NEC

μPD70320/322G/L
CMOS 16-BIT MICROCOMPUTER

PRODUCT DESCRIPTION

TABLE OF CONTENTS

This document describes the functions of a product under development. Certain parts of the document are subject to change without notice.

NEC MOS Integrated Circuit
μPD70322, μPD70320G/L One Chip Microcomputer

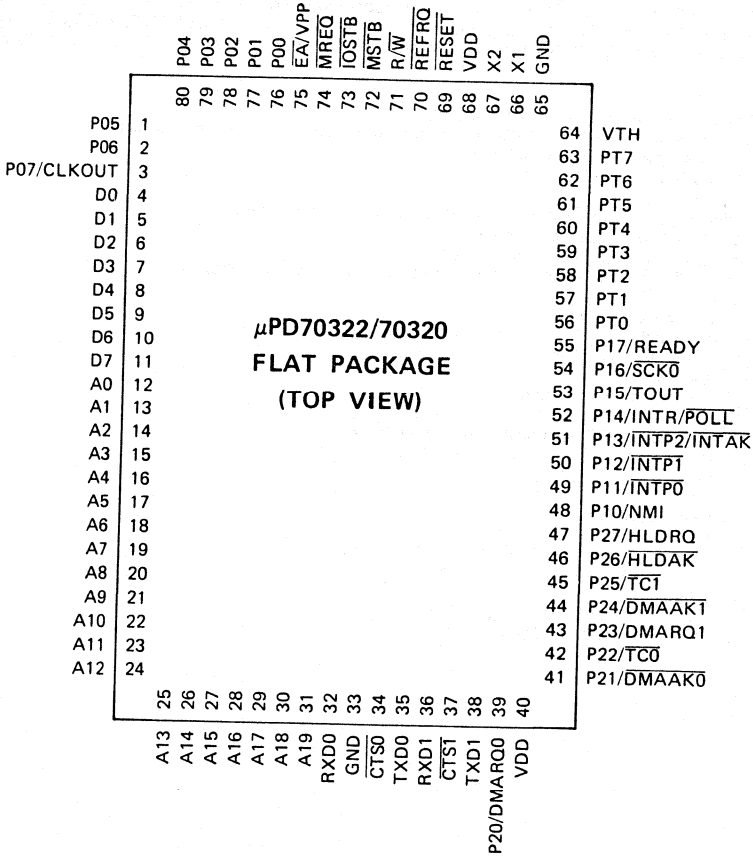
The μPD70322 (also known as V25tm) is a one -chip microcomputer which features one-chip integration of 16-bit CPU, ROM, RAM, serial interface, timer, DMA controller, interrupt controller and others. The μPD70322 is software-compatible with the 8/16-bit microprocessor μPD70108/70116 (also known as V20tm/V30tm). The μPD70320 is a μPD70322 without ROM.

Features

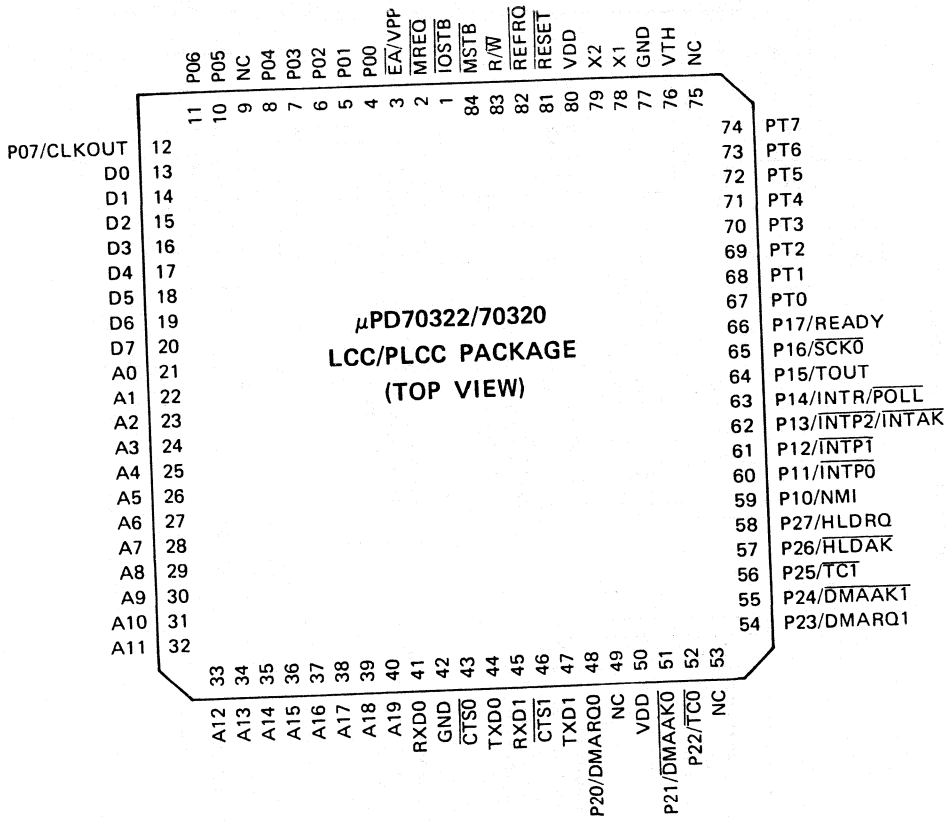
- internal 16-bit architecture, external 8-bit data bus.
- software-compatible with μPD70108/70116 (in native mode) (additional instructions available).
- minimum instruction cycle: 400 ns (10MHz, 5V).
- internal ROM: 16383W x 8 (μPD70322).
- internal RAM: 256W x 8.
- one-chip peripheral hardware memory mapping (special function register).
- input port (port T) with a comparator: 8 channels.,
- I/O lines (input ports: 4; input/output ports: 20).
- serial interface (with a built-in dedicated baud rate generator): 2 ch; asynchronous mode, I/O interface mode.
- interrupt controller
  - programmable priority (8 levels)
  - vector interrupt function
  - register bank switching function
  - macro-service function
- DRAM, pseudo SRAM refresh function
- DMA controller
- input/output instructions, FPO instruction interrupt function.
- 16-bit timers: 2.
- time base counter.
- built-in clock generator circuit.
- programmable wait function.
- standby function (STOP/HALT).
- variable instruction cycles: 400ns, 800ns, 1.6μs (10MHz, 5V).
- CMOS.
- single power supply.
- 80-pin plastic flat package (μPD70322G, μPD70320G).
- 84-pin PLCC (Plastic Leaded Chip Carrier) μPD70322L, μPD70320L).

Pin Connection Diagram

(1) 80-pin Plastic Flat Package (top view).

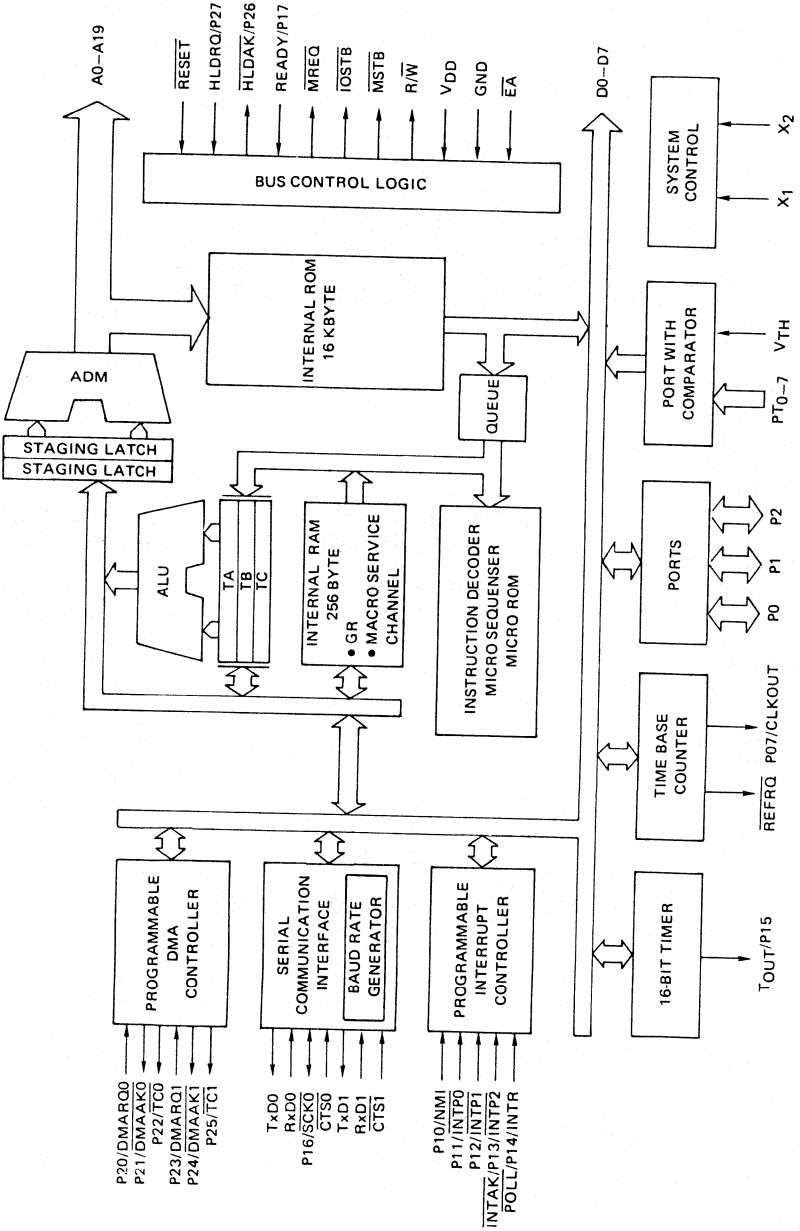


(2) 84-pin PLCC (top view)



Note: IC terminal should be fixed at the high level.

μPD70322/70320 block diagram



1. Pin functions

1.1 Port Pins

Pin Name	I/O	Port functions	Control functions
P00-P06	I/O	8-bit I/O port whose I/O can be specified at bit level	–
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,I	input/output port whose I/O can be specified and POLL input	external interrupt request input
P15/TOUT	I/O, O	I/O port whose I/O can be specified at bit level	timer output
P16/$\overline{\text{SCK0}}$			serial clock output
P17/READY	I/O, I		READY input
P20/DMARQ0	I/O, I	8-bit I/O port whose I/O can be specified at bit level	DMA request input (CH0)
P21/$\overline{\text{DMAAK0}}$	I/O, O		DMA acknowledgement output (CH0)
P22/$\overline{\text{TC0}}$			DMA end output (CH1)
P23/DMARQ1	I/O, I		DMA request input (CH1)
P24/$\overline{\text{DMAAK1}}$	I/O, O		DMA acknowledgement output (CH1)
P25/$\overline{\text{TC1}}$			DMA end output (CH1)
P26/$\overline{\text{HLDAK}}$	I/O, O		HOLD acknowledgement output
P27/HLDRQ	I/O, I		HOLD input
PT0-PT7	I	input port with 8-bit comparator	–

1.2 Non-port Pins

Pin Name	I/O	Function
TXD0	output	serial data output
TXD1		
RXD0	input	serial data input
RXD1		
$\overline{CTS0}$	I/O	CTS input in asynchronous mode; receive clock input in I/O interface mode
$\overline{CTS1}$	input	CTS input
$\overline{REFRQ}$	output	DRAM refresh pulse output
V_{TH}	input	comparator reference voltage input
$\overline{RESET}$		reset signal input
$\overline{EA}$		Input for setting ROM-less mode
X1		connector for crystal system clock oscillation. External clock input is carried out by
X2		Internal connection. Should be fixed to the high level from the outside.
D0-D7	I/O	8-bit data bus
A0-A19	output	20-bit address output
$\overline{MREQ}$		output indicating start of memory bus cycle.
$\overline{MSTB}$		Strobe output for memory read or memory write.
R/$\overline{W}$		Read cycle and write cycle identification signal output
$\overline{IOSTB}$		I/O read or I/O write strobe output
V_{DD}		Positive power supply pin
GND		GND pin
I.C.		Internal connection. Should be fixed to the high level from the outside.

2. CPU

The μPD70322/70320 has a CPU which is software-compatible with the native-mode operation of the μPD70116/70108.

2.1 Registers

The μPD70322/70320 CPU has a general-purpose register set compatible with the μPD70116/70108. It also has special function registers for the control of on-chip peripheral hardware. These registers are all mapped in the memory space. The general-purpose register set also serves as built-in RAM, providing a maximum 8-bank register set in the internal RAM.

The addresses of the registers are relocatable in 4-kilobyte units. These addresses are specified using the internal data area base register (IDB) which is one of the special function registers (see 2.4.2).

2.1.1 Register Bank

The general purpose register set is mapped in the built-in RAM area. The general purpose register set is bank-formatted. Up to 8 banks of it can be installed. Each bank uses 32 bytes. Of these 8 banks, banks 0 and 1 can also be used for macro-service channel (see 2.4.2) and DMA service channel (see 5.3.3). They can also be accessed as data memory (see 2.4.4).

Normally the CPU runs programs by using register bank 7, switching automatically to other register banks through the use of interrupts. Return to the original register bank from a register bank switched by interrupt is carried out only by a return instruction – the RETRBI instruction (additional instruction from μPD70108/70116) from the interrupt.

Fig. 2-1 shows the configuration of the register bank. The (+00H)-(+01H) in the register bank become reserve areas when the register bank is used. The general purpose register set is mapped in the area of (+08H)-(+1FH) by an offset from a initial address from each register bank. The area (+02H)-(+07H) is not for general use as it is used for the switching of register banks.

Area (+02H) holds the value which the register bank loads to the PC during register bank switching, an offset of the interrupt processing routine starting address.

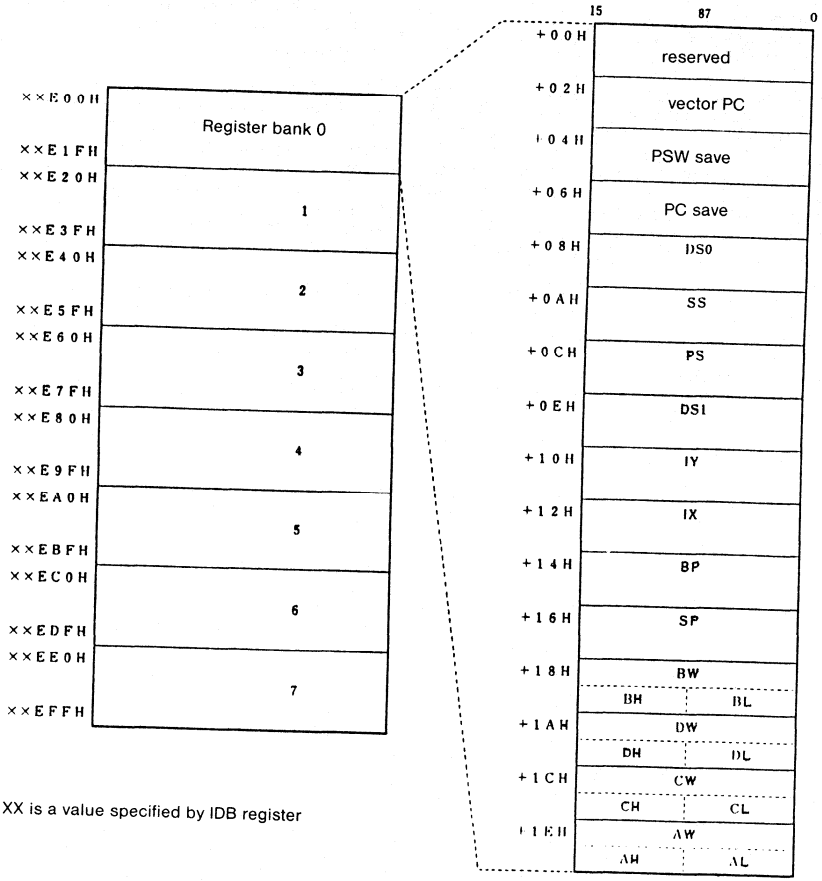
(+04H) is an area for saving the PSW when register banks are switched.

(+06H) is an area for saving the PC when register banks are switched.

After reset, register bank 7 is automatically selected.

Initialization of segment register (see 2.1.4) after reset is executed only in the register of register bank 7.

Fig. 2-1 Configuration of Register Bank



XX is a value specified by IDB register

(offset from starting address of each register bank)

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

Four 16-bit registers are used as general purpose registers; each register can be accessed as a 16-bit register as well as 8-bit registers by dividing it into higher and lower 8-bits (AH, AL, BH, BL, CH, CL, DH, DL).

These can be used as either 8-bit or 16-bit registers for a wide range of instructions including transfer, arithmetic, and logical operation instructions.

Each register is used as a default register for specific instruction processing 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 branching, repeat prefixing.
CL: shift instruction, rotation instruction, BCD operation.
DW: word multiplication/division, indirect addressing input/output.

These registers are mapped in the internal RAM. Their addresses are determined by adding the offset of each register to (IDB register* value* x 4096) + (0E00H) + (register bank number x 32). *See 2.4.2 for information on IDB register.

Fig. 2-1 Offset values for general purpose registers

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

2.1.3 Pointers (SO, BP) and Index Registers (IX, IY)

Base pointers or index registers are used during memory access using based addressing (BP), indexed addressing (IX, IY), or based indexed addressing (BP, IX, IY). They are also used as pointers during stack operations (SP). Like the general purpose registers they are used for instructions for transfer, arithmetic operations, and logical operations; however, in this case they cannot be used as 8-bit registers. Each of the registers is used as a default register for specific processing as follows:

SP: stack operations
IX: block transfer, source side of BCD string operations
IY: block transfer, destination side of BCD string operations.

These registers are mapped in internal RAM. Their addresses are determined by adding the offset of each register to (IDB register value x 4096) + (0E00H) + (register bank number x 32). Offset values for each register are indicated in Figure 2-2. *See 2.4.2 for information on IDB register.

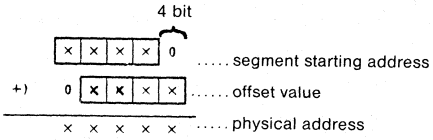
Fig. 2-2 Offset values for pointers and index registers

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

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

The CPU divides the memory space into logical segments of 64 kilobytes each, the starting address of each segment is specified by the segment register and the offset part of the initial address is specified by another register or by the effective address.

The physical address therefore is created in the following way:



There are four types of segment registers; PS (Program Segment), SS (Stack Segment), DS0 (Data Segment 0), and DS1 (Data Segment 1). The respective segments are used in the following cases:

PS: Program fetch

SS: Stack operation instructions, addressing using the BP as the base register.

DS0: general variable access, source block data access for block transfer instructions.

DS1: destination block data access for block transfer instructions.

However, other segments can be used instead of DS0 by using a segment override prefix, or other segments instead of SS may be used in the same way in addressing with BP base register.

During reset, the PS of register 7 is initialized for FFFFH and SS, DS0, and DS1 can be initialized for 000H. These registers are mapped using internal RAM and their addresses are determined by adding the offset of each register to (IDB register* value x 4096) + (0E00H) + (register bank number x 32) as indicated in Figure 2-3.

*See 2.4.2 for explanation of IDB register.

Fig. 2-3 Offset values for segment registers

Register	Offset value
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 (2.4.1) which is the area for the special function register (See 2.4.3) for controlling the internal RAM (also used with the general purpose register) and the on-chip peripheral hardware. These registers can be referenced by using FFFFFH or their own value x 4096 + FFFH (See 2.4.2)

2.1.6 Special function registers

The μPD70320/70322 has a group of register with special functions for setting up and controlling on-chip peripheral hardware modes. These register groups are memory mapped in the special function register areas inside the internal data areas and Read/Write is carried out in the same way as with regular memory (see 2.4.3).

The additional BTCLR instructions (See 13.1) can be used only for these special function registers.

2.2 Program Counter (PC)

This is a 16-bit binary counter for holding the offset information on the memory addresses of a program to be executed by the CPU.

The program counter is incremented each time instruction bytes are fetched from the instruction queue. A new location is loaded while executing branch, call, return, and break instructions.

0000H is loaded while resetting. PS is initialized or FFFFH during reset so that the CPU starts execution from FFFF0H after reset.

2.3 PSW (Program Status Word)

PSW is comprised of six types of status flags and five types of control flags as well as user flags.

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

Control Flags
- RB0-RB2 (Register Bank 0-2)
- DIR (Direction)
- IE (Interrupt Enable)
- BRK (Break)
- IBRK (I/O Break)

User flags
- F0 (User Flag 0)
- F1 (User Flag 1)

The status flags are automatically set (1) and reset (0) according to a number of instruction executions (data value). The CY flags can be set directly, reset, and reserved by instructions.

The control flags are set and reset by instructions controlling the CPU operations. The IE and BRK flags are always reset whenever an interrupt processing is started.

The user flags can be set, reset, and tested by instructions and can be freely used by the user.

When the PSW is processed in byte or word units, it is executed in the following way.

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

The least significant 8-bit PSW can be stored in the AH register and restored using MOV instructions. PSW can be saved separately and returned to stack using PUSH PSW and POP PSW instructions. The upper 4 bits of PSW are not affected by POP PSW instruction. The method of changing the upper 4 bits of the PSW is using the RETI or the RETRBI instruction. The others are automatically returned before the control flag is changed using interrupt generation. RESET input is used to initialize PSW at F002H using word image. The IBRK and RB0-RB2 flags are set (1) and the others are reset (0).

2.3.1 CY (Carry Flag)

(1) Binary addition and subtraction

When carrying out byte operations, the flag is set when there is a carry or a borrow from operation result bit 7; otherwise, it is reset.

When word operations are carried out, it is set when there is a carry or a borrow from operation result bit 15; otherwise, it is reset.

The flag is not changed by increment and decrement instructions.

(2) Logical operations

The flag is reset regardless of the results of the operations.

(3) Binary multiplication

The flag is reset when an unsigned byte operation gives 0 for AH; otherwise it is set. The flag is reset when a signed byte operation gives a sign expansion of AL for AH; otherwise it is set.

The flag is reset when an unsigned word operation gives 0 for DW; otherwise it is set. The flag is reset when a signed word operation gives a signed expansion of AW for DW; otherwise it is set.

With 8-bit immediate operations, it is reset when the product is within 16 bits and is set when it exceeds 16 bits.

(4) Binary division

Undefined

(5) Shift/Rotate

With shift and rotate which include CY, it is set if the bit shifted to CY is 1, and is reset if 0.

2.3.2 P (Parity Flag)

(1) Binary addition and subtraction, logical operation, shift.

This flag is set when the number of "1" bits in the lower 8 bits, representing the results of an operation, is even; it is reset when the number is odd.

When results are all 0, it is set.

(2) Binary Multiplication and Division undefinied

2.3.3 AC (Auxiliary Flag)

When working with byte operations, the flag is set when there is a carry from the lower 4 bits to the higher 4 bits or when there is a borrow from the higher 4 bits to the lower 4 bits; and is reset in all other cases.

In word operations, the same operations are carried out for the lower byte as for byte operations.

(2) Logical operations, binary multiplication and division, shift/rotate undefined

2.3.4. Z (Zero Flag)
(1) Binary addition and substraction, logical operations, shift/rotate.
For byte operations, the flag is set if the resulting 8 bits are 0; it is reset for all other values. For word operations, it is set if the resulting 16 bits are 0; it is reset for all other values.
(2) Binary multiplication and division undefined
2.3.5 S (Sign Flag)
(1) Binary addition/subtraction, logical operations, shift/rotate.
For byte operations, it is set when the resulting bit 7 is 1, and reset when 0.
For word operations, it is set when resulting bit 15 is 1, and reset when 0.
(2) Binary multiplication and division
undefined

2.3.6 V (Overflow Flag)
(1) Binary addition and subtraction
For byte operations, it is set if the carries from bits 7 and 6 are different and reset if the same. For word operations, it is set if the carries from bits 15 and 14 are different and reset if the same.
(2) Binary multiplication
The flag is reset when an unsigned byte operation gives 0 for AH; otherwise it is set. The flag is reset when a signed byte operation gives a sign expansion of AL for AH; otherwise it is set.
The flag is reset when an unsigned word operation gives 0 for DW; otherwise it is set. The flag is reset when a signed word operation, gives a signed expansion of AW for DW; otherwise it is set.
With 8-bit immediate operations, it is reset when the product is within 16 bits and is set when it exceeds 16 bits.
(3) Binary division
Reset.
(4) Logical operation
Reset.
(5) Shift/Rotate
In the case of left 1 bit shift/rotate,
when CY = most significant bit, is reset
when CY = most significant bit, is set in the operational results.
In the case of right 1 bit shift/rotate,
when most significant bit = next least significant bit after most significant bit, is reset
when most significant bit = next least significant bit after most significant bit, is set.
Undefined in the case of multibit shift/rotate
2.3.7 IBRK (I/O Break Flag)
Controls the software interrupt generation during input/output instructions.
When the execution of an I/O instruction is attempted with IBRK = 0, a software interrupt is automatically generated (interrupt vector 20), enabling a software simulation of the I/O instruction.
When IBRK = 1, input/output instructions are executed in the normal manner and software interrupts do not take place.
2.3.8 BRK (Break Flag)
Only in a condition where it is saved to the stack as a part of PSW can it be set using memory operation instruction and is effective after setting when it is restored in the PSW.
If BRK flag is set, software interrupt (interrupt vector 1) automatically takes place when one instruction is executed making it possible to trace each instruction.
2.3.9 IE (Interrupt Enable Flag)
It is set by the EI instruction, to enable the interrupt; it is reset by the DI instruction to disable the interrupt.
2.3.10 DIR (Direction Flag)
It is set by the SET1 DIR instruction and reset by the CLR1 DIR instruction.
When the DIR flag is set, processing is executed from the higher address to the lower address in block transfer/input output group instructions; when it is reset, processing is executed from lower addresses to higher addresses.
2.3.11 RB0-RB2 (Register Bank 0-2 Flag)
The RB0-RB2 is used to specify currently used register banks from among the eight register banks installed in the internal RAM.
2.3.12 F0, F1 (User Flag 0, 1 Flag)
Can be freely used by users.
The setting and resetting of these flags can be executed by instructions for PSW and can be used to execute set, reset, and test by using special function register flags.
User flags F0, F1 operate in the following way when operated by flag register.

Fig. 2-2 Format for user flag register (FLAG)

Symbol	7	6	5	4	3	2	1	0	address
FLAG	–	–	F1	–	F0	–	–	–	××FEAH

2.4 Memory Space

The μPD70322/70320 has a 1-megabyte memory space. A memory map is indicated in Fig. 2-3. The space up to 00000H-003FF is used as a vector area. However, it can be used for other purposes if it is not used for vectors. XXE00H-XXFFFH (XX indicates IDB register value) is internal data area. The location of this area can be changed in units of 4 kilobytes. The 4 byte FFFFCH-FFFFFH is reserved. In FFFFFH, IDB register is assigned. Wait cycles can be inserted in the memory space, programmable in each 128 kilobytes segment.

The 1 megabyte physical address space is designated by the offset value for the segment initiator location which is indicated by a segment starting address which is indicated by segment register and other register or effective address.

Fig. 2-3 Memory Map

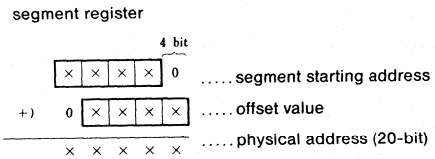
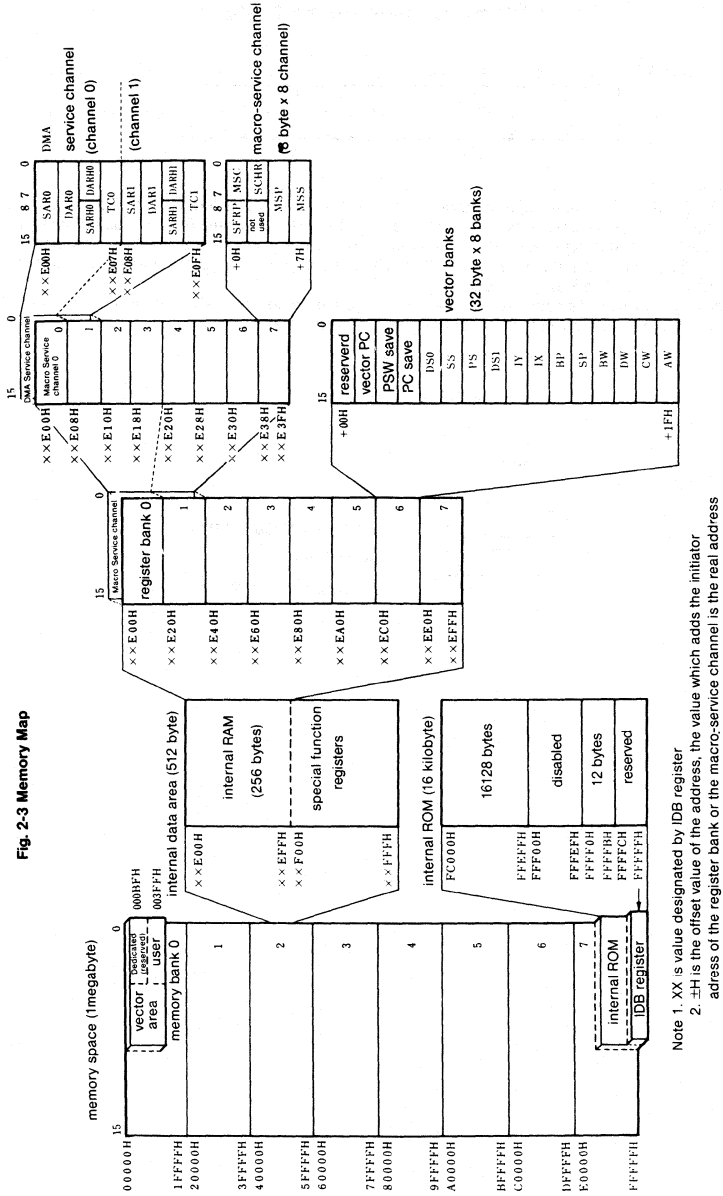


Fig. 2-3 Memory Map



Note 1. XX is value designated by IDB register
2. ±H is the offset value of the address, the value which adds the initiator adress of the register bank or the macro-service channel is the real address
3. Internal ROM is for the μPD70322 only.
4. Macro-servic channel is overlapped and assigned to register banks 0,1;
DMA service is overlapped and assigned to macro-service channel 0,1.

2.4.1 Internal Data Areas

The internal data areas are a 512 byte area containing internal RAM and special function register areas. 1-megabyte memory space can be divided in 4 kilobyte units. The internal data area base addresses are set up using the IDB register (internal data area base register). The higher 8 bits of the 20-bit internal data base address are set up using the IDB register and the lower 12 bits are fixed at E00H (beginning of area).

The internal data area is operated by memory operation instructions.

The internal data areas overlap the external memory space or the internal ROM areas (μPD70322 only).

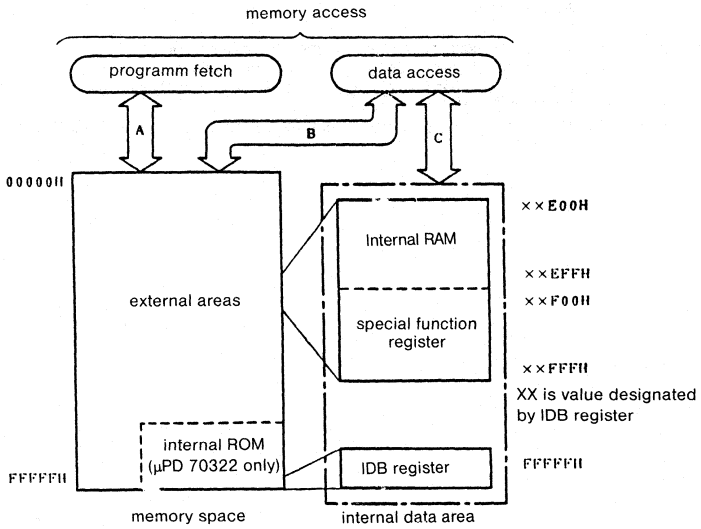
Memory access for all operations except program fetch can access internal data areas. It should be noted that internal data areas cannot be accessed by program fetch.

The least significant 256 byte of internal data areas (in XXE00H-XXEFFH, XX is the value specified by IDB register) is in the internal RAM area. In addition to the use as an ordinary RAM, the internal RAM has been assigned functionally register bank, macro-service channel and DMA service channel use. The internal RAM can be used in a way to disable access as ordinary RAM by resetting (0) bit 6 (RAMEN) of the processor control register (PRC) which is a special function register. In this case it still can be accessed as Register Banks.

The higher 256 byte of the internal areas (in XXF00H-XXFFFH, XX is the value specified by the IDB register) is a special function register area. The special function register has a register group that is mapped and has special functions assigned such as on-chip peripheral hardware mode registers and control registers.

The internal data areas are located in the FFE00H-FFFFFH location using RESET input, from the initialization of IDB register to FFH.

Fig. 2-4 Access Conditions for Memory Space



A. program fetch can access everything except the internal data areas.

B. data access outside of internal data areas or address data access corresponding to internal RAM areas during internal RAM access disable.

C. if it does not meet conditions in B, data access to the internal data areas takes precedence.

2.4.2 Internal Data Base Register (IDB)

This is a register for determining the physical address of internal data areas (areas for internal RAM and special function register) which can be located in 4 kilobyte steps. The higher 8-bit internal data area base address is designated by the IDB and the lower 12 bits are fixed at E00H (beginning address).

The IDB has two addresses assigned: XXFFFH inside the special function register (XX is the value of the IDB register itself) and the FFFFFH fixed address. The IDB can be modified or referenced by memory access to either of these two adresses with the same effect.

The IDB is set to FFH at reset time so that the internal data area base address is FFE00H.

