Bytes: 3/4/5/6

Transfers: 2

Flag operation:

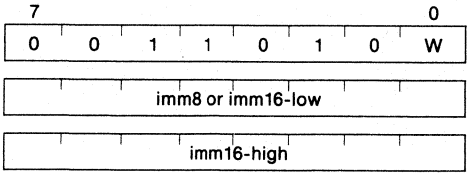
V	S	Z	AC	P	CY
0	X	X	U	X	0

Example:

```
XOR    BYTE PTR [IY],0FH
XOR    WORD_VAR,0FH
```

XOR acc,imm

Exclusive OR, accumulator with immediate data to accumulator



XORs the contents of the accumulator (AL or AW) specified by the first operand and the 8- or 16-bit immediate data specified by the second operand. Stores the result in the accumulator specified by the first operand.

When W=0: AL ← AL XOR imm8
When W=1: AW ← AW XOR imm16

Bytes: 2/3

Transfers: None

Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	0

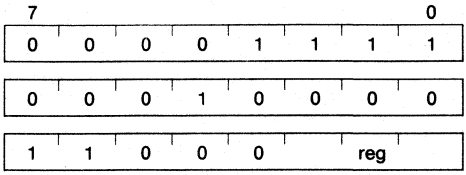
Example:

```
XOR     AL,0FFH
XOR     AW,8000H
```

BIT MANIPULATION

TEST1 reg8,CL

Test bit CL of the 8-bit register



When bit CL of reg8=0: Z ← 1
When bit CL of reg8=1: Z ← 0

Sets the Z flag to 1 when bit CL of the 8-bit register (specified by the first operand) is 0. Resets the Z flag to 0 when bit CL is 1. Only the lower 3 bits of CL are used to address the bit.

Bytes: 3

Transfers: 1

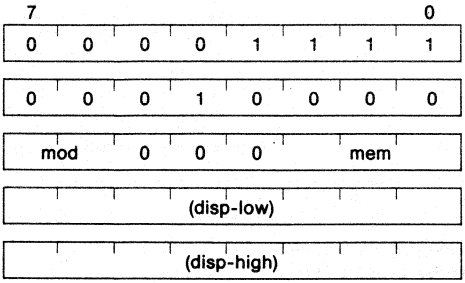
Flag operation:

V	S	Z	AC	P	CY
0	U	X	U	U	0

Example: TEST1 AL,CL

TEST1 mem8,CL

Test bit CL of the 8-bit memory



When bit CL of (mem8) = 0: Z ← 1
When bit CL of (mem8) = 1: Z ← 0

Sets the Z flag to 1 when bit CL of the 8-bit memory (addressed by the first operand) is 0. Resets the Z flag to 0 when the CL bit is 1. Only the lower 3 bits of CL are used to address the bit.

Bytes: 3/4/5

Transfers: 1

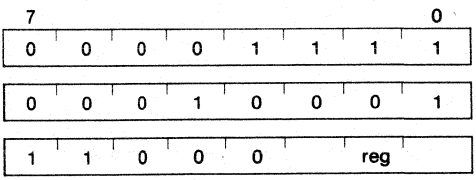
Flag operation:

V	S	Z	AC	P	CY
0	U	X	U	U	0

Example: TEST1 BYTE PTR [BW],CL

TEST1 reg16,CL

Test bit CL of the 16-bit register



When bit CL of reg16 = 0: Z ← 1
When bit CL of reg16 = 1: Z ← 0

Sets the Z flag to 1 when bit CL of the 16-bit register (specified by the first operand) is 0. Resets the Z flag to 0 when the bit is 1. Only the lower 4 bits of CL are used to address a bit.

Bytes: 3

Transfers: 1

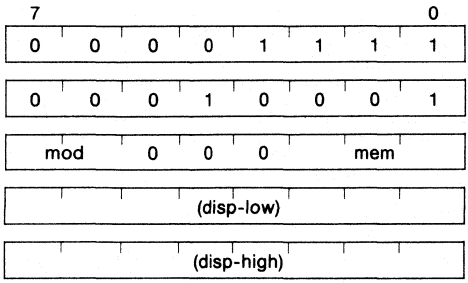
Flag operation:

V	S	Z	AC	P	CY
0	U	X	U	U	0

Example: TEST1 AW,CL

TEST1 mem16,CL

Test bit CL of the 16-bit memory



When bit CL of (mem16) = 0: Z ← 1
When bit CL of (mem16) = 1: Z ← 0

The first operand specifies the 16-bit memory location and the second operand (CL) specifies the bit position. When the bit specified by CL is 0, the Z flag is set to 1. When that bit is 1, the Z flag is reset to 0. Only the lower 4 bits of CL are used to address a bit.

Bytes: 3/4/5

Transfers: 1

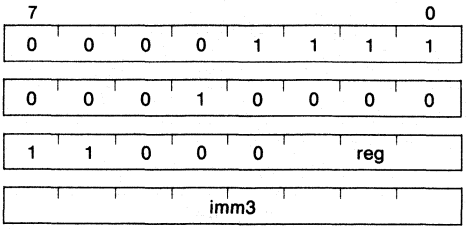
Flag operation:

V	S	Z	AC	P	CY
0	U	X	U	U	0

Example: TEST1 WORD PTR [BW],CL

TEST1 reg8, imm3

Test bit imm3 of the 8-bit register



When bit imm3 of reg8 = 0: Z ← 1
When bit imm3 of reg8 = 1: Z ← 0

Sets the Z flag to 1 when bit imm3 of the 8-bit register (specified by the first operand) is 0. Resets the Z flag to 0 when the bit is 1. Only the lower 3 bits of the immediate data are used to identify a bit.

Bytes: 4

Transfers: None

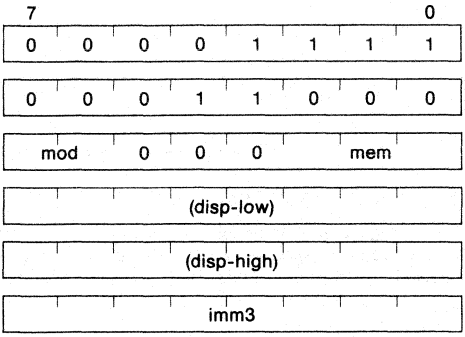
Flag operation:

V	S	Z	AC	P	CY
0	U	X	U	U	0

Example: TEST1 BH,1

TEST1 mem8,imm3

Test bit imm3 of the 8-bit memory



When bit imm3 of (mem8) = 0: Z ← 1
When bit imm3 of (mem8) = 1: Z ← 0

The first operand specifies the 8-bit memory location and the second operand (imm3) specifies the bit position. When the bit specified by imm3 is 0, the Z flag is set to 1. When that bit is 1, the Z flag is reset to 0. Only the lower 3 bits of the immediate data are used to address a bit.

Bytes: 4/5/6

Transfers: 1

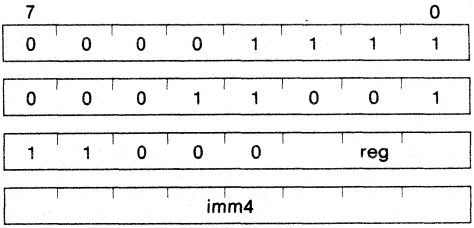
Flag operation:

V	S	Z	AC	P	CY
0	U	X	U	U	0

Example: TEST1 BYTE_VAR,5

TEST1 reg16, imm4

Test bit imm4 of the 16-bit register



When bit imm4 of reg16 = 0: Z ← 1
When bit imm4 of reg16 = 1: Z ← 0

The first operand specifies the 16-bit register and the second operand (imm4) specifies the bit position. When the bit specified by imm4 is 0, the Z flag is set to 1. When that bit is 1, the Z flag is reset to 0. Only the lower 4 bits of the immediate data are used to address a bit.

Bytes: 4

Transfers: None

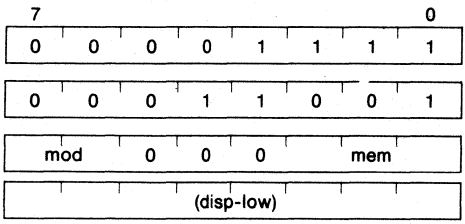
Flag operation:

V	S	Z	AC	P	CY
0	U	X	U	U	0

Example: TEST1 AW,15

TEST1 mem16,imm4

Test bit imm4 of the 16-bit memory



When bit imm4 of (mem16) = 0: Z ← 1
When bit imm4 of (mem16) = 1: Z ← 0

The first operand specifies the 16-bit memory and the second operand (imm4) specifies the bit position. When the bit specified by imm4 is 0, the Z flag is set to 1. When that bit is 1, the Z flag is reset to 0. The immediate data in the last byte of the instruction is valid only for the lower 4 bits.

Bytes: 4/5/6

Transfers: 1

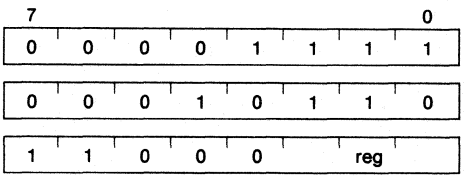
Flag operation:

V	S	Z	AC	P	CY
0	U	X	U	U	0

Example: TEST1 WORD PTR [BP],8

NOT1 reg8,CL

Not bit CL of the 8-bit register



Bit CL of reg8 ← $\overline{\text{bit CL of reg8}}$

The CL register (second operand) specifies which bit of the 8-bit register (specified by the first operand) is to be inverted. Only the lower 3 bits of the CL register are used.

Bytes: 3

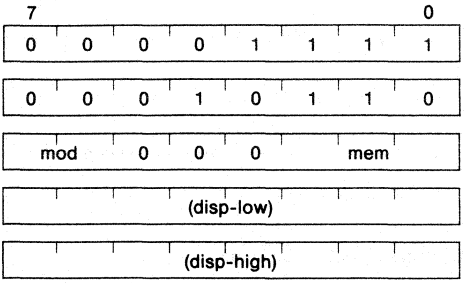
Transfers: None

Flag operation: None

Example: NOT1 BH,CL

NOT1 mem8,CL

Not bit CL of the 8-bit memory



Bit CL of (mem8) ← $\overline{\text{bit CL of (mem8)}}$

The CL register (second operand) specifies which bit of the 8-bit memory location (specified by the first operand) is to be inverted. Only the lower 3 bits of the CL register are used.

Bytes: 3/4/5

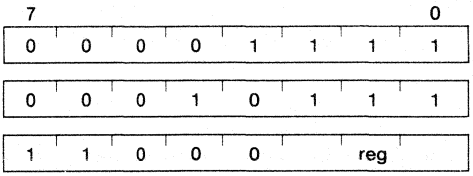
Transfers: 2

Flag operation: None

Example: NOT1 BYTE_VAR,CL

NOT1 reg16, CL

Not bit CL of the 16-bit register



Bit CL of reg16 ← $\overline{\text{bit CL of reg16}}$

The CL register (second operand) specifies which bit of the 16-bit register (specified by the first operand) is to be inverted. Only the lower 4 bits of the CL register are used.

Bytes: 3

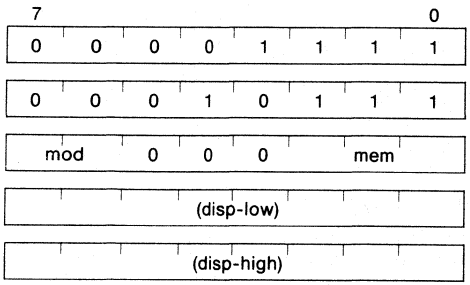
Transfers: None

Flag operation: None

Example: NOT1 AW,CL

NOT1 mem16,CL

Not bit CL of the 16-bit memory



Bit CL of (mem16) ← $\overline{\text{bit CL of (mem16)}}$

The CL register (second operand) specifies which bit of the 16-bit memory location (addressed by the first operand) is to be inverted. Only the lower 4 bits of the CL register are used.

Bytes: 3/4/5

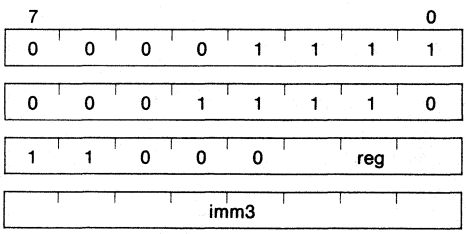
Transfers: 2

Flag operation: None

Example: NOT1 WORD_VAR,CL

NOT1 reg8,imm3

Not bit imm3 of the 8-bit register



Bit imm3 of reg8 ← $\overline{\text{bit imm3 of reg8}}$

Bit imm3 (second operand) specifies which bit of the 8-bit register (specified by the first operand) is to be inverted. Only the lower 3 bits of the immediate data at the fourth byte of the instruction are used.

Bytes: 4

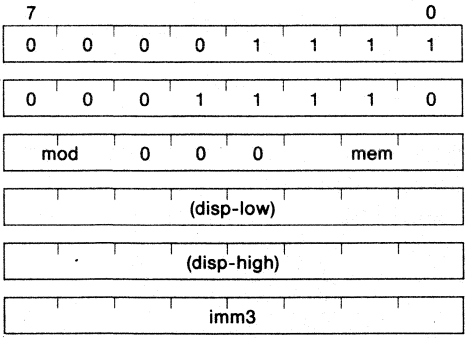
Transfers: None

Flag operation: None

Example: NOT1 AH,3

NOT1 mem8,imm3

Not bit imm3 of 8-bit memory



Bit imm3 of mem8 ← $\overline{\text{bit imm3 of mem8}}$

Bit imm3 (second operand) specifies which bit of the 8-bit memory location (addressed by the first operand) is to be inverted. Only the lower 3 bits of the immediate data are used in the last byte of the instruction.

Bytes: 4/5/6

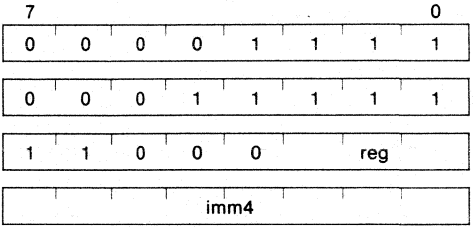
Transfers: 2

Flag operation: None

Example: NOT1 BYTE PTR [BW][IX]34H,4

NOT1 reg16,imm4

Not bit imm4 of the 16-bit register



Bit imm4 of reg16 ← $\overline{\text{bit imm4 of reg16}}$

Bit imm4 (second operand) specifies which bit of the 16-bit register (specified by the first operand) is to be inverted. Only the lower 4 bits of the immediate data are used in the fourth byte of the instruction.

Bytes: 4

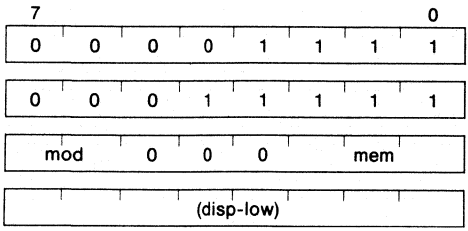
Transfers: None

Flag operation: None

Example: NOT1 BW,15

NOT1 mem16,imm4

Not bit imm4 of the 16-bit memory



Bit imm4 of (mem16) ← $\overline{\text{bit imm4 of (mem16)}}$

The bit imm4 (second operand) specifies which bit of the 16-bit memory location (addressed by the first operand) is to be inverted. Only the lower 4 bits of the immediate data are used in the last byte of the instruction.

Bytes: 4/5/6

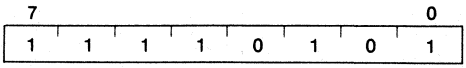
Transfers: 2

Flag operation: None

Example: NOT1 WORD_VAR,0

NOT1 CY

Not carry flag



CY ← $\overline{\text{CY}}$

Inverts the CY flag.

Bytes: 1

Transfers: None

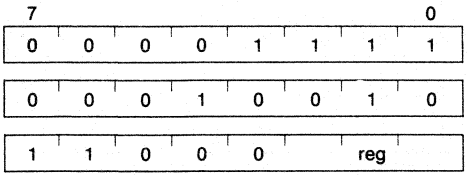
Flag operation:

V	S	Z	AC	P	CY
U	U	U	U	U	X

Example: NOT1 CY

CLR1 reg8,CL

Clear bit CL of the 8-bit register



Bit CL of reg8 ← 0

Clears the bit specified by CL of the 8-bit register (specified by the first operand) to 0. Only the lower three bits of CL are used.

Bytes: 3

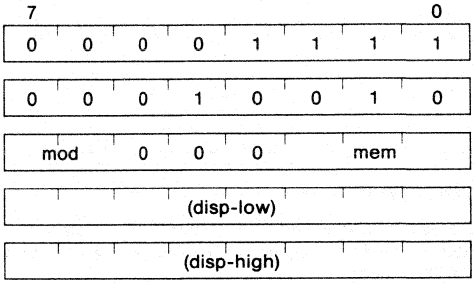
Transfers: None

Flag operation: None

Example: CLR1 AL,CL

CLR1 mem8,CL

Clear bit CL of the 8-bit memory



Bit CL of (mem8) ← 0

Clears the bit specified by CL of the 8-bit memory location (addressed by the first operand) to 0. Only the lower three bits of CL are used.

Bytes: 3/4/5

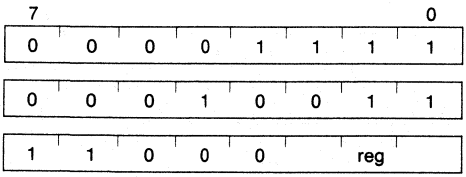
Transfers: 2

Flag operation: None

Example: CLR1 BYTE_VAR,CL

CLR1 reg16,CL

Clear bit CL of the 16-bit register



Bit CL of reg16 ← 0

Clears the bit specified by CL of the 16-bit register (specified by the first operand) to 0. Only the lower four bits of CL are used.

Bytes: 3

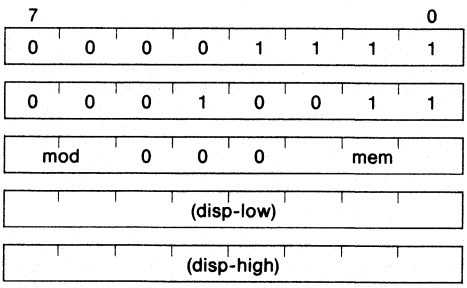
Transfers: None

Flag operation: None

Example: CLR1 AW,CL

CLR1 mem16,CL

Clear bit CL of the 16-bit memory



Bit CL of (mem16) ← 0

Clears the bit specified by CL of the 16-bit memory location (addressed by the first operand) to 0. Only the lower 4 bits of CL are used.

Bytes: 3/4/5

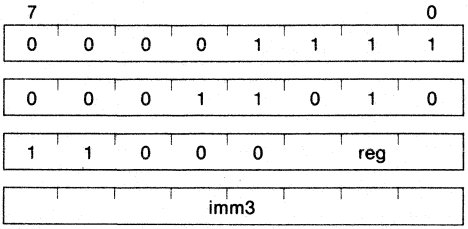
Transfers: 2

Flag operation: None

Example: CLR1 WORD_VAR,CL

CLR1 reg8,imm3

Clear bit imm3 of the 8-bit register



Bit imm3 of reg8 ← 0

Clears the bit specified by the 3-bit immediate data (second operand) of the 8-bit register (specified by the first operand) to 0. Only the lower 3 bits of the immediate data are used in the fourth byte of the instruction.