2.4.3 Special Function Register Areas

A group of registers with special functions for on-chip peripheral hardware mode registers and control registers assigned are mapped to XXF00H-XXFFFH (XX is the value designated by the IDB register). The IDB register is specially assigned to both XXFFFH (XX is the value designated by IDB register) and FFFFFH fixed adresses. Program fetches cannot be executed from these areas.

The special function register is operated by memory access. The additional BTCLR instruction (additional to the μPD70108/70116) is a special instruction used exclusively for this area and applies to the bit in the areas no matter where the area is located in the memory space.

Charts 2-4 and 2-5 show a list of special function registers. Meanings of individual items in the chart are as follows:

- SYMBOL . . . . . symbol which indicates internally stored special function addresses coded in the instruction operand column.
- R/W . . . . . indicates whether a given function register is Read/Write-capable
  - R/W: Read/Write-capable
  - R: Read only
  - W: Write only
- Operational method . . . . . each register indicates whether 16-bit operations, 8-bit operations or 1-bit operations are possible.
- RESET condition . . . . . indicates condition of each register after RESET input XX in the higher 8 bits of an address is specified by IDB register.

The address part which is not mentioned is reserved. Contents during Read are undefined. Operations during Write have no significance.

Table 2-4 Special Function Registers

address	name of special function register	Symbol	R/W	operation method (bit)	RESET CONDITION
XX F00H	port 0	P0	R/W	8/1	undefined
XX F01H	port 0 mode register	PM0			FFH
XX F02H	port 0 mode control register	PMC0			00H
XX F08H	port 1	P1			undefined
XX F09H	port 1 mode register	PM1			FFH
XX F0AH	port 1 mode control register	PMC1			00H
XX F10H	port 2	P2			undefined
XX F11H	port 2 mode register	PM2			FFH
XX F12H	port 2 mode control register	PMC2			00H
XX F38H	port T	PT	R	8	undefined
XX F3BH	port T mode register	PMT	R/W	8/1	00H
XX F40H	external interrupt mode register	INTM	R/W	8/1	00H
XX F44H	external interrupt macro-service control register 0	EMS0			undefined
XX F45H	external interrupt macro-service control register 1	EMS1			
XX F46H	external interrupt macro-service control register 2	EMS2			
XX F4CH	external interrupt request control register 0	EXIC0			47H
XX F4DH	external interrupt request control register 1	EXIC1			
XX F4EH	external interrupt request control register 2	EXCI2			
XX F60H	receive buffer register 0	RxB0	R	8	undefined
XX F62H	transmit buffer register 0	TxB0	W		
XX F65H	serial receive macro-service control register 0	SRMS0	R/W	8/1	undefined
XX F66H	serial trasmit macro-service control register 0	STMS0			
XX F68H	serial mode register 0	SCM0			00H
XX F69H	serial control register 0	SCC0			
XX F6AH	baud rate generator register 0	BRG0			
XX F6BH	serial error register 0	SCE0	R	8	00H
XX F6CH	serial error interrupt request control register 0	SEIC0	R/W	8/1	47H
XX F6DH	serial receive interrupt request control register 0	SRIC0			
XX F6EH	serial transmit interrupt request control register 0	STIC0			

Table 2-5 Special Function Registers (cont.)

address	name of special function register	Symbol	R/W	operation method (bit)	RESET CONDITION
XX F70H	receive buffer register 1	RxB1	R	8	undefined
XX F72H	transmit buffer register 1	TxB1	W		
XX F75H	serial receive macro-service control register 1	SRMS1	R/W	8/1	undefined
XX F76H	serial transmit macro-service control register 1	STMS1			undefined
XX F78H	serial mode register 1	SCMO1			00H
XX F79H	serial control register 1	SCC1			
XX F7AH	baud rate generator register 1	BRG1			
XX F7BH	serial error register 1	SCE1	R	8	00H
XX F7CH	serial error interrupt request control register 1	SEIC1	R/W	8/1	47H
XX F7DH	serial receive interrupt request control register 1	SRIC1			
XX F7EH	serial transmit interrupt request control register 1	STIC1			
XX F80H	timer register 0	TM0	R/W	16	undefined
XX F82H	modulo/timer register 0	MD0			
XX F88H	timer register 1	TM1			
XX F8AH	modulo/timer register 1	MD1			
XX F90H	timer control register 0	TMC0	R/W	8/1	00H
XX F91H	timer control register 1	TMC1			
XX F94H	timer unit macro-service control register 0	TMMS0	R/W	8/1	undefined
XX F95H	timer unit macro-service control register 1	TMMS1			
XX F96H	timer unit macro-service control register 2	TMMS2			
XX F9CH	timer unit interrupt request control register 0	TMIC0			47H
XX F9DH	timer unit interrupt request control register 1	TMIC1			
XX F9EH	timer unit interrupt request control register 2	TMIC2			
XX FA0H	DMA control register 0	DMAC0	R/W	8/1	undefined
XX FA1H	DMA mode register 0	DMAM0			00H
XX FA2H	DMA control register 1	DMAC1			undefined
XX FA3H	DMA mode register 1	DMAM1			00H
XX FACH	DMA interrupt request control register 0	DIC0			47H
XX FADH	DMA interrupt request control register 1	DIC1			
XX FE0H	standby control register	STBC	R/W	8/1	*undefined
XX FE1H	refresh mode register	RFM	R/W	8/1	FCH
XX FE8H	wait control register	WTC	R/W	16/8	FFFFH
XX FEAH	user flag register **	FLAG	R/W	8/1	00H
XX FEBH	processor control register	PRC	R/W	8/1	4EH
XX FECH	time base interrupt request control register	TBIC			00H
XX FFFH	internal data area base register	IDB			FFH

* If the standby control register (SB) is set once, it cannot be reset by instruction. It is cleared by power supply voltage.

** Bit operations exclusive of bit 3 and bit 5 of the user flag register (FLAG) are of no significance. Also, the content of user flag 0, 1 (F0, F1) of the flag register can also vary according to the F0, F1 operation of PSW (See 2.3.12)

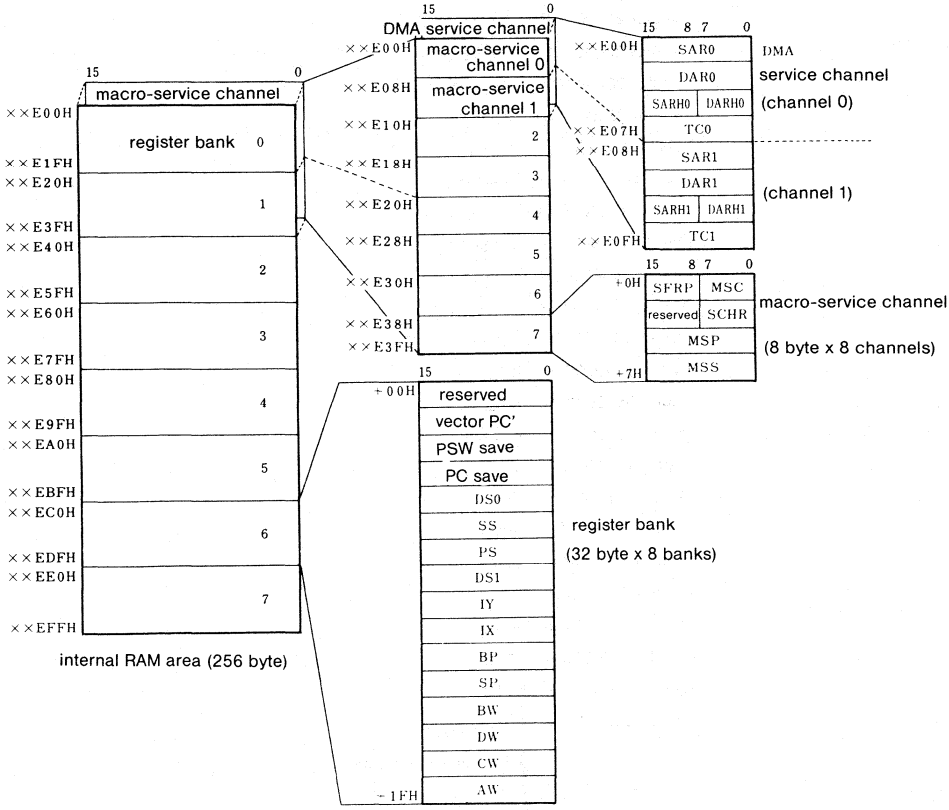
2.4.4 Internal RAM Areas

256 byte RAM is stored in XXE00H-XXEFFH (XX is the value designated by IDB)
The internal RAM is accessed by 16-bit units which enable high-speed processing.

8 register banks are assigned to the internal RAM. The macro-service channel and the DMA service are also overlapped and assigned.

The internal RAM makes it possible to disable memory access by resetting (0) bit 6 (RAMEN) of processor control register (PRC). It is also impossible to carry out program fetch from the internal RAM. When memory access has been disabled no access other than access as register is possible.

Fig. 2-5 Internal RAM Area Map



2.4.5 Vector Table Areas

In the 1 kilobyte area of 00000H-003FFH, the interrupt requests and the interrupt routine starting addresses corresponding to the break instructions are retained by the 256 vector portion (using 4 bytes for each vector)

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 19 (0004CH) : reserved
vector 20 (00050H) : input/output instruction
vector 21 (00054H) : reserved
vector 27 (0006CH) : reserved
vector 28 (00070H) : INTSER0
vector 29 (00074H) : INTSR0
vector 30 (00078H) : INTST0
vector 31 (0007CH) : reserved
vector 32 (00080H) : INTSER1
vector 33 (00084H) : INTSR1
vector 34 (00088H) : INTST1
vector 35 (0008CH) : reserved
vector 36 (00090H) : INTD0
vector 37 (00094H) : INTD1
vector 38 (00098H) : reserved
vector 39 (0009CH) : reserved
vector 40 (000A0H) : INTP0
vector 41 (000A4H) : INTP1
vector 42 (000A8H) : INTP2
vector 43 (000ACH) : reserved
vector 44 (000B0H) : INTTUO I
vector 45 (000B4H) : INTTU1
vector 46 (000B8H) : INTTU2
vector 47 (000BCH) : INTTB
vector 48 (000C0H) : user area
BRK imm8 instruction
vector 255 (003FCH) : INT input

In vectors 0-47, the interrupt vectors are designated (part of reserved area) and are not available for general use.

In vectors 48-255, they are for general use and can be used with 2 byte break instructions and the INT input. In the unused portions, they can be available for uses other than vectors.

The vectors are comprised of 4 bytes and the higher 2 bytes are loaded to the program segment PS while the lower 2 bytes are loaded to the program counter PC.

Example Vector 0

000H	001H
002H	003H

PC · (000H. 001H)

PS· (002H. 003H)

2.4.6 External Memory Areas

The μPD70322 can expand the external memory (ROM, RAM, and others) to the 00000H-FBFFFH areas.

The μPD70320 connects external memory (ROM, RAM, and others) to the 00000H-FFFFEH areas. However, the FFF00H-FFFEFH and the FFFFCH-FFFFEH areas are reserved.

The external memory is accessed by using address bus (A0-A19), data bus (D0-D7), and the $\overline{MREQ}$, $\overline{MSTB}$, and R/$\overline{W}$ signals. It also provides a refresh pulse output terminal (REFRQ) for pseudo-static memory refresh use. A pseudo-static memory can easily be connected, due to a function for automatic outputting of refresh address for dynamic memory refresh use; also the dynamic memory may be easily connected. It is also possible to insert wait cycles during the memory cycles in 128 kilobyte steps using software (See 4.1).

2.4.7 Internal ROM Areas

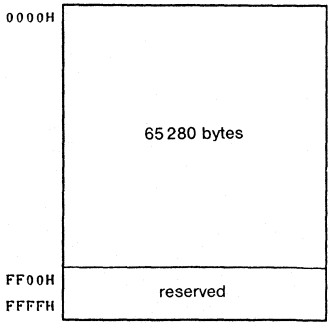
The μPD70322 has an internal mask ROM in the FC000H-FFFFFH areas. However, the FFF00H-FFFEFH area is used for testing internally and is not available for general use. The 4 byte FFFFCH-FFFFH are reserved. As a result, total 16140bytes can be utilized as a ROM area.

The internal ROM has an exclusive bus between the instruction queue so that the external memory space can carry out prefetch separately and rapidly which makes possible rapid instruction execution (Prefetch is possible with one clock, while external memories require a minimum of two clocks).

2.5 I/O Space

The μPD70322/70320 has a 64 kilobyte I/O space in addition to a 1 megabyte memory space. Fig. 2-6 shows a map of I/O space. The I/O space is accessed by using address bus (A0-A15), data bus (D0-D7) and $\overline{IOSTB}$, R/$\overline{W}$, $\overline{DMAAK0}$, and $\overline{DMAAK1}$ signals. 0 is output from the unused address bus's higher 4 bit (A16-A19). Insertion of wait cycles in the I/O cycle is software-specified.

Fig. 2-6 I/O Map (64 kilobyte)



3. INTERRUPTS

3.1 Interrupt Controller

The μPD70322/70320 has a high performance interrupt controller which controls multiprocessing of interrupts arising from 17 possible sources. The 17 interrupt sources in this interrupt controller are divided into a group of five external and 12 internal sources for control which carry out programmable multiprocessing control in groups. It is also possible to select from three types of response methods according to the characteristics of the interrupt sources: vector interrupt function, register bank switching function, and macro-service function.

External interrupts can be easily expanded by connecting the interrupt controllers like the μPD71059 and others.

Instructions of the interrupt controller are defjned by the interrupt control register which is provided for each interrupt source and by the macro-service control register.

EI and DI instructions are for all the interrupts, RETI and RETRBI instructions are for return from interrupt, and FINT instruction is used to indicate that interrupt processing for interrupt controller is completed.

3.2 Interrupt Sources

The μPD70322/70320 has 5 external and 12 internal sources. The 17 interrupt sources are divided into eight groups and are managed by the interrupt controller. The configuration inside this group is fixed by hardware. For the 8 groups of interrupts priority from 0-7 (0 being the highest), excluding NMI and INTR, and 5 groups of interrupts except INTTB can be arbitrarily set using software. The function supported by the interrupt controller differs according to interrupt source.

Interrupt sources are listed in Table 3-1.

Table 3-1 Interrupt Sources

Interrupt Source	Internal/ External	vector	macro-service	bank-switching	priority			multi-processing control
					register	between groups	inside group	
NMI (Non Maskable Interrupt)	E	2	none	none	no	0	–	no
INTR (INTerrupt Request)	E	external input	none	none	no	7	–	no
INTTU0 (INTerrupt from Timer Unit0)	I	44	yes	yes	yes	1	1	yes
INTTU1 (INTerrupt from Timer Unit1)		45					2	
INTTU2 (INTerrupt from Timer Unit2)		46					3	
INTD0 (INTerrupt from DMA channel0)	I	36	no	yes	yes	2	1	yes
INTD1 (INTerrupt from DMA channel1)		37					2	
INTP0 (INTerrupt from Peripheral 0)	E	40	yes	yes	yes	3	1	yes
INTP1 (INTerrupt from Peripheral 1)		41					2	
INTP2 (INTerrupt from Peripheral 2)		42					3	
INTSER0 (INTerrupt from Serial ERror of channel0)	I	28	no	yes	yes	4	1	yes
INTSR0 (INTerrupt from Serial Receiver of channel0)		29	yes				2	
INTST0 (INTerrupt from Serial Transmitter of channel0)		30	yes				3	
INTSER1 (INTerrupt from Serial ERror of channel1)	I	32	no	yes	yes	5	1	yes
INTSR1 (INTerrupt from Serial Receiver of channel1)		33	yes				2	
INTST1 (INTerrupt from Serial Transmitter of channel1)		34	yes				3	
INTTB (INTerrupt from Time Base counter)	I	47	no	no	no (fixed at 7)	6	–	yes

3.3 Interrupt Controller Functions

The interrupt controller regulates the priority among interrupts when interrupts with same priority occur simultaneously.

3.3.1 Multi-interrupt Priority Control

Multi-interrupt priority control is carried out in group units for interrupt excluding interrupt response using NMI and INTR, as well as macro-service.

Interrupt multiprocessing control is carried out in EI condition. As a result, when allowing multiprocessing it is necessary to change to EI condition during interrupt processing routine. In multiprocessing control, if interrupt requests with a higher priority than the interrupt being processed are received, the interrupt being processed is discontinued, and processing of interrupts with higher priority is carried out. If the priority is below the priority of the interrupt being processed, that interrupt is held. If, for the interrupt held the interrupt mask bit of the interrupt control register (provided for each interrupt source) is not set during the interrupt processing routine being executed and if the interrupt request flag is not reset, the interrupt being held will be accepted at the end of the current interrupt.

In interrupt response exclusive of NMI, INTR and software interrupt (incl. trap), it is necessary to execute the FINT instruction in order to indicate to the interrupt controller that the interrupt processing routine has been completed at the very last part of the interrupt processing routine. If this instruction is not executed, all succeeding interrupts will be received as having a priority not higher than the interrupt for which the FINT instruction has not been executed. The FINT is not necessary at the end of the NMI and INTR service routine. Interrupt response using NMI and INTR as well as macro-service functions do not contain multiprocessing control so that it can be received if it is in an enable mode (always for NMI).

The eight priority levels from 0 to 7 (0 has the highest priority) can be set up arbitrarily for each interrupt group. The priority simultaneously indicates the number of switching destination register banks when using the register bank switching function which will be described later. Priority is established by using the three bits PR0-2 in the interrupt control register which is provided for each maskable interrupt source. However, when setting this up, only the interrupt control registers of the interrupt sources which have the highest default priority inside the interrupt groups can be programmed, the others are ignored and use the default values inside the group.

3.3.2 Priority Control during Simultaneous Generation of Interrupts

NMI is the highest and INTR is the lowest of the priorities possible during simultaneous generation of interrupts. Priority exclusive of NMI and INTR is exactly the same as for the priority of multi-interrupts. Among the groups with the same priority, it complies with the priorities fixed by hardware (default priority). Even in an identical group, it complies in exactly the same way with the priority inside the group.

3.4 Interrupt Response Method

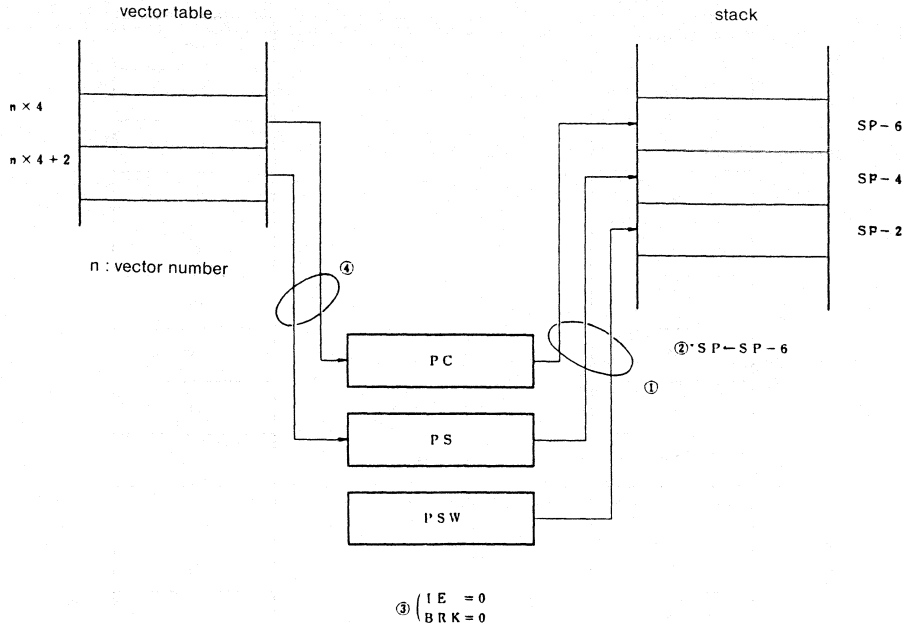
The μPD70322/70320 has three types of interrupt response method: vector interrupt function, bank switching function, and macro-service function. All of these functions can be selected to fit the purpose of the interrupt. The interrupt controller reacts to the interrupt requests according to the response method set up by the interrupt control register.

When receiving an interrupt using a vector interrupt function and a register bank switching function, the contents of PC, PS, and PSW are saved using the method applied to that functions. After PSW has been saved, each BRK flag is reset. As a result, single step interrupt and interrupt exclusive of interrupt response using NMI and macro-service are disabled (software interrupt exclusive of single step interrupt takes place) (See 3.9).

3.4.1 Vector Interrupt

When interrupt is received using vector interrupt, present contents of PSW as well as PC and PS contents are saved to the stack, a vector is selected from the vector table and is executed as an interrupt processing routine from the address indicated by the vector. All vectors are fixed except the INTR vector. When working with an INTR interrupt, an acknowledge cycle takes place and an interrupt vector is taken from the data bus (See 3.6 INTR). Interrupt vectors exclusive of INTR are indicated in Table 3-1. Return from interrupt is carried out by RETI instruction, however, when carrying out return from interrupts exclusive of NMI and INT, it is necessary to execute the FINT instruction. When carrying out return from interrupt, PC, PS, and PSW are returned from the stack.

Fig. 3-1 Operations for Interrupt Receive carried out in 1-4 order



3.4.2 Register Bank Switching Function

In the μPD70322/70320, the general purpose register sets are mapped in internal RAM and can contain a maximum of 8 register banks. These register banks are switched automatically during interrupt response and it is not necessary to carry out save processing to the stack of register groups which until now have been carried out using software and so it is now possible to respond to interrupt requests very fast.

When using register bank switching function, the ENCS bit of the interrupt control register which has been provided for each maskable interrupt source is set (1). One register bank can be designated for each interrupt group and it has the same number like the multiprocessing priority and is designated by PR0-2 of the interrupt control register.

The register bank switching sequence is carried out in the following way (Fig. 3-2)

(1) PSW contents are saved to a temporary register.
(2) register bank is switched.
(3) IE = 0, BRK = 0.
(4) the PC contents and the PSW in the temporary register are saved respectively to the save areas of the new register bank
(5) the offset of the start address of the interrupt processing routine is loaded from the vector PC area to PC.

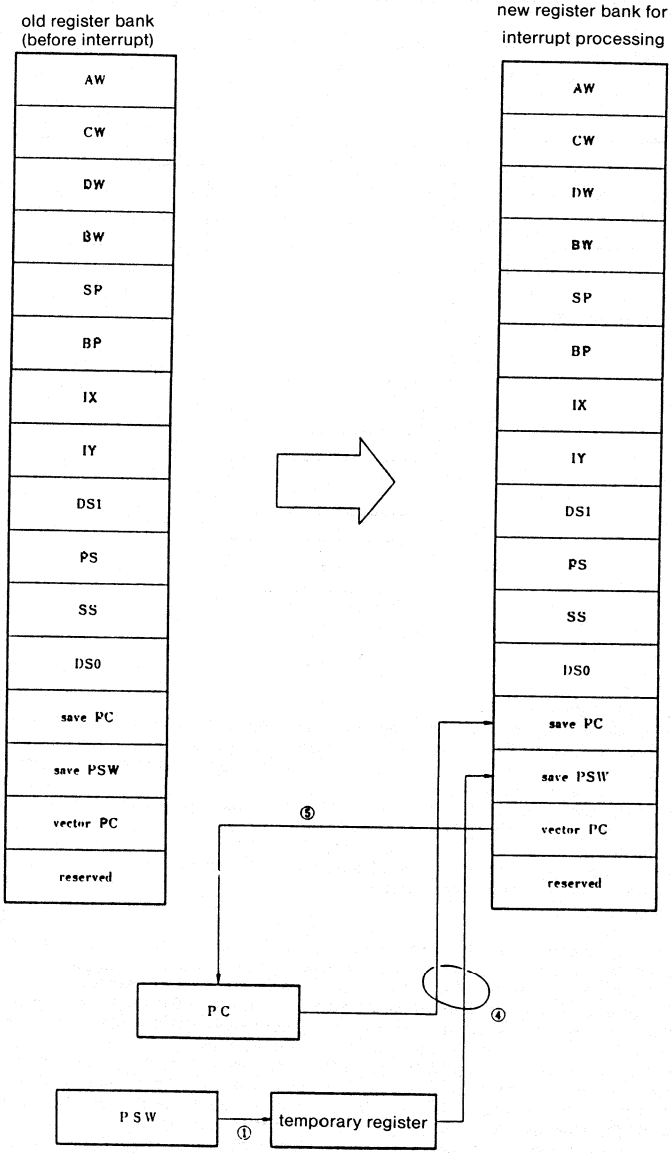
The register banks are thus switched and the interrupt processing routine is executed.

Return from register bank switching interrupt is carried out by executing RETRBI instruction after executing FINT instruction (the use of register bank switching function is limited to receiving of maskable interrupts). When RETRBI instruction is executed, PC and PSW are respectively reloaded from the save areas of the register banks as indicated in Fig. 3-3. (recovery of register banks is not carried out using RETI instruction so that return to main program can normally not be executed

When using the register bank switching function, it is necessary to initialize beforehand the PS from the register bank of the switching destination, the vectors PC, SS, and SP. The other registers should be initialized as needed. However, care must be taken with PC when modifying during the interrupt processing routine.

The register bank switching function can be used only for one interrupt in each interrupt group with the same priority (See 3.7 (1) IF).

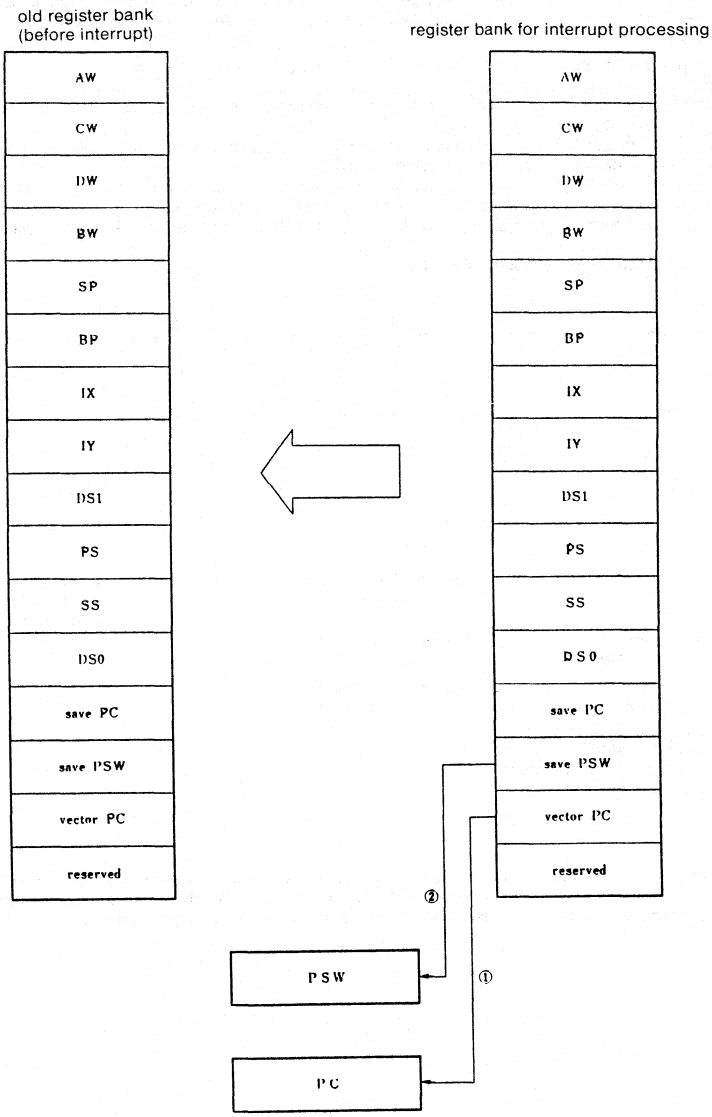
Fig. 3-2 Register bank switching sequence



(2) register bank switching

(3) IE-0, BRK=0

Fig. 3-3 Register bank return sequence



The macro-service function is a function which carries out data transfer between special function register areas and the memory space depending on the interrupt request. The function makes is possible to reduce the overhead (operations for save, initialization, and return of registers) on interrupt processing making it unnecessary to carry out simple processing of simple data transmission by interrupt processing using software. It is also unnecessary to execute program when processing with macro-service and it is now possible to process a portion of data with effective programming results which have traditionally been processed in 1 byte units using software. The macro-service function differs from other interrupt response modes in that the IMK bit (interrupt mask bit) of the interrupt control register which has been provided for every interrupt source is reset and macro service will be operated if the MS/INT bit (macro-service enable bit) is set whether it is in EI or in DI condition (See 3.7).

There are two types of operational mode for the macro-service function as follows:

(1) Normal mode

a pre-established number of data transfers are carried out, one byte or one word for every interrupt request occurrence.

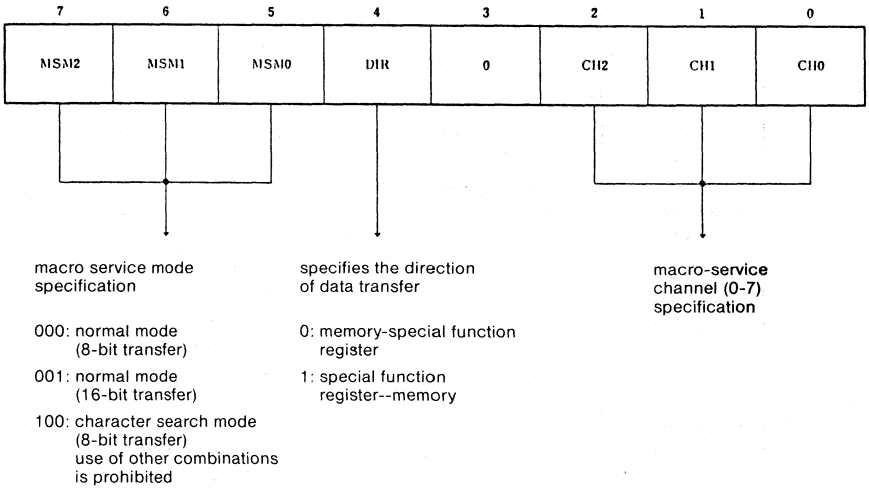
(2) Character search mode

One byte of data transfer is carried out for each interrupt request occurrence until a pre-established number of bytes has been transferred or the data coincide with pre-established 8-bit data.

The macro-service function is controlled by macro-service channels specified by macro-service control registers; a macro-service control register is provided for each interrupt source for which macro-service is possible.

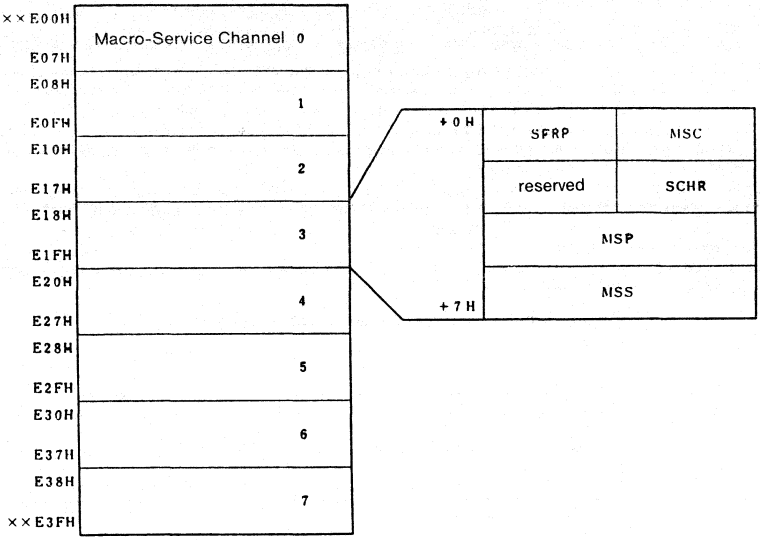
The macro-service control register is configured as indicated in Fig. 3-4 and are within the special function register area.

Fig. 3-4 Format of Macro-Service Control Register



The macro-service channel is assigned to the XXE00H-E3FH (XX is value designated by IDB) of internal RAM, as shown in Figure 3-5. The macro-service channel is used to define the transfer destination, transfer source, number of transfers, search character for the macro-service data and one can use a maximum of eight channels.

Fig. 3-5 Configuration of Macro-Service Channel



MSC (+0H): Number of transfers carried out by macro-service.

SFRP (+1H): Offset value of special function register address, XXF00H+SFRP (XX is specified by IDB) is special function register address.

SCHR (+2H): 8-bit data compared during character search mode

MSP (+4H): Offset value of memory address which is object of macro service data transfer

MSS (+6H): value of memory address segment which is object of data transfer in macro-service. The memory address which is the object of data transfer is MSS x 16 +MSP.

The MSC of the macro-service channel is decremented (–1) after each data transfer (8-bit/16-bit), MSP is incremented (+1). Afterwards the interrupt request flags are cleared unless when MSC is 0 or when the transferred data is equal with the search data (only during character search mode), the interrupt request flags force an interrupt by not being cleared.

3.5 NMI (Non-Maskable Interrupt)

The NMI is the highest priority interrupt which cannot be disabled. This interrupt is edge-detected. The direction of the edge is selected by the special function register INTM register bit 0 the ESNMI bit. When ESNMI bit is 0 and when the completion edge is 1, interrupt is generated by the starting edge. This interrupt is capable of vector response only and the vector type is fixed at 2. This input is used in conjunction with terminal P10 and the level can be checked by reading P10. When NMI is received it causes the DI condition and disables other interrupts.

3.6 INTR (Interrupt)

INTR is a maskable interrupt and the interrupt is detected by level (active high). INTR does not receive multiprocessing control by using interrupt controller and if it is an interrupt enable condition (IE = 1) it can be received at any time. However, its priority when there is a simultaneous generation of interrupts is the lowest. The INTR is capable of vector response and the vector address is supplied from the data bus by the interrupt acknowledge cycle. The interrupt acknowledge cycle is defined via INTAK ouput. The INTR terminal is used in conjunction with P14 and POLL and is selected by bit 4 of the special function register port 1 mode control register (PMC 1). As a result, interrupt does not take place even in interrupt enable condition (IE = 1) when the INTR function is not selected. INTAK is used in conjunction with P13 and INTP2 and the function is selected using PMC1 bit 3.