Bytes: 4

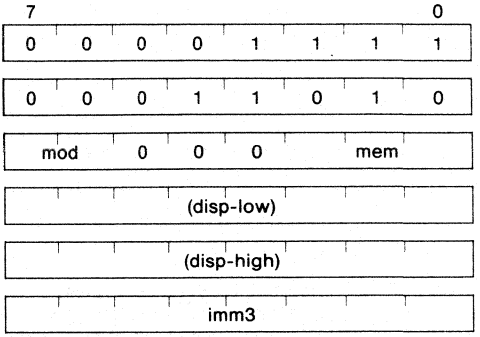
Transfers: None

Flag operation: None

Example: CLR1 BH,1

CLR1 mem8,imm3

Clear bit imm3 of the 8-bit memory



Bit imm3 of (mem8) ← 0

Clears the bit specified by the 3-bit immediate data (second operand) of the 8-bit memory location (addressed by the first operand) to 0. Only the lower 3 bits of immediate data are used in the last byte of the instruction.

Bytes: 4/5/6

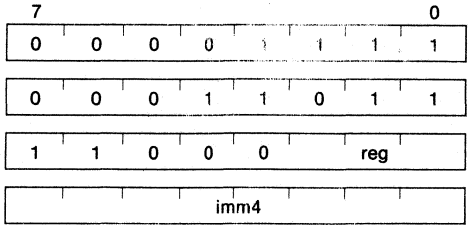
Transfers: 2

Flag operation: None

Example: CLR1 BYTE_VAR[BW],6

CLR1 reg16,imm4

Clear bit imm4 of the 16-bit register



Bit imm4 of reg16 ← 0

Clears the bit specified by the 4-bit immediate data (second operand) of the 16-bit register (specified by the first operand) to 0. Only the lower 4 bits of the immediate data are used in the fourth byte of the instruction.

Bytes: 4

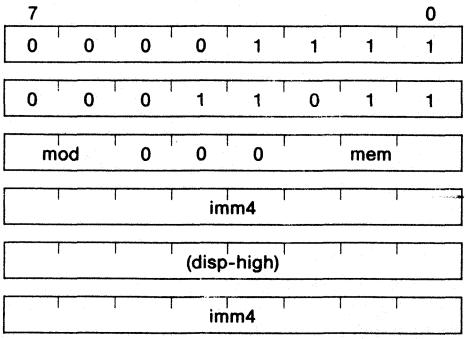
Transfers: None

Flag operation: None

Example: CLR1 CW,5

CLR1 mem16,imm4

Clear bit imm4 of the 16-bit memory



Bit imm4 of (mem16) ← 0

Clears the bit specified by the 4-bit immediate data (second operand) of the 16-bit memory location (addressed by the first operand) to 0. Only the lower 4 bits of immediate data are used in the last byte of the instruction.

Bytes: 4/5/6

Transfers: 2

Flag operation: None

Example: CLR1 WORD PTR [BP],0

CLR1 CY

Clear carry flag

7							0
1	1	1	1	1	0	0	0

CY ← 0

Clears the CY flag.

Bytes: 1

Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U	U	U	U	U	0

Example: CLR1 CY

CLR1 DIR

Clear direction flag

7							0
1	1	1	1	1	1	0	0

DIR ← 0

Clears the DIR flag. Sets index registers IX and IY to autoincrement when MOVBK, CMPBK, CMPM, LDM STM, INM, and OUTM are executed.

Bytes: 1

Transfers: None

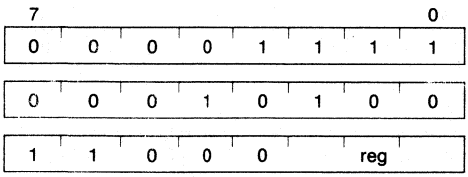
Flag operation:

DIR
0

ple: CLR1 DIR Exam

SET1 reg8,CL

Set bit CL of the 8-bit register



Bit CL of reg8 ← 1

Sets the bit specified by CL of the 8-bit register (specified by the first operand) to 1. Only the lower three bits of CL are used.

Bytes: 3

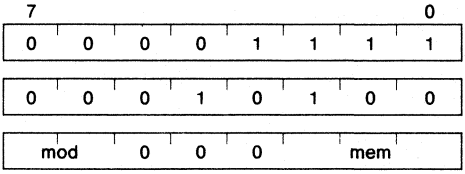
Transfers: None

Flag operation: None

Example: SET1 BL,CL

Set bit CL of the 8-bit memory

SET1 mem8,CL



Bit CL of (mem8) ← 1

Sets the bit specified by CL of the 8-bit memory location (addressed by the first operand) to 1. Only the lower three bits of CL are used.

Bytes: 3/4/5

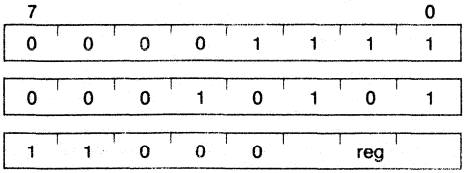
Transfers: 2

Flag operation: None

Example: SET1 BYTE PTR [BW],CL

SET1 reg16,CL

Set bit CL of the 16-bit register



Bit CL of reg16 ← 1

Sets the bit specified by CL of the 16-bit register (specified by the first operand) to 1. Only the lower four bits of CL are used.

Bytes: 3

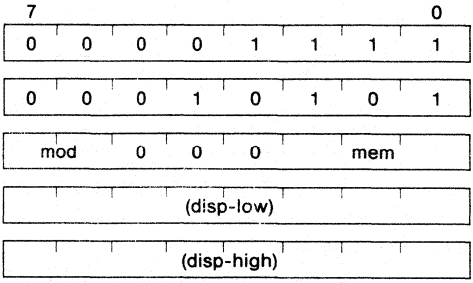
Transfers: None

Flag operation: None

Example: SET1 BW,CL

SET1 mem16,CL

Set bit CL of the 16-bit memory



Bit CL of (mem16) ← 1

Sets the bit specified by CL of the 16-bit memory location (addressed by the first operand) to 1. Only the lower 4 bits of CL are used.

Bytes: 3/4/5

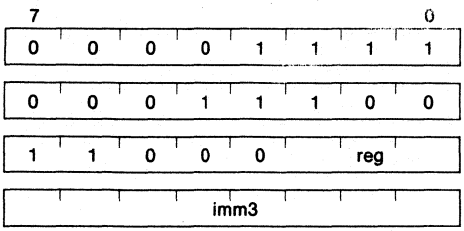
Transfers: 2

Flag operation: None

Example: SET1 WORD_VAR,CL

SET1 reg8,imm3

Set bit imm3 of the 8-bit register



Bit imm3 of reg8 ← 1

Sets the bit specified by the 8-bit immediate data (second operand) of the 8-bit register (specified by the first operand) to 1. Only the lower 3 bits of the immediate data are used in the fourth byte of the instruction.

Bytes: 4

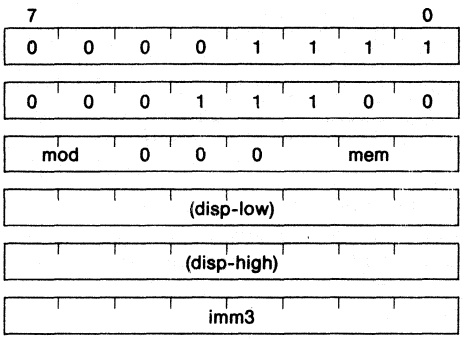
Transfers: None

Flag operation: None

Example: SET1 AL,4

SET1 mem8,imm3

Set bit imm3 of the 8-bit memory



Bit imm3 of (mem8) ← 1

Sets the bit specified by the 3-bit immediate data (second operand) of the 8-bit memory location (addressed by the first operand) to 1. Only the lower 3 bits of the immediate data are used in the last byte of the instruction.

Bytes: 4/5/6

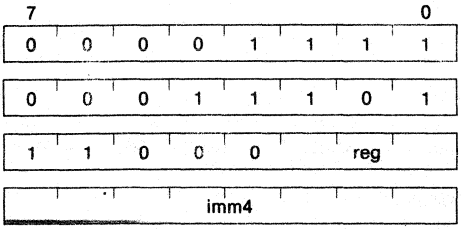
Transfers: 2

Flag operation: None

Example: SET1 BYTE_VAR,5

SET1 reg16,imm4

Set bit imm4 of the 16-bit register



Bit imm4 of reg16 ← 1

Sets the bit specified by the 4-bit immediate data (second operand) of the 16-bit register (specified by the first operand) to 1. Only the lower 4 bits of the immediate data are used in the 4th byte of the instruciton.

Bytes: 4

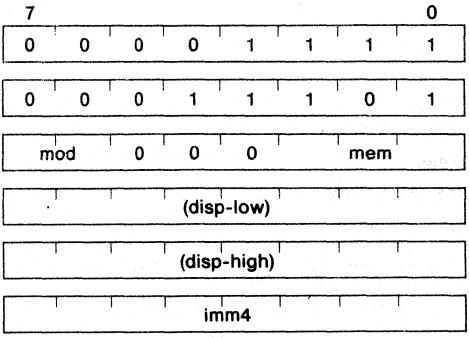
Transfers: None

Flag operation: None

Example: SET1 CW,0

SET1 mem16,imm4

Set bit imm4 of the 16-bit memory



Bit imm4 of (mem16) ← 1

Sets the bit specified by the 4-bit immediate data (second operand) of the 16-bit memory location (addressed by the first operand) to 1. Only the lower 4 bits of immediate data are used in the last byte of the instruction.

Bytes: 4/5/6

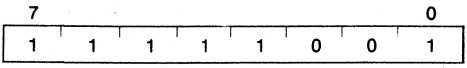
Transfers: 2

Flag operation: None

Example: SET1 Word_Var,15

SET1 CY

Set carry flag



CY ← 1

Sets the CY flag.

Bytes: 1

Transfers: None

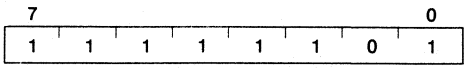
Flag operation:

V	S	Z	AC	P	CY
U	U	U	U	U	1

Example: SET1 CY

SET1 DIR

Set direction flag



Dir ← 1

Sets the DIR flag. Sets index registers IX and IY to auto-decrement when MOVBK, CMPBK, CMPM, LDM STM, INM, and OUTM are executed.

Bytes: 1

Transfers: None

Flag operation:

DIR
1

Example: SET1 DIR

SHIFT

SHL reg,1

Shift left register, single bit

7							0
1	1	0	1	0	0	0	W

1	1	1	0	0		reg	

CY ← MSB of reg, reg ← reg ×2
When MSB of reg ≠CY: V ← 1
When MSB of reg =CY: V ← 0

Performs a shift left (1 bit) of the 8- or 16-bit register specified by the first operand. Zero is loaded to the LSB of the specified register and the MSB is shifted to the CY flag. If the sign bit is the same after the shift, the V flag is cleared.

Bytes: 2

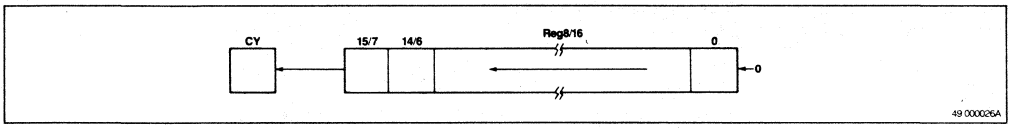
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X	X	X	U	X	X

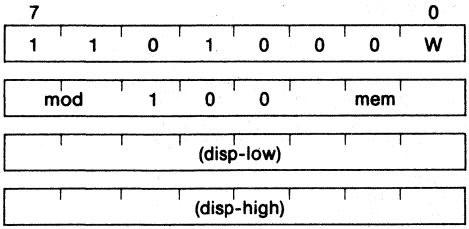
Example:

```
SHL     BH,1
SHL     AW,1
```



SHL mem,1

Shift left memory, single bit



CY ← MSB of (mem), (mem) ← (mem) ×2
When MSB of (mem) ≠ CY: V ← 1
When MSB of (mem) = CY: V ← 0

Performs a shift left (1 bit) of the 8- or 16-bit memory location addressed by the first operand. Zero is loaded to the addressed memory LSB and the MSB is shifted to the CY flag. If the sign bit (bit 7 or 15) remains the same after the shift, the V flag is cleared.

Bytes: 2/3/4

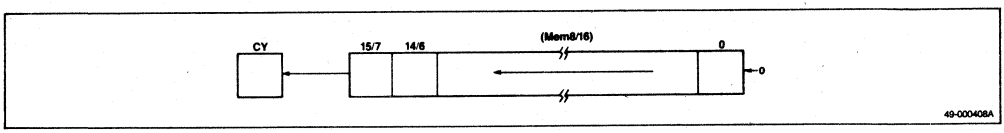
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X	X	X	U	X	X

Example:

```
SHL     BYTE PTR [IX],1
SHL     WORD_VAR,1
```



SHL reg, CL

Shift left register, variable bit

7							0
1	1	0	1	0	0	1	W

1	1	1	0	0			
					reg		

temp ← CL, while temp ≠ 0
repeat this operation, CY ← MSB of reg,
reg ← reg × 2, temp ← temp − 1

Performs a shift left of the 8- or 16-bit register specified by the first operand by the number in the CL register. Zero is loaded to the specified register's LSB. MSB is shifted to the CY flag.

Bytes: 2

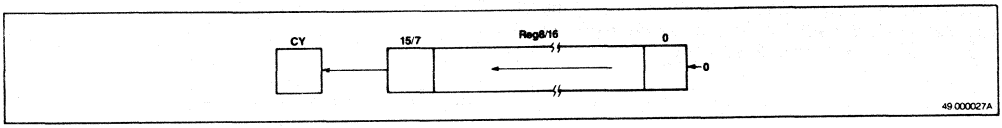
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

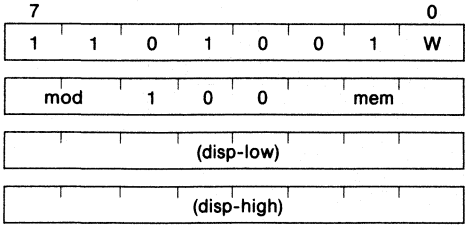
Example:

```
SHL     CL,CL
SHL     BW,CL
```



SHL mem, CL

Shift left memory, variable bit



temp ← CL, while temp ≠ 0,
repeat operation, CY ← MSB of (mem),
(mem) ← (mem) × 2, temp ← temp − 1

Performs a shift left of the 8- or 16-bit memory location addressed by the first operand by the number in the CL register. Zero is loaded to the addressed memory LSB and the MSB is shifted to the CY flag.

Bytes: 2/3/4

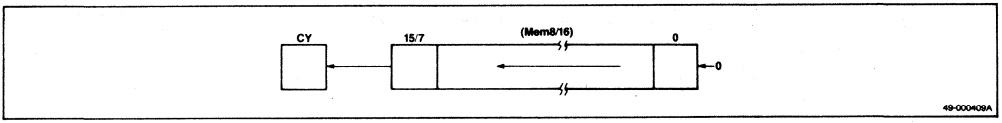
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

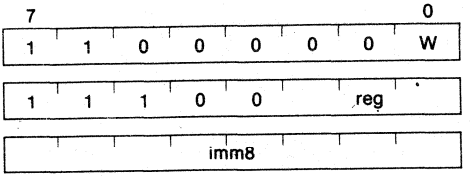
Example:

```
SHL     BYTE PTR [IY],CL
SHL     WORD PTR [IY],CL
```



SHL reg,imm8

Shift left register, multibit



Temp ← imm8, while temp ≠ 0,
repeat operation, CY ← MSB of reg,
reg ← reg × 2, temp ← temp − 1

Performs a shift left of the 8- or 16-bit register (specified by the first operand) by the 8-bit immediate data (second operand). Zero is loaded to the specified register's LSB. MSB is shifted to the CY flag.

Bytes: 3

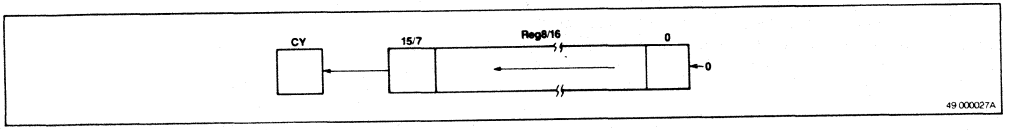
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

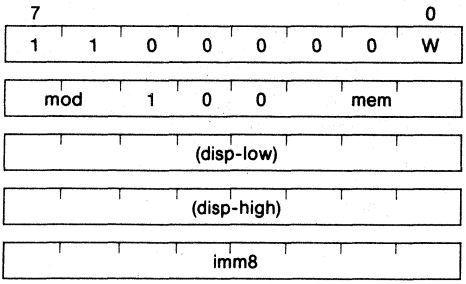
Example:

```
SHL     AH,3
SHL     DW,15
```



SHL mem,imm8

Shift left memory, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, CY ← MSB of (mem)
(mem) ← (mem) × 2, temp ← temp − 1

Performs a shift left of the 8- or 16-bit memory location addressed by the first operand by the bits specified by the 8-bit immediate data (second operand). Zero is loaded to the specified memory locations's LSB. The MSB is shifted to the CY flag.

Bytes: 3/4/5

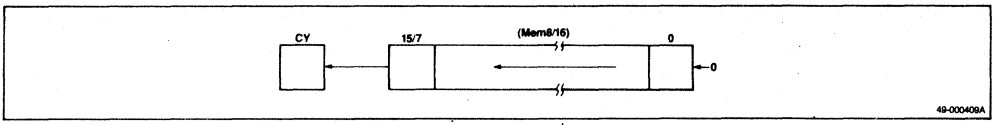
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

Example:

```
SHL     BYTE PTR [IX] [2],7
SHL     WORD_VAR,5
```



SHR reg,1

Shift right register, single bit

7							0
1	1	0	1	0	0	0	W

1	1	1	0	1		reg	

CY ← MSB of reg, reg ← reg ÷ 2
When MSB of reg ≠ bit following MSB of reg: V ← 1
When MSB of reg = bit following MSB of reg: V ← 0

Performs a logical shift right (1 bit) of the 8- or 16-bit register specified by the first operand. Zero is loaded to the MSB of the specified register and the LSB is shifted to the CY flag. If the sign bit (7 or 15) is the same after the shift, the V flag is cleared.

Bytes: 2

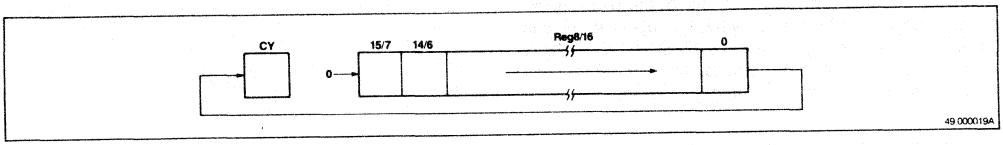
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X	X	X	U	X	X

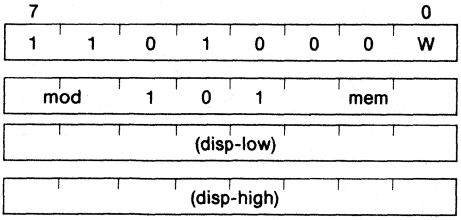
Example:

```
SHR     BH,1
SHR     AW,1
```



SHR mem,1

Shift right memory, single bit



CY ← MSB of (mem), (mem) ← (mem) ÷ 2
When MSB of (mem) ≠ bit following MSB of (mem):
V ← 1
When MSB of (mem) = bit following MSB of (mem):
V ← 0

Performs a logical shift right (1 bit) of the 8- or 16-bit memory location addressed by the first operand. Zero is loaded to the memory location's MSB and the LSB is shifted to the CY flag. If the sign bit (bit 7 or 15) remains the same after the shift, the V flag is cleared.