The external interrupt input can be expanded to a maximum of 64 by connecting the μPD71059 interrupt controller. When the interrupt is received, it causes the interrupt disable condition (IE = 0)

3.7 Interrupts other than NMI and INTR

The interrupts other than NMI and INTR receives multiprocessing control using the interrupt controller. When the interrupt is accepted, the interrupt is automatically set in disable condition (IE = 0). However, when an interrupt with a priority higher than that of the interrupt being processed is generated, that interrupt can be accepted by setting it in an interrupt disable condition during interrupt processing routine. When an interrupt is generated with lower or with same priority, the interrupt is held over.

The 15 interrupt sources are divided into six priority groups. It is possible to set up arbitrarily 8 levels from 0 to 7 (0 being the highest) of priorities for each group. However, the priority for INTTB (Time Base Counter Interrupt) is fixed at level 7 by hardware. These priority levels also express the numbers of switching destination banks when using the bank switching function. The priorities are initialized at level 7 by resetting.

When interrupts are generated simultaneously, the interrupt in a group established with higher priority is accepted. When interrupts which have been set up on the same level are generated simultaneously, the one with the highest priority among the groups which are fixed by hardware and software and inside the same group with the highest priority inside that group which is fixed by hardware, is accepted.

Each interrupt source has a register for interrupt control used inside the special function registers. The bit configuration of this control register is indicated in Fig. 3-6.

Fig. 3-6 Format of Interrupt Request Control Register

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

(1) IF (Interrupt Flag)

Flag indicates that there is an interrupt request. It indicates that there is an interrupt request and indicates that it is not served. This flag is set by the generation of the interrupt requests and is reset by interrupt acceptance, by BTCLR instruction (an instruction additional to the μPD70108/70116), and by other instructions.

(2) IMK (Interrupt Mask)

A bit which sets up interrupt mask. 1 indicates that interrupt is masked, 0 indicates that mask has been released.

(3) MS/INT (Macro-Service/Interrupt)

This is a bit which specifies whether an interrupt response is processed by macro-service or by vector interrupt or register bank switching function; 1 is used for macro-service function, 0 for vector interrupt or register bank switching function.

(4) ENCS (Enable Context Switching)

This is a bit which specifies whether the register bank switching function is used or not; 1 indicates that register bank switching function is used; 0 indicates that vector interrupt is used.

(5) PRO-2 (Priority 0-2)

This are the bits which indicate the priority of the interrupt group with specifications from 0 through 7. This specification is possible only for the interrupt registers which have the highest priority within the group and specification by other interrupt control registers is invalid. (During Read, 7 is fixed).

These priorities indicate the number of the register banks for switching destination within the register bank switching function.

3.8 External Interrupt

There are five external interrupt sources. Among these INTR is detected by level and all others are detected by edge. For the interrupts which are detected by edge exclusive of INTR the respective effective edges are designated by the external interrupt mode register (INTM) of the special function registers.

Fig. 3-7 Format of External Interrupt Mode Register (INTM)

Symbol	7	6	5	4	3	2	1	0	Address
INTM	0	ES2	0	ES1	0	ES0	0	ESNMI	××F40H

ESNMI: Designation of effective edge for NMI input

ESO-2: Designation of effective edge of iNTPO--2 input

Effective edge: 0: falling edge

1: rising edge

3.9 Software-Activated Interrupts

The μPD70322/70320 has a total of eight types of interrupts using software (Table 3-2). Six types are compatible with interrupts for μPD70108/70116 software (there is no interrupt for emulation mode, however). The other two types of interrupts have a special function for the μPD70322/70320.

The vectors for these interrupts are predefined.

When conditions for generation of that interrupt are realized – exclusive of BRK interrupt (single step interrupt) – it is accepted as usual (it has a greater priority than hardware interrupt). However, the BRK flag interrupt is generated when BRK = 1 (with no distinction made for hardware or software) and when the interrupt is accepted the BRK flag is automatically reset so that it has a lower priority than the other interrupts (for both hardware and software) and the BRK flag interrupt is not generated during interrupt processing.

Table 3-2 Software Interrupt

interrupt source	vector	priority
DIVU divide error	0	1
DIV divide error		
CHKIND boundary overflow	5	
BRKV	4	
BRK 3	3	
BRK imm8	48-255	
BRK flag (single step)	1	2
input/output instruction (IBRK flag)	20	1
FPO instruction	7	

3.9.1 General Software Interrupts

The execution sequence for receiving of software interrupt exclusive of input/output instruction interrupt and FPO instruction interrupt is identical to that of the vector interrupt. In other words, the address information for the following instruction (PC) and PSW are saved to the stack, IE = BRK = 0, and vector contents are loaded to PS and PC.

Each software interrupt is described as follows:

(1) DIVU divide error, DIV divide error.
Always occurs when a quotient overflow occurs due to the execution of a division instruction.

(2) CHKIND boundary overflow.
Takes place when it is determined whether the index value has exceeded the boundary by executing instruction (CHKIND) which checks to see if the index value has exceeded the boundary of predefined arrays.

(3) BRKV
Occurs when V (overflow flag) is set during execution of BRKV instruction.

(4) BRK3
Occurs with execution of BRK3 instruction.

(5) BRK imm
Occurs with execution of BRK imm instruction.

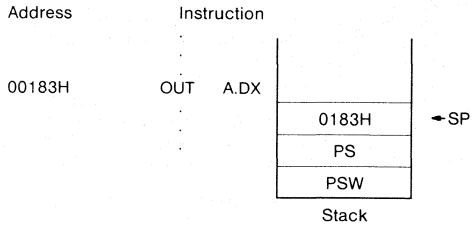
(6) BRK flag (single step)
When BRK is set at 1, occurs every time one instruction is executed.

3.9.2 Input/Output Instruction Interrupt

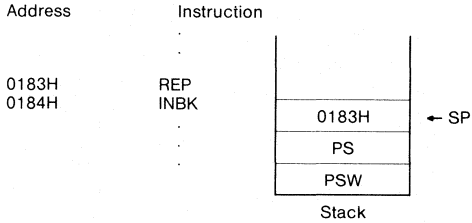
When IBRK = 0, interrupt takes place when an input/output instruction is attempted. The address information which is saved to stack when this interrupt is accepted differs from ordinary interrupt using software (see 3.91) in that it goes to the address where the input/output instructions are located. When the prefix is added to that input/output instruction, it goes to the address where the prefix is located. The other operations are the same as for ordinary interrupt using software. When returning from input/output instruction interrupt, it is necessary to adjust the PC value in the stack in order to return to normal. It is possible to use software to find out exactly which instructions have been executed to cause interrupt generation by making the address information which has been saved to stack the lead address. This function facilitates the transplantation of programs which have previously been used with the μPD70108/70116.

The contents of PSW are saved to stack immediately before interrupt has taken place and afterwards the flags are set automatically so that IE=BRK=0 and IBRK=1. IBRK is set to 1 so that the input/output instruction/during interrupt processing are executed as input/output instructions and it is automatically returned to the original condition (IBRK=0) by return from interrupt

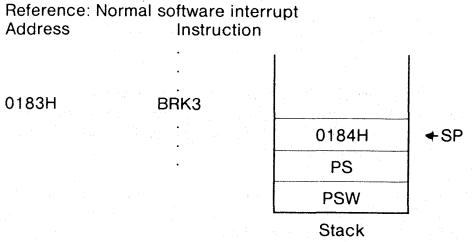
Example 1: I/O instruction without prefix



Example 2: I/O instruction with prefix



Reference: Normal software interrupt



3.9.3 FPO Instruction Interrupt

The external bus configuration of the μPD70322/70320 differs from that of the μPD70108/70116 in that the coprocessor for use in floating point operations can not be connected. As a result, when the use of the FPO instruction is attempted with this coprocessor, an interrupt is generated for the purpose of emulating the operation of the instruction. The PC value of this interrupt which is saved to stack becomes the starting address (see 3.9.2 for input/output instruction interrupt) (the prefix lead address when the prefix has been attached). As a result, the FPO instruction is decoded and software emulation is possible. It is necessary to adjust the PC value which has been saved to stack when returning from FPO instruction interrupt just as with the input/output instruction interrupt.

4. Bus Control

The μPD70322/70320 has bus control pins as shown in Table 4.
When using a multi-function pin, it is necessary to select the desired function by means of the port mode control register (PMCn).

Chart 4-1 Pin Functions for Bus Control

Name of Pin	Input/ Output	Function	Comments
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	strobe signal for memory read/memory write	
$\overline{IOSTB}$	Output	strobe signal for I/O cycle	
$\overline{REFRQ}$	Output	indicates memory refresh cycle	
HLDRQ	Input	bus hold request signal	for use with P27
$\overline{HILDAK}$	Output	bus hold acknowledgement signal	use with P26
$\overline{DMAAK0}$	Output	indicates DMA acknowledgement cycle	for use with P21
$\overline{DMAAK1}$	Output	indicates DMA acknowledgement cycle	for use with P24
READY	Input	insert wait in external bus cycle	for use with P17
$\overline{INTAK}$	Output	indicates interrupt acknowledgement cycle	for use with P13 and INTP2
$\overline{POLL}$	Input	polling input	for use with P14 and INTR

4.1 Programmable Wait Function

The μPD70322/70320 insertion of wait state during bus cycle (exclusive of the memory refresh cycle) can be specified by software. It specifies a 1 megabyte memory space in 8 units of 128 kilobyte and I/O space using the wait control register (WTC) as shown in Fig. 4-1. However, memory space block 6 (C000H-DFFFH) and block 7 (E000H-FFFFFH) and are set up in the same way.

The wait state specification can easily be programmed independent for each block with one out of four possibilities using READY pin with 0,1, 2 and more cycles, as indicated in Chart 4-2. When using READY pin control, the READY pin is used in conjunction with P17 so that bit 7 of port 1 mode control register (PMC1) must be set at 1. When bit 7 of PMC1 is 0, READY condition or wait state always goes to 2 state. If control by READY terminal is selected 2 wait states are inserted regardless of the READY pin condition. The READY pin is level-triggered and in case of a low level wait states are inserted.

Accessing of internal ROM (μPD70322 only) and internal data areas is not influenced by the programmable wait functions. This set-up applies to access of everything in the external areas with the exception of refresh time.

The WTC register is initialized to FFFFH at reset time.

Fig. 4-1 Format of Wait control Register (WTC)

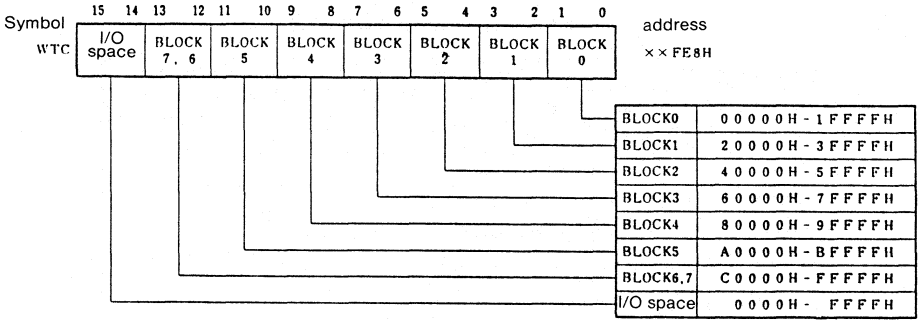


Table 4-2 Designation of wait State

BLOCKn/I/O space	Wait state
0 0	0 state
0 1	1 state
1 0	2 state
1 1	2 state + READY pin

4.2 Bus Hold Function

The μPD70322/70320 has a bus hold function. Input of external high level to HLDRQ terminal indicates that the external element wants to use the bus. When the μPD70322/70320 detects that the HLDRQ terminal is high level, it puts all of the A0-A19, D0-7, MREO, MSTB, and IOSTB outputs on high impedance, puts the HLDAK terminal on low level, and indicates that the external elements have been opened to the buses and switches to the hold mode. During the hold mode, the μPD70322/70320 stops execution of instructions and reception of prefetch data. Only the on-chip peripheral hardware which does not use a bus is operated. When the HLDRQ pin is checked and a low level is detected during hold mode, the HLDAK signal is placed on high level, this is an indication that the bus is not opened any more to the external elements, and execute is restarted after a one clock interval.

Even during HALT mode (one type of standby function: see 11.2), the bus hold requirement can be received and when the hold mode is released (if the HLDRQ signal is low level), it returns to HALT mode. The hold mode conditions is the same as normal mode.

During execution of one instruction following BUSLOCK prefix and during interrupt acknowledge operation, bus hold requests are not accepted.

The μPD70322/70320 can execute insertion of memory refresh cycle during hold mode and it is executed by setting refresh mode (RFM) register HLDRF (bit 6). The HLDAK signal is forced to a high level for each refresh timing and the refresh cycle is carried out after confirming that HLDRQ has gone to low level. Afterwards, if the HLDRQ signal reaches high level, it again shifts to the hold mode. If the HLDRQ signal remains at low level, the hold mode is released and instruction execution is restarted. Since HLDRQ pin is combined with P27 and HLDAK with P26, to use the bus hold function it is necessary to set bits 6 and 7 of the PORT 2 mode control register (PMC2) to 1.

4.3 Refresh Function

The μPD70322/70320 has a number of functions for refreshing of DRAM and the pseudo-SRAM. There are functions for insertion of refresh cycle on a regular basis for a series of bus cycles, for outputting of refresh address for the support of the DRAM and pseudo-SRAM power down self refresh mode, a function which generates a refresh cycle during HALT mode and a function for the insertion of wait state during refresh cycle.

4.3.1 Refresh Mode Register (RFM)

The RFM register is an 8-bit register which enables refresh function control. It can be accessed with 8/1-bit Read/Write operations using memory access.

The RFM register is initialized at FCH during reset.

The RFM register has the following bit functions and configuration:

address	7	6	5	4	3	2	1	0	symbol
RFM	RFLV	HLDRF	HLTRF	RFEN	RFW1	RFW0	RFT1	RFT0	××FE1H

RFT0 RFT1 are bits which specify refresh synchronization.

Refresh synchronization is selected from time base counter (see 7.1) output taps 3-6. Refresh cycle is generated at synchronous intervals as shown in Table 4-3.

Table 4-3 Refresh Synchronization

f_{CLK} = 5 MHZ (= ½ fx; fx = 10MHz)

RFT 1	RFT 0	refresh cycle
0	0	$2^4/f_{CLK}$ (3.2μs)
0	1	$2^5/f_{CLK}$ (6.4μs)
1	0	$2^6/f_{CLK}$ (12.8μs)
1	1	$2^7/f_{CLK}$ (25.6μs)

RFW0 RFW1-- are bits which specify the number of wait states to be inserted during refresh cycle.

The number of wait states during refresh cycle is defined by designation of RFW0, 1 independently of previously described programmable wait function (see 4.1) as shown in Table 4-4.

Table 4-4 Wait State during Refresh Cycle

RFW 1	RFW 0	wait state
0	0	0 state
0	1	1 state
1	0	2 state
1	1	2 state

RFEN is a bit which enables automatic insertion of refresh cycles

When it is 1, it permits automatic insertion of refresh cycles, when it is 0, it disables automatic insertion of refresh cycles, REFRQ pin output is controlled by RFLV bit contents (for further details, see description of RFLV bit).

HLTRF is a bit which enables automatic insertion of refresh cycles during HALT mode.

1 indicates enabling of automatic insertion, and 0 indicates disable. However, when RFEN bit=0, it is disabled regardles of the HLTRF bit contents.

HLDRF is a bit which enables automatic insertion of refresh cycles during hold mode.

1 indicates enable, 0 indicates disable. When in enable condition (1), it is forced to start HLDAK output at high level for each refresh timing, and inserts refresh cycle automatically.

RFLV is a bit which defines the output level for REFRQ signal.

Fig. 4-2 indicates the circuit configuration. Output is determined logically as in Chart 4-5.

The RFLV bit becomes master RFLV output at read time and is written for RFLV slave. Writing of master RFLV is carried out when refresh timing takes places.

Use of the RFLV bit enables support of power down self refresh mode for pseudo-SRAM.

Fig. 4-2 Control Circuitry Using RFLV Bit

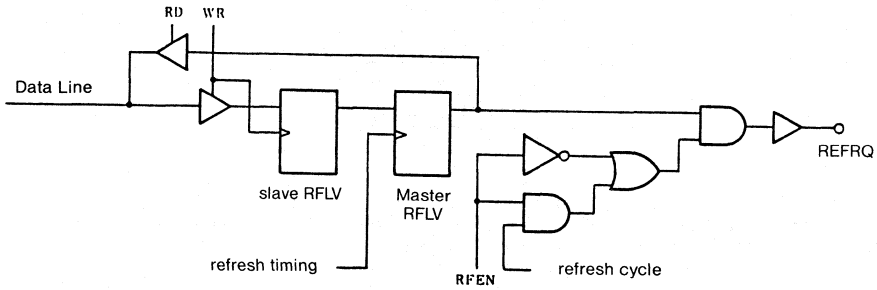


Table 4-5 Output Level for REFRQ Signal

RFLV	RFEN	REFRQ condition
0	0	0
0	1	0
1	0	1
1	1	refresh pulse output

Insertion of refresh cycle is carried out when RFEN bit goes to 1. At this time, MREQ, MSTB, and IOSTB go to high level, refresh address is output to A0-A8 and the high level is output to A9-A19, and refresh pulse is output from REFRQ pin.

Care must be taken when using the bit operation instructions as the RFLV bit does not go to read data up to the following refresh timing even with write.

4.4 Bus Usage Privileges

The priority for bus usage privileges for the μPD70322/70320 is as follows:

(1) Refresh cycle (see 4.3)

The refresh cycle will always take place if insertion of the refresh cycle is enabled. However, if insertion of refresh cycle at time of hold mode is enabled during hold mode, HLDAK signal is forced to high level and refresh cycle is carried out while waiting for HLDRQ signal to go to low level.

(2) Hold mode (see 4.2)

The system goes into the hold mode except during execution of one instruction following BUSLOCK prefix and during interrupt acknowledgement cycle.

(3) DMA cycle (see 5)

(4) Normal bus cycle

However, other requests are temporarily retained in the following cases:

- During execution of interrupt acknowledgement cycle and processing related to it.
- During execution of instructions with BUSLOCK prefix. Bus will not operate during stop mode. (See 11, Chart 11-2 for bus conditions).

4.5 Bus Timing

Figs. 4-3 through 4-10 show principal bus timings (exept for DMA)

Fig. 4-3 Memory Read Cycle

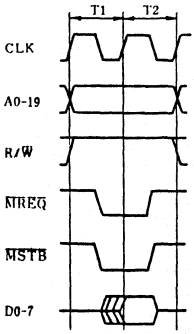


Fig. 4-4 Memory Write Cycle

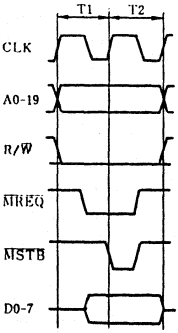


Fig. 4-5 I/O Read Cycle

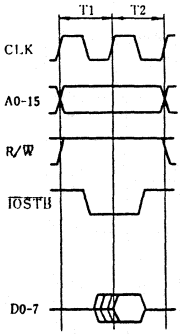


Fig. 4-6 I/O Write Cycle

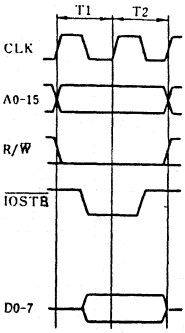


Fig. 4-7 Memory Read Cycle (During insertion of 1 wait state)

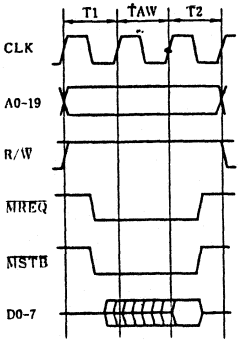


Fig. 4-8 Refresh Cycle (During 2 Wait State)

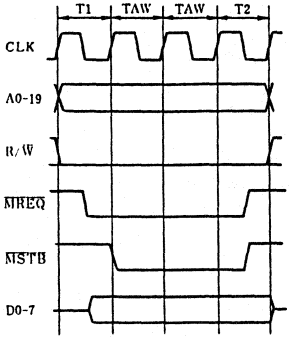


Fig. 4-9 Memory Write Cycle (During operation with READY terminal)

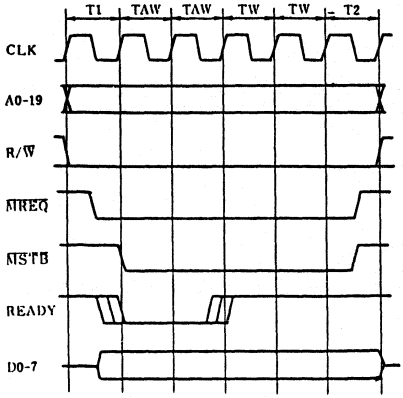


Fig. 4-10 Refresh Cycle (During insertion of 1 wait state)

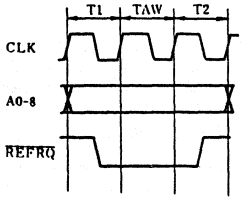


Fig. 4-11 Bus Hold Accept and Release Timing

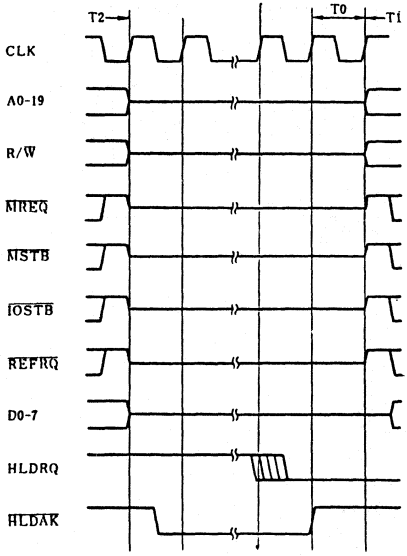


Fig. 4-12 Refresh Cycle during Hold Mode (0 number of wait states)

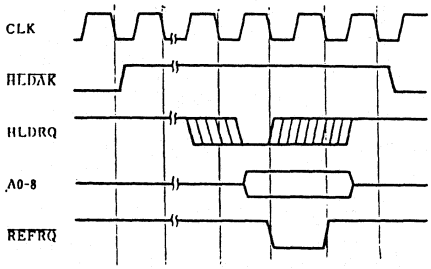
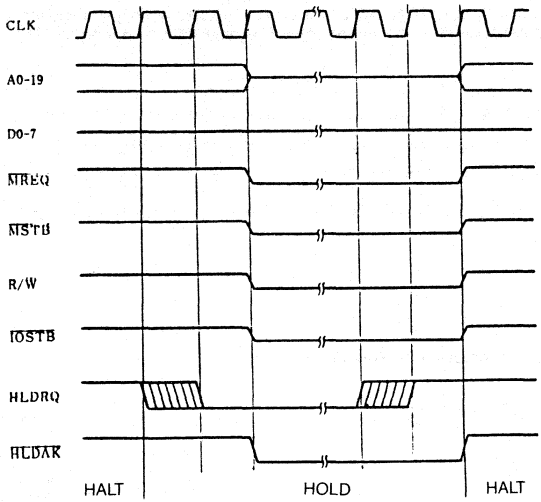


Fig. 4-13 HOLD Accept and Release during HALT Mode



5 DMA Controller

There is a built-in two-channel DMA controller which can directly specify 1 megabyte memory space in the μPD70322/70320.

5.1 Pin Functions

The DMA controller provides pins with the following functions. These pins can all be used in conjunction with the port; therefore it is necessary to put the bit of the corresponding port 2 mode control register (PMC2) during use to 1.

(1) DMARQ 0, DMARQ 1 (P20, P23).

Active high DMA request input pin.

(2) DMAAK 0, DMAAK 1 (P21, P24).

An active low DMA response signal pin. However, there is no output during memory-to-memory DMA transfer (burst mode, single step mode).

(3) TC 0, TC 1 (P22, P25).

An active low DMA completion signal output terminal. It is output when TC0 or TC1 of the DMA service channel are 0.

5.2 DMA Operation

Thera are four types of DMA transmit mode in the μPD70322/70320. Functions of each transfer mode are indicated in Table 5-1.

Table 5-1 Transfer Mode Functions

Mode	Transfer type	Function	DMA start	DMA Stop	Interrupts	During HALT	DMA request during DMA operation
single step	memory to memory	alternately repeats execution of 1 instruction and 1 DMA transfer for a specified number of times only using 1 DMA request	○ DMA rising edge ○ setting TDMA bit of DMA control register	software	receive all	carries out specified number of DMA transfers	DMA channel 1 is either retained or interrupted and then carries out channel 0 DMA.
burst	memory to memory	carries out successively a specified number of DMA transfers using 1 DMA request	○ DMARY rising edge ○ setting of TDMA bit of DMA control register	NMI input only	can receive NMI only	carries out specified number of DMA transfers	New DMA'S are held until DMA transfer is finished
single transfer	memory to I/O	carries out one DMA transfer each time DMA requests are generated	○ DMARQ rising edge	software control	receive all	same as usual	DMA request is processed after current DMA transfer is completed
demand release	memory to I/O	carries out transfer during high level period of DMARQ pin	○ DMARQ high level	○ stopped at low level of DMARQ during DMA transfer ○ all others use software control	○ not accepted during DMA transfer ○ in all other cases, all interrupts accepted	same as usual	New DMAs are held until DMA transfer is finished

In memory-to-memory DMA transfer, DMAAK signal is not output. In memory-to-I/O DMA transfer, DMAAK signal is output for every 1 DMA cycle.

The programmable wait function (see 4.1) is effective even during DMA transfer. In memory-to-memory transfer the specified wait state is inserted at every transfer destination and transfer source. In memory-to-I/O transfer a wait state which is slow between memory and I/O is inserted so as to complete one transfer in one bus cycle.

The bus hold function and refresh function are effective even during DMA transfer and DMA transfer is interrupted by them.

All interrupts which were generated and could not be received during DMA transfer are reatained.

DMA transfer during HALT mode can be carried out if there are requests. When DMA transfer has ended, it returns to HALT mode. If DMA transfer end interrupt occurs when it returns to HALT mode, the HALT mode is released.

Channel 0 is given priority when DMA requests are generated at the same time.

When DMA transfer is ended (when a specific number of DMA transfers is executed), it can generate an interrupt.

5.3 DMA Control Registers

The DMA mode register and DMA control register help to program the specification of DMA transfer mode. The DMA service channels are mapped in internal RAM in order to specify transfer destination, transfer source and number of transfers. There are also registers for interrupt control and they are provided for each channel.

5.3.1 DMA Mode Registers (DMAM0, DMAM1)

These are bit registers which designate DMA transfer mode. The DMAMn register (n=0,1) can be accessed with 8/1-bit Read/Write operations by memory access.

	Symbol	7	6	5	4	3	2	1	0	address
(channel 0)	DMAM0	MD2	MD1	MD0	W	EDMA	TDMA	0	0	××FA1H
(channel 1)	DMAM1									××FA3H

MD2 MD1 MD0 are bits which specify transfer mode

MD 2	MD 1	MD 0	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	disable
1	0	0	burst mode
1	0	1	single transfer mode (I/O--Memory)
1	1	0	single transfer mode (Memory--I/O)
1	1	1	disable

W A bit which specifies whether transfer processing is to be carried out by byte or by word.

EDMA A bit which specifies enable or disable for DMA transfer.

1 indicates enable, 0 enable disable. This bit is automatically cleared (0) when the DMA service channel terminal counter (TC) is 0.

TDMA Transfer Start Bit

This is effective only with single step mode or burst mode. DMA is started up when 1 is written into this bit. (However, only when EDMA is set (1)). Read level for this bit is always 0. It is insignificant with demand release mode and single transfer mode.

Fig. 5-1 Format for DMA Mode Registers (DMAM0, DMAM1)

symbol	7	6	5	4	3	2	1	0	address
DMAM0 (channel 0)	MD2	MD1	MD0	W	EDMA	TDMA	0	0	××FA1H
DMAM1 (channel 1)									××FA3H

EDMA	TDMA	Transfer Start
0	0	DMA disable condition
1	0	Hold status
1	1	DMA transfer start

EDMA	DMA transfer enable/disable
0	DMA disable condition
1	DMA enable condition

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 releasemode(memory--I/O)
0	1	0	demandreleasemode(I/O--memory)
0	1	1	
1	0	0	burst mode
1	0	1	single transfer mode (I/O--memory)
1	1	0	single transfer mode (memory--I/O)
1	1	1	disable

5.3.2 DMA Control Registers (DMAC0, DMAC1)

These are 8-bit registers which specify the influence on the source address and destination address in DMA transfer. The DMAC register (n=0,1) can be accessed with 8/1-bit Read/Write operations using memory access.

DMACn register contents are retained during RESET and are undefined.

As fig. 5-2 indicates, bit 1,0 (PS1, PS0) of DMACn register specifies the influence on the source side address offset value.

Fig. 5-2 Format of DMA Control Register (DMAC0, DMAC1)

symbol	7	6	5	4	3	2	1	0	address
DMAC0 (channel 0)	0	0	PD1	PD0	0	0	PS1	PS0	××FA0H
DMAC1 (channel 1)									××FA2H

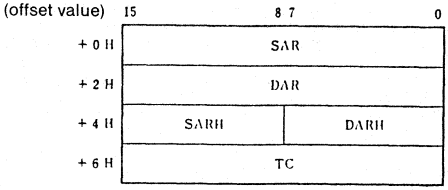
PS1/PD1	PS0/PD0	update mode for address offset value
0	1	increment of address offset value
1	0	decrement of address offset value
0	0	not used
1	1	

5.3.3 DMA Service Channel

This is used to specify transfer source, transfer destination, and number of transfers used in DMA transfer and it is mapped in internal RAM. The internal RAM addresses are assigned as follows: channel 0 to XXE00H-XXE07H and channel 1 to XXE08H-XXE0FH (XX is the value designated by IDB register). Care must be taken with these areas as they are assigned to the same areas as macro-service channel 0 and 1 as well as register bank 0.

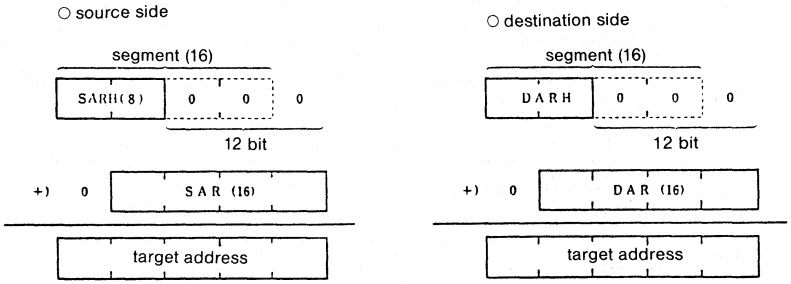
Designation of addresses for DMA source side and destination side is the same as the method for normal memory access and is specified by offset from segment and segment. However, only the higher 8 bits of the segments can be specified and the lower 8 bits are fixed to 0. Only this offset can be changed when changing the address using DMA transfer. As a result, a 64K-byte transfer is possible as far as any number of transfers goes, and when DMA transfer of data exceeds 61441 bytes (61441 times during byte transfer, 30720 times during word transfer), and care must be taken as there are cases in which it cannot be processed by a series of DMA transfers using addresses of transfer source and transfer destination. However, it is wise to be cautious even in cases in which data do not exceed 61441 bytes when segment and offset value are initialized. Fig. 5-3 shows configuration of DMA service channel; Fig. 5-4 shows method generation of DMA addresses.

Fig. 5-3 Format of DMA Service Channel



- SAR (+0H): specifies offset (least significant 16-bit of address of DMA transfer source side.
- DAR (+2H): specifies offset (least significant 16-bit) of address of DMA transfer destination side.
- DARH (+4H): specifies most significant 8-bit of segment value of DMA transfer destination side.
- SARH (+5H): specifies most significant 8-bit of segment value of DMA transfer source side address. However, the least significant 8-bit for the segment value is 0 fixed.
- TC (+6H): specifies number of DMA transfers.

Fig. 5-4 Method of DMA Address Generation



DMA service channel 0 is assigned to XXE00H and DMA service channel 1 is assigned to XXE08H (XX is value designated by IDB register).

The DMA service channel is automatically changed by DMA operations. TC value is decremented by 1 for every DMA transfer (byte data and word data are the same.).

Address offset value is changed in accordance with mode specified by DMA control register (DMACn): ±1 or unchanged for byte data, ±2 or unchanged for word data.

5.3.4 DMA Interrupt Request Control Registers (DIC0, DIC1).

These are 8-bit registers for control of interrupts generated by completion of a DMA transfer. Interrupt is generated when terminal counter (TC)=0.

The DICn (n=0,1) registers can be accessed with 8/1-bit Read/Write operations using memory access. The DICn register is initialized at 47H at reset time.

The macro-service functions are not supported in these interrupts. The DMA transfer completion interrupt of channel 0 (INTD0) and channel 1 (INTD1) form one group, the channel 0 taking a higher interrupt priority. INTD0 control is carried out by using the DIC0 register and the vector is 36. INTC1 control is carried out by DIC1 and the vector is 37 (see 2.4.5).

Fig. 5-5 Format of DMA Interrupt Request Registers (DIC0, DIC1).

(interrupt priority DF0>DF1)

symbol	7	6	5	4	3	2	1	0	address
DIC0	DF0	DMK0	0	ENCS	0	PR2	PR1	PR0	××FACH
DIC1	DF1	DMK1	0	ENCS	0	1	1	1	××FADH

(Note) The DIC1 register bit 2-0 is fixed at '1' using hardware. Bit 2-0 is a bit field (PR2-0) which specifies interrupt request priority by group and forms one group with the DIC0 register. The priority of the DIC1 register interrupt requests conforms to the setting of the PR2-0 bit of the DIC0 register.

The DF0/DF1 bit is an interrupt request flag for DMA transfer completion and the DMK0/DMK1 are masked bits for DMA transfer completion interrupt.
For description of other bit fields, see 3.7.

5.4 DMA Transfer Timing

Fig. 5-6 through 5-9 illustrate principal DMA transfer timing.

Fig. 5-6 Timing of burst mode (timing for 1 wait state insertion for transfer destination and no wait state insertion for transfer source when starting DMA using DMARQ signal when TC=1).

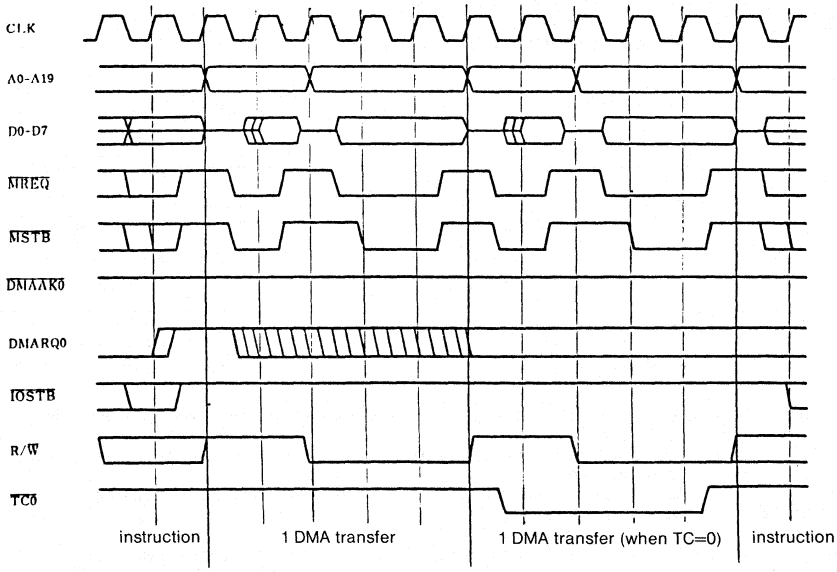


Fig. 5-7 Single Step Mode

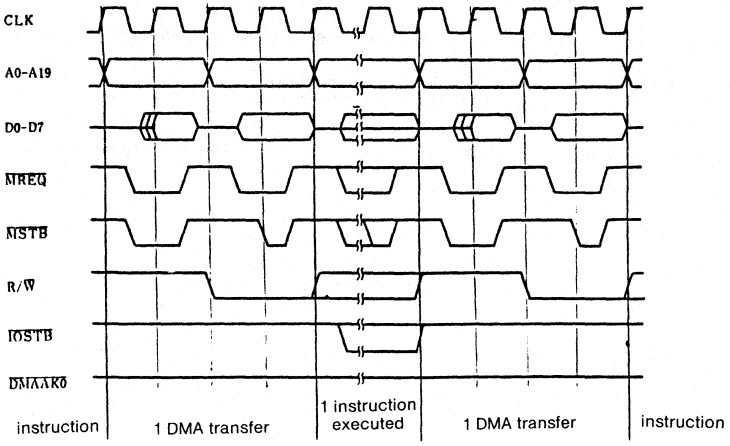


Fig. 5-8 Single transfer mode
(memory--I/O, no wait)

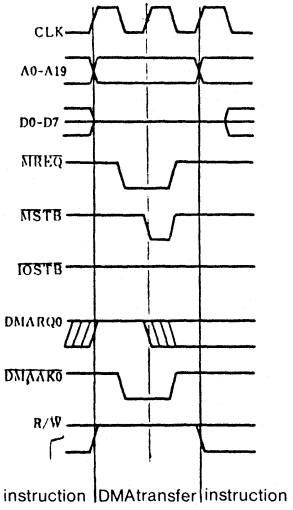


Fig. 5-9 Demand Release mode
(I/O--memory I/O; 1 wait, memory: no wait)

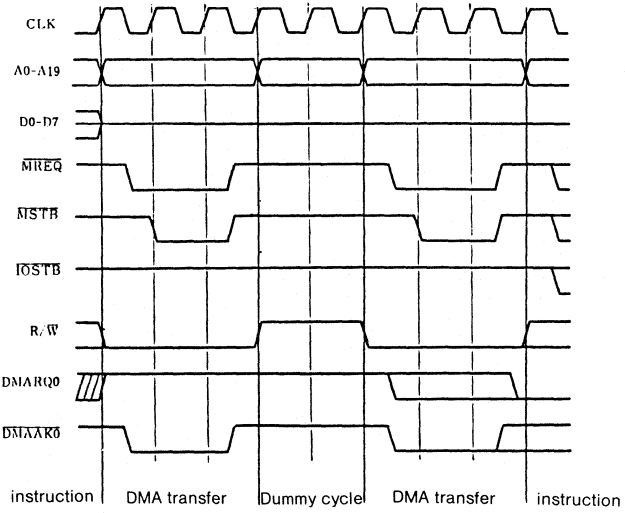
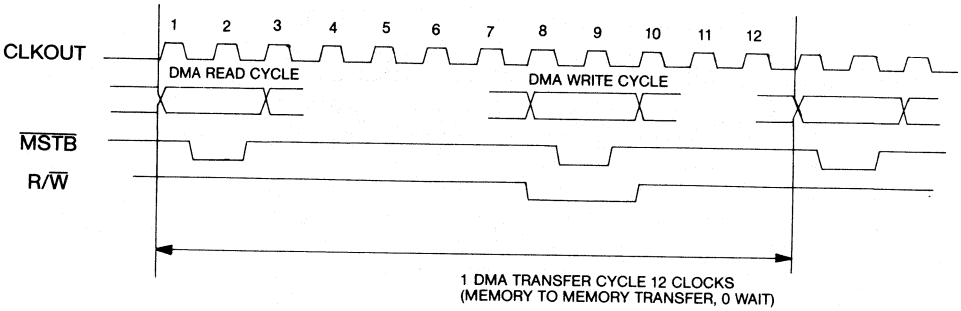


Fig. 5-10 Memory to Memory transfer mode



DMA TRANSFER CYCLE

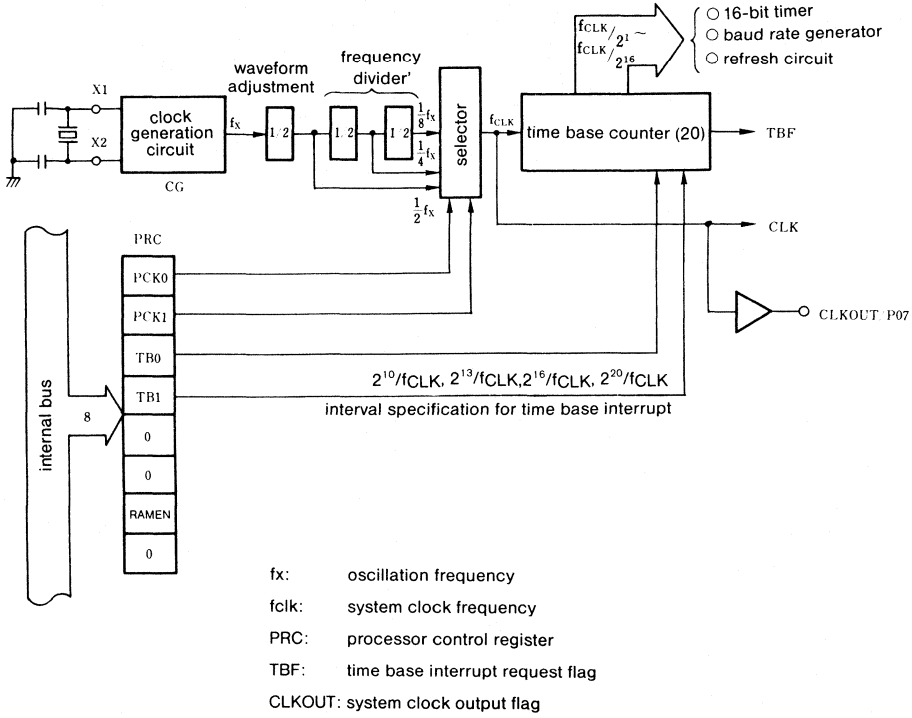
0 WAIT → 12 CLOCKS
1 WAIT → 12 CLOCKS
2 WAIT → 14 CLOCKS

6. Clock Generation Circuit

The Clock generation circuit supplies all types of clock to the CPU and to peripheral hardware and is a circuit which controls the CPU's operation mode. 6.1 Configuration of clock generation circuitry

Clock generation circuit is configured as in Fig. 6-1

Fig. 6-1 Block diagram of clock generation circuit



fx: oscillation frequency

fclk: system clock frequency

PRC: processor control register

TBF: time base interrupt request flag

CLKOUT: system clock output flag

The clock generation circuit uses a crystal oscillator connected to X1 and X2 pins or a ceramic oscillator. Clock generation circuit output undergoes a „waveform adjustment" (1/2 „frequency division"), selects „frequency division" ratio and is used as a system clock (CLK).

The CLK „frequency division" ratio can select oscillation frequencies of 1/2, 1/4, and 1/8 by specifying bit 0, 1 (PCK0, PCK1) of the processor control register (PRC).

Low-speed use of clock guarantees long periods of stable operation even if the voltage of a battery-driven system decreases.

6.2 Processor Control Register (PRC)

The PRC register is an 8-bit register which carries out concentrated control of CPU operations clock, time base interrupt periods, internal RAM access and other items related to the CPU and internal system control.

The PRC register can be accessed with 8/1-bit Read/Write operations using memory access.

It is initialized at 4EH using RESET input.

The PCK0,1 bits determine the system clock „frequency division" ratio. After „frequency division" of the frequency of the oscillator via PCK0,1, it is used as system clock (CLK).

The TB0,1 bits specify the time base interrupt interval. Four types of long interval time can be selected by using the TB0,1 bit.

The RAMEN bit controls enable for internal RAM access. It makes no distinction of internal RAM address in disable conditions (RAMEN bit „0") and accessing is always the object of external memory. When RAM is referenced as a register, internal RAM is always the object of accessing.

Fig. 6-2 Format of Processor Control Register (PRC)

mbol	7	6	5	4	3	2	1	0	address
PRC	0	RAMEN	0	0	TB1	TB0	PCK1	PCK0	××FEBH

PCK1	PCK0	'frequency division' ratio specification of system clock (CLK)
0	0	f_{CLK} = oscillation frequency (fx) x ½
0	1	f_{CLK} = oscillation frequency (fx) x ¼
1	0	f_{CLK} = oscillation frequency (fx) x $^1/_8$
1	1	disabled

TB1	TB0	interval designation for time base interrupt
0	0	generation of interrupt each $2^{10}/f_{CLK}$ interval
0	1	$2^{13}/f_{CLK}$
1	0	$2^{16}/f_{CLK}$
1	1	$2^{20}/f_{CLK}$

RAMEN	specification of internal RAM enable
0	internal RAM disable
1	internal RAM enable

7 Time Base Counter

The μPD70322/70320 stores a long interval timer function for clock function.

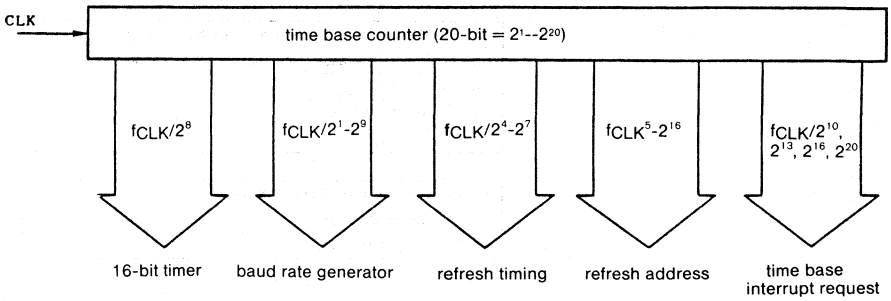
7.1 Configuration of Time Base Counter

Configuration of time base counter is illustrated in Fig. 7-1.

The time base counter is configured of 20 „frequency dividers" which divide the frequency of the system clock (CLK). The „frequency divider's lower side of the tap output is used for time count clock, baud rate generation input clock, refresh timing generation and refresh address generation. Of the 20 tap outputs, output taps 9, 12, 15, and 19 are used for time base interrupts.

The time base counter is cleared 00H only by RESET input and afterwards it is always incremented continnously.

Fig. 7-1 Configuration of Time Base Counter



7.2 Specification of Time Base Interval

The interrupt request interval time which is generated from the time base counter can be selected from four types (as indicated in Fig. 7-2) using bit 2,3 (TB0,1) of the processor control register (PRC).

Fig. 7-2 Interval Timer Mode of the Processor Control Register (PRC)

symbol	7	6	5	4	3	2	1	0	address
PRC	0	RAMEN	0	0	TB1	TB0	PCK1	PCK0	××FEBH

f_{CLK}= 5MHz (=½ fx, fx = 10MHz)

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

Note: time immediately after setting TB0, 1 bit until generation of initial interrupt request is undefined.

7.3 Time Base Interrupt Request Control Register (TBIC)

The TBIC is an 8-bit register used to carry out mask control for interrupt requests generated from the time base counter.

TBIC can be accessed with 8/1-bit Read/Write operations using memory access.

TBIC is initialized at 07H using RESET input.

Fig. 7-3 Format of time base interrupt request control register (TBIC)

symbol	7	6	5	4	3	2	1	0	address
TBIC	TBF	TBMK	0	0	0	1	1	1	×× FECH

Interrupt requests are generated once the output tap of the time base counter specified by processor control register (PRC) has gone to high level and the interrupt request flag (TBF) is set.

The TBIC bit 4.5 is fixed „0" and there is no context-switching function or macro-service function using the timer base counter interrupt. The TBIC bit 0-2 are fixed at „1", priority of time base interrupt (INTTB) is „7" fixed, and is fixed at the lowest position even among the other interrupts which have priority 7. Multiprocessing control, is accepted, however.

8. Serial Interface

8.1 Configuration of Serial Interface

The μPD70322/μPD70320 has two serial interface channels with built-in special baud rate generators. The serial interface has two types of operational mode: an asynchronous (start/stop transmission) mode which takes data bit synchronization and charakter synchronization using start bit in the asynchronous mode and an I/O interface mode which carries out data transmission by synchronizing in the serial clock which has been controlled in the same way as the μCOM-87 group and other serial data transmission modes.

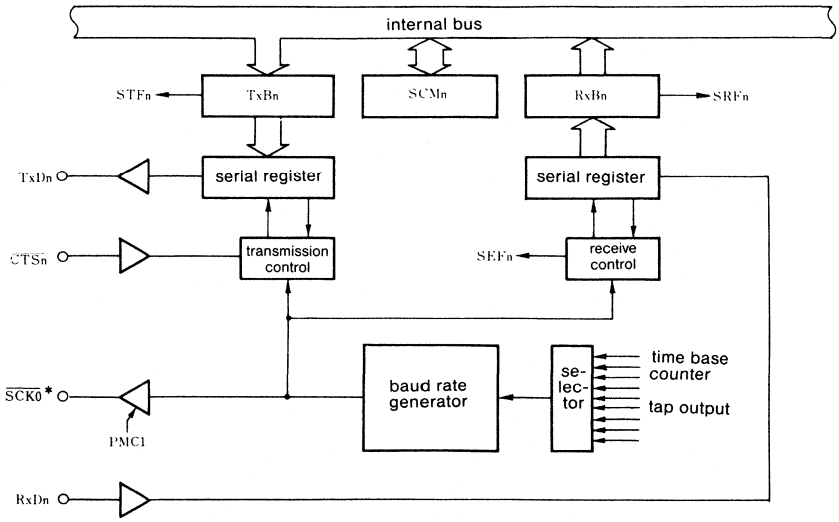
Fig. 8-1 gives a configuration diagram once for setting of serial interface asynchronous mode and once for I/O interface set-up.

The serial interface part is comprised of serial data input (RxDn), serial data output (TxDn), serial clock output (SCKO) transmission-enabling control input terminal (CTSn1), a transmission controller, an 8-bit serial register for send/receive, a transmission buffer (TxBn), a receive buffer (RxBn) and a baud rate generator.

It has serial registers and serial buffers for each transmission and receiving so that the transmission and reception can be carried out independently (all overlapping operations are possible). The CTSn terminal has functions for the receive clock input/output terminal during I/O interface mode so that all serial operations are possible and may overlap, even in I/O interface mode.

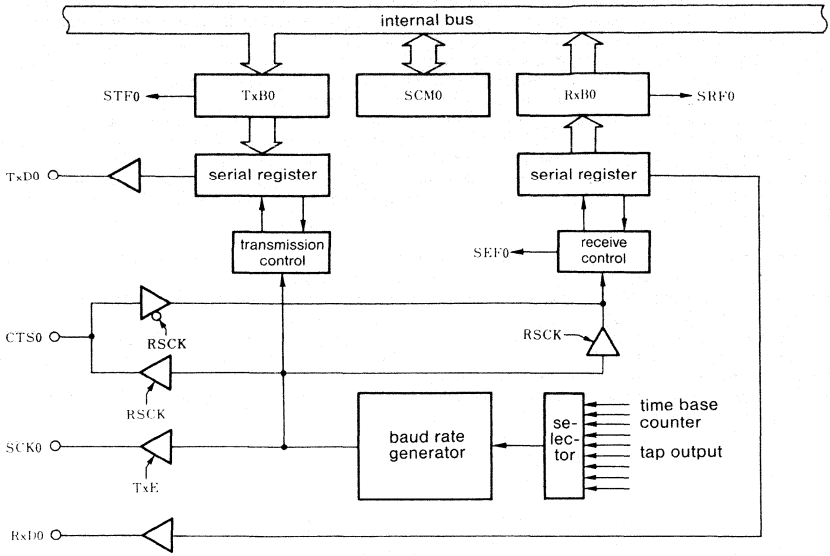
Fig. 8-1 Configuration of Serial Interface

(a) when asynchronous mode is set up (n-0, 1)



*(channel 0)

(b) when I/O interface mode is set up (channel 0 only)



8.2 Asynchronous Mode

During asynchronous mode, the specification of character length, number of stop bits, parity enable, odd/even parity is made by the serial mode register (SCMn).

(1) Transmission

In transmission operations bit 7 (TxRDY) of serial mode register (SCMn) is set (1) and the CTSn terminal goes to transmit enable condition at active time (0).

There are three methods of transmission start:

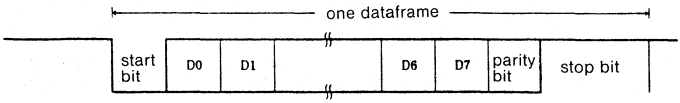
(i) transmission buffer (TxB) is set up in transmission condition when empty which generates transmission completion interrupt requests and carries out transmission data write operations to the transmission buffer within interrupt processing.

(ii) when transmission data are transmitted to transmission buffer at transmission enable condition, this transmission data are continuosly sent after the preceding transmission operation has ended.

(iii) in transmission disable condition, transmission data are writen beforehand in transmission buffer and the data reatined in the transmission buffer are sent afterwards when it is put in transmission enable condition.

*: there are no restrictions as such in the procedure for setting up the transmission enable condition; it is possible to make TxRDY=„1" active and set it at TxRDY=„1".

In the transmission data format, one data frame is comprised of start bit, character bit, parity bit and stop bit as the following figure indicates; the data transmitted is sent from the TxDn terminal starting with least significant bit (LSB). The TxDn terminal is in mark condition (1) during transmit disable or when it has no data which it is transmitting to serial register.



start bit ..... 1 bit
character bit ..... 7/8 bit
parity bit odd/even/0/unattached
stop bit ..... 1/2 bit

When the transmission buffer becomes empty the interrupt request for transmission completion is immediately generated and the transmission buffer goes to empty condition due to RESET input. When it is set to transmission enable condition at this time, the interrupt requests for transmission completion are generated. When transmit data from the transmit buffer are sent to the shift register by starting transmission operations, the transmission buffer goes to empty condition and there interrupt requests for transmission completion are generated.

Each time an interrupt request for transmission completion is generated continuous data transmission is possible by writing the transmission data in the transmission buffer without the mark condition (1) becoming inserted.

While transmission operations are being carried out the data being transmitted are sent one frame at a time until the end of the data or when switched to transmit disable condition. However, when new transmission data have already been written into the transmission buffer, sending from transmission buffer to shift register is disabled and transmission buffer contents are retained as they are. When it is again set to transmit enable condition, the transmission buffer contents coinciding with this timing are sent to shift register and interrupt request for transmission completion are generated at the same time that transmission has started.

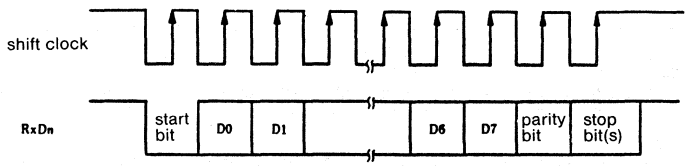
(2) Receiving

In receiving operations, it goes to receive enable condition when bit 6 (RxE) of serial mode register (SCMn) is set (1). (In receive disable condition (RxE=0) the hardware for receive is in a standby condition).

Sampling is carried out on the RxDn terminal input using the input clock to the baud rate generator; when the trailing edge is detected, receive operations are started up and a receive baud rate generator is started. When a RxDn pin input low level is detected using the initial timing signal from the receive baud rate generator, receiving operations are carried out after they have been recognized as a start bit. When high level has been detected by the initial timing signal, the baud rate generator is initialized without having recognized a start bit and operations are suspended.

Sampling of receiving data is carried out through synchronization with rise of a shift clock after the start bit has been detected, as indicated in the following figure.

Sampling timing of receive data



The receive interrupt requests are generated when the receive data from the shift register are sent to receive buffer (RxBn) when reception of data whose character length has been specified by serial mode register bit 3 (CL) has ended. During reception, receive error flag is set and receive error interrupt requests are generated, a parity check of even and odd numbers is carried out (when parity 1 bit=1*), and if they do not match (parity error), or when stop bit is low level (framing error), or when receive buffer is full and the subsequent data are sent to receive buffer (overrun error). (See 8.6).
Note: The PRTY1 bit is Bit 5 of the serial mode register.

8.3 I/O Interface Mode

I/O interface mode is identical to the μCOM-87 serial interface and is effective either when expanding I/O to external parts or when connecting peripheral controllers (A/D converter, LCD controller).

When using I/O interface mode, data transmission is carried out starting with the most significant bit (MSB) with 8-bit fixed character length and without parity bit. I/O interface mode is used on channel 0.

(1) Transmission

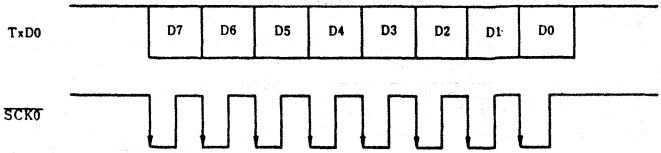
Transmission operations go to transmit enable condition when bit 7 (TxE) of serial mode register is set.

The SCKO terminal becomes transmission clock output pin in I/O interface mode. As with asynchronous mode, there are three types of transmission operation start-ups as follows:

(i) when transmission buffer (TxBO) is in empty condition, an interrupt request for transmission completion is generated by setting the buffer in transmit enable condition, and transmission data write operations to transmission buffer are carried out.

(ii) when transmission data are sent to transmission buffer (TxBO) in transmit enable condition, when the preceding transmission operation is completed, this transmission is continuously sent.

(iii) in transmit disable condition, transmission data are written in transmission buffer beforehand and when buffer is later put in transmit enable condition, the data retained in transmission buffer (TxBO) are sent.



Interrupt requests for transmission completion are generated as soon as transmission buffer (TxBO) is empty. Transmission buffer (TxBO) goes to empty condition due to RESET input. At this time, when it is set at transmit enable condition, interrupt requests for transmission completion are generated. When transmission data from transmission buffer (TxBO) are sent by starting transmission operations, the transmission buffer goes to empty condition and an interrupt request for transmission is generated.

(2) Receive

In receive operations, when bit 6 (RxE) in the serial mode register is set (1), it goes to receive enable condition. Receive data are input to serial register on receive clock rising edge. When the serial register receives 8-bit data, data are sent from the serial register to the receive buffer (RxBO) and an interrupt request for receive completion is generated.

Receive clocks in I/O interface mode are selected from both the external receive clocks and from the internal receive clocks by specifying bit 2 (RSCK) of the serial mode register (SCM0).

The CTSO terminal also functions as an input/output pin during I/O interface mode. Receive error flag is set and the interrupt request for receive error are generated at receive time when the receive buffer is full (RxBO) and when the following data have been sent to the receive buffer.

8.4 Serial Mode Register (SCM0, SCM1)

The SCMn register (n = 0, 1) is an 8-bit register which specifies the transmission mode for the serial interface and is set up at both channel 0 (SCM0) and channel 1 (SCM1). The assigned meanings of bits 7 to 2 on the SCMn vary according to specification of bits 1, 0 (MD1, MD0)

symbol	7	6	5	4	3	2	1	0	address
(channel 0) SCM0							MD 1	MD 0	××F68H
(channel 1) SCM1									××F78H

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

MD1 and MD0 bits are bit fields which specify transmission mode of serial interface. When they are set at MD1, MD0 = 0, 1, they go to asynchronous mode; when set at MD1, MD0 = 0, 0, they go to I/O interface mode. However, I/O interface mode can be set up only in SCM0.

SCMn can be accessed by 8/1-bit Read/Write operations by using memory access.

These registers are cleared to 00H by RESET input.

(1) Setting up of asynchronous mode

RxE a bit which carries out receive enable control.

When placed in receive disable condition (RxE = 0) during receive operations, receive processing is interrupted and no interrupt requests for receive completion are generated.

SL a bit which specifies stop bit

When SL bit is reset (0), the stop bit is 1 bit and it is 2 bits when set (1).

CL a bit which specifies character length.

When CL bis is reset (0) it is 7 characters long and 8 characters long when set (1).

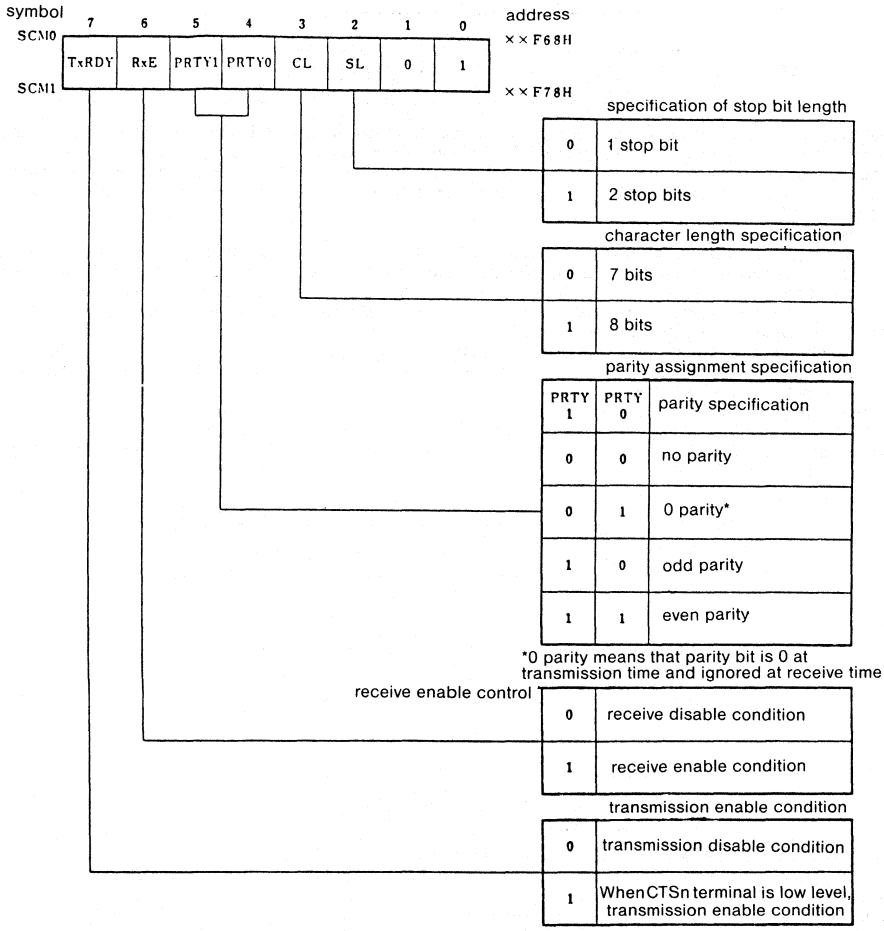
PRTY0 PRTY1 bits which specify parity assignment.

PRTY0, 1 bits specify no parity, odd and even number parity, and 0 parity. 0 parity makes parity bit "0" during transmission and ignores it during receive.

TxRDY is a bit which controls transmission enable condition.

When CTSn pin is low level and when TxRDY = 1, it causes the transmit enable condition.

Fig. 8-2 Format for Serial Mode Register (SCM0, SCM1) ... when setting up Asynchronous Mode



(2) Setting up of I/O Interface Mode

RSCK a bit which specifies source of serial receive clocks

When RSCK bit is reset (0), receive operations are carried out by external receive clock; when RSCK bit is set (1), receive operations are carried out by internal receive clock. Input/output for receive clock is carried out by CTSO pin.

TSK an output trigger bit for receive clock.

This is effective only when RSCK bit is set (1) and eight receive shift clocks are output from CTSO terminal by write operation of 1 to TSK bit.

RxE a bit which carries out receive enable control.

When RxE bit is set (1), it goes to receive enable condition; when reset (0), it goes to receive disable condition. When put in receive disable condition during receive operations, receive processing is interrupted at that point, and no interrupt requests for receive completion are generated.

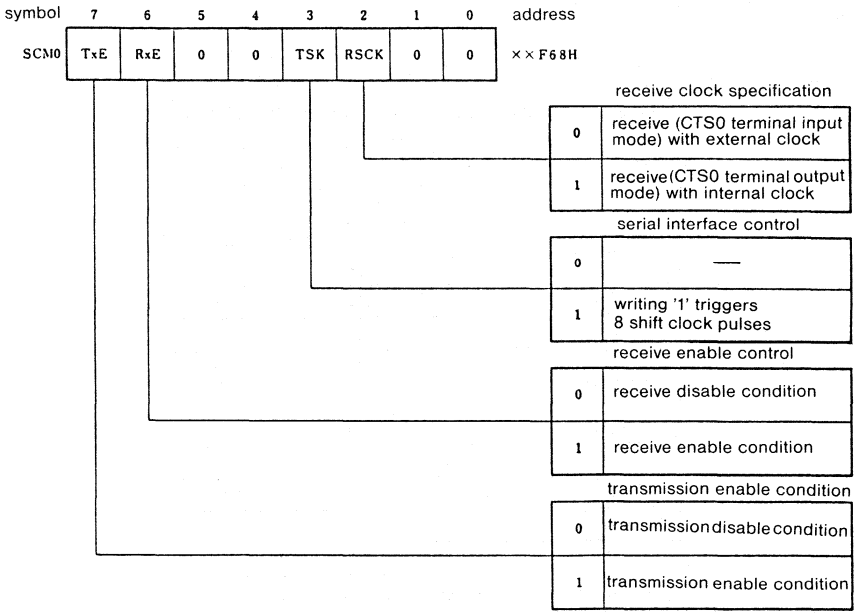
TxE a bit which carries out transmission enable control.

When TxE bit is set (1), it goes to transmission enable condition; when is is reset (0), it goes to transmission disable condition.

When transmission data are written into transmission buffer during transmission enable condition (TxE = 1), corresponding serial transmission is started after completion of running transmisson, and started immediately if no transmission is being carried out. When transmission data are written into transmission buffer during transmission disable condition (TxE = 0), serial transmission is not carried out, and data in the transmission buffer is retained unchanged. Afterwards, transmission processing of transmission data retained in the buffer is started at the same time when switching to transmission enable condition takes place.

Even if the TxE bit is reset (0) (transmission disable condition) during transmission operations, transmission operations are carried out until completion. However, the next transmission data which have already been stored in the transmission buffer at the point when it has been set to disable condition, the transmission following this transmission is omitted and the data are remaining in the buffer.

Fig. 8-3 Format for Serial Mode Register (SCM0) .... I/O Interface Mode



8.5 Baud Rate Generator

The baud rate generator is an 8-bit timer for the serial interface which generates shift clocks for transmission and receive. Each channel is provided with an own baud rate generator for transmission and receiving. The baud rate is the same both transmission and receiving and the baud rate is determined by writing the value to the baud rate generator register (BRGn).

The specification of the input clock for the baud rate generator is done by selecting the time base counter (see 7.1) output tap using the PRS3-0 bits of the serial control register (SCCn). The serial interface shift clock uses the baud rate generator output signals which have been divided by two. Setting up the baud rate generator for the transmit rate the parameter values satisfy the following formula:

$$B \times G = 10^6 \times \frac{CLK}{2n+1}$$

Where the parameters are defined as follows:

B: transmission baud rate (bps)
B = 110, ... 9600, 19200 . . .

G: set value for baud rate register (BRGn)
(1--G--255)

n: Input clock specification number (0--n--7)
for baud rate generator specified by serial control register (SCC).

CLK: system clock frequency (MHz)

Based on the above formula, the set values for the baud rate generator for all standard transmission baud rates when using a 10MHz crystal attached to the outside are as follows.

Chart 8-1 Set values for Baud Rate Generator (for reference)

f_{CLK} = 5MHz (=½ fx; fx = 10 MHz)

transfer baud rate	n	set value G for 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: input clock specification number of baud rate generator

8.5.1 Serial Control Registers (SCC0, SCC1)

The SCCn register (n=0, 1) is a register which controls serial interface transmission rate.
SCCn can be accessed by 8/1-bit Read/Write operations using memory access.
It is initialized at 00H using RESET input.
It specifies the output tap of the time base counter which is input to the baud rate generator in the PRS3-0 bit field.