Bytes: 2/3/4

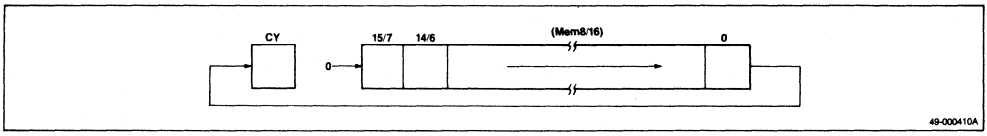
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X	X	X	U	X	X

Example:

```
SHR     BYTE_VAR [BW],1
SHR     WORD_VAR [IX],1
```



SHR reg,CL

Shift right register, variable bit

7							0
1	1	0	1	0	0	1	W

1	1	1	0	1			reg

temp ← CL, while temp ≠ 0,
repeat operation, CY ← MSB of reg,
reg ← reg ÷ 2, temp ← temp − 1

Performs a logical shift right of the 8- or 16-bit register (specified by the first operand) by the number in the CL register. Zero is loaded to the specified register's MSB. The LSB is shifted to the CY flag.

Bytes: 2

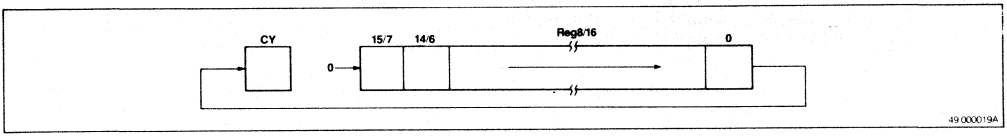
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

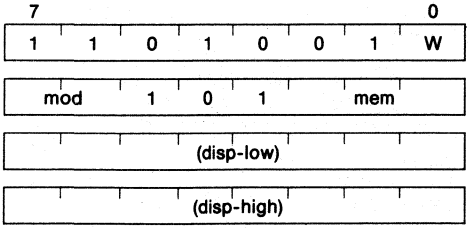
Example:

```
SHR     AL,CL
SHR     BW,CL
```



SHR mem,CL

Shift right memory, variable bit



temp ← CL, while temp ≠ 0,
repeat operation, CY ← MSB of (mem),
(mem) ← (mem) ÷ 2, temp ← temp − 1

Performs a logical shift right of the 8- or 16-bit memory location (addressed by the first operand) by the number in the CL register. Zero is loaded to the addressed memory MSB and the LSB is shifted to the CY flag.

Bytes: 2/3/4

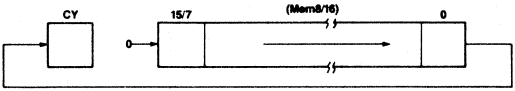
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

Example:

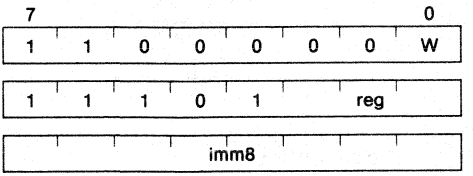
```
SHR     BYTE_VAR,CL
SHR     WORD PTR [IY],CL
```



49 000020A

SHR reg,imm8

Shift right register, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, CY ← MSB of reg,
reg ← reg ÷ 2, temp ← temp − 1

Performs a shift right of the 8- or 16-bit register (specified by the first operand) by the 8-bit immediate data (second operand). Zero is loaded to the specified register's MSB. The LSB is shifted to the CY flag.

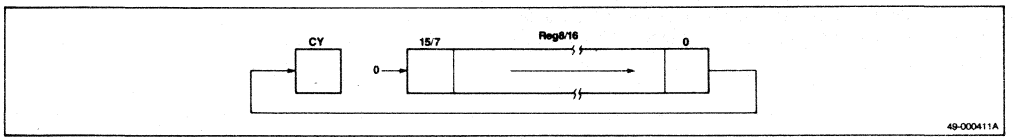
Bytes: 3

Transfers: None

Flag operation:

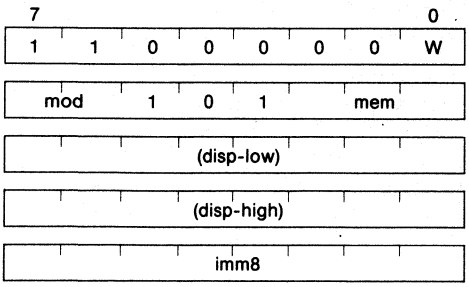
V	S	Z	AC	P	CY
U	X	X	U	X	X

Example:
SHR BL,6
SHR IX,2



SHR mem,imm8

Shift right memory, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, CY ← MSB of (mem),
(mem) ← (mem) ÷ 2, temp ← temp − 1

Performs a shift right of the 8- or 16-bit memory location (addressed by the first operand) by the bits specified by the 8-bit immediate data (second operand). Zero is loaded to the specified memory location's MSB. The LSB is shifted to the CY flag.

Bytes: 3/4/5

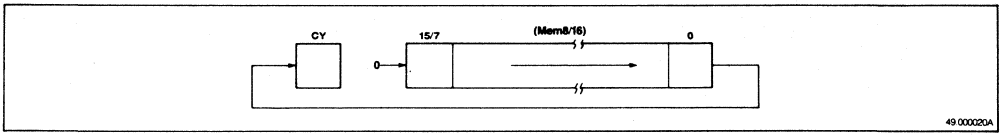
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

Example:

```
SHR     BYTE PTR [BW],2
SHR     WORD_VAR,13
```



SHRA reg,1

Shift right arithmetic

7							0
1	1	0	1	0	0	0	W

1	1	1	1	1		reg	

CY ← LSB of reg,
reg ← reg ÷ 2, V ← 0
MSB of operand does not change

Performs an arithmetic shift right (1 bit) of the 8- or 16-bit register specified by the first operand. A bit with the same value as the original bit is shifted to the specified register's MSB. The LSB is shifted to the CY flag. The sign remains unchanged after the shift.

Bytes: 2

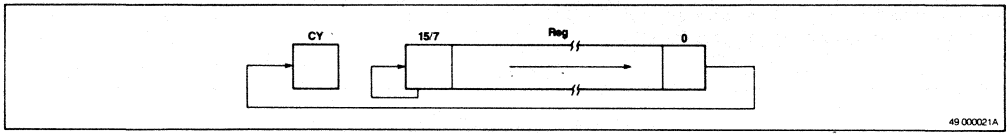
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	X

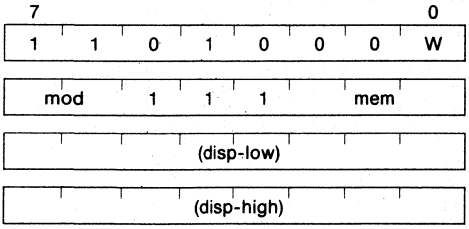
Example:

```
SHRA    CL,1
SHRA    AW,1
```



SHRA mem,1

Shift right arithmetic, memory, single bit



CY ← LSB of (mem),
(mem) ← (mem) ÷ 2, V ← 0
MSB of operand does not change

Performs an arithmetic shift right (1 bit) of the 8- or 16-bit memory location addressed by the first operand. A bit with the same value as the original bit is shifted to the memory location's MSB. The LSB is shifted to the CY flag. The sign remains unchanged after the shift.

Bytes: 2/3/4

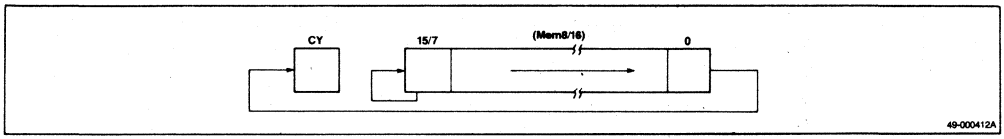
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	X

Example:

```
SHRA    BYTE_VAR,1
SHRA    WORD_VAR,1
```



SHRA reg,CL

Shift right arithmetic, register, variable bit

7							0
1	1	0	1	0	0	1	W

1	1	1	1	1		reg	

temp ← CL, while temp ≠ 0,
repeat operation, CY ← LSB of reg,
reg ← reg ÷ 2, temp ← temp − 1

Performs an arithmetic shift right of the 8- or 16-bit register (specified by the first operand) by the number of bits specified by the CL register. A bit with the same value as the original bit is shifted to the register's MSB. The LSB is shifted to the CY flag. The sign remains unchanged after the shift.

Bytes: 2

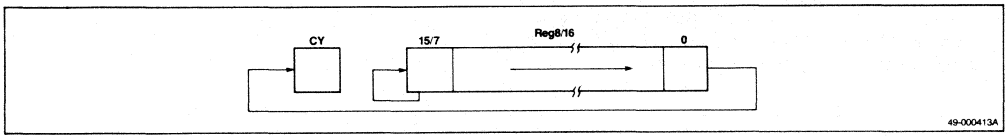
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

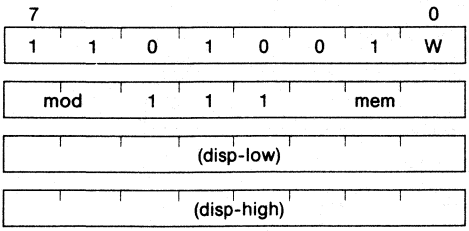
Example:

```
SHRA    BL,CL
SHRA    DW,CL
```



SHRA mem,CL

Shift right arithmetic, memory, variable bit



temp ← CL, while temp ≠ 0,
repeat operation, CY ← LSB of (mem),
(mem) ← (mem) ÷ 2, temp ← temp − 1,
MSB of operand does not change

Performs an arithmetic shift right of the 8- or 16-bit memory location (addressed by the first operand) by the number of bits specified in the CL register. A bit with the same value as the original bit is shifted to the memory location's MSB. The LSB is shifted to the CY flag. The sign remains unchanged after the shift.

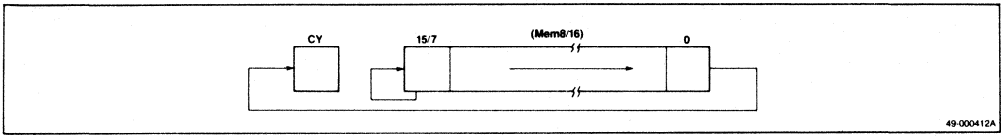
Bytes: 2/3/4
Transfers: 2

Flag Operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

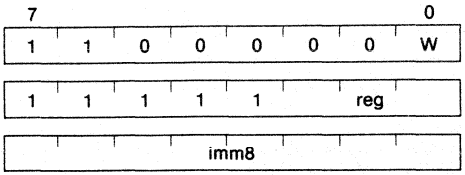
Example:

```
SHRA    BYTE_VAR,CL
SHRA    WORD_VAR,CL
```



SHRA reg,imm8

Shift right arithmetic, register, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, CY ← LSB of reg,
reg ← reg ÷ 2, temp ← temp − 1,
MSB of operand does not change

Performs an arithmetic shift right of the 8- or 16-bit register (specified by the first operand) by the 8-bit immediate data in the second operand. A bit with the same value as the original bit is shifted to the register's MSB. The LSB is shifted to the CY flag. The sign remains unchanged after the shift.

Bytes: 3

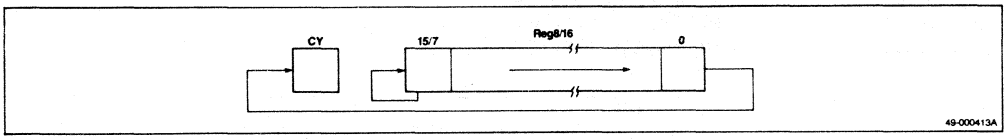
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

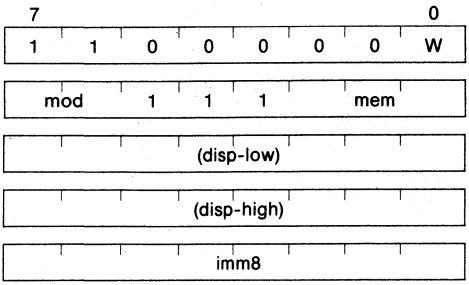
Example:

```
SHRA    CL,3
SHRA    BW,7
```



SHRA mem,imm8

Shift right arithmetic, memory, multibit



temp ← imm8, while temp ≠ 0,
repeat this operation, CY ← LSB of (mem),
(mem) ← (mem) ÷ 2, temp ← temp − 1,
MSB of operand does not change

Performs an arithmetic shift right of the 8- or 16-bit memory location (addressed by the first operand) by the number specified by the 8-bit immediate data in the second operand. A bit with the same value as the original bit is shifted to the register's MSB. The LSB is shifted to the CY flag. The sign remains unchanged after the shift.

Bytes: 3/4/5

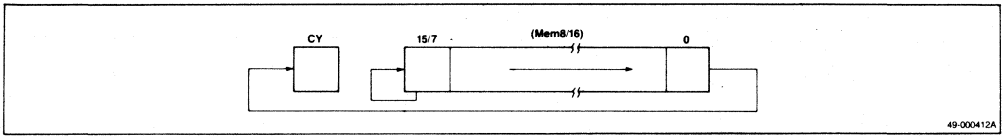
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

Example:

```
SHRA    BYTE_VAR,5
SHRA    WORD_VAR,7
```



ROTATE

ROL reg,1

Rotate left, register, single bit

7							0
1	1	0	1	0	0	0	W

1	1	0	0	0		reg	

CY ← MSB of reg, reg ← reg × 2 + CY
MSB of reg ≠ CY: V ← 1
MSB of reg = CY: V ← 0

Rotates the 8- or 16-bit register specified by the first operand left by one bit. If the MSB changes, the V flag is set. If the MSB stays the same, the V flag is cleared.

Bytes: 2

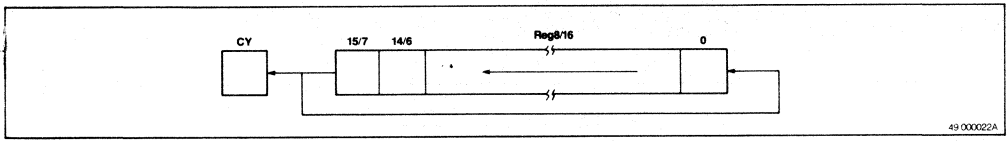
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X					X

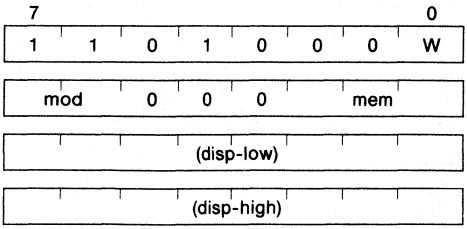
Example:

```
ROL     AH,1
ROL     DW,1
```



ROL mem,1

Rotate left, memory, single bit



CY ← MSB of (mem),
(mem) ← (mem) × 2 + CY
MSB of (mem) ≠ CY: V ← 1
MSB of (mem) = CY: V ← 0

Rotates the 8- or 16-bit memory location (addressed by the first operand) left by one bit. If the MSB changes, the V flag is set; if it stays the same, the V flag is cleared.

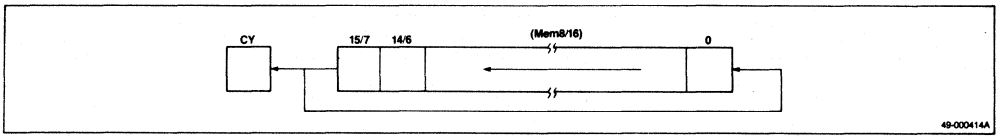
Bytes: 2/3/4
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

Example:

```
ROL     BYTE_VAR,1
ROL     WORD PTR [IX][7],1
```



ROL reg,CL

Rotate left, register, variable bit

7							0
1	1	0	1	0	0	1	W

1	1	0	0	0		reg	

temp ← CL, while temp ≠ 0,
repeat operation, CY ← MSB of reg,
reg ← reg × 2 + CY,
temp ← temp − 1

Rotates the 8- or 16-bit register specified by the first operand left by the number of bits specified by the CL register.

Bytes: 2

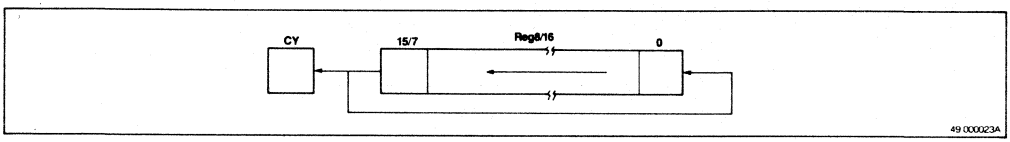
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U					X

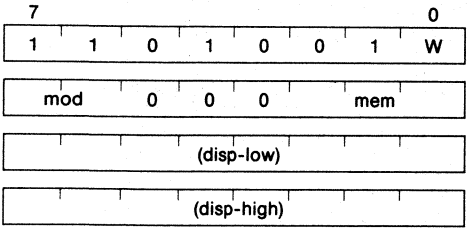
Example:

```
ROL    DL,CL
ROL    BP,CL
```



ROL mem,CL

Rotate left, memory, variable bit



temp ← CL, while temp ≠ 0,
repeat operation, CY ← MSB of (mem),
(mem) ← (mem) × 2 + CY,
temp ← temp − 1

Rotates the 8- or 16-bit memory location addressed by the first operand left by the number of bits specified in the CL register.

Bytes: 2/3/4

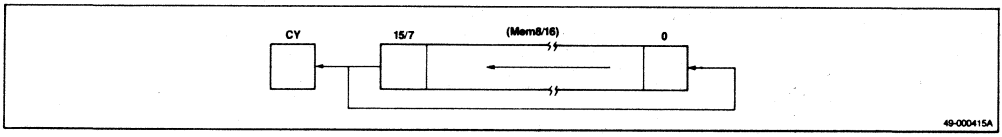
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U					X

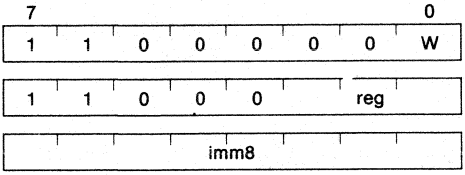
Example:

```
ROL     BYTE PTR [IX],CL
ROL     WORD_VAR,CL
```



ROL reg,imm8

Rotate left, register, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, CY ← MSB of reg,
reg ← reg × 2 + CY,
temp ← temp − 1

Rotates the 8- or 16-bit register (specified by the first operand) left by the number of bits specified by the 8-bit immediate data in the second operand. The register's MSB is shifted to the CY flag and to the LSB.