Fig. 8-4 Format for Serial Control Registers (SCC0, SCC1)

symbol	7	6	5	4	3	2	1	0	address
SCC0 (channel 0)	0	0	0	0	PRS 3	PRS 2	PRS 1	PRS 0	××F69H
SCC1 (channel 1)									××F79H

PRS 3	PRS 2	PRS 1	PRS 0	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

8.6 Serial Error Processing

The following three types of serial interface errors during reception can be detected.

(i) Parity error (asynchronous mode)
Transmit parity and receive parity are different.

(ii) Framing error (asynchronous mode)
Stop bit is not detected.

(iii) Overrun error (asynchronous mode, I/O interface mode).

Before taking over the previous receive data from RxB, the following reception is completed.

8.6.1 Serial Error Registers (SCE0, SCE1)

These are 8-bit registers which indicate three types of error flag conditions corresponding to each receive error. Both channel 0 and channel 1 are provided with them.

SCEn (n=0, 1) can be accessed only by 8-bit Read operations using memory access.

SCEn is initialized with 00H during RESET.

ERPn Parity error flag

ERP flag is set when transmit parity and receive parity do not agree and is reset (1) during receive data read from receive buffer.

ERFn Framing error flag

ERF flag is set (1) when stop bit is not detected and reset (0) during receive data read from receive buffer.

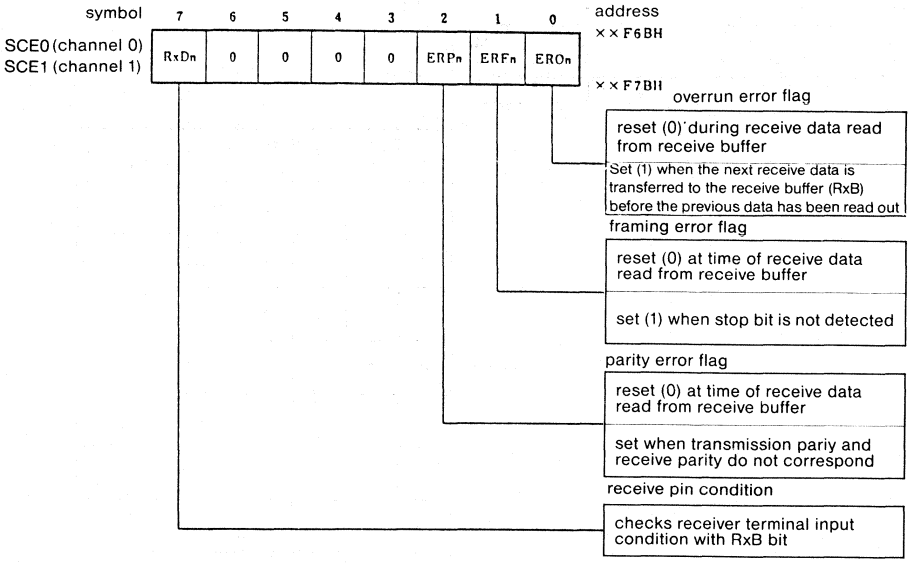
EROn Overrun error flag

ERO flag is set (1) when before receiving the preceding receive data from RxB the next receive is completed, and reset (0) during receive data read from receive buffer.

RxDn is a bit which checks receiver terminal input condition using RxB bit.

The serial error register (SCEn) is initialized at 00H using RESET input.

Fig. 8-5 Format of Serial Error Registers (SCE0, SCE1)



8.7 Break Detection Function

The μPD70322/70320 can be used to detect circuit break condition using software processing (asynchronous-mode only). Procedures for detecting a break condition are as follows:

(1) generation of receive error interrupt using the first framing error.
Receive data are checked inside receive error processing routine and are confirmed to be 00H.
At the same time, receive error flag is checked and the framing error is confirmed.

(2) generation of receive error interrupt using second framing error.
Framing error is again generated during break condition.
Receive data is again 00H, and continuous reception of 00H data which accompany framing error as well as confirmation of direct pin condition using bit 7 (RxDn) of serial error register (SCEn) are used to decide that the circuit is in break condition.

8.8 Interrupt Requests for Serial Interface

There are three types of interrupt requests which are generated by the serial interface and which correspond to the two channels: interrupt requests for transmission completion, for receive completion, and for receive error.

8.8.1 Control Registers for Interrupt Requests (SEICn, SRICn, STICn) n=0, 1

These are registers which control three types of interrupt requests generated from the serial interface: interrupt requests for receive error (SERn), for receive completion (SRFn), and for transmission completion (STFn). The three control registers for interrupt requests form one group and can be applied a priority specified for the serial interface interrupt request. The priorities inside the group are decided using hardware in the following way:

SEFn > SRFn > STFn

when SEF = 1, SRF is always set.

Fig. 8-6 Interrupt Control Registers (SEICn, SRICn, STICn) (n=0,1)

symbol	7	6	5	4	3	2	1	0	address
SEIC0	SEFn	SEMKn	MS/$\overline{INT}$	ENCS	0	PR2	PR1	PR0	××F6CH
SEIC1									××F7CH
SRIC0	SRFn	SRMKn	MS/$\overline{INT}$	ENCS	0	1	1	1	××F6DH
SRIC1									××F7DH
STIC0	STFn	STMKn	MS/$\overline{INT}$	ENCS	0	1	1	1	××F6EH
STIC1									××F7EH

(Note) The SRICn and STICn bits 2-0 are fixed at '1' using hardware. Bit 2-0 is a bit field (PR2-0) which specifies priority of interrupt requests according to group and form a group within SEICn. Interrupt request priority for SRICn and STICn conform to set-up of PRs-0 of SEICn.

Bits SEFn, SRFn, and STFn are interrupt request flags and are all set (1) respectively according to generation of receive error, receive completion, and transmission completion, and are reset, by acceptance of interrupt requests or by software.

See 3.7 for description of other bit fields.

SEICn, SRICn, and STICn can be accessed by 8/1-bit Read/write operations using memory access. SEICn, SRICn, and STICn are initialized at 47H using RESET input.

8.8.2 Macro-Service Control Registers (SRMSn, STMSn) n=0, 1

SRMSn is an 8-bit register which specifies macro-service processing mode which accompanies receive completion of serial interface. STMSn is an 8-bit register which specifies the macro-service processing mode and the channel which accompany the transmission completion of the serial interface. SRMSn and STMSn correspond to the two serial interface channels.

SRMSn and STMSn can be accessed by 8/1-bit Read/Write operations using memory access.

See 3.4.3 for description of each macro-service regsiter bit.

Fig. 8-7 Format of Macro-Service Control Register (SRMSn, STMSn) n=0,1

symbol	7	6	5	4	3	2	1	0	address
SRMS0	MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0	××F65H
SRMS1									××F75H
STMS0	MSM2	MSM1	MSM0	DIR	0	CH2	CH1	CH0	××F66H
STMS1									××F76H

9. Timer Unit

The μPD70322/70320 timer unit can be used as an interval timer, a one shot timer, and as a square wave output.

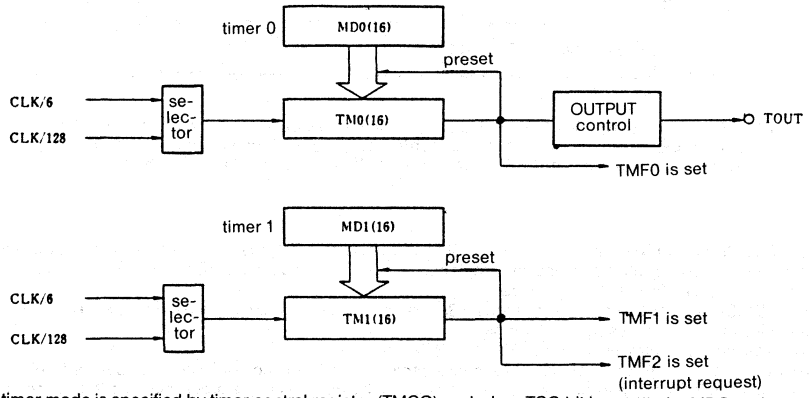
9.1 Configuration and Operation of Timer Unit

The timer unit is comprised of two 16-bit timer registers, two 16-bit modulo/timer registers and an 8-bit timer control register. Configuration and operation of each operational mode are described as follows.

(1) Interval timer mode

When timer unit is set up in interval timer mode, both timers 0 and 1 can be used as in Fig. 9-1.

Fig. 9-1 Configuration of Timer Unit during Interval Timer Mode



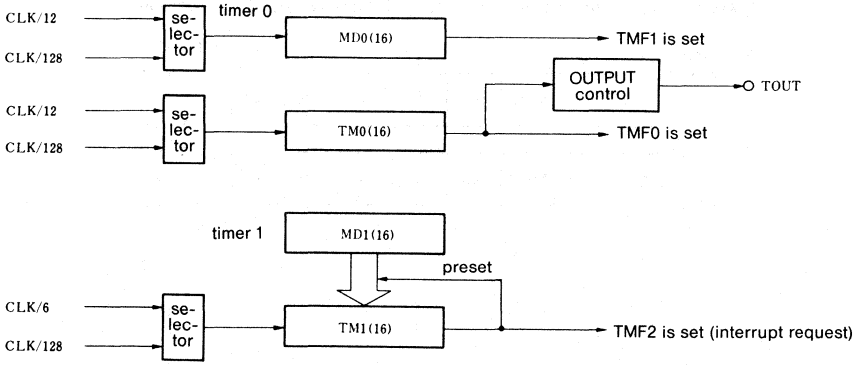
Interval timer mode is specified by timer control register (TMCO) and when TSO bit is set (1), the MDO register value is loaded into the TMO register, and the clock specified by TCLKO is down counted. When an underflow is generated during down count, the MDO register value is again reloaded into the TMO register and the down count is again repeated.

The same down count operations are executed for register of timer 1.

(2) One shot timer mode

When timer unit is set up in one shot timer mode, channel 0 is used as indicated in Fig. 9-2. However, it is still possible to operate timer 1 (channel 1) simultaneously as an interval timer.

Fig. 9-2 Configuration of Timer Unit during One Shot Timer Mode



The one shot timer mode is specified by the timer control register (TMCO) and when TSO/MSO bit is set (1), the TMO/MDO register down counts clocks specified by TCLKO/MCLKO bit. When an underflow is generated during counting, the count operations are suspended. TMO/MDO register is suspended while retaining 000H.

9.2 Timer Control Registers (TMC0, TMC1)

The TMCO register is an 8-bit register which controls operations of TMO and MDO registers. The TMC1 register is an 8-bit register which controls operations of TM1 and MD1 registers.

TMC0 and TMC1 registers can be accessed by 8/1-bit Read/Write operations using memory access. They are initialized at 00H using RESET input. TMC0 and IMC1 registers have different formats as shown in the following figures.

(interval timer mode, one shot timer mode)

symbol	7	6	5	4	3	2	1	0	address
TMC0	TS0	TCLK0	MS0	MCLK0	ENTO	ALV	MOD1	MOD0	××F90H

(interval timer mode)

	7	6	5	4	3	2	1	0	
TMC1	TS1	TCLK1	0	0	0	0	0	0	××F91H

Operational mode for timer 0 and timer 1 which are comprised respectively of TM0 and MD0, and TM1 and MD1 are specified by bits 0, 1 (MOD0, 1) of TMC0 and TMC1 registers.

MOD0 and MOD1 are bits which specify the operational modes for timer 0 and timer 1.

When MOD0 = 0 and MOD1 = 0, the interval timer operation mode is set. When MOD0 = 1 and MOD1 = 0, the one-shot timer mode is set.

During the interval timer operation mode, TM0 and TM1 work as timer registers which down-count the set values, where as MD0 and MD1 work as modulo registers which retain the set values for the intervals. Under the one-shot timer operation mode both TMO and MDO work as timer registers down counting the set values. The timer 1, however has ists TMC1 bits 0 and 1 fixed as "0", capable of operating only as an interval timer.

As a result, timer 0 can operate as a.16-bit interval timer or as two 16-bit one shot timers comprised of TMO and MDO using TMCO register. Timer 1 can be operated as a 16-bit interval timer comprised of TM1 and MD1 using TMC1 register.

Timer 0 can also output rectangular waves to the TOUT pin using the TMCO register. However, TOUT pin used with P15 to output rectangular waves to TOUT pin so that bit 5 (PMC15) of port 1 mode control register must be put on control mode.

ALV is a bit which specifies the active level for TOUT pin output.

The active level of TOUT pin output when ENTO bit is reset (0) goes to low level when ALV bit is reset (0) and high active when set (1).

ENTO is a bit which specifies operations for square waves output to TOUT pin.

When ENTO bit is reset (0), the TOUT pin level is specified by ALV bit. When ENTO bit is set (1), the TOUT pin level is reserved every time the interrupt request flag of the timer unit is set.

Descriptions of other bits of TMC0 and TMC1 registers are given according to operational mode as follows.

TCLK is a bit which specifies TMn register count clock.

Chart 9-1 gives reference values for system clocks with 5MHz frequency.

TSn is a bit which controls the operations of timer n.

When TSn bit is set (1), the value of the MDn register is set into the TMn regsiter and the down count of TMn register is started. When the TSn bit is cleared (0) the TMn register down count is suspended with TMn and MDn register contents retained unchanged.

During down count, underflow is generated and when TSn bit is set (1) again, the value of MDn register is again reloaded into the TMn register and down count operations are restarted.

Chart 9-1 Count Time (n=0,1) for Timer Register (TMn) During Interval Timer Mode

f_{CLK} = 5 MHz (=½ fx; fx = 10MHz)

TCLKn	count clock	resolution	full count
0	$f_{CLK}/6$	1.2 μs	78.6 ms
1	$f_{CLK}/128$	25.6 μs	1.7 s

(2) One Shot Timer MOde (MOD0=1. MOD1=0) However, timer 0 only.

TCLK is a bit which specifies TMO register count clock.

Table 9-2 indicates reference values when system clock frequency (TCLK) is 5MHz.

TSO is a bit which controls TMO register operations.

When TSO bit is set, it is down counted from values of TMO register which have been retained at that time; TSO bit is cleared (0) by underflow generation, and count operations are suspended. When TSO is cleared (0), count is suspended while retaining TMO register value unchanged.

MCLKO is a bit which specifies MDO register count clocks.

Table 9-2 indicates reference values when system clock frequency (CLK) is 5MHz. When it is specified in interval timer mode, the MCLKO does not affect the count operation.

MSO is a bit which controls count operations for MD register.

When MSO bit is set (0), it is down counted from the MDO register values which are retained at that time; MSO bit is cleared (0) by underflow generation and count operations are suspended. When MSO bit is cleared (0), count is suspended while MDO register values are retained unchanged.

The MSO bit does not affect count operations during interval timer operations.

Chart 9-2 Count Time for Timer Register 0 (TM0) and Modulo Timer Register 0 (MD0) During One Shot Timer Mode f_{CLK} = 5MHz(=½ fx ; fx = 10 MHz)

TCLK0 MCLK0	count clock	resolution	full count
0	$f_{CLK}/12$	2.4 μs	157.3 ms
1	$f_{CLK}/128$	25.6 μs	1.7 s

Note: the TM0 register has different count clocks depending on whether it has been specified in interval timer mode or specified in one shot time mode.

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

ol

7	6	5	4	3	2	1	0
TS	TCLK	MS	MCLK	ENTO	ALV	MOD	MOD
0	0	0	0			1	0

C0

address
××F90H

operational mode specifications

MOD 1*	MOD 0	specification
0	0	interval timer mode
0	1	one shot timer mode
1	x	disable

*'0' must be written in MOD1

effective level specification for TOUT pin

0	TOUT pin active level is specified at low level
1	TOUT pin active level is specified at high level

operation specification for TOUT pin

0	TOUT pin level fixed at ALV bit
1	TOUT pin level reversed by TMF0 set timing

clock source specified for MD0 register (during one shot timer mode)

0	CLK/12 specified as MD0 register count clock
1	CLK/128 specified as MD0 register count clock

operation control for MD0 register (during one shot timer mode)

0	count stopped (MD0 retained)
1	MD0 down count started

clock source specification for TM0 register

during interval timer	0	$f_{CLK}/6$ specified as TM0 count clock
	1	$f_{CLK}/128$ specified as MD0 count clock
during one shot mode	0	$f_{CLK}/12$ specified as TM0 count clock
	1	$f_{CLK}/128$ specified as TM0 count clock

operation control for TM0 register

during interval timer mode	0	count stopped (TM0, MD0 retained)
	1	MD0 contents loaded to TM0; TM0 down count started
during one shot timer mode	0	count stopped (TM0 retained)
	1	TM0 down count started

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

Symbol	7	6	5	4	3	2	1	0	Address
TMC1	TS 1	TCLK 1	0	0	0	0	0	0	××F91H

Clock source specification for TM1 register

0	specifies $f_{CLK}/6$ as count clock
1	specifies $f_{CLK}/128$ as count clock

Operation control for TM1 register

0	count stop (maintains TM1, MD1)
1	Starts TM1 down count by loading the contents of MD1on TM1

9.3 Timer Unit Interrupt Requests

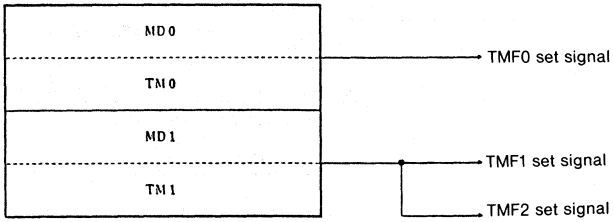
Three interrupt requests (TMF0-2) are generated from the timer unit. The generation condition for the interrupt requests coming from the timer unit differ according to the specification of the timer operation mode.

When they are set up to interval timer mode, the TMF0 is set (1) by the timing of underflow generated by the TM0 register countdown, and TMF1 and TMF2 are set (1) by underflow generated by TM1 register countdown (Fig. 9-5a).

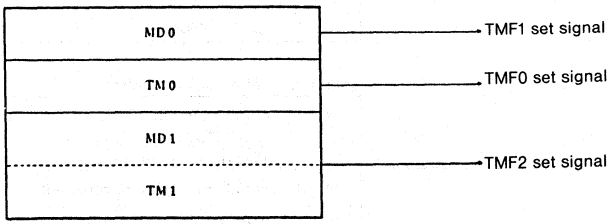
When TM0 and MD0 registers are set up in one shot timer mode, TMF0 is set (1) by underflow generated by TMO register countdown and TMF1 is set (1) by underflow generated by MD0 register countdown. In this case, TMF2 is set (1) by the underflow generated by TM1 regsiter countdown which operates as an interval timer.

Fig. 9-5 Interrupt Requests from Timer Unit

a. when TM0 and MD0 are specified as Interval Timer Mode



b. when TM0 and MD0 are specified as One Shot Timer Mode



TMF0-2 timer unit interrupt request flag 0-2

9.3.1 Interrupt Request Control Registers for Timer Unit (TMIC0, TMIC1, TMIC2)

The TMICn (n=0-2) register is an 8-bit register which controls three interrupt requests which are generated from the timer unit. These three interrupts requests form one group and priority for the timer unit interrupt requests as specified by program. Within that group, priority is fixed using hardware as follows.

TMF0 > TMF1 > TMF2

Fig. 9-6 Format of Interrupt Request Control Registers for Timer Units (TMICO, TMIC1, TMIC2)

Symbol	7	6	5	4	3	2	1	0	Address
TMIC0	TMF0	TMMK0	MS/$\overline{\text{INT}}$	ENCS	0	PR2	PR1	PR0	××F9CH
TMIC1	TMF1	TMMK1	MS/$\overline{\text{INT}}$	ENCS	0	1	1	1	××F9DH
TMIC2	TMF2	TMMK2	MS/$\overline{\text{INT}}$	ENCS	0	1	1	1	××F9EH

(Note) Bit 2-0 for TMIC1 and TMIC2 are fixed at "1" by hardware. Bit 2-0 is a bit field (PR2-0) which specifies the priority of interrupt requests in the group and it forms one group with TMIC0. Priority for TMCI1 and TMIC2 interrupt requests conforms with setting of PR2-0 for TMIC0.

See 3.7 for explanation of each of the TMICn register bits.

The TMICn regsiter can be accessed by 8/1 bit Read/Write operations using memory access.

The TMICn regsiter is initialized at 07H by RESET input.

9.3.2 Macro-Service Control Registers for Timer Unit (TMMS0, TMMS1, TMMS2)

These are 8-bit registers which control macro-service started by the three types of interrupt requests generated from the timer unit.

The TMMS0 register controls macro-service started by the TMF0 flag. The TMMS1 and TMMS2 both control macro-service which is started by TMF1 flag (for TMMS1) and TMF2 flag (for TMMS2).

TMMSn (n=0-2) can be accessed by 8/1-bit Read/Write operations using memory access.

Fig. 9-7 Format of Macro-Service Control Registers for Timer Unit (TMMS0, TMMS1, TMMS2).

Symbol	7	6	5	4	3	2	1	0	Address
TMMS0									××F94H
TMMS1	MSM 2	MSM 1	MSM 0	DIR	0	CH 2	CH 1	CH 0	××F95H
TMMS2									××F96H

See 3.4 for explanation of TMMSn regiter bits.

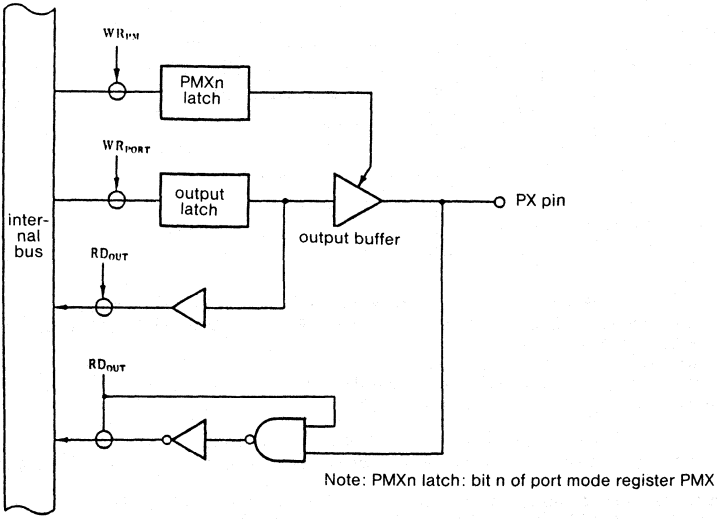
10 PORT FUNCTIONS

10.1 Port 0-2

10.1.1 Hardware Configuration

The μPD70322/μPD70320 ports 0-2 are basically comprised of three state bidirectional ports as indicated in Fig. 10-1. Each port mode register bit is set (1) by RESET input and thus specified as input port. All port pins are placed in a high impedance condition. The output latch contents are not influenced by RESET input.

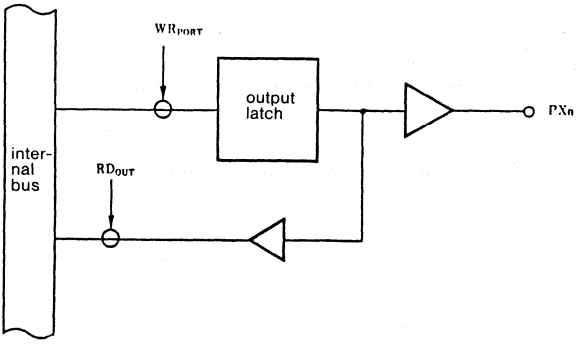
Fig. 10-1 Configuration of Port 0-2



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

The output latch is effective and data exchange between output latch and accumulator can be carried out by transfer instructions. Output latch contents can be set without restriction by logical operation instructions. Once data are written into the output latch, they are retained until the next instruction to operate the port is executed.

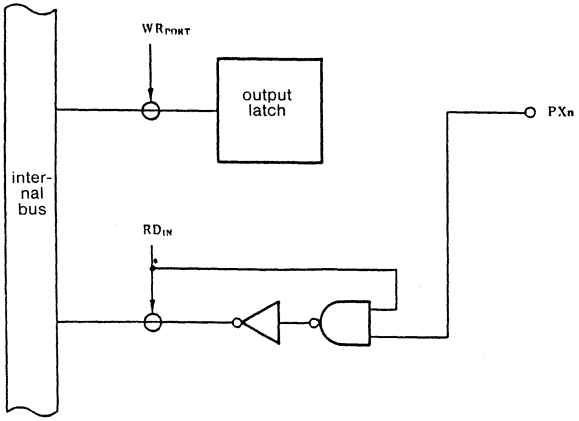
Fig. 10-2 Port for Output Port Specification



(2) when specified as input port (PMXn=1)

Port pin level can be loaded to the accumulator using transfer instructions. Even in this case writing into the output latch is possible and data sent from accumulator using transfer instructions are latched completely by the output latch regardless of port input/output specification. However, the bit output buffer specified at input port goes to high impedance condition so that there is no output to the port pin. (When the bit for input/output specification has been switched to the output port the contents of the output latch are not output to the port terminal). The contents of the output latch of the bit specified as input port can not be loaded to the accumulator. (Fig. 10-3)

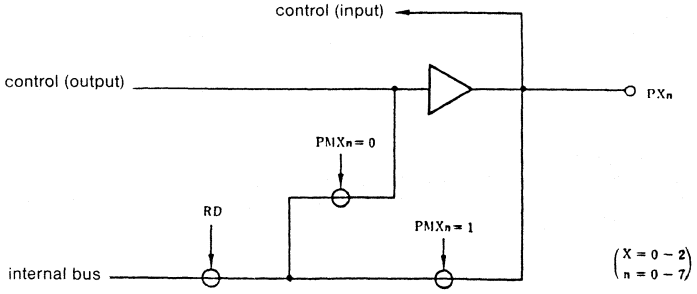
Fig. 10-3 Port for Input Port Specification



(3) with control specification (PMCXn=1)

In working with port 0-2, the bit of port mode control register (PMCX) is set (1) so that it can be used as input or output for control signals in units of bits regardless of the port mode register (PMX) set up. When a pin is used for a control signal the condition of the control signal can be checked by executing the port access instructions.

Fig. 10-4 Port for Control Signal Specification



(ii) when port is control signal output

When the port mode register (PMXn) is set (1) it is possible to read the control signal pin condition when the port read instruction is executed.

When the port mode register is reset (0) it is possible to read the condition of the internal control signal.

(ii) When port is control signal input

When the port mode register is set (1), it is possible to read the pin condition of the control signal when the port read instruction is executed.

10.1.2 Port Functions

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

This is a special 8-bit input/output port. Besides functioning as a general purpose input/output port whose input/output can be specified in bit units, it can also function as a system clock pin (for use with P07). Switching for these can be carried out in bit units by specifying the port 0 mode register (PM0) as well as the port 0 mode control register (PMC0).

Chart 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)

This is an 8-bit register used to define the use as port/system clock output for port 0 in bit units. As a result, the PMC0 register can be accessed by 8/1-bit Read/Write operations using memory access. If the corresponding bit of the PMC0 register is set (1) it defines the system clock output mode (P07), if reset, they go to port mode. All the bits of the PMC0 register during RESET input are reset (0) and it goes to port mode.

Fig. 10-5 Format for Port 0 Mode Control Register (PMC0)

Symbol	7	6	5	4	3	2	1	0	Address
PMC0	PMC0 7	0	0	0	0	0	0	0	××F02H

0	P0n=port mode
1	P07=system clock output

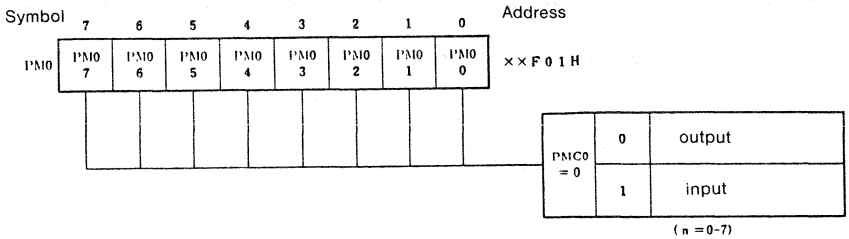
(ii) port 0 mode register (PM0)

PM0 is an 8-bit register which specifies input/output for port 0 using bit units.

PM0 can be accessed by 8/1-bit Read/Write operations using memory access. When the corresponding bit in PMC0 is "0", the PM0 becomes valid.

All bits are set (1) using RESET inout

Fig. 10-6 Format of Port 0 Mode Register (PM0)



(2) P10-17 (Port 1) . . . . . three state input/output

This is a special 8-bit input/output port. Besides functioning as a general-purpose input/output port whose input and output can be specified in bit units, it functions as a number of control pins. Switching for these can be carried out in bit units by specifying the port 1 mode register (PM1) as well as port 1 mode control register (PMC1).

The P10-P13 terminals can read the pin levels by direct accessing of port 1 (P1).

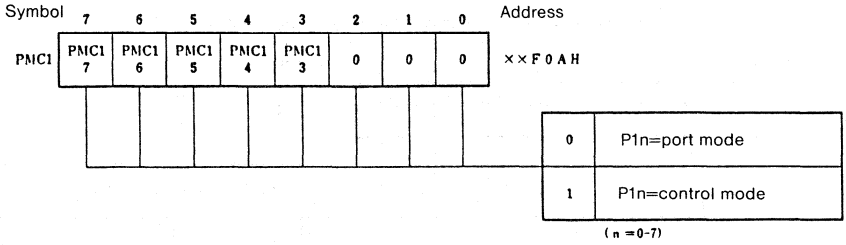
Table 10-2 Port 1 Operations (n=0-7)

	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	INTR input	input port (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

(i) Mode control register (PMC1) for port 1

This is an 8-bit register which can specify in bit units the use of port 1 for port-control signals or as input/output. As a result, the PMC1 register can be accessed by 8/1-bit Read/Write operations using memory access. When the corresponding bit in PMC1 register is set (1) it defines the control signal input/output mode, if it is reset (0), it is in the port mode. However, the P10-P12 pins are fixed in port mode.

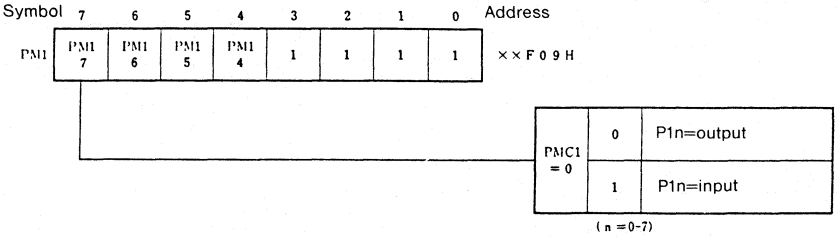
Fig. 10-7 Format for port 1 mode control register (PMC1)



(ii) Port 1 mode register (PM1)

The PM1 is an 8-bit register which specifies input or output for port 1 in bit units. As a result, the PM1 can be accessed by 8/1 bit Read/Write operations using memory access. When the corresponding bit of PMC1 is "0" PM1 becomes valid.

Fig. 10-8 Format of port 1 mode register (PPM1)



(3) P20-27 (port 2) . . . . . three state input/output.

This is a special 8-bit input/output port. A side from functioning as a general-purpose input/output port for which input/output can be specified in bit units it also functions as a number of control pins. Switching for these can be carried out in bit units by specifying the port 2 mode register (PM2) as well as the port 2 mode control register (PMC2).

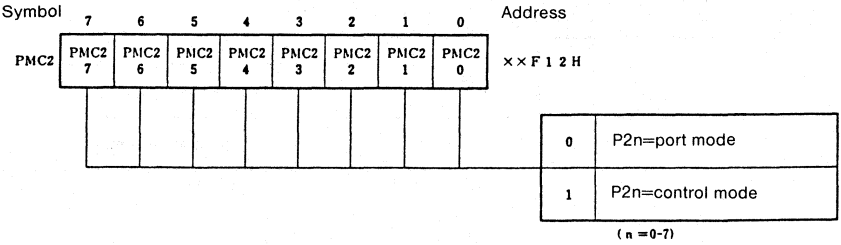
Chart 10-3 port 2 operations

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

(i) Port 2 Mode Control Register (PMC2)

This is an 8-bit register which can specify the use of port 2 for port/control signals or as input/output port for port 2 in bit units. If the corresponding bit of the PM2 is set (1) it defines the control signal input/output mode, if it reset (0), it goes to port mode. During reset input, the PMC2 register is reset (0), and it goes to port mode.

Fig. 10-9 Format of Port 2 Mode Control Register (PMC2)

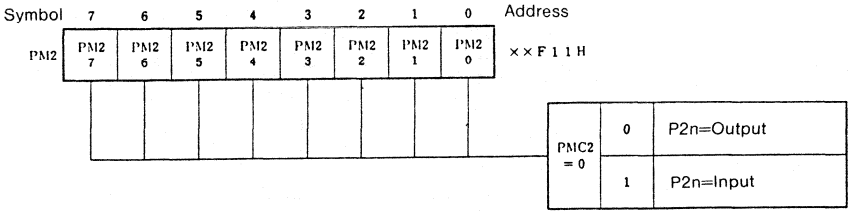


(ii) Port 2 Mode Register (PM2)

The PM2 is an 8-bit register which specifies input/output for port 2 in bit units. As a results, the PM2 can be accessed by 8/1 bit Read/Write operations using memory access. When the corresponding bit in PMC2 is 0, PM2 becomes valid.

All bits are set (1) by RESET input.