Bytes: 3

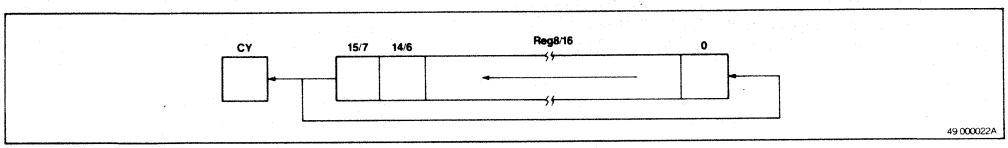
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U					X

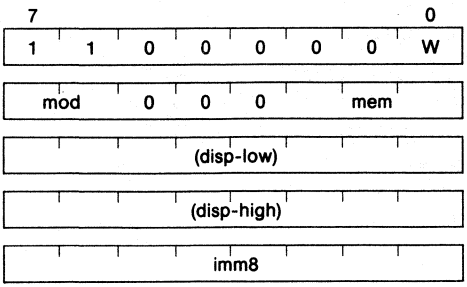
Example:

```
ROL     DH,3
ROL     IY,7
```



ROL mem,imm8

Rotate left, memory, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, CY ← MSB of (mem),
(mem) ← (mem) × 2 + CY,
temp ← temp − 1

Rotates the 8- or 16-bit memory location (addressed by the first operand) left by the number of bits specified by the 8-bit immediate data in the second operand. The memory location's MSB is shifted to the CY flag and to the LSB.

Bytes: 3/4/5

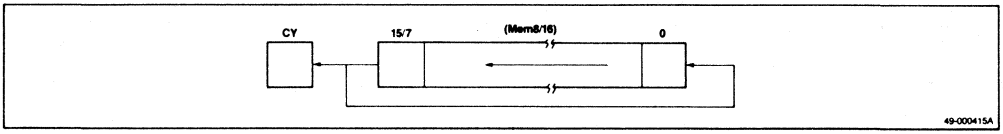
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U					X

Example:

```
ROL     BYTE_VAR,7
ROL     WORD_VAR,2
```



ROR reg,1

Rotate right, register, single bit

7							0
1	1	0	1	0	0	0	W

1	1	0	0	1		reg	

CY ← LSB of reg, reg ← reg ÷ 2,
MSB of reg ← CY
MSB of reg ≠ bit following MSB of reg: V ← 1
MSB of reg = bit following MSB of reg: V ← 0

Rotates the 8- or 16-bit register (specified by the first operand) right by 1 bit. If the MSB of the specified register changes, the overflow flag is set. If the MSB stays the same, the overflow flag is cleared.

Bytes: 2

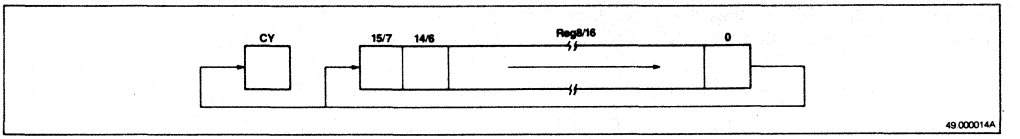
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X					X

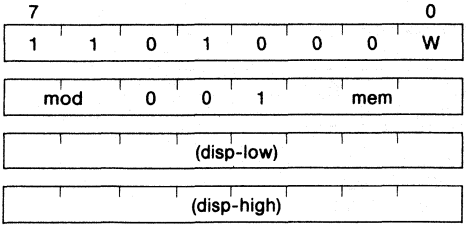
Example:

```
ROR     AL,1
ROR     CW,1
```



ROR mem,1

Rotate right, memory, single bit



CY ← LSB of (mem), (mem) ← (mem) ÷ 2
MSB of (mem) ← CY
MSB of (mem) ≠ bit following MSB of (mem): V ← 1
MSB of (mem) = bit following MSB of (mem): V ← 0

Rotates the 8- or 16-bit memory location addressed by the first operand right by 1 bit. If the MSB of the addressed memory changes, the overflow flag is set. If the MSB stays the same, the overflow flag is cleared.

Bytes: 2/3/4

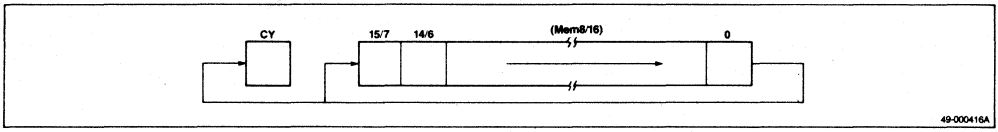
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

Example:

```
ROR     BYTE_VAR,1
ROR     WORD PTR [BW],1
```



ROR reg,CL

Rotate right, register, variable bit

7							0
1	1	0	1	0	0	1	W

1	1	0	0	1		reg	

temp ← CL, while CL ≠ 0,
repeat operation,
CY ← LSB of reg, reg ← reg ÷ 2,
MSB of reg ← CY,
temp ← temp − 1

Rotates the 8- or 16-bit register (specified by the first operand) right by the number of bits specified by the CL register.

Bytes: 2

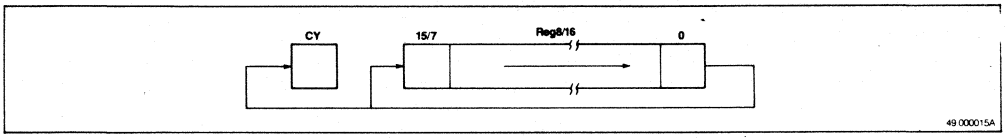
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U					X

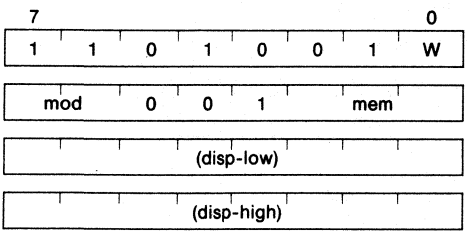
Example:

```
ROR     AH,CL
ROR     AW,CL
```



ROR mem,CL

Rotate right, memory, variable bit



temp ← CL, while temp ≠ 0,
repeat operation,
CY ← LSB of (mem), (mem) ← (mem) ÷ 2,
MSB of (mem) ← CY,
Temp ← temp − 1

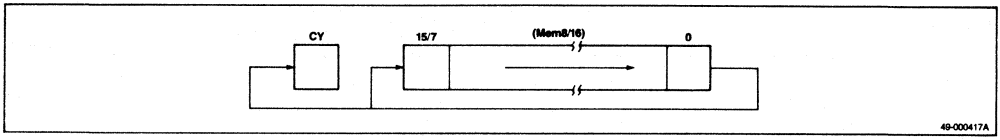
Rotates the 8- or 16-bit memory location (specified by the first operand) right by the number of bits specified by the CL register.

Bytes: 2/3/4
Transfers: 2
Flag operation:

V	S	Z	AC	P	CY
U					X

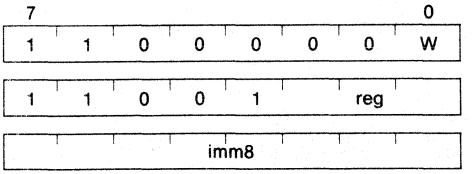
Example:

```
ROR     BYTE_VAR,CL
ROR     WORD PTR [IX]2,CL
```



ROR reg,imm8

Rotate right, register, multibit



temp ← imm8, while temp ≠ 0,
repeat operation,
CY ← LSB of reg, reg ← reg ÷ 2,
MSB of reg ← CY,
temp ← temp − 1

Rotates the 8- or 16-bit register (specified by the first operand) right by the number of bits specified by the 8-bit immediate data in the second operand. The register's LSB is shifted to the MSB and the CY flag.

Bytes: 3

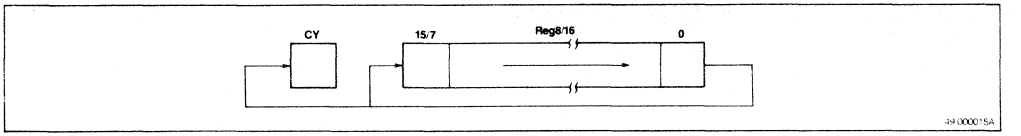
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U					X

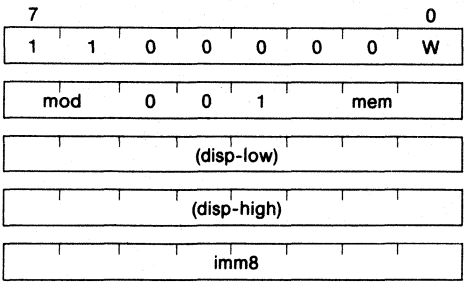
Example:

```
ROR   AL,2
ROR   IX,3
```



ROR mem,imm8

Rotate right, memory, multibit



temp ← imm8, while temp ≠ 0,
repeat operation,
CY ← LSB of (mem), (mem) ← (mem) ÷ 2,
temp ← temp − 1

Rotates the 8- or 16-bit memory location addressed by the first operand right by the number of bits specified by the 8-bit immediate data in the second operand. The memory location's LSB is shifted to the MSB as well as to the CY flag.

Bytes: 3/4/5

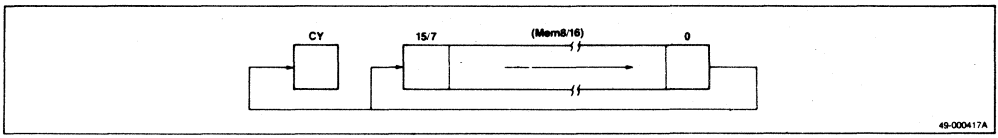
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U					X

Example:

```
ROR     BYTE_VAR,6
ROR     WORD_VAR [IX],7
```



ROLC reg,1

Rotate left with carry, register, single bit

7							0
1	1	0	1	0	0	0	W

1	1	0	1	0		reg	

tmpcy ← CY, CY ← MSB of reg,
Reg ← reg × 2 + tmpcy,
MSB of reg = CY: V ← 0
MSB of reg ≠ CY: V ← 1

Rotates the 8- or 16-bit register specified by the first operand left, including the CY flag, by one bit. If the register's MSB changes, the V flag is set. If it stays the same, the V flag is cleared.

Bytes: 2

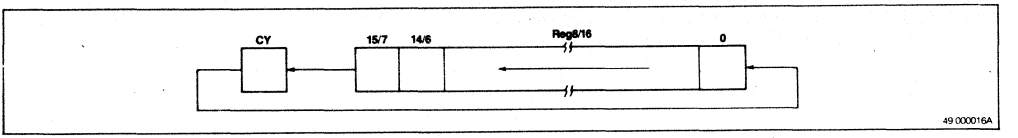
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X					X

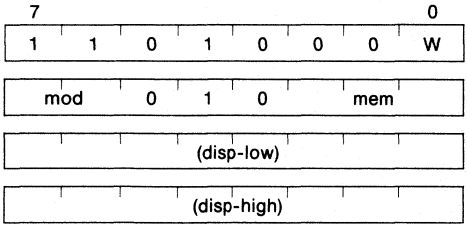
Example:

```
ROLC    BL,1
ROLC    IY,1
```



ROLC mem,1

Rotate left with carry, memory, single bit



tmpcy ← CY, CY ← MSB of (mem),
(mem) ← (mem) × 2 + tmpcy,
MSB of (mem) = CY: V ← 0
MSB of (mem) ≠ CY: V ← 1

Rotates the 8- or 16-bit memory location (addressed by the first operand) left by one bit. The rotation includes the CY flag. If the MSB of the memory location changes, the V flag is set. If it stays the same, the V flag is cleared.

Bytes: 2/3/4

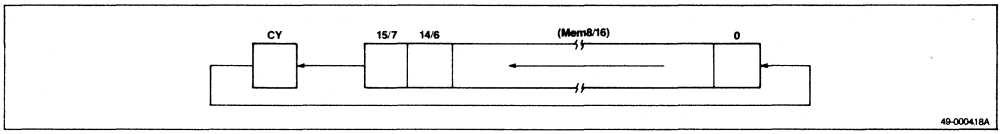
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

Example:

```
ROLC    BYTE_VAR,1
ROLC    WORD PTR [IY],1
```



ROLC reg,CL

Rotate left with carry, register, variable bit

7							0
1	1	0	1	0	0	1	W

1	1	0	1	0		reg	

temp ← CL, while temp ≠ 0,
repeat operation, tmpcy ← CY,
CY ← MSB of reg, reg ← reg × 2 + tmpcy,
temp ← temp − 1

Rotates the 8- or 16-bit register (specified by the first operand) left by the number in the CL register. Rotation includes the CY flag.

Bytes: 2

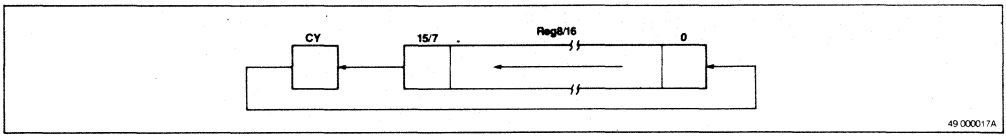
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U					X

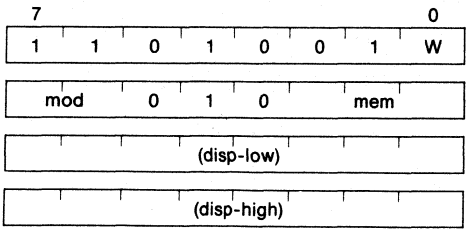
Example:

```
ROLC    AL,CL
ROLC    BW,CL
```



ROLC mem,CL

Rotate left with carry, memory, variable bit



temp ← CL, while temp ≠ 0,
repeat operation, tmpcy ← CY,
CY ← MSB of (mem),
(mem) ← (mem) × 2 + tmpcy,
temp ← temp − 1

Rotates the 8- or 16-bit memory location (addressed by the first operand) left by the number in the CL register. Rotation includes the CY flag.

Bytes: 2/3/4

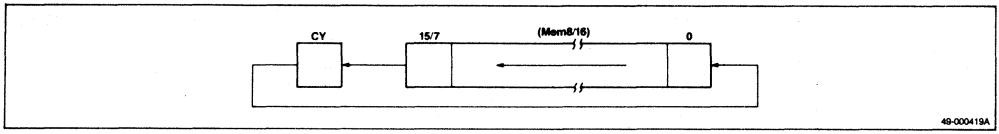
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

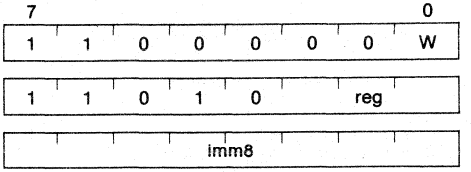
Example:

```
ROLC    BYTE PTR [IY],CL
ROLC    WORD_VAR,CL
```



ROLC reg,imm8

Rotate left with carry, register, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, tmpcy ← CY,
CY ← MSB of reg, reg ← reg × 2 + tmpcy,
temp ← temp − 1

Rotates the 8- or 16-bit register (specified by the first operand) left by the number of bits specified by the 8-bit immediate data of the second operand. Rotation includes the CY flag.

Bytes: 3

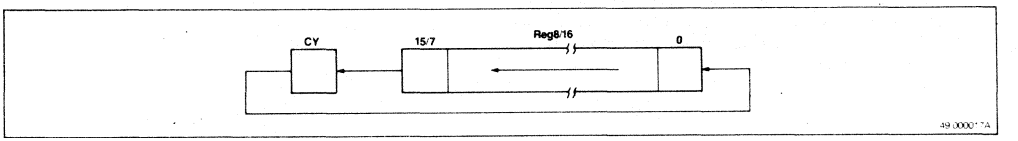
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U					X

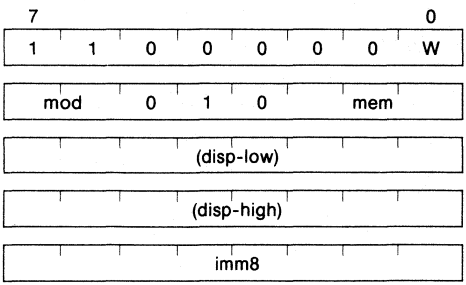
Example:

```
ROLC    BL,3
ROLC    AW,14
```



ROLC mem,imm8

Rotate left with carry, memory, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, tmpcy ← CY,
CY ← MSB of (mem),
(mem) ← (mem) × 2 + tmpcy,
temp ← temp − 1

Rotates the 8- or 16-bit memory location (addressed by the first operand) left by the number of bits specified by the 8-bit immediate data of the second operand. Rotation includes the CY flag.

Bytes: 3/4/5

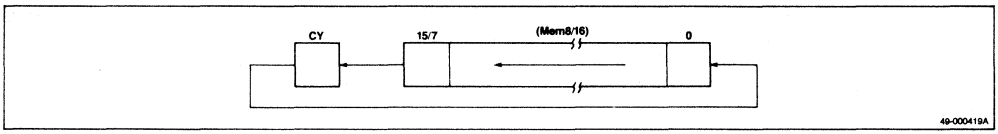
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U					X

Example:

```
ROLC    BYTE_VAR,3
ROLC    WORD_VAR,5
```



RORC reg,1

Rotate right with carry, register, single bit

7							0
1	1	0	1	0	0	0	W

1	1	0	1	1		reg	

tmpcy ← CY, CY ← LSB of reg,
reg ← reg ÷ 2, MSB of reg ← tmpcy,
MSB of reg ≠ bit following MSB of reg: V ← 1,
MSB of reg = bit following MSB of reg: V ← 0

Rotates the 8- or 16-bit register, specified by the first operand, right (including the CY flag) by one bit. If the MSB changes, the V flag is set. If it remains unchanged, the V flag is cleared.

Bytes: 2

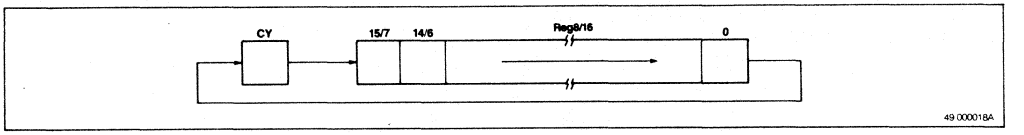
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X					X

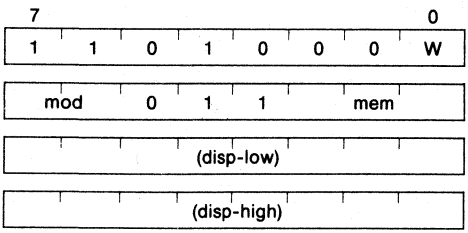
Example:

```
RORC    BH,1
RORC    BP,1
```



RORC mem,1

Rotate right with carry, memory, single bit



tmpcy ← CY, CY ← LSB of (mem),
(mem) ← (mem) ÷ 2, MSB of (mem) ← tmpcy,
MSB of (mem) ≠ bit following MSB of (mem): V ← 1
MSB of (mem) = bit following MSB of (mem): V ← 0

Rotates the 8- or 16-bit memory location (addressed by the first operand) right (including the CY flag) by one bit. If the MSB changes, the V flag is set. If it remains unchanged, the V flag is cleared.

Bytes: 2/3/4

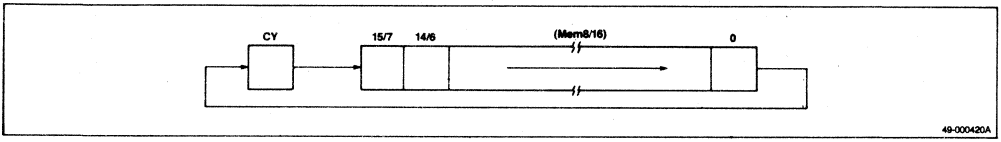
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

Example:

```
RORC    BYTE PTR [BW],1
RORC    WORD_VAR [BW] [IX],1
```



RORC reg,CL

Rotate right with carry, register, variable bit

7							0
1	1	0	1	0	0	1	W

1	1	0	1	1		reg	

temp ← CL, while temp ≠ 2,
repeat operation, tmpcy ← CY,
CY ← LSB of reg, reg ← reg ÷ 2
MSB of reg ← tmpcy, temp ← temp − 1,

Rotates the 8- or 16-bit register specified by the first operand right (including the CY flag) by the number in the CL register.

Bytes: 2

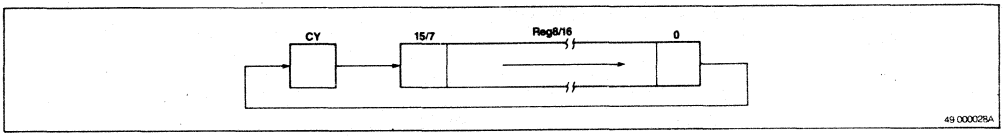
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X					X

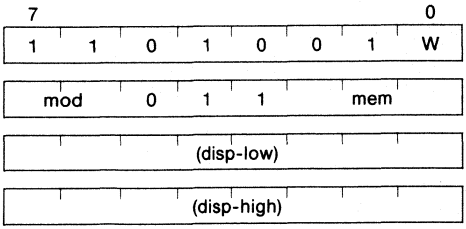
Example:

```
RORC    AL,CL
RORC    CW,CL
```



RORC mem,CL

Rotate right with carry, memory, variable bit



temp ← CL, while temp ≠ 0,
repeat operation, tmpcy ← CY,
CY ← LSB of (mem), reg ← reg ÷ 2,
MSB of (mem) ← tmpcy, temp ← temp − 1

Rotates the 8- or 16-bit memory location specified by the first operand right (including the CY flag) by the number in the CL register.

Bytes: 2/3/4

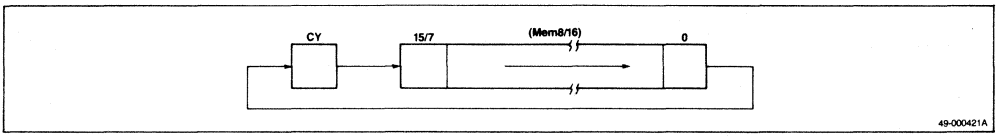
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

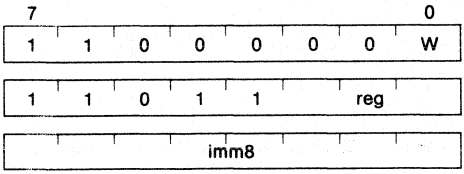
Example:

```
RORC    BYTE_VAR,CL
RORC    WORD_VAR [BP],CL
```



RORC reg,imm8

Rotate right with carry, register, multibit



temp ← imm8, while temp ≠ 0,
repeat operation, tmpcy ← CY,
CY ← LSB of reg, reg ← reg ÷ 2,
MSB of reg ← tmpcy, temp ← temp − 1

Rotates the 8- or 16-bit register specified by the first operand right (including the CY flag) by the number of bits specified by the 8-bit immediate data of the second operand.

Bytes: 3

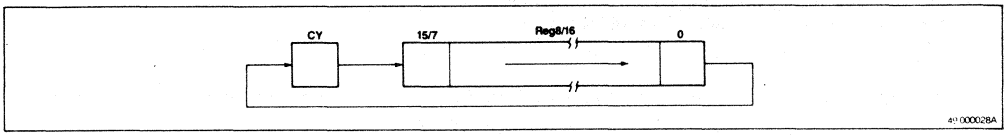
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
X					X

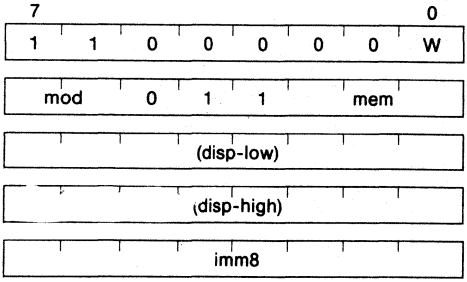
Example:

```
RORC    CH,5
RORC    BW,10
```



RORC mem,imm8

Rotate right with carry, memory multibit



temp ← imm8, while temp ≠ 0,
repeat operation, tmpcy ← CY,
CY ← LSB of (mem), (mem) ← (mem) ÷ 2,
MSB of (mem) ← tmpcy, temp ← temp − 1

Rotates the 8- or 16-bit memory location addressed by the first operand right (including the CY flag) by the number of bits specified by the 8-bit immediate data of the second operand.

Bytes: 3/4/5

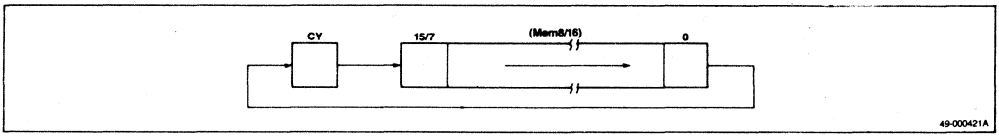
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U					X

Example:

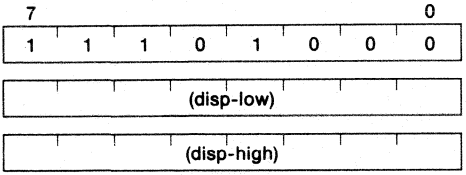
```
RORC    BYTE_VAR,3
RORC    WORD PTR [BW],10
```



SUBROUTINE CONTROL

CALL near-proc

Call, relative, same segment



(SP − 1, SP − 2) ← PC,
SP ← SP − 2,
PC ← PC + disp

Saves the PC to the stack and loads the 16-bit displacement to the PC. Enables calls to any address within the current segment.

Bytes: 3

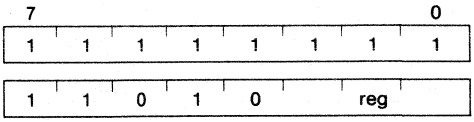
Transfers: 1

Flag operation: None

Example: CALL NEAR_PROC

CALL regptr16

Call, register, same segment



(SP − 1, SP − 2) ← PC,
SP ← SP − 2,
PC ← regptr16

Saves the PC to the stack and loads the value of the 16-bit register specified by the operand to the PC. Enables calls to any address within the current segment.

Bytes: 2

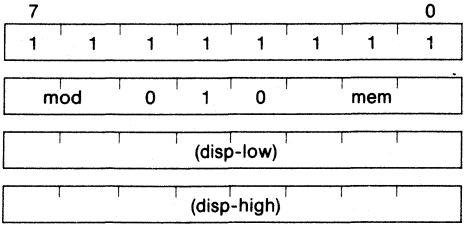
Transfers: 1

Flag operation: None

Example: CALL BX

CALL memptr16

Call, memory, same segment



(SP − 1, SP − 2) ← PC,
SP ← SP − 2, PC ← (memptr16)

Saves the PC to the stack and loads the contents of the 16-bit memory location addressed by the operand to the PC. Enables calls to any address within the current segment.

Bytes: 2/3/4

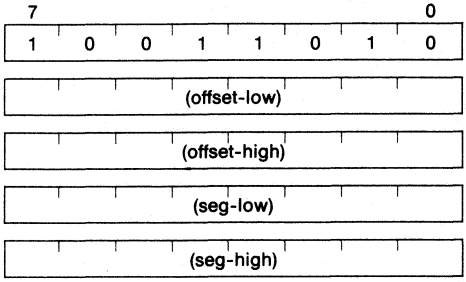
Transfers: 2

Flag operation: None

Example: CALL TABLE_ENTRY [IX]

CALL far-proc

Call, direct, external segment



(SP − 1, SP − 2) ← PS,
(SP − 3, SP − 4) ← PC,
SP ← SP − 4,
PS ← seg,
PC ← offset

Saves the PS and PC to the stack. Loads the fourth and fifth bytes of the instruction to the PS and the second and third bytes to the PC. Enables calls to any address in any segment.

Bytes: 5

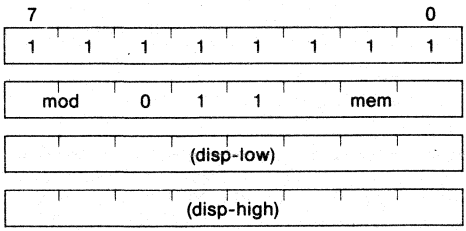
Transfers: 2

Flag operation: None

Example: CALL FAR_PROC

CALL memptr32

Call, memory, external segment



(SP − 1, SP − 2) ← PS,
(SP − 3, SP − 4) ← PC,
SP ← SP − 4,
PS ← (memptr32 + 3, memptr32 + 2),
PC ← (memptr32 + 1, memptr32)

Saves the PS and PC to the stack. Loads the higher two bytes of the 32-bit memory addressed by the operand to the PS. Loads the lower two bytes to the PC. Enables calls to any address in any segment.

Bytes: 2/3/4

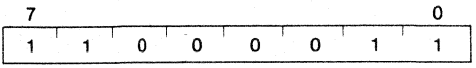
Transfers: 4

Flag operation: None

Example: CALL FAR_TABLE [IY]

RET (no operand)

Return from procedure, same segment



PC ← (SP + 1, SP),
SP ← SP + 2

Used for returning from intrasegment calls. Restores the PC from the stack. The assembler automatically distinguishes this instruction from the other RET instruction with no operand.

Bytes: 1

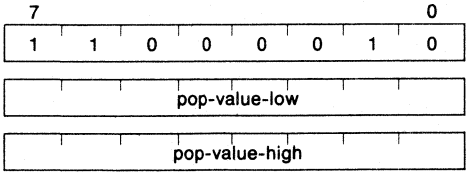
Transfers: 1

Flag operation: None

Example: RET

RET pop-value

Return from procedure, SP jump, same segment



PC ← (SP + 1, SP),
SP ← SP + 2,
SP ← SP + pop-value

Restores the PC from the stack and adds the 16-bit pop-value specified by the operand. Effective for jumping a desired number of parameters when the parameters saved in the stack become unnecessary to the program. Used for returning from intrasegment calls. The assembler automatically distinguishes this instruction from the other RET pop-value instruction.

Bytes: 3

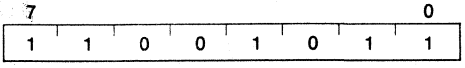
Transfers: 1

Flag operation: None

Example: RET 8

RET (no operand)

Return from procedure, external segment



PC ← (SP + 1, SP),
PS ← (SP + 3, SP + 2),
SP ← SP + 4

Restores the PC and PS from the stack. Used for returning from intersegment calls. The assembler automatically distinguishes this instruction from the RET instruction without an operand.

Bytes: 1

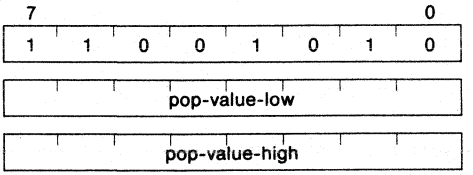
Transfers: 2

Flag operation: None

Example: RET

RET pop-value

Return from procedure, SP jump, intersegment



PC ← (SP + 1, SP),
PS ← (SP + 3, SP + 2),
SP ← SP + 4,
SP ← SP + pop-value

Restores the PC and PS from the stack and adds the 16-bit pop-value specified by the operand to the SP. This command is effective for jumping the SP value when the parameters saved in the stack subsequently become unnecessary to the program. Used for returning from intersegment calls. The assembler automatically distinguishes this instruction from the other RET pop-value instruction.

Bytes: 3

Transfers: 2

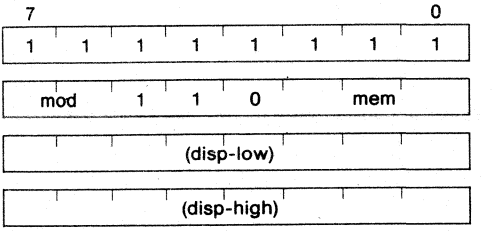
Flag operation: None

Example: RET 4

STACK OPERATION

PUSH mem16

Push, 16-bit memory



(SP − 1, SP − 2) ← (mem16),
SP ← SP − 2

Saves the contents of the 16-bit memory location addressed by the operand to the stack.

Bytes: 2/3/4

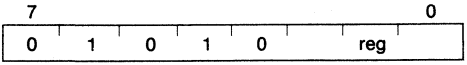
Transfers: 2

Flag operation: None

Example: PUSH DATA [IX]

PUSH reg16

Push, 16-bit register



(SP − 1, SP − 2) ← reg16,
SP ← SP − 2

Saves the 16-bit register specified by the operand to the stack.

Bytes: 1

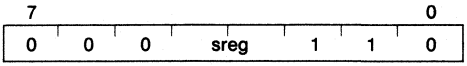
Transfers: 1

Flag operation: None

Example: PUSH IY

PUSH sreg

Push, segment register



(SP − 1, SP − 2) ← sreg,
SP ← SP − 2

Saves the segment register specified by the operand to the stack.

Bytes: 1

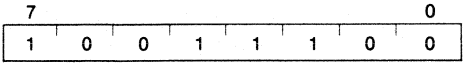
Transfers: 1

Flag operation: None

Example: PUSH PS

PUSH PSW

Push, program status word



(SP − 1, SP − 2) ← PSW,
SP ← SP − 2

Saves the PSW to the stack.

Bytes: 1

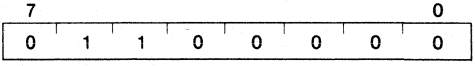
Transfers: 1

Flag operation: None

Example: PUSH PSW

PUSH R

Push, register set



temp ← SP,
(SP − 1, SP − 2) ← AW,
(SP − 3, SP − 4) ← CW,
(SP − 5, SP − 6) ← DW,
(SP − 7, SP − 8) ← BW,
(SP − 9, SP − 10) ← temp,
(SP − 11, SP − 12) ← BP,
(SP − 13, SP − 14) ← IX,
(SP − 15, SP − 16) ← IY,
SP ← SP − 16

Saves eight 16-bit registers (AW, BW, CW, DW, SP, BP, IX, and IY) to the stack.

Bytes: 1

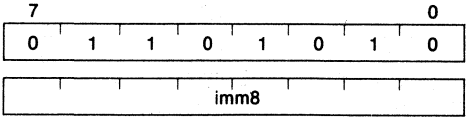
Transfers: 8

Flag operation: None

Example: PUSH R

PUSH imm8

Push, 8-bit immediate data, sign expansion



(SP − 1, SP − 2) ← Sign expansion of imm8,
SP ← SP − 2

Expands the sign of the 8-bit immediate data specified by the operand. Saves the data as 16-bit data to the stack addressed by the SP.

Bytes: 2

Transfers: 1

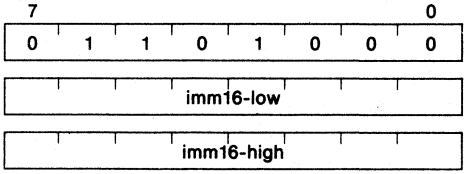
Flag operation: None

Example:

```
PUSH    5
PUSH    −1
```

PUSH imm16

Push, 16-bit immediate data



(SP − 1, SP − 2) ← imm16,
SP ← SP − 2

Saves the 16-bit immediate data described by the operand to the stack addressed by the SP.

Bytes: 3

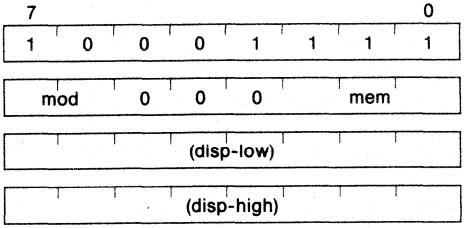
Transfers: 1

Flag operation: None

Example: PUSH 1234H

POP mem16

Pop, 16-bit memory



(mem16) ← (SP + 1, SP),
SP ← SP + 2

Transfers the contents of the stack to the 16-bit memory location addressed by the operand.

Bytes: 2/3/4

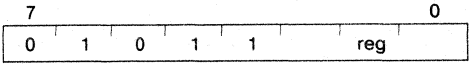
Transfers: 2

Flag operation: None

Example: POP DATA

POP reg16

Pop, 16-bit register



reg16 ← (SP + 1, SP), SP ← SP + 2

Transfers the contents of the stack to the 16-bit register specified by the operand.

Bytes: 1

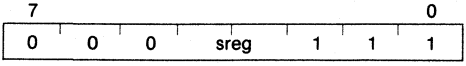
Transfers: 1

Flag operation: None

Example: POP BP

POP sreg

Pop, segment register



sreg ← (SP + 1, SP), SP ← SP + 2

Transfers the contents of the stack to the segment register (except PS) specified by the operand. External interrupts NMI and INT, and single-step breaks will not be acknowledged between this instruction and the next.

Bytes: 1

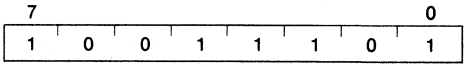
Transfers: 1

Flag operation: None

Example: POP DS1

POP PSW

Pop, program status word



PSW ← (SP + 1, SP), SP ← SP + 2

Transfers the contents of the stack to the PSW.

Bytes: 1

Transfers: 1

Flag operation:

MD*	V	DIR	IE	BRK	S	Z
R	R	R	R	R	R	R

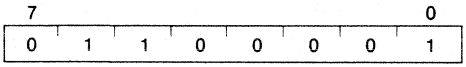
			AC	P	CY
			R	R	R

*The Mode flag (MD) can only be modified by POP PSW during Native mode calls from 8080 Emulation mode; i.e. between the execution of BRKEM and RETEM instructions. In Native mode outside of Emulation mode, the MD flag will remain set to 1 regardless of the contents of the stack. Do not alter the MD flag during Native mode calls from Emulation mode, or during Native mode interrupt service routines which may be executed by interrupting Emulation mode execution.

Example: POP PSW

POP R

Pop, register set



IY ← (SP + 1, SP),
IX ← (SP + 3, SP + 2),
BP ← (SP + 5, SP + 4),
BW ← (SP + 9, SP + 8),
DW ← (SP + 11, SP + 10),
CW ← (SP + 13, SP + 12),
AW ← (SP + 15, SP + 14),
SP ← SP + 16

Restores the contents of the stack to the following 16-bit registers: AW, BW, CW, DW, BP SP, IX, and IY.

Bytes: 1

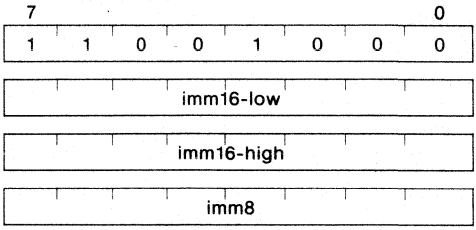
Transfers: 7

Flag operation: None

Example: POP R

PREPARE imm16,imm8

Prepare new stack frame



(SP − 1, SP − 2) ← BP,
SP ← SP − 2,
temp ← SP,
When imm8 > 0, repeat these operations "imm8 − 1" times:
  (SP − 1, SP − 2 ← (BP − 1, BP − 2)
  SP ← SP − 2 (*1, see notes)
  BP ← BP − 2
and perform these operations:
  (SP − 1, SP − 2) ← temp
  SP ← SP − 2 (*2, see notes)

Then perform these operations:
  BP ← temp
  SP ← SP − imm16

Notes: When imm8=1, *1 is not performed,
When imm8=0, *1 and *2 are not performed.

Used to generate "stack frames" required by the block structures of high-level languages such as Pascal and Ada. The stack frame includes a local variable area as well as pointers. These frame pointers point to other frames containing variables that can be referenced from the current procedure.

The first operand (16-bit immediate data) specifies (in bytes) the size of the local variable area. The second operand (8-bit immediate data) specifies the depth (or lexical level) of the procedure block. The frame base address generated by this instruction is set in the BP base pointer.