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



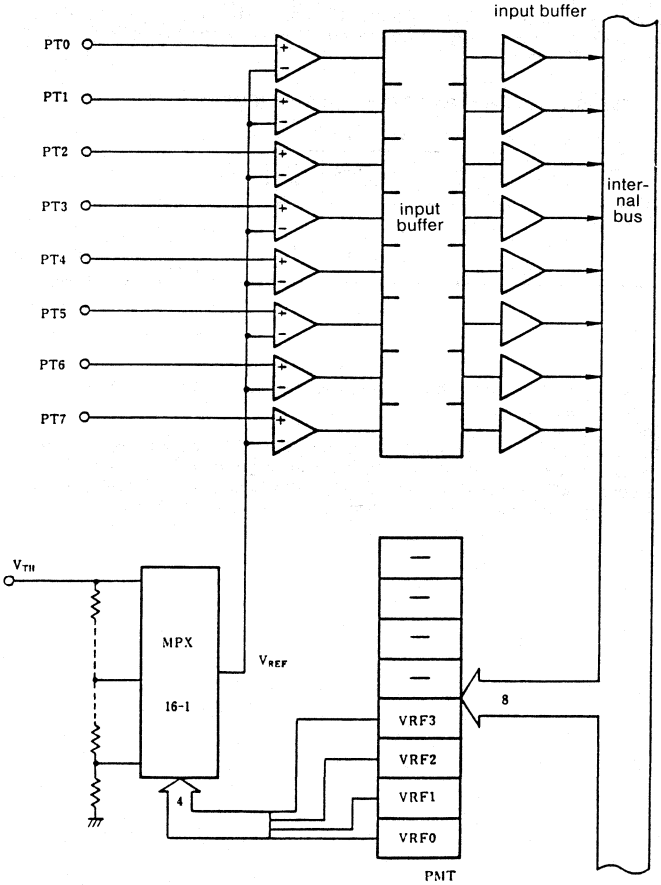
10.2 Port T (PT0-PT7)

Port T is an 8-bit input port which can vary the threshold voltage (reference voltage) in 16 stages. Comparator operations are carried out by analog input.

10.2.1 Hardware Configurations

Port T contains a multiplex circuitry (MPX) which selects one of PT0-PT7 comparator inputs, a V_{th} pin for standard power supply input to generate a matching voltage (V_{ref}) in 16 steps ranging from 1/16 X V_{th} up to 16/16 X V_{th}, of a port mode T register (PMT) which controls MPX and of 8 latches (Fig. 10-11). The V_{ref} and PT0-PT7 input (selected by setting up PMT) are compared using a comparator and the result is then latched into the port T input latch.

Fig. 10-11 Block diagram of Port T



10.2.2 Port T Mode Register (PMT)

PMT sets up the comparison voltage (V_{ref}) for the comparator as one of the 16 steps as indicated in Fig. 10-12.

PMT can be accessed by 8/1 bit Read/Write operations using memory access. All of the PMT bits are reset (0) by RESET input.

Fig. 10-12 Format of Port T Mode Register (PMT)

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	$V_{TH} \times 15/16$

11 STANDBY FUNCTIONS

The μPD70322/70320/70320 has two standby function modes which control the clock operation.

- HALT mode . . . . . a mode which suspends clock supply for the CPU. However, a number of CPU status data and RAM are all retained and peripheral hardware continues operation. Intermittent operation by combination with normal operation mode is used to lower total system power consumption.
- STOP mode . . . . . a mode which stops the oscillator which leads to a complete stop of the entire system. Internal RAM and port output data are retained as they require only very low power consumption.

Setting up of various modes is carried out using the HALT and STOP instructions.

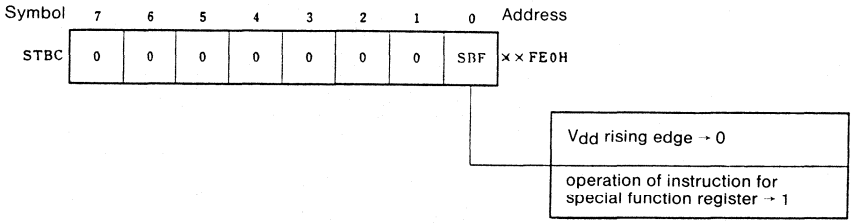
11.1 Standby Control Register (STBC)

The STBC is an 8-bit register which controls the standby flag (SBF).

SBF is used for the return decision from STOP condition. SBF is reset (0) only by starting power supply voltage (V_{DD}) and set (1) only by instruction execution for special function registers. SBF can be tested to distinguish whether there is a release after reset or return from STOP mode.

STBC is initialized by reset.

Fig. 11-1 Format for Standby Control Register (STBC)



11.2 HALT Mode

This is a mode which suspends clock supply for the CPU.

Setting up the HALT mode during CPU empty time reduces the overall power consumption for the system. When the HALT instruction is executed, it goes into HALT condition.

In HALT mode, the CPU clock is suspended, program execution is stopped and the contents of all of the registers and the internal RAM immediately before the suspension are retained. Table 11-2 illustrates conditions of all relevant hardware blocks.

11.2.1 Release from the HALT Mode

HALT mode is released by a nonmaskable interrupt (NMI) request, unmasked maskable interrupt request and RESET input. (Fig. 11-12) It goes from HALT mode to macro-service and DMA processing using macro-service requests or DMA processing requests (Fig. 11-3). When macro-service and DMA processing are completed, it again returns to HALT mode. However, if conditions such as those illustrated in Chart 11-1 appear during macro-service and DMA processing, the HALT mode is released.

(1) Release from HALT mode through an interrupt request

(i) when a HALT mode is set during an interrupt processing routine, it is released by generating unmasked maskable interrupt requests with a priority higher than that of the interrupt under processing or the generation of nonmaskable interrupt requests.

(ii) in all other cases

The HALT mode is released by generating a nonmaskable interrupt request or by generation of unmasked maskable interrupt requests regardless of their priority.

(2) Release by RESET input

Identical to normal reset operations

Fig. 11-2 Release from HALT Mode using Interrupt Requests

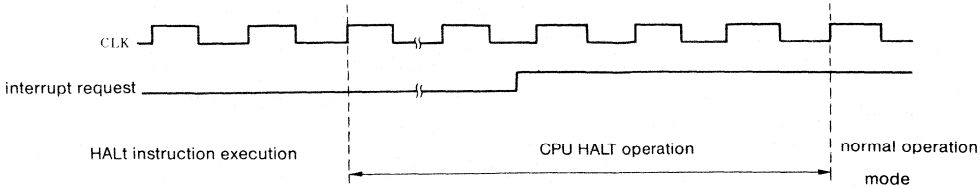


Fig. 11-3 Starting Macro-Service/DMA During HALT Mode

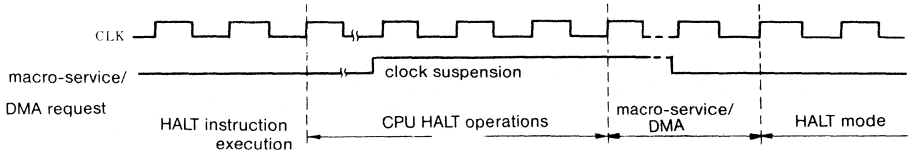


Chart 11-1 Operations after releasing HALT mode using interrupt requests

Release Source	EI Condition	DI Condition
Nonmaskable interrupt requests	branches to vector address after release	branches to vector address after release
maskable interrupt request	branches to vector address after release	executes next instruction after release
macro-service request	when macro-service is started and macro-service counter is OH, it branches to vector address. If macro-service counter does not reach OH, it goes to HALT condition a second time.	when macro-service starts and macro-service counter is OH, HALT mode is released and the next instruction is executed
DMA request	when DMA starts and terminal counter is at OH, it branches to vector address. If terminal counter does not reach OH, it goes to HALT condition a second time	when DMA starts, and terminal counter is OH, HALT mode is released and the next instruction is executed

11.3 STOP Mode

This is a mode which suspends the oscillator. It results in a very low power consumption as the complete system is suspended. Execution of STOP instruction causes it to go into STOP condition. In STOP mode, all of the clocks are suspended. Program execution is suspended and all of the register values immediately before suspension and the contents of the internal RAM are retained. Table 11-2 illustrates the condition of all hardware blocks.

11.3.1 Release from STOP Mode

STOP mode is released by using NMI request or by RESET input.

(1) Release by NMI Request (Fig. 11-4)

When the effective edge is input via the NMI pin, oscillation is restarted. The time base counter (TBC) starts operation and measures a period of several tens of milliseconds until the oscillation stabilizes.

As a result, after release from the stop mode, clocks are not immediately supplied, but the clock supply starts after the computed time of transmission stability using TBC.

(2) Release by RESET Input

Identical to normal reset operation.

Fig. 11-4 Release from STOP Mode Using NMI Input

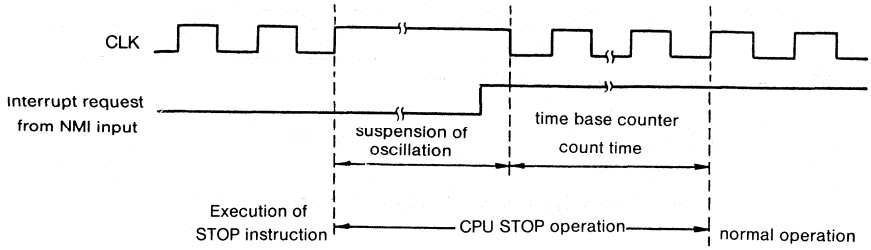


Table 11-2 HALT Mode/STOP Mode

Item		HALT Mode	STOP Mode
Oscillation		Operation	Stops
Internal System Clock		Stops	
16-Bit timer		Operation	
Time base counter			
HOLD circuitry			
Serial interface			
Interrupt request controller			
DMA controller			
I/O lines		Retained	Retained
Bus lines	A0-A19	Retained	Retained
	D0-D7	High impedance	High impedance
R/W output		High level	High level
Refresh operation		Operation/Stop	Stopped
Data retention		CPU status, RAM contents and internal data are all retained	CPU status, RAM contents and internal data are all retained
Released by		○ nonmaskable interrupt ○ maskable interrupt request ○ $\overline{\text{RESET}}$ input ○ macro-service request* ○ DMA*	○ nonmaskable interrupt request ○ $\overline{\text{RESET}}$ input

* again returns to HALT mode after processing of macro service and DMA

12. OPERATIONS AFTER RESET

When a low level is input to the RESET input pin a system reset takes place and the hardware goes into the condition as illustrated in Table 12-1. When the RESET input goes to high level the reset condition is turned off and the program execution is started. The contents should be initialized in the program as needed.

Table 12-1 Conditions after Hardware Reset

Hardware (symbol)			Address* (least significant 12-bit)	Condition 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	FFH
ports	port register	P0, P1, P2	F00H, F08H, F10H	undefined
		PT	F38H	
	port mode register	PM0, PM1, PM2	F01H, F09H, F11H	FFH
		PMT	F3BH	00H
	port 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 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

*XX in address (most significant 8-bit) is value designated by IDB register

Table 12-2 Conditions After Hardware Reset (continued)

Hardware (symbol)			Address* (least significant 12-bit)	Condition after Reset
serial interface	serial mode register	SCM0, SCM1	F68H, F78H	00H
	serial control register	SCC0, SCC1	F69H, F79H	00H
	baud rate generator set up 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	
time 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
external interrupt	external interrupt mode register	INTM	F40H	00H
	interrupt request control register	EXIC0-ECIC2	F4CH-F4EH	47H
	macro-service control register	EMS0-EMS2	F44H-F46H	undefined

* the XX of the higher 8-bit address is the value specified by the IDB register.

** the standby control register (STBC) cannot be reset using instructions once it has been set. It is cleared by a rise in the power supply voltage.

13 INSTRUCTION SET

The μPD70322/70320 instruction set is upward compatible with the μPD70108/70116 in native mode.

13.1 Instructions in addition to the μPD70108/70116

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

In addition, in contrast to the μPD70108/70116 instruction set, there are some instructions for the μPD70322/70320 which must be used with care.

○ input/output instruction . . . . . when the PSW IBRK flag is reset (0)
primitive input/output instruction . . . . . interrupt takes place without executing instruction

○ FPO instruction . . . . . interrupt takes place without executing instruction

13.2 Instruction Set Operations

Table 13-1 Description of Types of Operands

Identifier	Description
reg	8/16 bit general purpose register
reg 8	8-bit general purpose register
reg 16	16-bit general purpose register
dmem	8/16-bit memory location
mem	8/16-bit memory location
mem 8	8-bit memory
mem 16	16-bit memory
mem 32	32-bit memory location
imm	constant in 0-FFFFH range
imm 3	constant in 0-7 range
imm 4	constant in 0-FH range
imm 8	constant in 0-FFH range
imm 16	constant in 0-FFFFH range
acc	register AW or AL
sreg	segment register
src-table	name of 256-bit translation table
src-block	name of block addressed by register IX
dst-block	name of block addressed by register IY
near-proc	procedure within current code segments
far-proc	procedure within other code segments
near-label	label within present code segments
short-label	label from instruction end to −128/+127 byte range
far-label	label within other code segments
memptr16	words containing offset of locations within current program segments to which control is about to be transferred
memptr 32	double words containing segment base and offset addresses of locations inside other program segments to which control is about to be transferred.
regptr 16	16-bit general purpose register which contains offset of locations inside same code segment to which control is about to be transferred
pop-value	number of bytes taken from stack (0-64K, normally an even number.)
fp-op	immediate value which distinguishes instruction code for external floating point operations chip
R	register set

Table 13-2 Description of Operation Code

Identifier	Description
W	word/byte field(0-1)
reg	register field (000-111)
mem	memory field (000-111)
mod	mode field (00-10)
S : W	when S : W = 01, data = 16-bit, for all other cases, data = 8-bit when S : W = 11, data sign is expanded and forms a 16-bit operand
X, XXX, YYY, ZZZ	data for distinguishing instruction code for external floating point operations chip

Table 13-3 Description of Operands

Identifier	Description
AW	accumulator (16-bit)
AH	accumulator (higher byte)
AL	accumulator (lower byte)
BW	register BW (16-bit)
CW	register CW (16-bit)
CL	register CW (lower byte)
DW	register DW (16-bit)
SP	stack pointer (16-bit)
PC	program counter (16-bit)
PSW	program status word (16-bit)
IX	index register (source) (16-bit)
IY	index register (destination) (16-bit)
PS	program segment register (16-bit)
DS1	data segment 1 register (16-bit)
DS0	data segment 0 register (16-bit)
SS	segment register (16-bit)
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
MD	mode flag
(...)	memory contents indicated in ( )
disp	displacement (8/16-bit)
ext-disp 8	signed 16-bit displacement expanded from a signed 8-bit displacement
temp	temporary register (8/16/32-bit)
tmpcy	temporary carry flag (1-bit)
seg	immediate segment data (16-bit)
offset	immediate offset data (16-bit)
←	transfer direction
+	addition
–	subtraction
X	multiplication
:	division
%	modulo
∧	logical product ("and")
∨	logical sum ("or")
⊻	exclusive "or" ("or" else)
x x H	hexadecimal two-digit number
x x x x H	hexadecimal four-digit number

Table 13-4 Description of Flag Operations

Identifier	Description
(blank)	no change
0	cleared to 0
1	set to 1
x	is set or cleared according to results
U	undefined
R	number previously saved is restored

Table 13-5 Memory Addressing

mem \ mod	0 0	0 1	1 0
0 0 0	BW+IX	BW+IX+disp 8	BW+IX+disp 16
0 0 1	BW+IY	BW+IY+disp 8	BW+IY+disp 16
0 1 0	BP+IX	BP +IX+disp 8	BP +IX+disp 16
0 1 1	BP+IY	BP +IY+disp 8	BP +IY+disp 16
1 0 0	IX	IX +disp 8	IX +disp 16
1 0 1	IY	IY +disp 8	IY +disp 16
1 1 0	DIRECT ADDRESS	BP +disp 8	BP +disp 16
1 1 1	BW	BW+disp 8	BW+disp 16

Table 13-6 Selection of 8/16-bit general purpose register

reg	W=0	W=1
0 0 0	AL	AW
0 0 1	CL	CW
0 1 0	DL	DW
0 1 1	BL	BW
1 0 0	AH	SR
1 0 1	CH	BP
1 1 0	DH	IX
1 1 1	BH	IY

Table 13-7 Selection of segment register

sreg	
0 0	DS1
0 1	PS
1 0	SS
1 1	DS0

Starting with the following page, illustration sets will be explained in chart from.

In the column indicating the number of clocks, when there is an instruction (with W bit) with byte or word processing, the figure on the left side of the slash (/) indicates value for byte processing (W=0) and the figure on the right side indicates value for word processing (W=1) (instruction for primitive block transfer/input/output is coded in each column).

Number of clocks includes the following times needed for processing:

- ○ decode
- ○ EA generation
- ○ operand fetch
- ○ execution

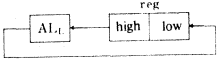
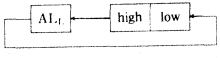
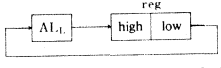
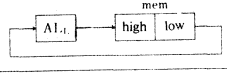
Instruction bytes have been prefetched.

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flag AC	flag CY	flag V	flag P	flag S	flag Z
data transfer instruction	MOV	reg,reg	1 0 0 0 1 0 1 W	1 1 reg reg	2		reg←reg						
		mem,reg	1 0 0 0 1 0 0 W	mod reg mem	2 - 4		(mem)←reg						
		reg,mem	1 0 0 0 1 0 1 W	mod reg mem	2 - 4		reg←(mem)						
		mem,imm	1 1 0 0 0 1 1 W	mod 0 0 0 mem	3 - 6		(mem)←imm						
		reg,imm	1 0 1 1 W reg		2 - 3		reg←imm						
		acc,dmem	1 0 1 0 0 0 0 W		3		when W = 0 AL← (dmem) when W = 1 AH← (dmem + 1), AL← (dmem)						
		dmem,acc	1 0 1 0 0 0 1 W		3		when W = 0 (dmem)← AL when W = 1 (dmem+1)← AH, (dmem)← AL						
		sreg,reg16	1 0 0 0 1 1 1 0	1 1 0 sreg reg	2		sreg←reg16 sreg : SS, DS0, DS1						
		sreg,mem16	1 0 0 0 1 1 1 0	mod 0 sreg mem	2 - 4		sreg←(mem16) sreg : SS, DS0, DS1						
		reg16,sreg	1 0 0 0 1 1 0 0	1 1 0 sreg reg	2		reg16←sreg						
		mem16,sreg	1 0 0 0 1 1 0 0	mod 0 sreg mem	2 - 4		(mem16)←sreg						
		DS0,reg16, mem32	1 1 0 0 0 1 0 1	mod reg mem	2 - 4		reg16←(mem32) DS0←(mem32 + 2)						
		DS1,reg16, mem32	1 1 0 0 0 1 0 0	mod reg mem	2 - 4		reg16←(mem32) DS1←(mem32 + 2)						
		AH,PSW	1 0 0 1 1 1 1 1		1		AH←S, Z, F1, AC, F0, P, $\overline{\text{IBRK}}$, CY						
		PSW,AH	1 0 0 1 1 1 1 0		1		S, Z, F1, AC, F0, P, $\overline{\text{IBRK}}$, CY←AH	×	×		×	×	×
	LDEA	reg16,mem16	1 0 0 0 1 1 0 1	mod reg mem	2 - 4		reg16←mem16						
	TRANS	src-table	1 1 0 1 0 1 1 1		1		AL←(BW + AL)						
	XCH	reg,reg	1 0 0 0 0 1 1 W	1 1 reg reg	2		reg↔reg						
		mem,reg reg,mem	1 0 0 0 0 1 1 W	mod reg mem	2 - 4		(mem)↔reg						
		AW,reg16 reg16,AW	1 0 0 1 0 reg		1		AW↔reg16						

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flag AC	flag CY	flag V	flag P	flag S	flag Z
repeat instructions	REPC		0 1 1 0 0 1 0 1		1		While CW = 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (-1). If there is a waiting interrupt, it is processed When CY = 1, exit the loop.						
	REPNC		0 1 1 0 0 1 0 0		1		Same as above When CY = 0, exit the loop.						
	REP REPE REPZ		1 1 1 1 0 0 1 1		1		While CW = 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (-1) If there is a waiting interrupt, it is processed. If the primitive block transfer instruction is CMPBK or CMPM and Z = 1, exit the loop.						
	REPNE REPNZ		1 1 1 1 0 0 1 0		1		Same as above CMPM and Z = 0, exit the loop.						
block transfer instructions	MOVBK	dst-block, src-block	1 0 1 0 0 1 0 W		1		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						
	CMPBK	src-block, dst-block	1 0 1 0 0 1 1 W		1		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	×	×	×	×	×	×
	CMPM	dst-block	1 0 1 0 1 1 1 W		1		when W = 0 (IY)← (IX) DIR = 0 : IX← IX + 1, IY← IY + 1 when W = 1 (IY + 1, IY)← (IX + 1, IX) DIR = 0 : IX← IX + 2, IY← IY + 2	×	×	×	×	×	×
	LDM	src-block	1 0 1 0 1 1 0 W		1		when W = 0 (IY)← (IX) DIR = 0 : IX← IX + 1, IY← IY + 1 when W = 1 (IY + 1, IY)← (IX + 1, IX) DIR = 0 : IX← IX + 2, IY← IY + 2						
	STM	dst-block	1 0 1 0 1 0 1 W		1		when W = 0 (IY)← (IX) DIR = 0 : IX← IX + 1, IY← IY + 1 when W = 1 (IY + 1, IY)← (IX + 1, IX) DIR = 0 : IX← IX + 2, IY← IY + 2						

instruction group	mnemonic	operand	operation code		no. of bytes	no. of clocks	operation	flag AC	CY	V	P	S	Z
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0									
Bit field transfer instructions	INS	reg8,reg8	0 0 0 0 1 1 1 1	0 0 1 1 0 0 0 1	3		16-bit field ← AW						
			1 1 reg reg										
		reg8,imm4	0 0 0 0 1 1 1 1	0 0 1 1 1 0 0 1	4		16-bit field ← AW						
			1 1 0 0 0 reg										
	EXT	reg8,reg8	0 0 0 0 1 1 1 1	0 0 1 1 0 0 1 1	3		AW ← 16-bit field						
			1 1 reg reg										
		reg8,imm4	0 0 0 0 1 1 1 1	0 0 1 1 1 0 1 1	4		AW ← 16-bit field						
			1 1 0 0 0 reg										
I/O instructions	IN	acc,imm8	1 1 1 0 0 1 0 W		2		When W = 0 AL ← (imm8) When W = 1 AH ← (imm8 + 1), AL ← (imm8)						
		acc,DW	1 1 1 0 1 1 0 W		1		When W = 0 AL ← (DW) When W = 1 AH ← (DW + 1), AL ← (DW)						
	OUT	imm8,acc	1 1 1 0 0 1 1 W		2		When W = 0 ← (imm8) ← AL When W = 1 ← (imm8 + 1) ← AH, (imm8) ← AL						
		DW,acc	1 1 1 0 1 1 1 W		1		When W = 0 (DW) ← AL When W = 1 (Dw + 1) ← AH, (DW) ← AL						
Primitive I/O instructions	INM	dst-block, DW	0 1 1 0 1 1 0 W		1		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						
	OUTM	DW, src-block	0 1 1 0 1 1 1 W		1		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						

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flag AC	flag CY	flag V	flag P	flag S	flag Z
addition/subtraction instructions	ADD	reg,reg	0 0 0 0 0 0 1 W	1 1 reg reg	2		reg←reg+reg	×	×	×	×	×	×
		mem,reg	0 0 0 0 0 0 0 W	mod reg mem	2-4		(mem)←(mem)+reg	×	×	×	×	×	×
		reg,mem	0 0 0 0 0 0 1 W	mod reg mem	2-4		reg←reg+(mem)	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 S W	1 1 0 0 0 reg	3-4		reg←reg+imm	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 S W	mod 0 0 0 mem	3-6		(mem)←(mem)+imm	×	×	×	×	×	×
		acc,imm	0 0 0 0 0 1 0 W		2-3		when W = 0 AL← AL + imm when W=1 AW← AW + imm	×	×	×	×	×	×
	ADDC	reg,reg	0 0 0 1 0 0 1 W	1 1 reg reg	2		reg←reg+reg+CY	×	×	×	×	×	×
		mem,reg	0 0 0 1 0 0 0 W	mod reg mem	2-4		(mem)←(mem)+reg+CY	×	×	×	×	×	×
		reg,mem	0 0 0 1 0 0 1 W	mod reg mem	2-4		reg←reg+(mem)+CY	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 S W	1 1 0 1 0 reg	3-4		reg←reg+imm+CY	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 S W	mod 0 1 0 mem	3-6		(mem)←(mem)+imm+CY	×	×	×	×	×	×
		acc,imm	0 0 0 1 0 1 0 W		2-3		when W = 0 AL← AL + imm when W=1 AW← AW + imm	×	×	×	×	×	×
	SUB	reg,reg	0 0 1 0 1 0 1 W	1 1 reg reg	2		reg←reg−reg	×	×	×	×	×	×
		mem,reg	0 0 1 0 1 0 0 W	mod reg mem	2-4		(mem)←(mem)−reg	×	×	×	×	×	×
		reg,mem	0 0 1 0 1 0 1 W	mod reg mem	2-4		reg←reg−(mem)	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 S W	1 1 1 0 1 reg	3-4		reg←reg−imm	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 S W	mod 1 0 1 mem	3-6		(mem)←(mem)−imm	×	×	×	×	×	×
		acc,imm	0 0 1 0 1 1 0 W		2-3		when W = 0 AL← AL + imm when W=1 AW← AW + imm	×	×	×	×	×	×
	SUBC	reg,reg	0 0 0 1 1 0 1 W	1 1 reg reg	2		reg←reg−reg−CY	×	×	×	×	×	×
		mem,reg	0 0 0 1 1 0 0 W	mod reg mem	2-4		(mem)←(mem)−reg−CY	×	×	×	×	×	×
		reg,mem	0 0 0 1 1 0 1 W	mod reg mem	2-4		reg←reg−(mem)−CY	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 S W	1 1 0 1 1 reg	3-4		reg←reg−imm−CY	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 S W	mod 0 1 1 mem	3-6		(mem)←(mem)−imm−CY	×	×	×	×	×	×
		acc,imm	0 0 0 1 1 1 0 W		2-3		when W = 0 AL← AL + imm when W=1 AW← AW + imm	×	×	×	×	×	×

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	CY	V	P	S	Z
BCD operation instructions	ADD4S		0 0 0 0 1 1 1 1	0 0 1 0 0 0 0 0	2		dst BCD string←dst BCD string+src BCD string *	U	×	U	U	U	×
	SUB4S		0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 0	2		dst BCD string←dst BCD string−src BCD string *	U	×	U	U	U	×
	CMP4S		0 0 0 0 1 1 1 1	0 0 1 0 0 1 1 0	2		dst BCD string−src BCD string *	U	×	U	U	U	×
	ROL4	reg 8	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 0 0	3								
		mem 8	0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 0 0	3-5								
	ROR4	reg 8	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 1 0	3								
		mem 8	0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 1 0	3-5								
increase/decrease instruction	INC	reg8	1 1 1 1 1 1 1 0	1 1 0 0 0 reg	2		reg8←reg8+1	×		×	×	×	×
		mem	1 1 1 1 1 1 1 W	mod 0 0 0 mem	2-4		(mem)←(mem) +1	×		×	×	×	×
		reg16	0 1 0 0 0 reg		1		reg16←reg16+1	×		×	×	×	×
	DEC	reg8	1 1 1 1 1 1 1 0	1 1 0 0 1 reg	2		reg8←reg8−1	×		×	×	×	×
		mem	1 1 1 1 1 1 1 W	mod 0 0 1 mem	2-4		(mem)←(mem)−1	×		×	×	×	×
		reg16	0 1 0 0 1 reg		1		reg16←reg16−1	×		×	×	×	×

n: 1/2 of number of BCD digits
*: number of BCD digits is given on CL register ; values from 1 through 254 can be set up.

instruction group	mnemonic	operand	operation code		no. of bytes	no. of clocks	operation	flags					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
multiplication operation	MULU	reg8	1 1 1 1 0 1 1 0	1 1 1 0 0 reg	2		AW←AL×reg8 AH=0 : CY←0, V←0 AH≠0 : CY←1, V←1	U	×	×	U	U	U
		mem8	1 1 1 1 0 1 1 0	mod 1 0 0 mem	2-4		AW←AL×(mem8) AH=0 : CY←0, V←0 AH≠0 : CY←1, V←1	U	×	×	U	U	U
		reg16	1 1 1 1 0 1 1 1	1 1 1 0 0 reg	2		DW,AW←AW×reg16 DW=0 : CY←0, V←0 DW=1 : CY←1, V←1	U	×	×	U	U	U
		mem16	1 1 1 1 0 1 1 1	mod 1 0 0 mem	2-4		DW,AW←AW×(mem16) DW=0 : CY←0, V←0 DW=1 : CY←1, V←1	U	×	×	U	U	U
	MUL	reg8	1 1 1 1 0 1 1 0	1 1 1 0 1 reg	2		AW← AL X reg8 AH = AL sign expansion : CY← 0, V← 0 AH = AL sign expansion : CY← 1, V← 1	U	×	×	U	U	U
		mem8	1 1 1 1 0 1 1 0	mod 1 0 1 mem	2-4		AW← AL X (mem8) AH = AL sign expansion : CY← 0, V← 0 AH = AL sign expansion : CY← 1, V← 1	U	×	×	U	U	U
		reg16	1 1 1 1 0 1 1 1	1 1 1 0 1 reg	2		DW, AW← AW X reg16 DW = AW sign expansion : CY← 0, V← 0 DW = AW sign expansion : CY← 1, V← 1	U	×	×	U	U	U
		mem16	1 1 1 1 0 1 1 1	mod 1 0 1 mem	2-4		DW, AW← AW X (mem16) DW = AW sign expansion : CY← 0, V← 0 DW = AW sign expansion : CY← 1, V← 1	U	×	×	U	U	U
		reg16, (reg16,)* imm8	0 1 1 0 1 0 1 1	1 1 reg reg	3		reg16← reg16 X imm8 product - 16 bit : CY← 0, V← 0 product - 16 bit : CY← 1, V← 1	U	×	×	U	U	U
		reg16, mem16, imm8	0 1 1 0 1 0 1 1	mod reg mem	3-5		reg16← (mem16) X imm8 product - 16 bit : CY← 0, V← 0 product - 16 bit : CY← 1, V← 1	U	×	×	U	U	U
		reg16, (reg16,)* imm16	0 1 1 0 1 0 0 1	1 1 reg reg	4		reg16← reg16 X imm16 product - 16 bit : CY← 0, V← 0 product - 16 bit : CY← 1, V← 1	U	×	×	U	U	U
		reg16, mem16, imm16	0 1 1 0 1 0 0 1	mod reg mem	4-6		reg16← (mem16) X imm16 product - 16 bit : CY← 0, V← 0 product - 16 bit : CY← 1, V← 1	U	×	×	U	U	U

*: second operand may be omitted. If it is omitted, it designates the same register as the first operand

instruction group	mnemonic	operand	operation code		no. of bytes	no. of clocks	operation	flags					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Signed division instructions	DIV	reg8	1 1 1 1 0 1 1 0	1 1 1 1 1 reg	2		temp· AW When temp ÷ reg8 > 0 and reg8 > 7FH or temp ÷ reg8 > 0 and temp ÷ reg8 < 0 to 7FH-1 AH· temp%reg8, AI· temp ÷ reg8 When temp ÷ reg8>0 and temp ÷ reg8 >7FH or When 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←(3,2),PC←(1,0)	U	U	U	U	U	U
		mem8	1 1 1 1 0 1 1 0	mod 1 1 1 mem	2 - 4		temp· AW When temp ÷ (mem8) > 0 and (mem8) > 7FH or temp ÷ (mem8) > 0 and temp ÷ (mem8) < 0 to 7FH–1 AH· temp%(mem8), AL· temp ÷ (mem8) When temp ÷ (mem8) >0 and temp ÷ (mem8) > 7FH or When 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←(3,2),PC←(1,0)	U	U	U	U	U	U
		reg16	1 1 1 1 0 1 1 1	1 1 1 1 1 reg	2		temp· DW, AW When temp ÷ reg16 > 0 and reg16 > 7FFFH or temp ÷ reg16 > 0 and temp ÷ reg16 < 0 to 7FFFH–1 DW· temp%reg16, AW· temp ÷ reg16 When temp ÷ reg16 >0 and temp ÷ reg16 > 7FFFH or When 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←(3,2),PC←(1,0)	U	U	U	U	U	U
		mem16	1 1 1 1 0 1 1 1	mod 1 1 1 mem	2 - 4		temp· DW, AW When temp ÷ (mem16) > 0 and (mem16) > 7FFFH or temp ÷ (mem16) > 0 and temp ÷ (mem16) < 0 to 7FFFH–1 DW· temp%(mem16), AW· temp ÷ (mem16) When temp ÷ (mem16) >0 and temp ÷ (mem16) > 7FFFH or When 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←(3,2),PC←(1,0)	U	U	U	U	U	U

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0		no. of clocks	operation	flags AC	CY	V	P	S	Z
Unsigned division instructions	DIVU	reg8	1 1 1 1 0 1 1 0	1 1 1 1 0 reg	2		temp ← AW When temp ÷ reg8 > 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←(3,2),PC←(1,0)	U	U	U	U	U	U
		mem8	1 1 1 1 0 1 1 0	mod 1 1 0 mem	2 - 4		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←(3,2),PC←(1,0)	U	U	U	U	U	U
		reg16	1 1 1 1 0 1 1 1	1 1 1 1 0 reg	2		temp ← DW, AW When temp ÷ reg16 > FFFFH DW ← temp%reg16, AW ← temp ÷ reg16 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←(3,2),PC←(1,0)	U	U	U	U	U	U
		mem16	1 1 1 1 0 1 1 1	mod 1 1 0 mem	2 - 4		temp ← DW, AW When temp ÷ (mem16) > FFFFH DW ← temp%(mem16), AW ← temp ÷ (mem16) 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←(3,2),PC←(1,0)	U	U	U	U	U	U