First the old BP value is saved to the stack so that BP of the calling procedure can be restored when the called procedure terminates. The frame pointer (BP value saved to the stack) that indicates the range of variables that can be referenced by the called procedure is placed on the stack. This range is always a value one less than the lexical level of the procedure. If the lexical level of a procedure is greater than one, the pointers of that procedure will also be saved on the stack. This enables the frame pointer of the calling procedure to be copied when frame pointer copy is performed within the called procedure.

Next, the new frame pointer value is set in the BP and the area for local variables used by the procedure is reserved in the stack. In other words, SP is decremented only for the amount of stack memory required by the local variables.

Bytes: 4

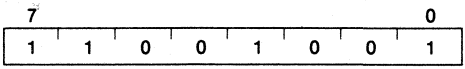
Transfers:
When imm8 = 0: none
When imm8 > 1: 1 + 2(imm8-1)

Flag operation: None

Example: PREPARE 10, 3

DISPOSE (no operand)

Dispose a stack frame



SP ← BP,
BP ← (SP + 1, SP),
SP ← SP + 2

Releases the last stack frame generated by the PREPARE instruction. A value that points to the preceding frame is loaded in the BP and the bottom of the frame value is loaded in SP.

Bytes: 1

Transfers: 1

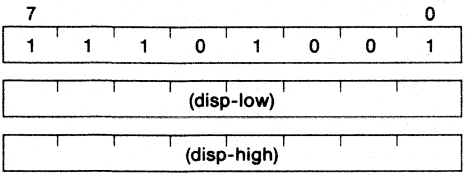
Flag operation: None

Example: DISPOSE

BRANCH

BR-near-label

Branch Relative, Same Segment BR near-label



PC ← PC + disp

Loads the current PC value plus a 16-bit displacement value to the PC. If the branch address is in the current segment, the assembler automatically generates this instruction.

Bytes: 3

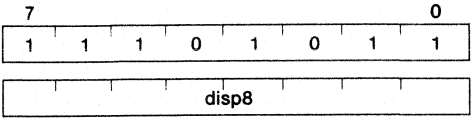
Transfers: None

Flag operation: None

Example: BR LABEL1

BR short-label

Branch short relative, same segment



PC ← PC + ext-disp8

Loads the current PC value plus an 8-bit (actually, sign-extended 16-bit) displacement value to the PC. When the branch address is in the current segment and within ±127 bytes of the instruction, the assembler automatically generates this instruction.

Bytes: 2

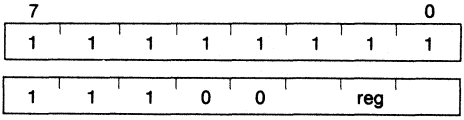
Transfers: None

Flag operation: None

Example: BR SHORT_LABEL

BR regptr16

Branch register, same segment



PC ← regptr16

Loads the contents of the 16-bit register specified by the operand to the PC. This instruction can branch to any address in the current segment.

Bytes: 2

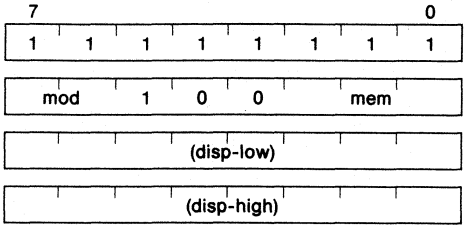
Transfers: None

Flag operation: None

Example: BR BX

BR memptr16

Branch memory, same segment



PC ← (memptr16)

Loads the contents of the 16-bit memory location addressed by the operand to the PC. This instruction can branch to any address in the current segment.

Bytes: 2/3/4

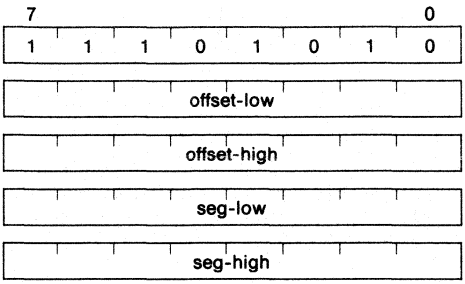
Transfers: 1

Flag operation: None

Example: BR TABLE [IX]

BR far-label

Branch direct, external segment



PC ← offset,
PS ← seg

Loads the 16-bit offset data (second and third bytes of the instruction) to the PC and the 16-bit segment data (fourth and fifth bytes) to the PS. This instruction can branch to any address in any segment.

Bytes: 5

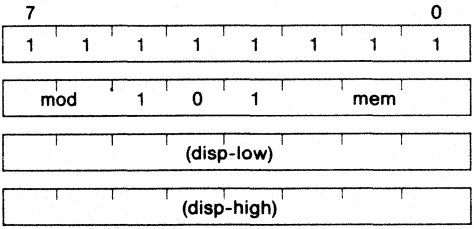
Transfers: None

Flag operation: None

Example: BR FAR_LABEL

BR memptr32

Branch memory, external segment



PS ← (memptr32 + 3, memptr32 + 2)
PC ← (memptr32 + 1, memptr32)

Loads the upper two bytes and lower two bytes of the 32-bit memory addressed by the operand to the PS and PC, respectively. This instruction can branch to any address in any segment.

Bytes: 2/3/4

Transfers: 2

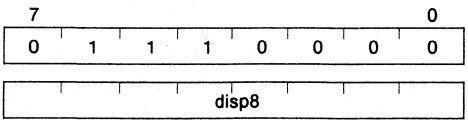
Flag operation: None

Example: BR FAR_SEGMENT [IY]

CONDITIONAL BRANCH

BV short-label

Branch if overflow



When V = 1, PC ← PC + ext-disp8

When the V flag is 1, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

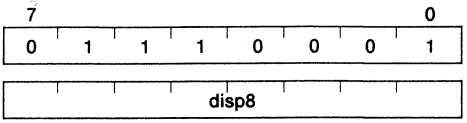
Transfers: None

Flag operation: None

Example: BV OVERFLOW_ERROR

BNV short-label

Branch if not overflow



When V = 0, PC ← PC + ext-disp8

When the V flag is 0, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

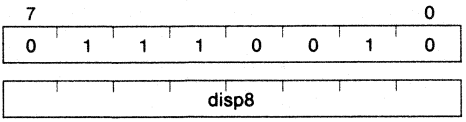
Transfers: None

Flag operation: None

Example: BNV NO_ERROR

BC short-label
BL short-label

Branch if carry/lower



When CY = 1, PC ← PC + ext-disp8

When the CY flag is 1, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

Transfers: None

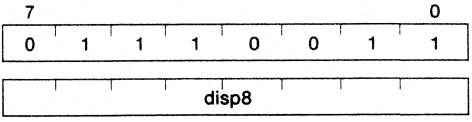
Flag operation: None

Example:

```
BC      CARRY_SET
BL      LESS_THAN
```

BNC short-label
BNL short-label

Branch if not carry/not lower



When CY = 0, PC ← PC + ext-disp8

When the CY flag is 0, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

Transfers: None

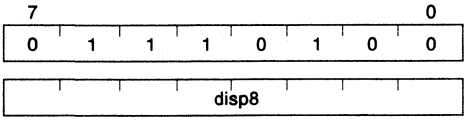
Flag operation: None

Example:

```
BNC     CARRY_CLEAR
BNL     GREATER_OR_EQUAL
```

BE short-label
BZ short-label

Branch if equal/zero



When Z = 1, PC ← PC + ext-disp8

When the Z flag is 1, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

Transfers: None

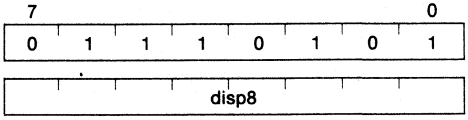
Flag operation: None

Example:

```
BE    EQUALITY
BZ    ZERO
```

BNE short-label
BNZ short-label

Branch if not equal/not zero



When Z = 0, PC ← PC + ext-disp8

When the Z flag is 0, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

Transfers: None

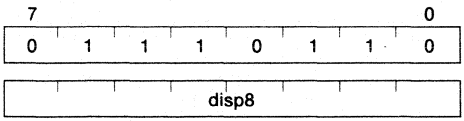
Flag operation: None

Example:

```
BNE    NOT_EQUAL
BNZ    NOT_ZERO
```

BNH short-label

Branch if not higher



When CY OR Z = 1, PC ← PC + ext-disp8

When the logical sum of the CY and Z flags is 1, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

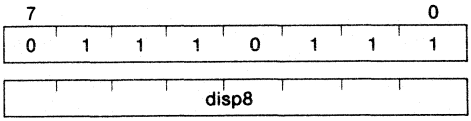
Transfers: None

Flag operation: None

Example: BNH NOT_HIGHER

BH short-label

Branch if higher



When CY OR Z = 0, PC ← PC + ext-disp8

When the logical sum of the CY and Z flags is 0, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

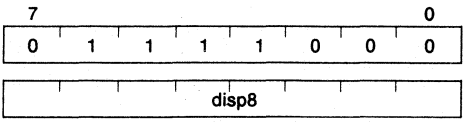
Transfers: None

Flag operation: None

Example: BH HIGHER

BN short-label

Branch if negative



When S = 1, PC ← PC + ext-disp8

When the S flag is 1, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

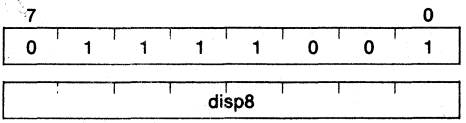
Transfers: None

Flag operation: None

Example: BN NEGATIVE

BP short-label

Branch if positive



When S = 0, PC ← PC + ext-disp8

When the S flag is 0, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

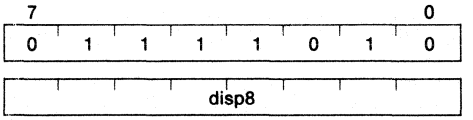
Transfers: None

Flag operation: None

Example: BP POSITIVE

BPE short-label

Branch if parity even



When P = 1, PC ← PC + ext-disp8

When the P flag is 1, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) dispacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

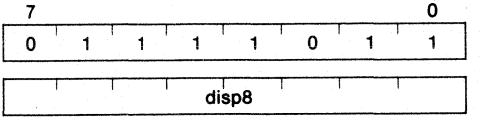
Transfers: None

Flag operation: None

Example: BPE PARITY_EVEN

BPO short-label

Branch if parity odd



When P = 0, PC ← PC + ext-disp8

When the P flag is 0, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

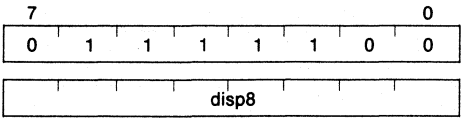
Transfers: None

Flag operation: None

Example: BPO PARITY_ODD

BLT short-label

Branch if less than



When S XOR V = 1, PC ← PC + ext-disp8

When the exclusive OR of the S and V flags is 1, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment. When the conditions are unsatisfied, proceeds to the next instruction.

Bytes: 2

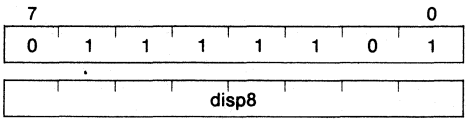
Transfers: None

Flag operation: None

Example: BLT LESS_THAN

BGE short-label

Branch if greater than or equal



When S XOR V = 0, PC ← PC + ext-disp8

When the Exclusive OR of the S and V flags is 0, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment. When the conditions are unsatisfied, proceeds to the next instruction.

Bytes: 2

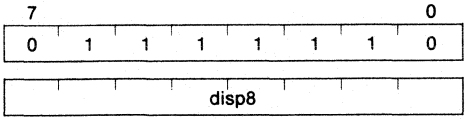
Transfers: None

Flag operation: None

Example: BGE GREATER_OR_EQUAL

BLE short-label

Branch if less than or equal



When (S XOR V) OR Z = 1, PC ← PC + ext-disp8

When the Exclusive OR of the S and V flags and the logical sum of that result and the Z flag is 1, loads the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment. When the conditions are unsatisfied, proceeds to the next instruction.

Bytes: 2

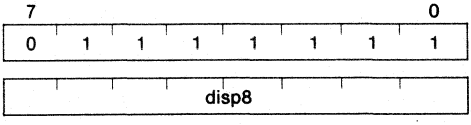
Transfers: None

Flag operation: None:

Example: BLE LESS_OR_EQUAL

BGT short-label

Branch if greater than



When (S XOR V) OR Z = 0, PC ← PC + ext-disp8

When the exclusive OR of the S and V flags and the logical sum of that result and the Z flag is 0, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment. When the conditions are unsatisfied, proceeds to the next instruction.

Bytes: 2

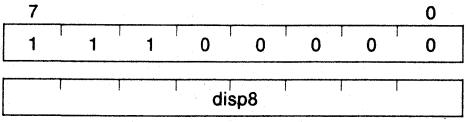
Transfers: None

Flag operation: None

Example: BGT GREATER

DBNZNE short-label

Decrement and branch if not zero and not equal



CW ← CW − 1
When CW ≠ 0 and Z = 0, PC ← PC + ext-disp8

When the 16-bit register CW is decremented (−1), the resultant CW value is not 0, and the Z flag is cleared, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

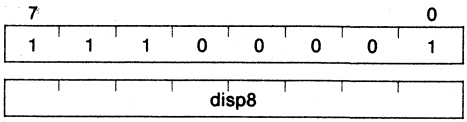
Transfers: None

Flag operation: None:

Example: PBNZNE LOOP_AGAIN

DBNZE short-label

Decrement and branch if not zero and equal



CW ← CW − 1
When CW ≠ 0 and Z = 1, PC ← PC + ext-disp8

When the 16-bit register CW is decremented (−1), the CW is not zero, and the Z flag is set, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

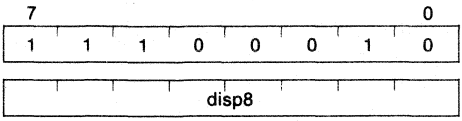
Transfers: None

Flag operation: None

Example: DBNZE LOOP_AGAIN

DBNZ short-label

Decrement and branch if not zero



CW ← CW − 1
When CW ≠ 0, PC ← PC + ext-disp8

When the 16-bit register CW is decremented (−1) and the CW value is not zero, load the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

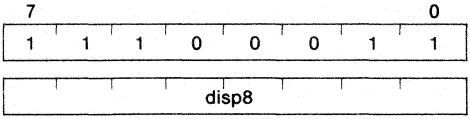
Transfers: None

Flag operation: None

Example: DBNZ LOOP_AGAIN

BCWZ short-label

Branch if CW equals zero



If CW = 0, PC ← PC + ext-disp8

When the 16-bit register CW is 0, load the current PC value plus the 8-bit (actually sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment.

Bytes: 2

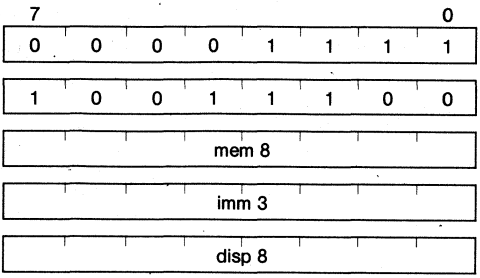
Transfers: None

Flag operation: None

Example: BCWZ CW_ZERO

BTCLR mem 8, imm 3, short-label

Bit test and if true then clear and branch else no operation



When the condition of the bit of the special function register is 1, execution of BTCLR can be used to reset that bit (0) and branch to the short label described in the operand.

Bytes: 5

Transfers: None

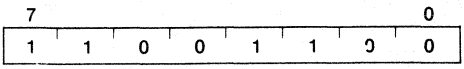
Flag operation: None

Example: BTCLR 9CH, 7, TIMER_INT

BREAK

BRK 3

Break, vector 3



(SP − 1, SP − 2) ← PSW
(SP − 3, SP − 4) ← PS
(SP − 5, SP − 6) ← PC
SP ← SP − 6
IE ← 0
BRK ← 0
PC ← (13, 12)
PS ← (15, 14)

Saves the PSW, PS, and PC to the stack and resets the IE and BRK flags to 0. Then loads the lower two bytes and higher two bytes of vector 3 of the interrupt vector table to the PC and PS, respectively.

Bytes: 1

Transfers: 5

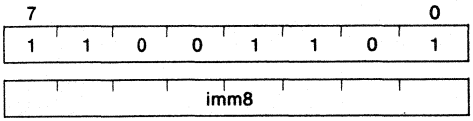
Flag operation:

IE	BRK				
0	0				

Example: BRK 3

BRK imm8 (≠3)

Break, immediate data



(SP − 1, SP − 2) ← PSW
(SP − 3, SP − 4) ← PS
(SP − 5, SP − 6) ← PC
SP ← SP − 6
IE ← 0
BRK ← 0
PC ← (imm8 ×4 + 1, imm8 ×4)
PS ← (imm8 ×4 + 3, imm8 ×4 + 2)

Saves the PSW, PS, and PC to the stack and resets the IE and BRK flags to 0. Then loads the lower two bytes and upper two bytes of the interrupt vector table (4 bytes) specified by the 8-bit immediate data to the PC and PS, respectively.

Bytes: 1

Transfers: 5

Flag operation:

IE	BRK				
0	0				

Example: BRK 10H ;PC = (40H,41H),
;PS = (42H,43H)

BRKV (no operand)

Break if overflow

7							0
1	1	0	0	1	1	1	0

When V = 1,
(SP − 1, SP − 2) ← PSW
(SP − 3, SP − 4) ← PS
(SP − 5, SP − 6) ← PC
SP ← SP − 6
IE ← 0
BRK ← 0
PC ← (011H, 010H)
PS ← (013H, 012H)

When the V flag is set, saves the PSW, PS, and PC to the stack and resets the IE and BRK flags to 0. Then loads the lower two bytes and upper two bytes of vector 4 of the interrupt vector table to the PC and PS, respectively. When the V flag is reset, proceeds to the next instruction.

Bytes: 1

Clocks:
When V = 1: 52, μPD70108
52, μPD70116 odd addresses
When V = 0: 40, μPD70116 even addresses

Transfers: 5

Flag operation:

IE	BRK				
0	0				

Example: BRKV

RETI (no operand)

Return from interrupt

7							0
1	1	0	0	1	1	1	1

PC ← (SP + 1, SP)
PS ← (SP + 3, SP + 2)
PSW ← (SP + 5, SP + 4)
SP ← SP + 6

Restores the contents of the stack to the PC, PS, and PSW. Used for return from interrupt processing.

Bytes: 1

Clocks:
39, μPD70108
39, μPD70116 odd addresses
27, μPD70116 even addresses

Transfers: 3

Flag operation:

MD*	V	DIR	IE	BRK	S	Z
R	R	R	R	R	R	R

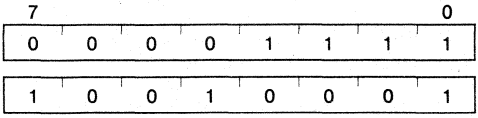
AC	P	CY				
R	R	R*				

*The Mode flag (MD) can only be modified by RETI during Native mode calls from 8080 Emulation mode; i.e. between the execution of BRKEM and RETEM instructions. In Native mode outside of Emulation mode, the MD flag will remain set to 1 regardless of the contents of the stack. Do not alter the MD flag during Native mode calls from Emulation mode, or during interrupt service routines which may be executed by interrupting Emulation mode execution. The RETI instruction should be used to exit Native mode service routines and to return to Emulation mode. The RETI instruction should be the last instruction executed in the Native mode service routine.

Example: RETI

RETRBI (no operand)

Return from Register Bank interrupt



7							0
0	0	0	0	1	1	1	1
1	0	0	1	0	0	0	1

PC ← Save PC
PSW ← Save PSW

Return instruction for register bank interrupt. This is used when returning from the interrupt processing routine which has used register bank switching function. It can not be used for return from vector interrupt.

Bytes: 2

Transfers: 2

Flag operation:

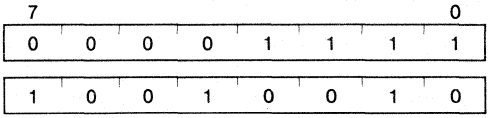
V	DIR	IE	BRK	S	Z
R	R	R	R	R	R

AC	P	CY				
R	R	R				

Example: RETRBI

FINT (no operand)

Finish of interrupt



7							0
0	0	0	0	1	1	1	1
1	0	0	1	0	0	1	0

Indicates to the CPU that interrupt processing for interrupt controller is completed. For all interrupts exclusive of NMI, INTR and software interrupt, it is necessary to execute before the return instruction from interrupt. It cannot be used for NMI, INTR and software interrupt.

Bytes: 2

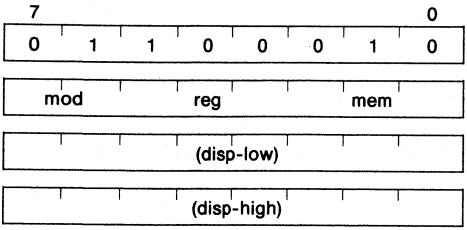
Transfers: None

Flag operation: None

Example: FINT

CHKIND reg16,mem32

Check index



When (mem32) > reg16 or (mem32 + 2) < reg16
(SP − 1, SP − 2) ← PSW
(SP − 3, SP − 4) ← PS
(SP − 5, SP − 6) ← PC
SP ← SP − 6
IE ← 0
BRK ← 0
PS ← (23, 22)
PC ← (21, 20)

Used to check whether the index value in reg16 is within the defined array bounds. Initiates a BRK 5 when the index does not satisfy the condition. The definition region should be set beforehand in the two words (first word for the lower limit and second word for the upper limit) of memory.

Transfers:
When interrupt condition is fulfilled: 7
When interrupt condition is not fulfilled: 2

Flag operation:

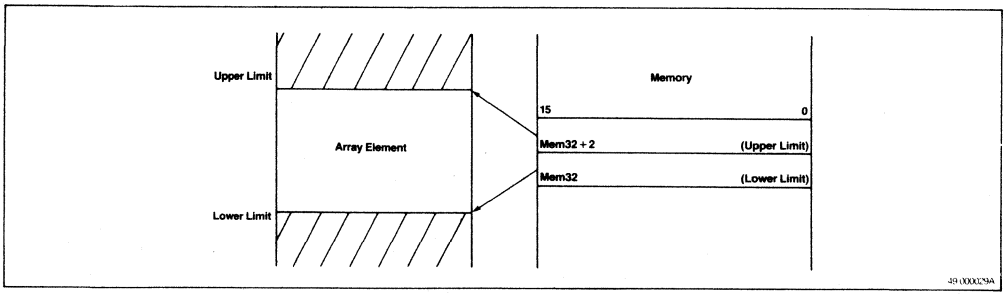
When interrupt condition is fulfilled:

IE	BRK				
0	0				

Example:
When interrupt condition is not fulfilled: None:

Example:

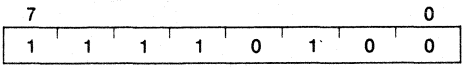
```
                MOV     IX,23
                CHKIND  IX,BOUNDS1      ;OK
                MOV     BW,87
                CHKIND  BW,BOUNDS2      ;causes
                                        ;BRK 5
BOUNDS1 DW      5,37
BOUNDS2 DW      2,80
```



CPU CONTROL

HALT (no operand)

Halt



Sets the halt state. The halt state is released by the RESET, NMI, or INT input.

Bytes: 1

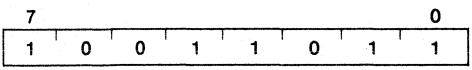
Transfers: None

Flag operation: None

Example: HALT

POLL (no operand)

Poll and wait



Keeps the CPU in the idle state until the POLL pin becomes an active low level.

Bytes: 1

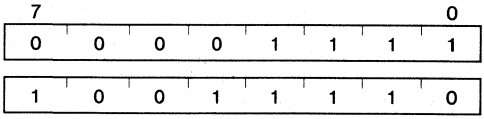
Transfers: None

Flag operation: None

Example: POLL

STOP (no operand)

Stop



Initiates Stop mode. The stop mode is released by RESET or NMI

Bytes: 2

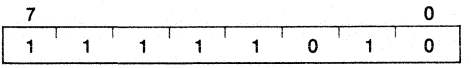
Transfers: None

Flag operation: None

Example: STOP

DI (no operand)

Disable interrupt



IE ← 0

Resets the IE flag and disables the external maskable interrupt input (INT). Does not disable the external non-maskable interrupt input (NMI) or software interrupt instructions.

Bytes: 1

Transfers: None

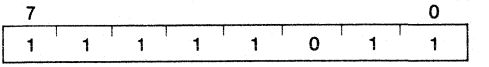
Flag operation:

IE					
0					

Example: DI

EI (no operand)

Enable interrupt



EI ← 1

Sets the EI flag and enables the external maskable interrupt input (INT). The system does not enter the interrupt-enable state until executing the instruction immediately after EI.

Bytes: 1

Transfers: None

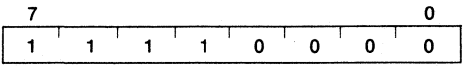
Flag operation:

IE
1

Example: EI

BUSLOCK (no operand)

Bus lock prefix



In the large-scale mode (S/LG = 0)

Outputs the buslock signal ($\overline{\text{BUSLOCK}}$) while the instruction immediately after the BUSLOCK instruction is being executed. When BUSLOCK is used for a block operation instruction with a repeat prefix, the BUSLOCK signal is kept at an active low level until the end of the block operation instruction.

Hold request is inhibited when $\overline{\text{BUSLOCK}}$ is active. The BUSLOCK instruction is effective when you do not want to acknowledge a hold request during block operations.

In small-scale mode (S/LG = 1)

The $\overline{\text{BUSLOCK}}$ signal is not an output. However, the BUSLOCK instruction can be used to delay a hold acknowledge response to a hold request until execution of the locked instruction is completed.

Bytes: 1

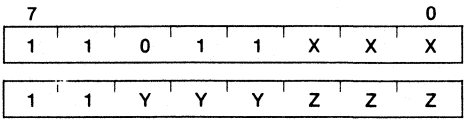
Transfers: None

Flag operation : None

Example: BUSLOCK REP MOVBKB

FPO1 fp-op

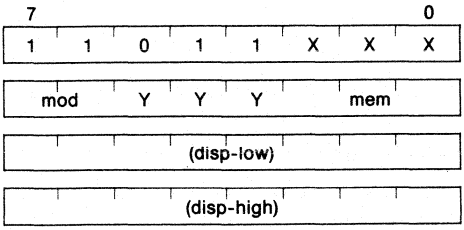
Floating point operation 1, register



instruction not executed, interrupt takes place

FPO1 fp-op,mem

Floating point operation 1, memory

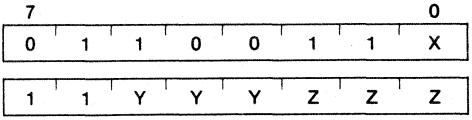


Data bus ← (mem)

instruction not executed, interrupt takes place

FPO2 fp-op

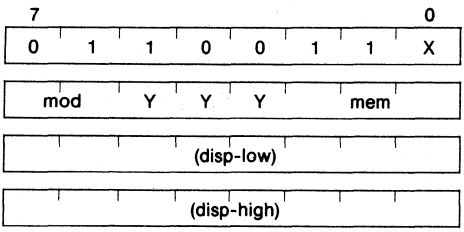
Floating point operation 2, register



instruction not executed, interrupt takes place

FPO2 fp-op, mem

Floating point operation 2, memory

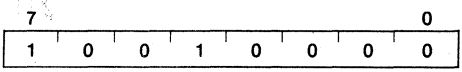


Data bus ← (mem)

instruction not executed, interrupt takes place

NOP (no operand)

No operation



Causes the processor to do nothing for three clocks.

Bytes: 1

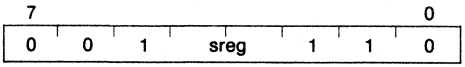
Transfers: None

Flag operation: None

Example: NOP

SEGMENT OVERRIDE PREFIXES

DS0:
DS1:
PS:
SS:



When appended to the operand, specifies the segment register to be used for access of a memory operand expecting segment override.

You can define the segment override by assembler directive "ASSUME" without describing the segment override prefix directly (see Assembler Operating Manual).

Bytes: 1

Transfers: None

Flag operation: None

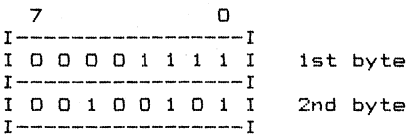
Example:

```
MOV     IX,DS1:[IY]
REP     MOVBKB DEST_BLK,SS:SRC_BLK
```

REGISTER BANK INSTRUCTIONS

(See also appendix C)

MOVSPA (no operand)

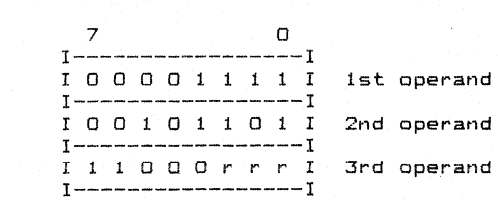


Copies the values of SS and SP of the previously register bank to the now current register bank. This copy shall done after register bank switch.

Helpful to keep the stack continuation after (but also before) register bank switching.

```
Example:        MOVSPA
```

BRKCS reg16



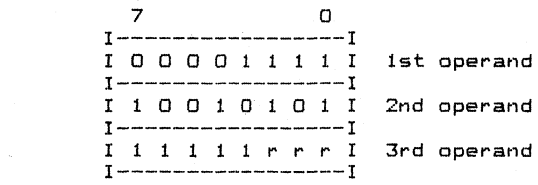
Switching the register bank by the value of "reg16", saves PC and PSW and loads the value of "VPC" to PC.

Useful to call subroutine with register bank switching.

This action is the same as register bank switching with hardware interrupt.

```
Example:    MOV     AW,3        ;register bank no.3
            BRKCS   AW          ;switch to reg.bank no.3
```

MOVSPB reg16

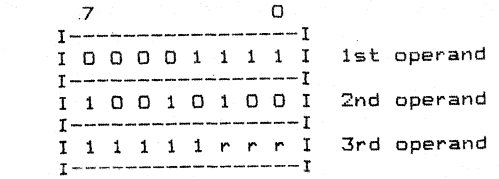


Copies the values of SS and SP of the current register bank to the corresponding registers of another register bank.

Useful to keep the stack continuation before and after register bank switching.

```
Example:        MOV     BW,5        ;reg.bank no.5
                MOVSPB  BW
```

TSKSW reg16



Saves PC and PSW, then switch to the register bank corresponding to the value of "reg16" and loads the values of "PC save" and PSW save" to PC and PSW.

Useful to switch tasks with register bank switching.

```
Example:  MOV    AW,0     ;reg.bank no.0
          TSKSW  AW       ;switch to reg.bank no.0
```

APPENDIX 5A: ADDITIONAL INSTRUCTIONS

The instruction set is upward compatible with the μPD70108/70116 in native mode.

The new instructions in addition to the μPD70108/70116 are listed as follows.

(1) Conditional branch instruction

○ BTCLR . . . . . Bit test instruction for special function register.
When the condition of the bit of the special function register is 1, execution of BTCLR can be used to reset that bit (0) and branch to the short label described in the operand.

Coding format

Mnemonic	Operand		
	special function register address	special function branch register bit	destination
BTCLR	**mem8**	**imm3**	**short-label**

(2) Interrupt instruction

○ RETRBI . . . . . .return instruction for register bank interrupt. This is used when returning from the interrupt processing routine which has used register bank switching function. It can not be used for return from vector interrupt.

Coding format

Mnemonic	operand
RETRBI	none

○ FINT . . . . . .instruction which indicates that interrupt processing for interrupt controller is completed.
When used for interrupts exclusive of NMI, INTR and software interrupt, it is necessary to execute before the return instruction from interrupt. It can not be used for NMI, INTR or for software interrupt.

Coding format

Mnemonic	operand
FINT	none

(3) CPU instruction

○ STOP . . . . . .transition instruction to STOP condition

Coding format

Mnemonic	operand
STOP	none

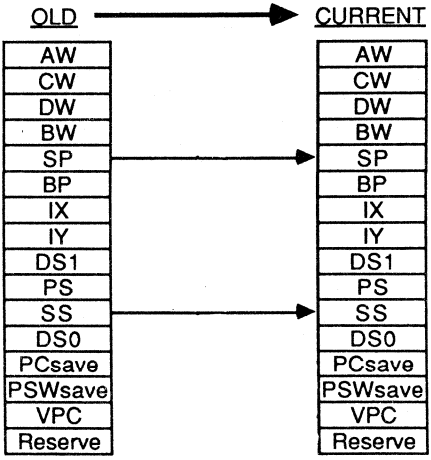
(4) Register bank switch instructions

I. MOVSPA.....Move Stack Pointer After register bank switch

After the register bank switching by the BRKCS instruction or due to an interrupt, this instruction copies the values of the SS and SP registers of the previously register bank into the SS and SP registers of the current register bank.
In the V25/V35 mode, the SS and SP are set in each register bank. Therefore, if the current register bank switch to another register bank, then also the SS and SP are switched. That cause a stack which is not contiously.
By this instruction, the values of the SS and SP registers of the "old" register bank can be copied into the SS and SP registers of the "currrent" register bank. This maintain the continuity of the stack.

Coding format

Mnemonic	Operand
MOVSPA	none



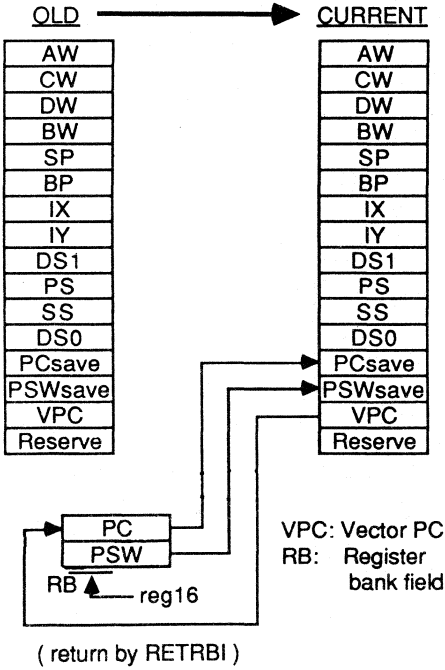
VPC: Vector PC

II. BRKCS.....BReaK for Context Switch (register bank switch)

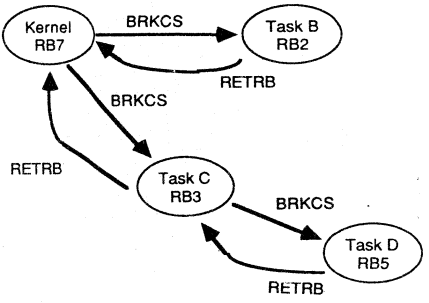
When this instruction is executed, the register bank specified by the operand "reg16" (the reg16 contains the register bank number) is selected for the now starting current register bank. Then the values of the PC and PSW registers are stored in the 'PC save' and 'PSW save' registers of the selected register bank. In addition, the program jumps to the address regarding the values of the PS and VPC registers in the selected register bank and the CPU set the Interrupt Disable (DI) state.
For return from this current register bank, use the RETRBI instruction.

Coding format

Mnemonic	Operand
BRKCS	reg 16



Example: Nested Tasks

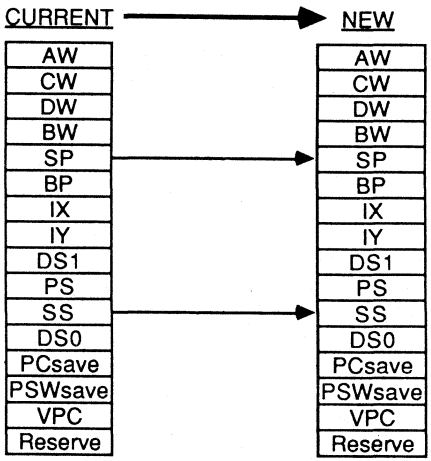


III. MOVSPB.....Move Stack Pointer Before register bank switch

Before register bank switching with the TSKSW or BRKCS instrcutions, this instruction copies the values of the SS and SP registers of the current register bank into the correspondig register bank, which is specified with the operand "reg16" (reg16 contains the number of the register bank).
In the V25/V35 mode, the SS and SP are set in each register bank. Therefore, if the current register bank switch to another register bank, then also the SS and SP are switched. That cause a stack which is not continously.
By this instruction, the values of the SS and SP registers of the current register bank can be copied into the SS and SP registers of each other register bank. This maintain the continuity of the stack.

Coding format

Mnemonic	Operand
MOVSPB	reg 16



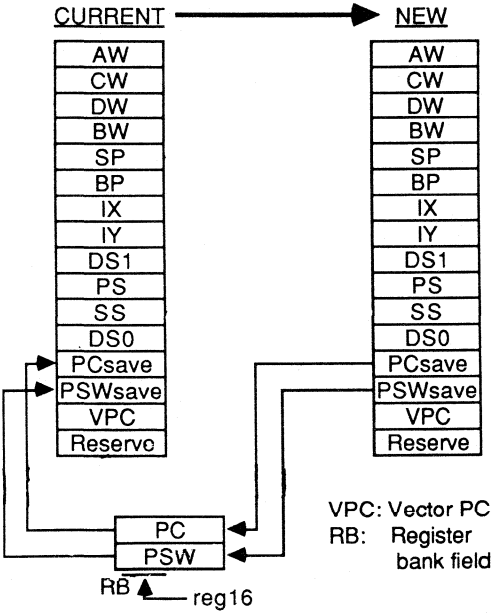
VPC: Vector PC

IV. TSKSW.....TaSK SWitching

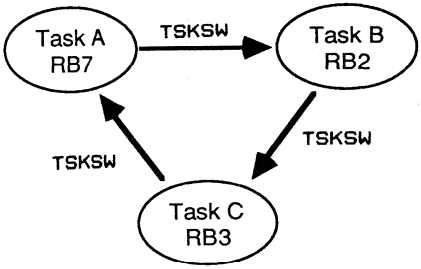
This instruction selects a register bank and is used for high-speed task selection.
The values of the PC and PSW are stored in the 'PC save' and 'PSW save' of the current register bank. Then the register bank selected by the operand 'reg16' is selected as the new register bank. After loading the value of the 'PSW save' register of the selected register bank into the PSW, the program jumps to the address abtained from the values of the PS and 'PC save' registers.

Coding format

Mnemonic	Operand
TSKSW	reg 16



Example: Circulations of Tasks



APPENDIX B
ADDRESSING MODES

ADDRESSING MODES

1 Instruction Address

The current address of is automatically incremented every time an instruction is executed. In addition, the microprocessor is provided with the following addressing methods for controlling execution procedure of instructions.