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	CY	V	P	S	Z
BCD complement instructions	ADJBA		0 0 1 1 0 1 1 1		1		When (AL∧ 0FH)> 9 or AC = AL· AL + 6 AH←AH + 1,AC←1,CY←AC,AL←AL∧0FH	×	×	U	U	U	U
	ADJ4A		0 0 1 0 0 1 1 1		1		When (AL∧ 0FH)> 9 or AC = 1 AL←AL + 6,CY←CY∨AC,AC←1 When AL>9FH or CY = 1 AL←AL + 60H,CY←1	×	×	U	×	×	×
	ADJBS		0 0 1 1 1 1 1 1		1		When (AL∧ 0FH)> 9 or AC = 1 AL←AL − 6,AH←AH − 1,AC←1 CY←AC,AL←AL∧0FH	×	×	U	U	U	U
	ADJ4S		0 0 1 0 1 1 1 1		1		When (AL∧ 0FH)> 9 or AC = 1 AL←AL − 6,CY←CY∨AC,AC←1 When AL> 9FH or CY = 1 AL←AL − 60H,CY←1	×	×	U	×	×	×
Data Conversion instructions	CVTBD		1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	2		AH←AL ÷ 0AH,AL←AL % 0AH	U	U	U	×	×	×
	CVTDB		1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	2		AL←AH × 0AH + AL, AH←0	U	U	U	×	×	×
	CVTBW		1 0 0 1 1 0 0 0		1		When AL < 80H, AH · 0, all other times AH · FFH						
	CVTWL		1 0 0 1 1 0 0 1		1		When AW < 8000H, DW · 0, all other times DW · FFFFH						
Comparison instructions	CMP	reg,reg	0 0 1 1 1 0 1 W	1 1 reg reg	2		reg − reg	×	×	×	×	×	×
		mem,reg	0 0 1 1 1 0 0 W	mod reg mem	2 - 4		(mem) − reg	×	×	×	×	×	×
		reg,mem	0 0 1 1 1 0 1 W	mod reg mem	2 - 4		reg − (mem)	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 S W	1 1 1 1 1 reg	3 - 4		reg − imm	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 S W	mod 1 1 1 mem	3 - 6		(mem) − imm	×	×	×	×	×	×
		acc,imm	0 0 1 1 1 1 0 W		2 - 3		When W = 0 AL−imm When W = 1 AW−imm	×	×	×	×	×	×
Complement instructions	NOT	reg	1 1 1 1 0 1 1 W	1 1 0 1 0 reg	2		reg←$\overline{reg}$						
		mem	1 1 1 1 0 1 1 W	mod 0 1 0 mem	2 - 4		(mem)←($\overline{mem}$)						
	NEG	reg	1 1 1 1 0 1 1 W	1 1 0 1 1 reg	2		reg←$\overline{reg}$ + 1	×	×	×	×	×	×
		mem	1 1 1 1 0 1 1 W	mod 0 1 1 mem	2 - 4		(mem)←($\overline{mem}$) + 1	×	×	×	×	×	×

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	CY	V	P	S	Z
Logical operation instructions	TEST	reg,reg	1 0 0 0 0 1 0 W	1 1 reg reg	2		reg ∧ reg	U	0	0	×	×	×
		mem,reg reg,mem	1 0 0 0 0 1 0 W	mod reg mem	2 - 4		(mem) ∧ reg	U	0	0	×	×	×
		reg,imm	1 1 1 1 0 1 1 W	1 1 0 0 0 reg	3 - 4		reg ∧ imm	U	0	0	×	×	×
		mem,imm	1 1 1 1 0 1 1 W	mod 0 0 0 mem	3 - 6		(mem) ∧ imm	U	0	0	×	×	×
		acc,imm	1 0 1 0 1 0 0 W		2 - 3		When W = 0 AL ∧ imm 8 When W = 1 AW ∧ imm 16	U	0	0	×	×	×
	AND	reg,reg	0 0 1 0 0 0 1 W	1 1 reg reg	2		reg←reg ∧ reg	U	0	0	×	×	×
		mem,reg	0 0 1 0 0 0 0 W	mod reg mem	2 - 4		(mem)←(mem) ∧ reg	U	0	0	×	×	×
		reg,mem	0 0 1 0 0 0 1 W	mod reg mem	2 - 4		reg←reg ∧ (mem)	U	0	0	×	×	×
		reg,imm	1 0 0 0 0 0 0 W	1 1 1 0 0 reg	3 - 4		reg←reg ∧ imm	U	0	0	×	×	×
		mem,imm	1 0 0 0 0 0 0 W	mod 1 0 0 mem	3 - 6		(mem)←(mem) ∧ imm	U	0	0	×	×	×
		acc,imm	0 0 1 0 0 1 0 W		2 - 3		When W = 0 AL· AL ∧ imm 8 When W = 1 AW· AW ∧ imm 16	U	0	0	×	×	×
	OR	reg,reg	0 0 0 0 1 0 1 W	1 1 reg reg	2		reg←reg ∨ reg	U	0	0	×	×	×
		mem,reg	0 0 0 0 1 0 0 W	mod reg mem	2 - 4		(mem)←(mem) ∨ reg	U	0	0	×	×	×
		reg,mem	0 0 0 0 1 0 1 W	mod reg mem	2 - 4		reg←reg ∨ (mem)	U	0	0	×	×	×
		reg,imm	1 0 0 0 0 0 0 W	1 1 0 0 1 reg	3 - 4		reg←reg ∨ imm	U	0	0	×	×	×
		mem,imm	1 0 0 0 0 0 0 W	mod 0 0 1 mem	3 - 6		(mem)←(mem) ∨ imm	U	0	0	×	×	×
		acc,imm	0 0 0 0 1 1 0 W		2 - 3		When W = 0 AL · AL ∨ imm 8 When W = 1 AW · AW ∨ imm 16	U	0	0	×	×	×
	XOR	reg,reg	0 0 1 1 0 0 1 W	1 1 reg reg	2		reg←reg ∀ reg	U	0	0	×	×	×
		mem,reg	0 0 1 1 0 0 0 W	mod reg mem	2 - 4		(mem)←(mem) ∀ reg	U	0	0	×	×	×
		reg,mem	0 0 1 1 0 0 1 W	mod reg mem	2 - 4		reg←reg ∀ (mem)	U	0	0	×	×	×
		reg,imm	1 0 0 0 0 0 0 W	1 1 1 1 0 reg	3 - 4		reg←reg ∀ imm	U	0	0	×	×	×
		mem,imm	1 0 0 0 0 0 0 W	mod 1 1 0 mem	3 - 6		(mem)←(mem) ∀ imm	U	0	0	×	×	×
		acc,imm	0 0 1 1 0 1 0 W		2 - 3		When W = 0 AL· AL ∀ imm 8 When W = 1 AW· AW ∀ imm 16	U	0	0	×	×	×

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	CY	V	P	S	Z
Bit operation instructions	TEST1	reg8,CL	0 0 0 1 0 0 0 0	1 1 0 0 0 reg	3		reg8 bit NO.CL = 0 : Z← 1 = 1 : Z← 0	U	0	0	U	U	×
		mem8,CL	0 0 0 0	mod 0 0 0 mem	3 5		(mem8) bit NO.CL = 0 : Z← 1 = 1 : Z← 0	U	0	0	U	U	×
		reg16,CL	0 0 0 1	1 1 0 0 0 reg	3		reg16 bit NO.CL = 0 : Z← 1 = 1 : Z← 0	U	0	0	U	U	×
		mem16,CL	0 0 0 1	mod 0 0 0 mem	3 5		(mem16) bit NO.CL = 0 : Z← 1 = 1 : Z← 0	U	0	0	U	U	×
		reg8,imm3	1 0 0 0	1 1 0 0 0 reg	4		reg8 bit NO.imm3 = 0 : Z← 1 = 1 : Z← 0	U	0	0	U	U	×
		mem8,imm3	1 0 0 0	mod 0 0 0 mem	4 6		(mem8) bit NO.imm3 = 0 : Z← 1 1 = 1 : Z← 0 0	U	0	0	U	U	×
		reg16,imm4	1 0 0 1	1 1 0 0 0 reg	4		reg16 bit NO.imm4 = 0 : Z← 1 = 1 : Z← 0	U	0	0	U	U	×
		mem16,imm4	1 0 0 1	mod 0 0 0 mem	4 6		(mem16) bit NO.imm4 = 0 : Z← 1 1 = 1 : Z← 0 0	U	0	0	U	U	×
	NOT1	reg8,CL	0 1 1 0	1 1 0 0 0 reg	3		reg8 bit NO.CL← $\overline{\text{reg8 bit NO.CL}}$						
		mem8,CL	0 1 1 0	mod 0 0 0 mem	3 5		(mem8) bit NO.CL← $\overline{\text{(mem8) bit NO.CL}}$						
		reg16,CL	0 1 1 1	1 1 0 0 0 reg	3		reg16 bit NO.CL← $\overline{\text{16 bit NO.CL}}$						
		mem16,CL	0 1 1 1	mod 0 0 0 mem	3 5		(mem16) bit NO.CL← $\overline{\text{(mem16) bit NO.CL}}$						
		reg8,imm3	1 1 1 0	1 1 0 0 0 reg	4		reg8 bit NO.imm3← $\overline{\text{reg8 bit NO.imm3}}$						
		mem8,imm3	1 1 1 0	mod 0 0 0 mem	4 6		(mem8) bit NO.imm3← $\overline{\text{(mem8) bit NO.imm3}}$						
		reg16,imm4	1 1 1 1	1 1 0 0 0 reg	4		reg16 bit NO.imm4← $\overline{\text{reg16 bit NO.imm4}}$						
		mem16,imm4	↓ 1 1 1 1	mod 0 0 0 mem	4 6		(mem)16 bit NO.imm4← $\overline{\text{(mem16) bit NO.imm4}}$						

2nd byte 3rd byte *1st byte – OFH

	NOT1	CY	1 1 1 1 0 1 0 1		1		CY←$\overline{\text{CY}}$		×				

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	CY	V	P	S	Z
Bit operation instructions	CLR1	reg8,CL	0 0 0 1 0 0 1 0	1 1 0 0 0 reg	3		reg8 bit NO.CL← 0						
		mem8,CL	0 0 1 0	mod 0 0 0 mem	3 - 5		(mem8) bit NO.CL← 0						
		reg16,CL	0 0 1 1	1 1 0 0 0 reg	3		reg16 bit NO.CL← 0						
		mem16,CL	0 0 1 1	mod 0 0 0 mem	3 - 5		(mem16) bit NO.CL← 0						
		reg8,imm3	1 0 1 0	1 1 0 0 0 reg	4		reg8 bit NO.imm3← 0						
		mem8,imm3	1 0 1 0	mod 0 0 0 mem	4 - 6		(mem8) bit NO.imm3← 0						
		reg16,imm4	1 0 1 1	1 1 0 0 0 reg	4		reg16 bit NO.imm4← 0						
		mem16,imm4	1 0 1 1	mod 0 0 0 mem	4 - 6		(mem16) bit NO.imm4← 0						
	SET1	reg8,CL	0 1 0 0	1 1 0 0 0 reg	3		reg8 bit NO.CL← 1						
		mem8,CL	0 1 0 0	mod 0 0 0 mem	3 - 5		(mem8) bit NO.CL← 1						
		reg16,CL	0 1 0 1	1 1 0 0 0 reg	3		reg16 bit NO.CL← 1						
		mem16,CL	0 1 0 1	mod 0 0 0 mem	3 - 5		(mem16) bit NO.CL← 1						
		reg8,imm3	1 1 0 0	1 1 0 0 0 reg	4		reg8 bit NO.imm3← 1						
		mem8,imm3	1 1 0 0	mod 0 0 0 mem	4 - 6		(mem8) bit NO.imm3← 1						
		reg16,imm4	1 1 0 1	1 1 0 0 0 reg	4		reg16 bit NO.imm4← 1						
		mem16,imm4	1 1 0 1	mod 0 0 0 mem	4 - 6		(mem16) bit NO.imm4← 1						

2nd byte 3rd byte *1st byte – OFH

	CLR1	CY	1 1 1 1 1 0 0 0		1		CY←0		0				
		DIR	1 1 1 1 1 1 0 0		1		DIR←0						
	SET1	CY	1 1 1 1 1 0 0 1		1		CY←1		1				
		DIR	1 1 1 1 1 1 0 1		1		DIR←1						

instruction group	mnemonic	operand	operation code		no. of bytes	no. of clocks	operation	flags					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Shift instructions	SHL	reg,1	1 1 0 1 0 0 0 W	1 1 1 0 0 reg	2		MSB of CY ← reg, reg ← regX2 When MSB of reg = CY V ← 1 When MSB of reg = CY V ← 0	U	×	×	×	×	×
		mem,1	1 1 0 1 0 0 0 W	mod 1 0 0 mem	2 - 4		MSB of CY ← (mem), (mem) ← (mem)X2 When MSB of (mem) = CY, V ← 1 When MSB of (mem) = CY, V ← 0	U	×	×	×	×	×
		reg,CL	1 1 0 1 0 0 1 W	1 1 1 0 0 reg	2		temp + CL, while temp = 0, repeat this operation. MSB of CY ← reg, reg ← regX2 temp ← temp − 1	U	×	U	×	×	×
		mem,CL	1 1 0 1 0 0 1 W	mod 1 0 0 mem	2 - 4		temp + CL, while temp = 0, repeat this operation. MSB of CY ← (mem), (mem) ← (mem)X2 temp ← temp − 1	U	×	U	×	×	×
		reg,imm8	1 1 0 0 0 0 0 W	1 1 1 0 0 reg	3		temp ← imm8, while temp = 0, repeat this operation. MSB of CY ← reg, reg ← regX2 temp ← temp − 1	U	×	U	×	×	×
		mem,imm8	1 1 0 0 0 0 0 W	mod 1 0 0 mem	3 - 5		temp ← imm8, while temp = 0, repeat this operation. MSB of CY ← (mem), (mem) ← (mem)X2 temp ← temp − 1	U	×	U	×	×	×

n: number of shifts

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	CY	V	P	S	Z
Shift instructions	SHR	reg,1	1 1 0 1 0 0 0 W	1 1 1 0 1 reg	2		CY ← LSB of reg, reg ← reg – 2 MSB of reg = bit following MSB of reg: V ← 1 MSB of reg = bit following MSB of reg: V ← 0	U	×	×	×	×	×
		mem,1	1 1 0 1 0 0 0 W	mod 1 0 1 mem	2 4		CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) = bit following MSB of (mem): V ← 1 MSB of (mem) = bit following MSB of (mem): V ← 0	U	×	×	×	×	×
		reg,CL	1 1 0 1 0 0 1 W	1 1 1 0 1 reg	2		temp ← CL, while temp = 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) ÷ 2 temp ← temp – 1	U	×	U	×	×	×
		mem,CL	1 1 0 1 0 0 1 W	mod 1 0 1 mem	2 4		temp ← CY, while temp = 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) ÷ 2 temp ← temp – 1	U	×	U	×	×	×
		reg,imm8	1 1 0 0 0 0 0 W	1 1 1 0 1 reg	3		temp ← imm8, while temp = 0, repeat this operation. CY ← LSB of reg, reg ← reg ÷ 2 temp ← temp – 1	U	×	U	×	×	×
		mem,imm8	1 1 0 0 0 0 0 W	mod 1 0 1 mem	3 5		temp ← imm8, while temp = 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) ÷ 2 temp ← temp – 1	U	×	U	×	×	×
	SHRA	reg,1	1 1 0 1 0 0 0 W	1 1 1 1 1 reg	2		CY ← LSB of reg, reg ← reg – 2, V ← 0 MSB of operand does not change.	U	×	0	×	×	×
		mem,1	1 1 0 1 0 0 0 W	mod 1 1 1 mem	2 4		CY ← MSB of (mem), (mem) ← (mem) – 2, V ← 0 MSB of operand does not change	U	×	0	×	×	×
		reg,CL	1 1 0 1 0 0 1 W	1 1 1 1 1 reg	2		temp ← CL, while temp = 0, repeat this operation. CL ← LSB of reg, reg ← reg ÷ 2 temp ← temp – 1, MSB of operand does not change.	U	×	U	×	×	×
		mem,CL	1 1 0 1 0 0 1 W	mod 1 1 1 mem	2 4		temp ← CL, while temp = 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) ÷ 2 temp ← temp – 1, MSB of operand does not change.	U	×	U	×	×	×
		reg,imm8	1 1 0 0 0 0 0 W	1 1 1 1 1 reg	3		temp ← imm8, while temp = 0, repeat this operation. CL ← LSB of reg, reg ← reg ÷ 2 temp ← temp – 1, MSB of operand does not change.	U	×	U	×	×	×
		mem,imm8	1 1 0 0 0 0 0 W	mod 1 1 1 mem	3 5		temp ← imm8, while temp = 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) ÷ 2 temp ← temp – 1, MSB of operand does not change.	U	×	U	×	×	×

n: number of shifts

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	flags CY	flags V	flags P	flags S	flags Z
Rotation instructions	ROL	reg,1	1 1 0 1 0 0 0 W	1 1 0 0 0 reg	2		CY ← MSB of reg, reg ← reg x 2 + CY MSB of reg = CY: V ← 1 MSB of reg = CY: V ← 0		×	×			
		mem,1	1 1 0 1 0 0 0 W	mod 0 0 0 mem	2 4		CY ← MSB of (mem), (mem) ← (mem) x 2 + CY MSB of (mem) = CY: V ← 1 MSB of (mem) = CY: V ← 0		×	×			
		reg,CL	1 1 0 1 0 0 1 W	1 1 0 0 0 reg	2		temp ← CL, while temp = 0, repeat this operation. CY ← MSB of reg, reg ← reg x 2 + CY temp ← temp –1		×	U			
		mem,CL	1 1 0 1 0 0 1 W	mod 0 0 0 mem	2 4		temp ← CL, while temp = 0, repeat this operation. CY ← MSB of (mem), (mem) ← (mem) x 2 + CY temp ← temp –1		×	U			
		reg,imm8	1 1 0 0 0 0 0 W	1 1 0 0 0 reg	3		temp ← imm8, while temp = 0, repeat this operation. CY ← MSB of reg, reg ← reg x 2 + CY temp ← temp –1		×	U			
		mem,imm8	1 1 0 0 0 0 0 W	mod 0 0 0 mem	3 5		temp ← imm8, while temp = 0, repeat this operation. CY ← MSB of (mem), (mem) ← (mem) x 2 + CY temp ← temp –1		×	U			
	ROR	reg,1	1 1 0 1 0 0 0 W	1 1 0 0 1 reg	2		CY ← LSB of reg, reg – reg x 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		×	×			
		mem,1	1 1 0 1 0 0 0 W	mod 0 0 1 mem	2 4		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		×	×			
		reg,CL	1 1 0 1 0 0 1 W	1 1 0 0 1 reg	2		temp ← CL, while temp = 0, repeat this operation. CY ← LSB of reg, reg ← reg ÷ 2 MSB of (mem) ← CY temp ← temp – 1		×	U			
		mem,CL	1 1 0 1 0 0 1 W	mod 0 0 1 mem	2 4		temp ← CL, while temp = 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← CY temp ← temp –1		×	U			
		reg,imm8	1 1 0 0 0 0 0 W	1 1 0 0 1 reg	3		temp ← imm8, while CL = 0, repeat this operation. CY ← LSB of reg, reg ← reg ÷ 2 MSB of reg ← CY temp ← temp –1		×	U			
		mem,imm8	1 1 0 0 0 0 0 W	mod 0 0 1 mem	3 5		temp ← imm8, while CL = 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← CY temp ← temp –1		×	U			

n: number of shifts

instruction group	mnemonic	operand	operation code		no. of bytes	no. of clocks	operation	flags					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Rotate instructions	ROLC	reg,1	1 1 0 1 0 0 0 W	1 1 0 1 0 reg	2		tmpcy ← CY, CY ← MSB of reg reg ← reg x 2 + tmpcy MSB of reg = CY: V ← 1 MSB of reg = CY: V ← 0		×	×			
		mem,1	1 1 0 1 0 0 0 W	mod 0 1 0 mem	2 - 4		tmpcy ← CY, CY ← MSB of (mem) (mem) ← (mem) x 2 + tmpcy MSB of (mem) = CY: V ← 1 MSB of (mem) = CY: V ← 0		×	×			
		reg,CL	1 1 0 1 0 0 1 W	1 1 0 1 0 reg	2		temp ← CL, while CY = 0, repeat this operation tmpcy ← CY, CY ← MSB of reg reg ← reg x 2 + tmpcy temp ← temp – 1		×	U			
		mem,CL	1 1 0 1 0 0 1 W	mod 0 1 0 mem	2 - 4		temp ← CL, while CY = 0, repeat this operation tmpcy ← CY, CY – MSB of (mem) (mem) ← (mem) x 2 + tmpcy temp ← temp –1		×	U			
		reg,imm8	1 1 0 0 0 0 0 W	1 1 0 1 0 reg	3		temp ← imm8, while CL = 0, repeat this operation tmpcy ← CY, CY ← MSB of reg reg ← reg x 2 + tmpcy temp ← temp – 1		×	U			
		mem,imm8	1 1 0 0 0 0 0 W	mod 0 1 0 mem	3 - 5		temp ← imm8, while CL = 0, repeat this operation tmpcy ← CY, CY ← MSB of (mem) reg ← (mem) x 2 + tmpcy temp ← temp – 1		×	U			

n: number of shifts

instruction group	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	CY	V	P	S	Z
Rotate instruction	RORC	reg,1	1 1 0 1 0 0 0 W	1 1 0 1 1 reg	2		tmpcy ← CY, CY ← LSB of CY 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		×	×			
		mem,1	1 1 0 1 0 0 0 W	mod 0 1 1 mem	2–4		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		×	×			
		reg,CL	1 1 0 1 0 0 1 W	1 1 0 1 1 reg	2		temp ← CL, while CL = 0, repeat this operation tmpcy ← CY, CY ← LSB of reg reg ← reg – 2 MSB of reg ← tmpcy temp ← temp – 1		×	U			
		mem,CL	1 1 0 1 0 0 1 W	mod 0 1 1 mem	2–4		temp ← CL, while CL = 0, repeat this operation tmpcy ← CY, CY ← LSB of (mem) (mem) ← (mem) – 2 MSB of (mem) ← tmpcy temp ← temp – 1		×	U			
		reg,imm8	1 1 0 0 0 0 0 W	1 1 0 1 1 reg	3		temp ← imm8, while CL = 0, repeat this operation tmpcy ← CY, CY ← LSB of reg reg ← reg – 2 MSB of reg ← tmpcy temp ← temp – 1		×	U			
		mem,imm8	1 1 0 0 0 0 0 W	mod 0 1 1 mem	3–5		temp ← imm8, while CL = 0, repeat this operation tmpcy ← CY, CY ← LSB of (mem) (mem) ← (mem) – 2 MSB of (mem) ← tmpcy temp ← temp – 1		×	U			

n: number of shifts

instruction group 群	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation	flags AC	CY	V	P	S	Z
Subroutine control instructions	CALL	near proc	1 1 1 0 1 0 0 0		3		(SP−1,SP−2)←PC,SP←SP−2 PC←PC+disp						
		regptr16	1 1 1 1 1 1 1 1	1 1 0 1 0 reg	2		(SP−1,SP−2)←PC,PC←regptr16 SP←SP−2						
		memptr16	1 1 1 1 1 1 1 1	mod 0 1 0 mem	2 4		(SP−1,SP−2)←PC,SP←SP−2 PC←(memptr16)						
		far proc	1 0 0 1 1 0 1 0		5		(SP−1,SP−2)←PS,(SP−3,SP−4)←PC SP←SP−4 PS←seg,PC←offset						
		memptr32	1 1 1 1 1 1 1 1	mod 0 1 1 mem	2 4		(SP−1,SP−2)←PS,(SP−3,SP−4)←PC SP←SP−4 PS←(memptr32+2),PC←(memptr32)						
	RET		1 1 0 0 0 0 1 1		1		PC←(SP+1,SP) SP←SP+2						
		pop value	1 1 0 0 0 0 1 0		3		PC←(SP+1,SP) SP←SP+2,SP←SP+pop value						
			1 1 0 0 1 0 1 1		1		PC←(SP+1,SP) PS←(SP+3,SP+2) SP←SP+4						
		pop value	1 1 0 0 1 0 1 0		3		PC←(SP+1,SP) PS←(SP+3,SP+2) SP←SP+4,SP←SP+pop value						

命令群	mnemonic	operand	operation code		no. of bytes	no. of clocks	operation	flags					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Stack manipulation instructions	PUSH	mem16	1 1 1 1 1 1 1 1	mod 1 1 0 mem	2 4		(SP−1,SP−2)←(mem16) SP←SP−2						
		reg16	0 1 0 1 0 reg		1		(SP−1,SP−2)←reg16 SP←SP−2						
		sreg	0 0 0 sreg 1 1 0		1		(SP−1,SP−2)←sreg SP←SP−2						
		PSW	1 0 0 1 1 1 0 0		1		(SP−1,SP−2)←PSW SP←SP−2						
		R	0 1 1 0 0 0 0 0		1		Push registers on the stack						
		imm	0 1 1 0 1 0 S 0		2 3		(SP−1,SP−2)←imm SP←SP−2, When S = 1, sign expansion						
	POP	mem16	1 0 0 0 1 1 1 1	mod 0 0 0 mem	2 4		(mem16)←(SP+1,SP) SP←SP+2						
		reg16	0 1 0 1 1 reg		1		reg16←(SP+1,SP) SP←SP+2						
		sreg	0 0 0 sreg 1 1 1		1		sreg←(SP+1,SP) SP←SP+2 sreg : SS,DS0,DS1						
		PSW	1 0 0 1 1 1 0 1		1		PSW←(SP+1,SP) SP←SP+2	R	R	R	R	R	R
		R	0 1 1 0 0 0 0 1		1		Pop registers from the stack						
	PREPARE	imm16,imm8	1 1 0 0 1 0 0 0		4		Prepare New Stack Frame						
	DISPOSE		1 1 0 0 1 0 0 1		1		Dispose of Stack Frame						
Branch instructions	BR	near label	1 1 1 0 1 0 0 1		3		PC←PC+disp						
		short label	1 1 1 0 1 0 1 1		2		PC←PC+ext disp8						
		regptr16	1 1 1 1 1 1 1 1	1 1 1 0 0 reg	2		PC←regptr16						
		memptr16	1 1 1 1 1 1 1 1	mod 1 0 0 mem	2 4		PC←(memptr16)						
		far label	1 1 1 0 1 0 1 0		5		PS←seg PC←offset						
		memptr32	1 1 1 1 1 1 1 1	mod 1 0 1 mem	2 4		PS←(memptr32+2) PC←(memptr32)						

命令群	mnemonic	operand	operation code 7 6 5 4 3 2 1 0	operation code 7 6 5 4 3 2 1 0	no. of bytes	no. of clocks	operation		flags AC	CY	V	P	S	Z
Conditional branch instructions	BV	short label	0 1 1 1 0 0 0 0		2		if V = 1	PC←PC + ext disp8						
	BNV	〃	0 0 0 1		〃		if V = 0	〃						
	BC BL	〃	0 0 1 0		〃		if CY = 1	〃						
	BNC BNL	〃	0 0 1 1		〃		if CY = 0	〃						
	BE BZ	〃	0 1 0 0		〃		if Z = 1	〃						
	BNE BNZ	〃	0 1 0 1		〃		if Z = 0	〃						
	BNH	〃	0 1 1 0		〃		if CY∨Z = 1	〃						
	BH	〃	0 1 1 1		〃		if CY∨Z = 0	〃						
	BN	〃	1 0 0 0		〃		if S = 1	〃						
	BP	〃	1 0 0 1		〃		if S = 0	〃						
	BPE	〃	1 0 1 0		〃		if P = 1	〃						
	BPO	〃	1 0 1 1		〃		if P = 0	〃						
	BLT	〃	1 1 0 0		〃		if S∀V = 1	〃						
	BGE	〃	1 1 0 1		〃		if S∀V = 0	〃						
	BLE	〃	1 1 1 0		〃		if (S∀V)∨Z = 1	〃						
	BGT	〃	1 1 1 1		〃		if (S∀V)∨Z = 0	〃						
	DBNZNE	〃	1 1 1 0 0 0 0 0		〃		CW = CW − 1 if Z = 0 and CW ≠ 0	〃						
	DBNZE	〃	0 0 0 1		〃		CW = CW − 1 if Z = 1 and CW ≠ 0	〃						
	DBNZ	〃	0 0 1 0		〃		CW = CW − 1 if CW ≠ 0	〃						
	BCWZ	〃	0 0 1 1		〃		if CW = 0	〃						
	BTCLR *	mem8 imm3 short-label	0 0 0 0 1 1 1 1	1 0 0 1 1 1 0 0	5		If special register bit = 1 Special register bit ← 0	〃						

*Newly added instruction for the μPD70322/70320.

instruction group	mnemonic	operand	operation code		no. of bytes	no. of clocks	operation	flags					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Interrupt instructions	BRK	3	1 1 0 0 1 1 0 0		1		(SP−1,SP−2)←PSW,(SP−3,SP−4)←PS (SP−5,SP−6)←PC,SP←SP−6 IE←0,BRK←0 PS←(15,14),PC←(13,12)						
		imm8 (≠3)	1 1 0 0 1 1 0 1		2		(SP−1,SP−2)←PSW,(SP−3,SP−4)←PS, (SP−5,SP−6)←PC,SP←SP−6 IE←0,BRK←0 PS←(n×4+3,n×4+2),PC←(n×4+1,n×4) n=imm8						
	BRKV		1 1 0 0 1 1 1 0		1		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 PS←(19,18),PC←(17,16)						
	RETI		1 1 0 0 1 1 1 1		1		PC←(SP+1,SP),PS←(SP+3,SP+2), PSW←(SP+5,SP+4),SP←SP+6	R	R	R	R	R	R
	RETRBI*		0 0 0 0 1 1 1 1	1 0 0 1 0 0 0 1	2		PC←Save PC,PSW←Save PSW	R	R	R	R	R	R
	FINT *		0 0 0 0 1 1 1 1	1 0 0 1 0 0 1 0	2		Indicates that interrupt service routine to the interrupt controller built in the CPU has been completed						
	CHKIND	reg16,mem32	0 1 1 0 0 0 1 0	mod reg mem	2-4		When (mem32) >reg 16 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)						

*Newly added instruction for the μPD70322/70320.

instruction group	mnemonic	operand	operation code		no. of bytes	no. of clocks	operation	flags				
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
CPU control instructions	HALT		1 1 1 1 0 1 0 0		1		CPU Halt						
	STOP *2		0 0 0 0 1 1 1 1	1 0 0 1 1 1 1 0	2		CPU Stop						
	POLL		1 0 0 1 1 0 1 1		1		Poll and wait n: No. of samplings of $\overline{\text{POLL}}$ pin						
	DI		1 1 1 1 1 0 1 0		1		IE← 0						
	EI		1 1 1 1 1 0 1 1		1		IE← 1						
	BUSLOCK		1 1 1 1 0 0 0 0		1		Bus Lock Prefix						
	FPO1	fp-op	1 1 0 1 1 X X X	1 1 Y Y Y Z Z Z	2		No Operation						
	*3	fp-op,mem	1 1 0 1 1 X X X	mod Y Y Y mem	2 - 4		data bus ← (mem)						
	FPO2	fp-op	0 1 1 0 0 1 1 X	1 1 Y Y Y Z Z Z	2		No Operation						
	*3	fp-op,mem	0 1 1 0 0 1 1 X	mod Y Y Y mem	2 - 4		data bus ← (mem)						
	NOP		1 0 0 1 0 0 0 0		1		No Operation						
	*1		0 0 1 sreg 1 1 0		1		Segment override prefix						

*1: DS0:, DS1:, PS:, SS:
*2: Newly added instruction for the μPD70322/70320
*3: Does not execute on the μPD70322/70320, but generates an interrupt.

EXPLANATION OF INSTRUCTIONS

CONTENTS

The following sections include instruction formats, descriptions, and examples for the μPD70320/70322 instruction set.

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

7							0
1	0	0	0	1	0	1	W

1	1	reg	reg

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

7							0
1	0	0	0	1	0	0	W

mod	reg	mem

(disp-low)

(disp-high)

(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

7							0
1	0	0	0	1	0	1	W
mod		reg			mem		
(disp-low)							
(disp-high)							

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

7							0
1	1	0	0	0	1	1	W
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm8- or imm16-low							
imm16-high							

(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

7				3	2		0
1	0	1	1	W		reg	

imm8- or imm16-low

imm16-high

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

7							0
1	0	1	0	0	0	0	W

addr-low

addr-high

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

7							0
1	0	1	0	0	0	1	W

addr-low

addr-high

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

7							0
1	0	0	0	1	1	1	0

1	1	0	sreg	reg

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

7							0
1	0	0	0	1	1	1	0

mod	0	sreg	mem

(disp-low)

(disp-high)

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

7							0
1	0	0	0	1	1	0	0

1	1	0	sreg	reg

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

7							0
1	0	0	0	1	1	0	0
mod		0	sreg			mem	
(disp-low)							
(disp-high)							

(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

7							0
1	1	0	0	0	1	0	1
mod			reg			mem	
(disp-low)							
(disp-high)							

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

7							0
1	1	0	0	0	1	0	0

mod	reg	mem

(disp-low)

(disp-high)

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

7							0
1	0	0	1	1	1	1	1

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

7							0
1	0	0	1	1	1	1	0

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

7							0
1	0	0	0	1	1	0	1

mod	reg	mem

(disp-low)

(disp-high)

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

7							0
1	1	0	1	0	1	1	1

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

7							0
1	0	0	0	0	1	1	W

1	1	reg			reg		

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

7							0
1	0	0	0	0	1	1	W

mod	reg	mem

(disp-low)

(disp-high)

(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

7							0
1	0	0	1	0	reg		

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

7							0
0	1	1	0	0	1	0	1

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

7							0
0	1	1	0	0	1	0	0

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)

7							0
1	1	1	1	0	0	1	1

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

7							0
1	1	1	1	0	0	1	0

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

7							0
1	0	1	0	0	1	0	W

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

7							0
1	0	1	0	0	1	1	W

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

7							0
1	0	1	0	1	1	1	W

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

7							0
1	0	1	0	1	1	0	W

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

7							0
1	0	1	0	1	0	1	W

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)

7							0
0	0	0	0	1	1	1	1
0	0	1	1	0	0	0	1
1	1	reg2				reg1	

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.