1.1 Direct Addressing

A 2- or 4-byte immediate data in an instruction is directly loaded to the PC alone or to both the PS and PC. The immediate data is then used as a branch address.
This addressing method is employed when executing the following instructions.

```
CALL  far-proc
CALL  memptr16
CALL  memptr32
BR    far-label
BR    memptr16
BR    memptr32
```

1.2 Relative Addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and added to the contents of the PC. The result of this addition is used as a branch address.
The sign bit of an 8-bit displacement value is extended and added to the contents of the PC as a 16-bit data.
When the addition is performed, the contents of the PC indicate the first address of the next instruction.
This addressing method is employed when executing the following instructions.

```
CALL  near-proc
BR    near-label
BR    short-label
Conditional Branch Instruction  short-label
```

1.3 Register Addressing

The contents of any 16-bit register specified by the register-specifying field (3 bits) of an instruction are loaded to the PC as a branch address. Unlike when an immediate data is used as a branch address, this addressing method allows all the eight 16-bit registers (AW, BW, CW, DW, IX, IY, SP, and BP) to be used. This addressing method is used when execution the following instructions:

		Example	
CALL	regptr 16	CALL	AW
BR	regptr 16	BR	BW

1.4 Register Indirect Addressing

A 16-bit register (IX, IY, or BW) is specified by the register-specifying field in an instruction. The specified register then addresses the contents of the memory (word or double word). The addressed contents are then loaded to the PC (or to both the PC and PS) as a branch address.

CALL	memptr16	CALL WORD PTR [IX]
CALL	memptr32	CALL DWORD PTR [IY]
BR	memptr16	BR WORD PTR [BW]
BR	memptr32	BR DWORD PTR [IX]

NOTE: Instruction code memptr 16 and memptr 32 are generated by the assembler in response to keywords WORD PTR and DWORD PTR, respectively.

1.5 Indexed Addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and is added to the contents of a 16-bit register that serves as an index register (IX or IY). The result of this addition addresses memory operand (word or double word) and it is loaded to PC as branch address.

		Example	
CALL	memptr 16	CALL	var [IX][2]
CALL	memptr 32	CALL	var [IY]
BR	memptr 16	BR	var [IY]
BR	memptr 32	BR	var [IX + 4]

1.6 Based Addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and is added to the contents of a 16-bit register (BP or BW) that serves as base register. The contents of the memory addressed by the result of this addition (word or double word) are loaded to the PC as a branch address.

This addressing method is employed when executing the following instructions:

		Example	
CALL	memptr16	CALL	var [BP + 2]
CALL	memptr32	CALL	var [BP]
BR	memptr16	BR	var [BW][2]
BR	memptr32	BR	var [BP]

NOTE: Instruction code memptr 16 is generated by the assembler if variable var has a word attribute. If it has a double word attribute, instruction code memptr 32 is generated.

1.7 Based Indexed Addressing

A 1- or 2-byte immediate data in an instruction byte is treated as a signed displacement value. This value is added to the contents of a 16-bit register that serves as a base register (BP or BW) and to the contents of a 16-bit register that serves as an index register (IX or IY). The result of this addition addresses the contents of the memory (word or double word). The addressed memory contents are loaded to the PC as a branch address.
This addressing method is employed when executing the following instructions:

```
CALL  memptr16
CALL  memptr32
BR    memptr16
BR    memptr32
```

Example

```
CALL  var [BP][IX]
CALL  var [BW + 2][IY]
BR    var [BW][2][IX]
Br    var [BP + 4][IY]
```

NOTE: Instruction code memptr 16 is generated by the assembler if variable var has a word attribute. If it has a double word attribute, instruction code memptr 32 is generated.

2 Memory Operand Address

This section describes several addressing methods for addressing registers or the memory when executing instructions.

2.1 Register Addressing

The contents of the register-specifying field (reg=3-bit field, sreg=2-bit field) in an instruction addresses a register. The 3-bit field "reg" is used in pairs with one bit (bit W) that is in the same instruction and indicates whether a word or a byte register is to be specified. Eight types of word registers (AW, BW, CW, DW, BP, SP, IX, and IY) and eight types of byte registers (AL, AH, BL, BH, CL, CH, DL, and DH) are specified.
The 2-bit field "sreg" specifies four types of segment registers (PS, SS, DS0, and DS1).
On some occasions, the operation code of an instruction specifies a register.
This addressing method is employed when executing the following instructions that have the following operand-writing formats:

Format	Items
reg	AW, BW, CW, DW, SP, BP, IX, IY, AL, AH, BL, BH, CL, CH, DL, DH
reg16	AW, BW, CW, DW, SP, BP, IX, IY
reg8	AL, AH, BL, BH, CL, CH, DL, DH
sreg	PS, SS, DS0, DS1
acc	AW, AL

Example

When MOV reg,reg is specified.

```
MOV BP,SP
MOV AL,CL
```

2.2 Immediate Addressing

A 1- or 2-byte immediate data in an instruction is used as is. This addressing method is employed when executing the instructions that have the following operand-writing formats:

Format	Items
imm	8/16-bit immediate data
imm16	16-bit immediate data
imm8	8-bit immediate data
pop-value	16-bit immediate data

If imm alone is specified, the assembler checks the value of imm written as an operand or the attribute of other operands that may be written at the same time and judges whether the value of imm is 8 bits or 16 bits. The status of word/byte-specifying bit W is then determined.

Example

When MOV reg,imm is specified

```
MOV AL, 5:  byte
```

When MUL reg16,reg16,imm16 is specified

```
MUL AW,BW,1000H
```

2.3 Direct Addressing

The immediate data in an instruction addresses the memory. This addressing method is employed when executing the instructions that have the following operand-writing formats:

Format	Item
mem	16-bit variable that specifies an 8- or 16-bit memory data
dmem	16-bit variable that specifies an 8- or 16-bit memory data
imm4	4-bit variable that indicates the bit length of the bit field data

Example

When MOV mem,imm is specified

```
MOV WORD-VAR, 2000H
```

When MOV acc,dmem is specified

```
MOV AL,BYTE-VAR
```

2.4 Register Indirect Addressing

A 16-bit register (IX, IY, and BW) specified by the memory-specifying field in an instruction addresses the memory.
This addressing method is employed when executing the instructions that have the following operand-writing formats:

Format	Method
mem	[IX],[IY],[BW]

Example

When SUB mem,reg is specified

```
SUB [IX],AW
```

2.5 Autoincrement/-decrement Addressing

This addressing method falls into the category of register indirect addressing.
The contents of a default register addresses a register or memory. Then the contents of the default register are automatically incremented/decremented by one if a byte process is performed. If it is a word process, the register contents are incremented/decremented by two. Stated another way, the address is automatically modified by this addressing function for processing the next byte/word operand. This addressing method is always applicable to default registers and is employed when executing the instructions that have the following operand formats:

Format	Default register
dst-block	IY
src-block	IX

This addressing will control block data operations if it is used in combination with a counter (CW) that counts the number of repetitions of a byte/word operand operation.

2.6 Indexed Addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and is added to the contents of a 16-bit register that serves as an index register (IX or IY). The result of this addition addresses a memory operand.
This addressing is useful when accessing an array of data. The displacement value indicates the start address of the array.
The contents of the index register determines the address of the data to be accessed.
This addressing method is employed when executing the instructions that have the following operand-writing formats:

Format	Method
mem	var [IX], var [IY]
mem16	var [IX]
mem8	var [IX]

Example

When TEST mem,imm is specified

```
TEST BYTE-VAR[IX],7FH
TEST BYTE-VAR[IX+8], 7FH
TEST WORD-VAR [IX][8],7FFFH
```

NOTE: If variable var has a byte attribute, a byte operand is specified. If it has a word attribute, a word operand is specified. An instruction code is generated by the assembler to each operand.

2.7 Based Addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and is added to the contents of a 16-bit base register that serves as a base register (BP or BW). The result of this addition addresses a memory operand.
This addressing is useful to access structural data that are stored at separate memory locations. The base register indicates the start address of each structural data and the displacement value selects one piece of data from each structural data.
This addressing method is employed when executing the instructions that have the following operand-writing format:

Format	Method
mem	var[BP], var[BW]
mem16	var[BP]
mem8	var[BP]

Example

When SHL mem,1 is specified

```
SHL BYTE-VAR[BP],1
SHL WORD-VAR[BP+2],1
SHL BYTE-VAR[BP][4],1
```

Note: If variable var has a byte attribute, a byte operand is specified. If it has a word attribute, a word operand is specified. An instruction code corresponding to each operand is generated by the assembler.

2.8 Based Indexed Addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value that is added to the contents of two 16-bit registers.
One of the registers serves as a base register (BP or BW) and the other as an index register (IX or IY). The result of the addition addresses a memory operand.
Since this addressing method allows accessing one data by modifying the contents of both the base and index registers, it is very useful when accessing structural data that are stored at separate memory locations and include data array.
For example, the contents of the base register indicates the first address of each structual data. The displacement value in turn indicates the number of offsets from that first address to the first address of a data array.
Then the index register can indicate a specific data in the data array.
This addressing method is employed when executing instructions that have the following operand-writing format:

Format	Item
mem	var [base register][index register]
mem16	var [base register][index register]
mem8	var [base register][index register]

Example

When PUSH mem16 is specified

```
PUSH WORD-VAR [BP][IX]
PUSH WORD-VAR [BP+2][IX+6]
PUSH WORD-VAR [BP][4][IX][8]
```

2.9 Bit Addressing

A 3- or 4-bit immediate data in an instruction or lower 3- or 4-bit of the CL register specifies one bit of the 8- or 16-bit register or memory.

By using this addressing method, a specific single bit in a register or the memory can be tested (for 0 or 1), set, cleared, or inverted without affecting the contents of other bits. That is, unlike when setting or resetting a bit by using the AND or OR instruction, a byte or word data does not have to be prepared to operate one bit.

This method is employed when executing the instructions that have the following operand-writing formats:

Format	Item
imm4	Bit number of word operand
imm3	Bit number of byte operand
CL	CL

Example

```
TEST1    reg8,CL
TEST1    AL,CL
NOT1     reg8,imm3
NOT1     CL,5
CLR1     mem16,CL
CLR1     WORD-VAR[IX],CL
SET1     mem16,imm4
SET1     WORD-VAR[BP],9
```

2.10 Addressing special function register.

When a special function register is addressed, the 1-byte of immediate data in the instruction byte is used as the offset value (unsigned) from the beginning of the special function register area.

This addressing mode apply only to the BTCLR insruction.

Description	Descriptive method
sfr	An 8-bit variable which specifies an 8-bit special function register

Example:

```
BTCLR EXIC, 7, 45
```

Note: Special function registers are mapped into memory, from address xxF00 to XXFFFH (xx is the IDB register value). Refer to "3.4.3 Special function register area" for details.

APPENDIX 5C

OVERVIEW OF INSTRUCTIONS

(ALPHABETIC ORDER)

EUROPEAN DISTRIBUTORS

AUSTRIA
A & D
ABRAHAMCZIK & DEMEL
GES. MBH. & CO KG
EICHENSTRASSE 58-64/1
1120 WIEN
TEL.: (222) 85 76 61
TLX.: 134 273

BELGIUM
INTRA ELECTRONICS P.V.B.A.
BOSUIL 80, BUS 1
2100 DEURNE
TEL.: (03) 3 25 23 20
TLX.: 31102

MALCHUS ELECTRONICS P.V.B.A.
PLANTIN EN MORETUSLEI 172
2000 ANTWERPEN
TEL.: (03) 2 35 32 72
TLX.: 33 637

DENMARK
MER-EL A/S
VED KLAEDEBO 18
2970 HOERSHOLM
TEL.: (2) 57 10 00
TLX.: 37 360

FINLAND
OY FERRADO A/B
P.O. BOX 54
VALIMONTIE 1
00380 HELSINKI 38
TEL.: (90) 55 00 02
TLX.: 122 214

FRANCE
ASAP
2 AVENUE DES CHAUMES
78180 MONTIGNY LE BRETONNEUX
TÉL.: (1) 30 43 82 33
TLX.: 698 887

C.C.I.
5, RUE MARCELIN BERTHELOT
B.P. 92
92164 ANTONY
TÉL.: (1) 46 66 21 82
TLX.: 203 881

CGE COMPOSANTS
32 RUE GRANGE DAME ROSE
92360 MEUDON
TÉL.: (1) 46 30 24 25
TLX.: 632 118

SERTRONIQUE
60 RUE SAGEBIEN
CEDEX 43
72040 LE MANS
TÉL.: (16) 43 84 24 60
TLX.: 720 019

GERMANY
BIT-ELECTRONIC AG
DINGOLFINGER STRASSE 6
8000 MÜNCHEN 80
TEL.: (0 89) 41 80 07-0
TLX.: 5 212 931

GLEICHMANN + CO ELECTRONICS GMBH
WORMSER STRASSE 34
6710 FRANKENTHAL
TEL.: (0 62 33) 2 50 56
TLX.: 4 65 270

GLYN GMBH
SCHÖNE AUSSICHT 30
6272 NIEDERNHAUSEN
TEL.: (0 61 27) 80 77
TLX.: 4 186 911

H3W ELEKTRONIK VERTRIEBS GMBH
STAHLGRUBERRING 12
8000 MÜNCHEN 82
TEL.: (0 89) 42 92 71
TLX.: 5 214 514

MICROSCAN GMBH
ÜBERSEERING 31
2000 HAMBURG 60
TEL.: (0 40) 6 32 00 30
TLX.: 2 13 288

REIN ELEKTRONIK GMBH
LÖTSCHERWEG 66
4054 NETTETAL 1
TEL.: (0 21 53) 73 31 11
TLX.: 8 54 251

SYSTEM ELEKTRONIK VERTRIEBS GMBH
HEESFELD 4
3300 BRAUNSCHWEIG
TEL.: (05 31) 31 40 95
TLX.: 9 52 351

ULTRATRONIK GMBH
GEWERBESTRASSE 4
8036 HERRSCHING
TEL.: (0 81 52) 37 09-0
TLX.: 5 26 459

UNIELECTRONIC VERTRIEBS GMBH
LISE-MEITNER-STRASSE 8
6072 DREIEICH 1 B. FRANKFURT
TEL.: (0 61 03) 3 51 75
TLX.: 4 11 213

ITALY
ADELSY S.R.L.
VIA DEL FONDITORE, 5
LOCALITA ROVERI
40127 BOLOGNA
TEL.: (51) 532119
TLX.: 510 226

CLAITRON S.P.A.
VIA GALLARATE, 211
20151 MILANO
TEL.: (2) 3010091
TLX.: 313 843

MELCHIONI S.P.A.
VIA COLLETTA, 37
20135 MILANO
TEL.: (2) 57941
TLX.: 315 293

NETHERLANDS
INNOCIRCUIT
MALCHUS ELECTRONICA
ADVIESGROEP
MALCHUS B.V.
FOKKERSTRAAT 511-513
3125 BD SCHIEDAM
TEL.: (10) 4277777
TLX.: 21598

INTRA ELECTRONICS B.V.
GULBERG 33
5674 TE NUENEN
TEL.: (0 40) 83 80 09
TLX.: 59418

NORWAY
JAKOB HATTELAND ELECTRONIC A/S
P.B. 25
5578 NEDRE VATS
TEL.: (47) 63 111
TLX.: 428 50

PORTUGAL
AMPEREL S.A.
AV. FONTES PEREIRA DE MELO 47, 4D
1000 LISBOA
TEL.: (1) 53 26 98
TLX.: 18588

SPAIN
COMELTA SA
EMILIO MUNOZ 41. NAVE 1-1-2
28037 MADRID
TEL.: (1) 7 54 30 77
TLX.: 42 007

VENCO
CALLE GALILEO 249
08028 BARCELONA
TEL.: (3) 330 97 51
TLX.: 98 266

SWEDEN
NAX AB KOMPONENTBOLAGET
BOX 4115
17104 SOLNA
TEL.: (8) 98 51 40
TLX.: 17912

SWITZERLAND
MEMOTEC AG
GASWERKSTRASSE 32
4901 LANGENTHAL
TEL.: (63) 28 11 22
TLX.: 9 82 550

UNITED KINGDOM
ANZAC COMPONENTS LIMITED
822, YEOVIL ROAD
SLOUGH TRADING ESTATE
SLOUGH
BERKSHIRE
TEL.: (62 86) 44 11
TLX.: 847 949

CELDIS LIMITED
37, LOVEROCK ROAD
READING
BERKSHIRE
RG3 1EL
TEL.: (7 34) 56 51 71
TLX.: 848 370

DIALOGUE DISTRIBUTION LIMITED
WICAT HOUSE
403, LONDON ROAD
CAMBERLEY
SURREY
GO15 3HL
TEL.: (2 76) 68 20 01
TLX.: 858 944

IMPULSE ELECTRONICS LIMITED
HAMMOND HOUSE
CATERHAM
SURREY
CR3 6XG
TEL.: (8 83) 4 64 33
TLX.: 291 496

S.T.C. MULTICOMPONENTS LIMITED
EDINBURGH WAY
HARLOW
ESSEX
CM20 2DF
TEL.: (2 79) 44 29 71
TLX.: 818 763

V.S.I. ELECTRONICS (UK) LIMITED
ROYDONBURY INDUSTRIAL PARK
HORSECROFT ROAD
HARLOW
ESSEX
CM19 5BY
TEL.: (2 79) 2 96 66
TLX.: 81 387

NEC OFFICES

NEC Electronics (Europe) GmbH, Oberrather Str. 4, 4000 Düsseldorf 30, W. Germany,
Tel. (02 11) 65 03 01, Telex 8 58 996-0

NEC Electronics (Germany) GmbH, Oberrather Str. 4, 4000 Düsseldorf 30,
Tel. (02 11) 65 03 02, Telex 8 58 996-0
- Königstr. 12, 3000 Hannover 1, Tel. (05 11) 31 60 91, Telex 9 230 109
- Arabellastr. 17, 8000 München 81, Tel. (0 89) 92 10 03-0, Telex 5 22 971
- Heilbronner Str. 314, 7000 Stuttgart 30, Tel. (07 11) 89 09 10, Telex 7 252 220

NEC Electronics (BNL) - Boschdijk 187a, NL-5612 HB Eindhoven, Tel. (0 40) 44 58 45,
Telex 51 923

NEC Electronics (Scandinavia) - Svärdvägen 25 B, S-18233 Danderyd,
Tel. (08) 75 36 020, Telex 13 839

NEC Electronics (France) S.A., 9, rue Paul Dautier, B.P. 187,
F-78142 Velizy Villacoublay Cedex, Tél. (1) 39 46 96 17, Télex 699 499

NEC Electronics (France) S.A., Representacion en Espana, Edificio «La Caixa»,
Paseo de la Castellana 51, E-28046 Madrid, Tél. (1) 41 94 150, Télex 41 316

NEC Electronics Italiana S.R.L., Via Fabio Filzi, 25A, I-20124 Milano, Tel. (02) 67 09 108,
Telex 315 355
- Rome Office, Via Monte Cervialto, 131, I-00139 Rome,
Tel. (06) 8 11 12 91, Telex 623 323

NEC Electronics (UK) Ltd., Cygnus House, Sunrise Park Way, Milton Keynes, MK14 6NP,
Tel. (09 08) 69 11 33, Telex 777 565
- Dublin Office, 34/35 South William Street, Dublin 2, Ireland, Tel. (00 01) 71 02 00