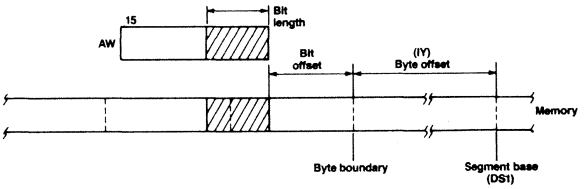
Bytes: 3

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)

7							0
0	0	0	0	1	1	1	1
0	0	1	1	1	0	0	1
1	1	0	0	0	reg8		
imm4							

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.

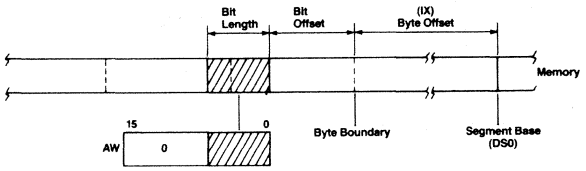
Bytes: 3

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)

7							0
0	0	0	0	1	1	1	1
0	0	1	1	1	0	1	1
1	1	0	0	0	reg		
imm4							

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

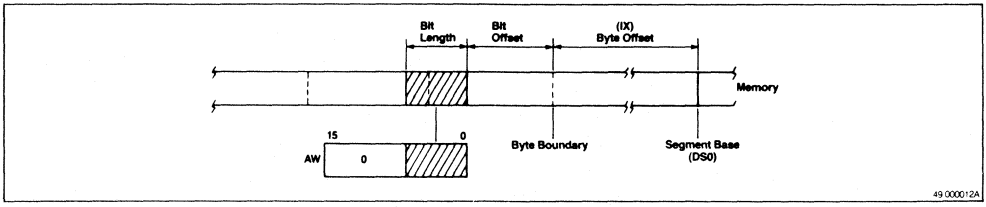
CL = 3, IX = base, AW = unknown

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

7							0
1	1	1	0	0	1	0	W

imm8

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

7							0
1	1	1	0	1	1	0	W

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

7							0
1	1	1	0	0	1	1	W

imm8

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

7							0
1	1	1	0	1	1	1	W

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

OUT DW,acc

Output to indirectly specified (by DW) I/O device

7							0
1	1	1	0	1	1	1	W

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

7							0
0	1	1	0	1	1	0	W

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

7							0
0	1	1	0	1	1	1	W

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

7							0
0	0	0	0	0	0	1	W

1	1		reg			reg	

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

7							0
0	0	0	0	0	0	0	W

mod	reg	mem

(disp-low)

(disp-high)

(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

7							0
0	0	0	0	0	0	1	W

mod	reg	mem

(disp-low)

(disp-high)

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

7							0
1	0	0	0	0	0	S	W
1	1	0	0	0	reg		
imm8 or imm16-low							
imm16-high							

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

7							0
1	0	0	0	0	0	S	W
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm8 or imm16-low							
imm16-high							

(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

7							0
0	0	0	0	0	1	0	W

imm8 or imm16-low

imm16-high

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

7							0
0	0	0	1	0	0	1	W

1	1	reg	reg

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

7							0
0	0	0	1	0	0	0	W
mod		reg			mem		
(disp-low)							
(disp-high)							

(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

7							0
0	0	0	1	0	0	1	W
mod		reg			mem		
(disp-low)							
(disp-high)							

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

7							0
1	0	0	0	0	0	S	W
1	1	0	1	0	reg		
imm8 or imm16-low							
imm16-high							

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

7							0
1	0	0	0	0	0	S	W
mod		0	1	0	mem		
(disp-low)							
(disp-high)							
imm8 or imm16-low							
imm16-high							

(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

7							0
0	0	0	1	0	1	0	W

imm8 or imm16-low

imm16-high

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

7							0
0	0	1	0	1	0	1	W

1	1	reg	reg

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

7							0
0	0	1	0	1	0	0	W

mod	reg	mem

(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

7							0
0	0	1	0	1	0	1	W

mod	reg	mem

(disp-low)

(disp-high)

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

7							0
1	0	0	0	0	0	S	W
1	1	1	0	1	reg		
imm8 or imm16-low							
imm16-high							

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

7							0
1	0	0	0	0	0	S	W
mod		1	0	1	mem		
(disp-low)							
(disp-high)							
imm8 or imm16-low							
imm16-high							

(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

7							0
0	0	1	0	1	1	0	W
imm8 or imm16-low							
imm16-high							

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

7							0
0	0	0	1	1	0	1	W
1	1	reg			reg		

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

7							0
0	0	0	1	1	0	0	W

mod	reg	mem

(disp-low)

(disp-high)

(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

7							0
0	0	0	1	1	0	1	W

mod	reg	mem

(disp-low)

(disp-high)

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

7							0
1	0	0	0	0	0	S	W
1	1	0	1	1	reg		
imm8 or imm16-low							
imm16-high							

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

7							0
1	0	0	0	0	0	S	W
mod		0	1	1	mem		
(disp-low)							
(disp-high)							
imm8 or imm16-low							
imm16-high							

(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

7							0
0	0	0	1	1	1	0	W

imm8 or imm16-low

imm16-high

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.

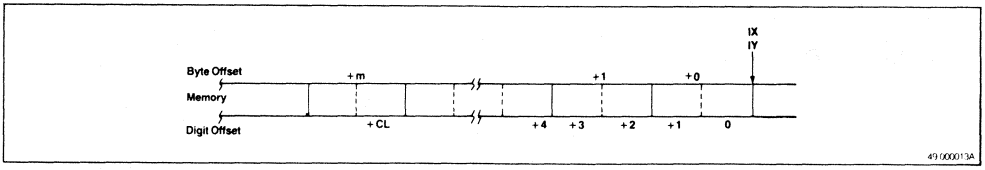
Bytes: 2

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.

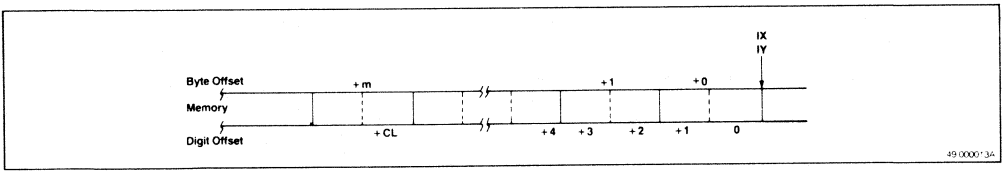
Bytes: 2

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

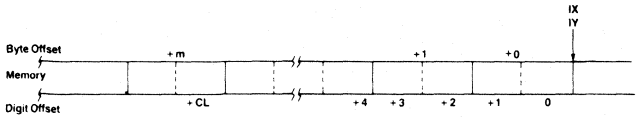
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U	U	X	U	U	X

Example:

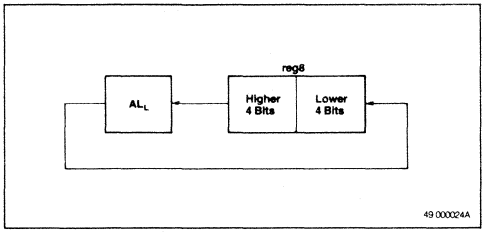
```
;μPD70116 BCD string operation
  MOV      AW,PS      ;Set both data
                      ;segments to
  MOV      DS0,AW     ;same as program
  MOV      DS1,AW     ;segment
  MOV      IX,OFFSET STR0
                      ;Point to BCD strings
  MOV      IY,OFFSET STR1
  MOV      CL,8       ;Eight 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

7							0
0	0	0	0	1	1	1	1
0	0	1	0	1	0	0	0
1	1	0	0	0	reg		



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

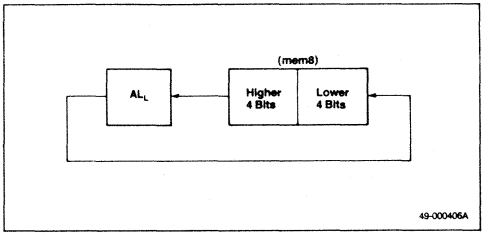
Example:

```
MOV    BL,95H
MOV    AL,03H
ROL4   BL   ;BL = 53H, AL = X9H
```

ROL4 mem8

Rotate left nibble, 8-bit memory

7							0
0	0	0	0	1	1	1	1
0	0	1	0	1	0	0	0
mod		0	0	0	mem		
(disp-low)							
(disp-high)							



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

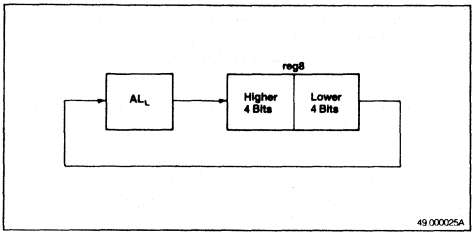
Example:

```
MOV    BYTE PTR [IX],12H
MOV    AL,03H
ROL4   [IX]   ;[IX] = 23H, AL = X1H
```

ROR4 reg8

Rotate right nibble, 8-bit register

7							0
0	0	0	0	1	1	1	1
0	0	1	0	1	0	1	0
1	1	0	0	0	reg		



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

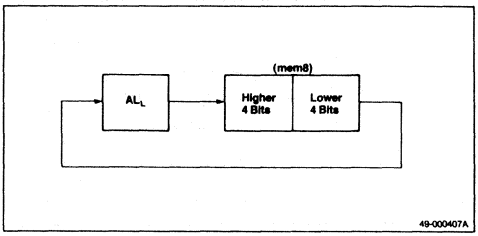
Example:

```
MOV    CL,95H
MOV    AL,03H
ROR4   CL   ;CL = 39H, AL = X5H
```

ROR4 mem8

Rotate right nibble, 8-bit memory

7							0
0	0	0	0	1	1	1	1
0	0	1	0	1	0	1	0
mod		0	0	0	mem		



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

7							0
1	1	1	1	1	1	1	0

1	1	0	0	0	reg		

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

7							0
1	1	1	1	1	1	1	W

mod	0	0	0	mem	

(disp-low)

(disp-high)

(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

7							0
0	1	0	0	0		reg	

reg16 ← reg16 + 1

Increments 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:

```
INC    BW
INC    IX
```

DEC reg8

Decrement 8-bit register

7							0
1	1	1	1	1	1	1	0

1	1	0	0	1		reg	

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

7							0
1	1	1	1	1	1	1	W

mod	0	0	1	mem

(disp-low)

(disp-high)

(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

7							0
0	1	0	0	1	reg		

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

7							0
1	1	1	1	0	1	1	0

1	1	1	0	0	reg		

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

7							0
1	1	1	1	0	1	1	0

mod		1	0	0	mem		

(disp-low)

(disp-high)

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

7							0
1	1	1	1	0	1	1	1

1	1	1	0	0	reg		

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

7							0
1	1	1	1	0	1	1	1

mod		1	0	0	mem		

(disp-low)

(disp-high)

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

7							0
1	1	1	1	0	1	1	0

1	1	1	0	1	reg		

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

7							0
1	1	1	1	0	1	1	0

mod		1	0	1	mem		

(disp-low)

(disp-high)

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

7							0
1	1	1	1	0	1	1	1

1	1	1	0	1	reg	

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

7							0
1	1	1	1	0	1	1	1

mod	1	0	1	mem

(disp-low)

(disp-high)

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

7							0
0	1	1	0	1	0	1	1

1	1	reg	reg

imm8

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

7							0
0	1	1	0	1	0	1	1

mod	reg	mem

(disp-low)

(disp-high)

imm8

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

7							0
0	1	1	0	1	0	0	1

1	1	reg	reg

imm16-low

imm16-high

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

7							0
0	1	1	0	1	0	0	1

mod	reg	mem

(disp-low)

(disp-high)

imm16-low

imm16-high

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

7							0
1	1	1	1	0	1	1	0

1	1	1	1	0	reg		

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

7							0
1	1	1	1	0	1	1	0

mod		1	1	0	mem		

(disp-low)

(disp-high)

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

7							0
1	1	1	1	0	1	1	1

1	1	1	1	0	reg

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

7							0
1	1	1	1	0	1	1	1

mod	1	1	0	mem

(disp-low)

(disp-high)

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

7							0
1	1	1	1	0	1	1	0

1	1	1	1	1	reg		

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

7							0
1	1	1	1	0	1	1	0

mod		1	1	1	mem		

(disp-low)

(disp-high)

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

7							0
1	1	1	1	0	1	1	1

1	1	1	1	1		reg	

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

7							0
1	1	1	1	0	1	1	1

mod		1	1	1	mem		

(disp-low)

(disp-high)

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

7							0
0	0	1	1	0	1	1	1

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

7							0
0	0	1	0	0	1	1	1

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

7							0
0	0	1	1	1	1	1	1

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

7							0
0	0	1	0	1	1	1	1

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

7							0
1	1	0	1	0	1	0	0

0	0	0	0	1	0	1	0

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

7							0
1	1	0	1	0	1	0	1

0	0	0	0	1	0	1	0

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

7							0
1	0	0	1	1	0	0	0

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

7							0
1	0	0	1	1	0	0	1

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

7							0
0	0	1	1	1	0	1	W

1	1	reg			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. 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

7							0
0	0	1	1	1	0	0	W

mod		reg			mem		

(disp-low)							

(disp-high)							

(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

7							0
0	0	1	1	1	0	1	W

mod	reg	mem

(disp-low)

(disp-high)

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

7							0
1	0	0	0	0	0	S	W

1	1	1	1	1	reg

imm8 or imm16-low

imm16-high

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

7							0
1	0	0	0	0	0	S	W
mod		1	1	1	mem		
(disp-low)							
(disp-high)							
imm8 or imm16-low							
imm16-high							

(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

7							0
0	0	1	1	1	1	0	W
imm8 or imm16-low							
imm16-high							

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

7							0
1	1	1	1	0	1	1	W

1	1	0	1	0	reg		

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

7							0
1	1	1	1	0	1	1	W

mod		0	1	0	mem		

(disp-low)

(disp-high)

(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

7							0
1	1	1	1	0	1	1	W

1	1	0	1	1	reg		

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

7							0
1	1	1	1	0	1	1	W

mod		0	1	1	mem		

(disp-low)

(disp-high)

(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

7							0
1	0	0	0	0	1	0	W

1	1	reg	reg

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

7							0
1	0	0	0	0	1	0	W

mod	reg	mem

(disp-low)

(disp-high)

(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

7							0
1	1	1	1	0	1	1	W
1	1	0	0	0	reg		
imm8 or imm16-low							
imm16-high							

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

7							0
1	1	1	1	0	1	1	W
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm8 or imm16-low							
imm16-high							

(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

7							0
1	0	1	0	1	0	0	W

imm8 or imm16-low

imm16-high

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

7							0
0	0	1	0	0	0	1	W

1	1	reg	reg

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

7							0
0	0	1	0	0	0	0	W

mod	reg	mem

(disp-low)

(disp-high)

(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

7							0
0	0	1	0	0	0	1	W

mod	reg	mem

(disp-low)

(disp-high)

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

7							0
1	0	0	0	0	0	0	W
1	1	1	0	0	reg		
imm8 or imm16-low							
imm16-high							

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

7							0
1	0	0	0	0	0	0	W
mod		1	0	0	mem		
(disp-low)							
(disp-high)							
imm8 or imm16-low							
imm16-high							

(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

7							0
0	0	1	0	0	1	0	W

imm8 or imm16-low

imm16-high

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

7							0
0	0	0	0	1	0	1	W

1	1	reg	reg

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

7							0
0	0	0	0	1	0	0	W

mod	reg	mem

(disp-low)

(disp-high)

(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

7							0
0	0	0	0	1	0	1	W

mod	reg	mem

(disp-low)

(disp-high)

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

7							0
1	0	0	0	0	0	0	W
1	1	0	0	1	reg		
imm8 or imm16-low							
imm16-high							

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

7							0
1	0	0	0	0	0	0	W
mod		0	0	1	mem		
(disp-low)							
(disp-high)							
imm8 or imm16-low							
imm16-high							

(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

7							0
0	0	0	0	1	1	0	W

imm8 or imm16-low

imm16-high

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

7							0
0	0	1	1	0	0	1	W

1	1	reg	reg

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

7							0
0	0	1	1	0	0	0	W
mod		reg			mem		
(disp-low)							
(disp-high)							

(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

7							0
0	0	1	1	0	0	1	W
mod		reg			mem		
(disp-low)							
(disp-high)							

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

7							0
1	0	0	0	0	0	0	W
1	1	1	1	0	reg		
imm8 or imm16-low							
imm16-high							

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

7							0
1	0	0	0	0	0	0	W
mod		1	1	0	mem		
imm8 or imm16-low							
imm16-high							
(disp-low)							
(disp-high)							

(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

7							0
0	0	1	1	0	1	0	W

imm8 or imm16-low

imm16-high

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

7							0
0	0	0	0	1	1	1	1

0	0	0	1	0	0	0	0

1	1	0	0	0	reg		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	0
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	1
1	1	0	0	0	reg		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	1
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	0
1	1	0	0	0	reg		
imm3							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	0	0	0
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm3							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	0	0	1
1	1	0	0	0	reg		
imm4							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	0	0	1
mod		0	0	0	mem		
(disp-low)							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	1	0
1	1	0	0	0	reg		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	1	0
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	1	1
1	1	0	0	0	reg		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	1	1
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
1	1	0	0	0	reg		
imm3							

Bit imm3 of reg8 ← $\overline{\text{bit imm3 of reg8}}$

Bit imm3 (second operand) specifies which bit of the 8-bi register (specified by the first operand) is to be inverted Only the lower 3 bits of the immediate data at the fourtl 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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm3							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1
1	1	0	0	0	reg		
imm4							

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

7							0
0	0	0	0	1	1	1	1

0	0	0	1	1	1	1	1

mod		0	0	0		mem	

(disp-low)

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

7							0
1	1	1	1	0	1	0	1

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	1	0
1	1	0	0	0	reg		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	1	0
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	1	1
1	1	0	0	0	reg		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	1	1
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	0	1	0
1	1	0	0	0	reg		
imm3							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	0	1	0
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm3							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	0	1	1
1	1	0	0	0	reg		
imm4							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	0	1	1
mod		0	0	0	mem		
imm4							
(disp-high)							
imm4							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	0	0
1	1	0	0	0	reg		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	0	0
mod		0	0	0	mem		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	0	1
1	1	0	0	0	reg		

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	0	1
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	0	0
1	1	0	0	0	reg		
imm3							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	0	0
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm3							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	0	1
1	1	0	0	0	reg		
imm4							

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

7							0
0	0	0	0	1	1	1	1
0	0	0	1	1	1	0	1
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm4							

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

7							0
1	1	1	1	1	0	0	1

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

7							0
1	1	1	1	1	1	0	1

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.

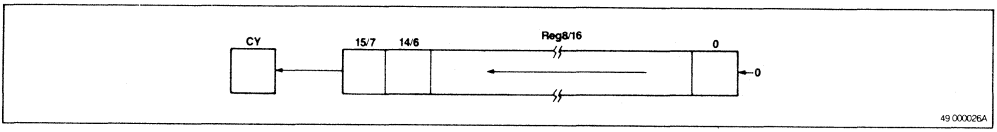
Bytes: 2

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

7							0
1	1	0	1	0	0	0	W
mod		1	0	0	mem		
(disp-low)							
(disp-high)							

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

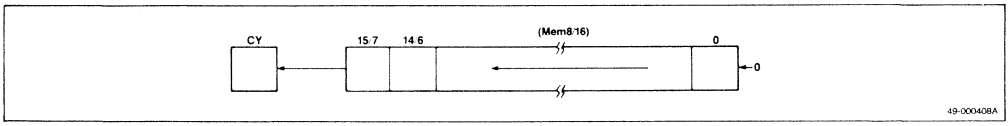
Transfers: 2

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.

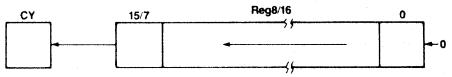
Bytes: 2

Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

Example:
SHL CL,CL
SHL BW,CL



49 000027A

SHL mem, CL

Shift left memory, variable bit

7							0
1	1	0	1	0	0	1	W

mod	1	0	0	mem

(disp-low)

(disp-high)

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

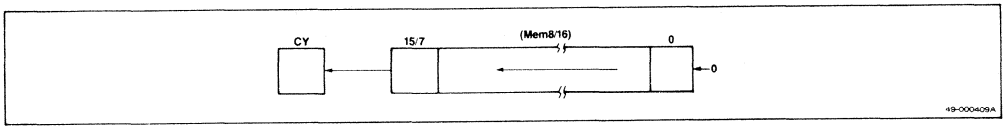
Transfers: 2

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

7							0
1	1	0	0	0	0	0	W

1	1	1	0	0	reg		

imm8

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.

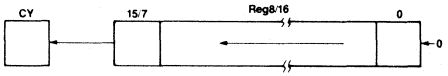
Bytes: 3

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

7							0
1	1	0	0	0	0	0	W
mod		1	0	0	mem		
(disp-low)							
(disp-high)							
imm8							

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

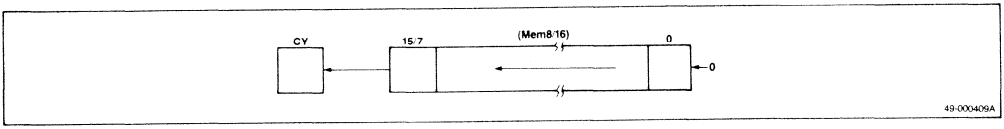
Transfers: 2

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

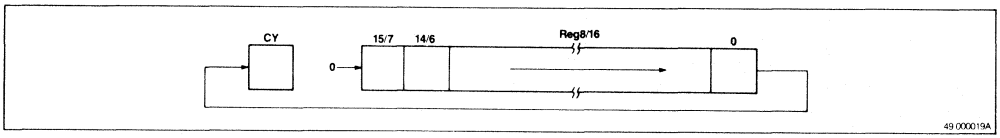
Transfers: None

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

7							0
1	1	0	1	0	0	0	W
mod		1	0	1	mem		
(disp-low)							
(disp-high)							

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

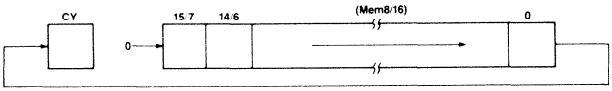
Transfers: 2

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

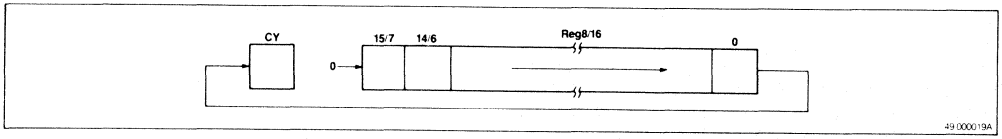
Transfers: None

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

7							0
1	1	0	1	0	0	1	W

mod	1	0	1	mem

(disp-low)

(disp-high)

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

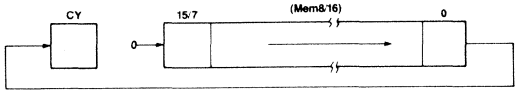
Transfers: 2

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

7							0
1	1	0	0	0	0	0	W
1	1	1	0	1	reg		
imm8							

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.

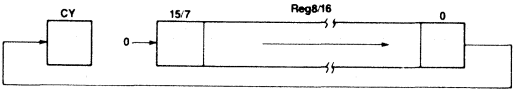
Bytes: 3

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

7							0
1	1	0	0	0	0	0	W

mod	1	0	1	mem

(disp-low)

(disp-high)

imm8

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

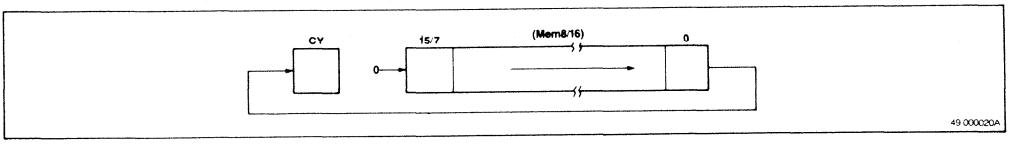
Transfers: 2

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

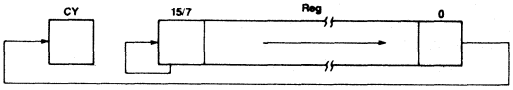
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	X

Example:

```
SHRA    CL,1
SHRA    AW,1
```



49 000021A

SHRA mem,1

Shift right arithmetic, memory, single bit

7							0
1	1	0	1	0	0	0	W
mod		1	1	1	mem		
(disp-low)							
(disp-high)							

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

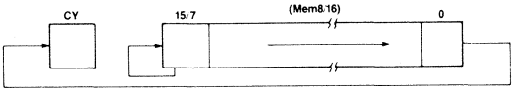
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
0	X	X	U	X	X

Example:

```
SHRA    BYTE_VAR,1
SHRA    WORD_VAR,1
```



49-000412A

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

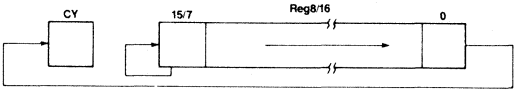
Transfers: None

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

7							0
1	1	0	1	0	0	1	W
mod		1	1	1	mem		
(disp-low)							
(disp-high)							

temp ← CL, while temp ≠ 0,
repeat operation, CY ← LSB of (mem),
(mem) ← (mem) ÷ 2, temp ← temp − 1,
MSB of operand does not change

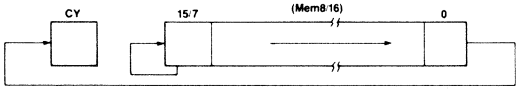
Performs an arithmetic shift right of the 8- or 16-bit memory location (addressed by the first operand) by the number of bits specified in the CL register. A bit with the same value as the original bit is shifted to the memory location's MSB. The LSB is shifted to the CY flag. The sign remains unchanged 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



49-000412A

SHRA reg,imm8

Shift right arithmetic, register, multibit

7							0
1	1	0	0	0	0	0	W
1	1	1	1	1		reg	
				imm8			

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

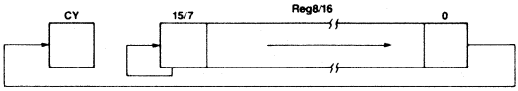
Transfers: None

Flag operation:

V	S	Z	AC	P	CY
U	X	X	U	X	X

Example:

```
SHRA    CL,3
SHRA    BW,7
```



49-000413A

SHRA mem,imm8

Shift right arithmetic, memory, multibit

7							0
1	1	0	0	0	0	0	W
mod		1	1	1	mem		
(disp-low)							
(disp-high)							
imm8							

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

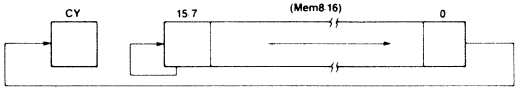
Transfers: 2

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

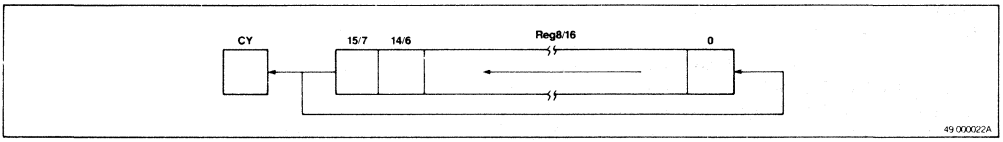
Transfers: None

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

7							0
1	1	0	1	0	0	0	W
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

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.

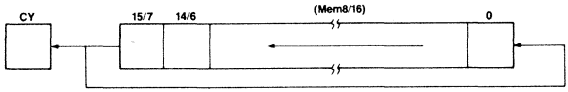
Bytes: 2/3/4
Transfers: 2

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.

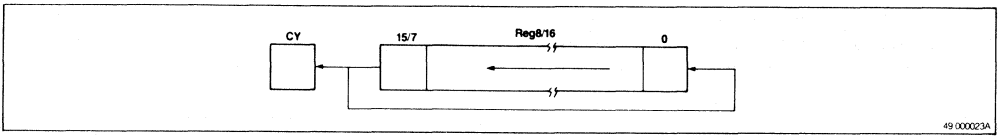
Bytes: 2

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

7							0
1	1	0	1	0	0	1	W

mod	0	0	0	mem

(disp-low)

(disp-high)

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

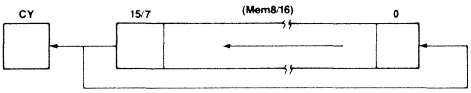
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
U					X

Example:

```
ROL     BYTE PTR [IX],CL
ROL     WORD_VAR,CL
```



49-000415A

ROL reg,imm8

Rotate left, register, multibit

7							0
1	1	0	0	0	0	0	W
1	1	0	0	0	reg		
imm8							

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

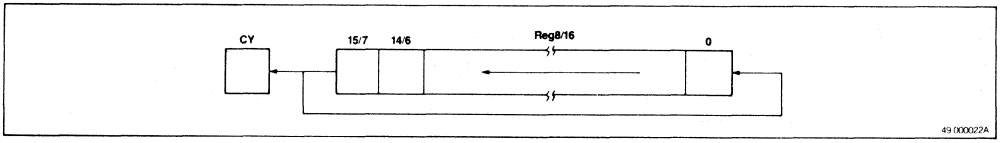
Flag operation:

V	S	Z	AC	P	CY
U					X

Example:

ROL DH,3

ROL IY,7



ROL mem,imm8

Rotate left, memory, multibit

7							0
1	1	0	0	0	0	0	W
mod		0	0	0	mem		
(disp-low)							
(disp-high)							
imm8							

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

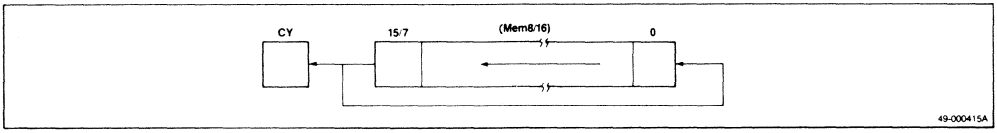
Transfers: 2

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

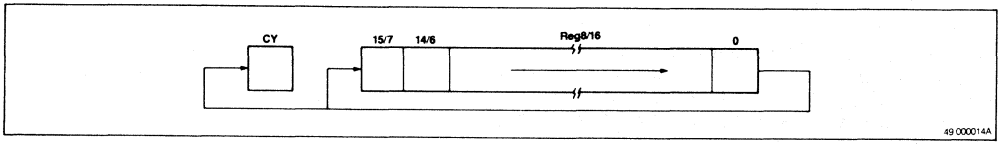
Transfers: None

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

7							0
1	1	0	1	0	0	0	W

mod	0	0	1	mem

(disp-low)

(disp-high)

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

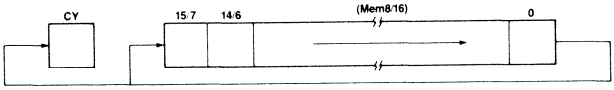
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

Example:

```
ROR     BYTE_VAR,1
ROR     WORD PTR [BW],1
```



49-000416A

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

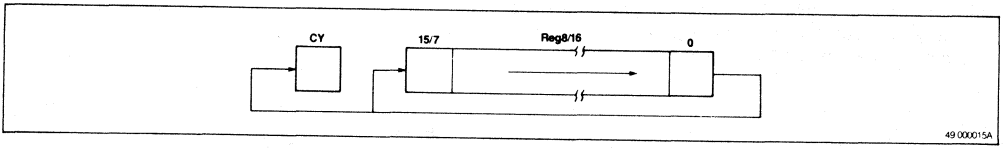
Transfers: None

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

7							0
1	1	0	1	0	0	1	W

mod	0	0	1	mem

(disp-low)

(disp-high)

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

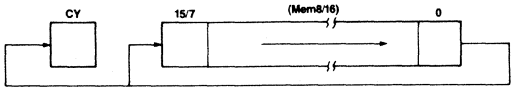
Transfers: 2

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

7							0
1	1	0	0	0	0	0	W

1	1	0	0	1	reg		

imm8

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.

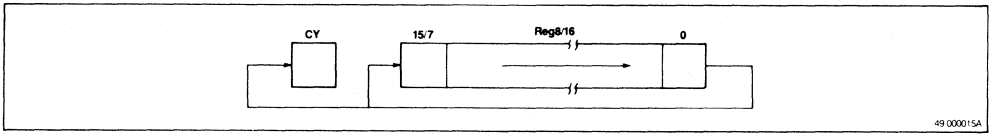
Bytes: 3

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

7							0
1	1	0	0	0	0	0	W
mod		0	0	1	mem		
(disp-low)							
(disp-high)							
imm8							

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.

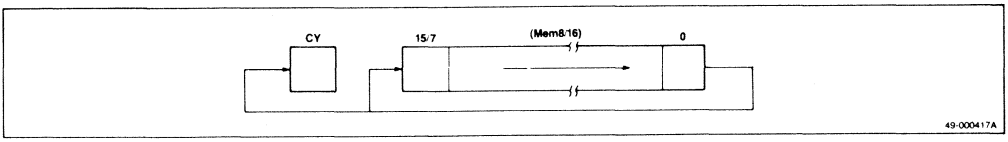
Bytes: 3/4/5

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.

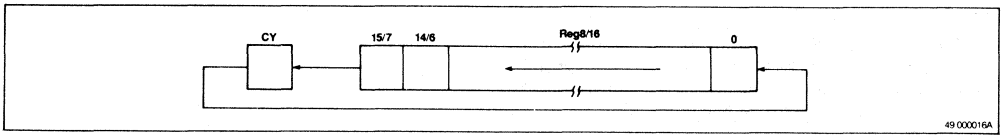
Bytes: 2

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

7							0
1	1	0	1	0	0	0	W

mod	0	1	0	mem

(disp-low)

(disp-high)

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

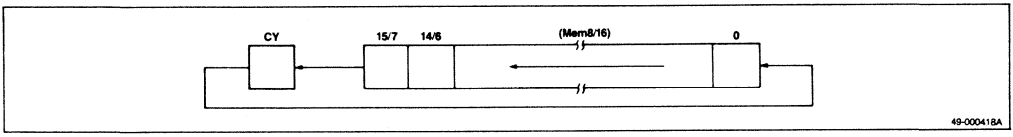
Transfers: 2

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.

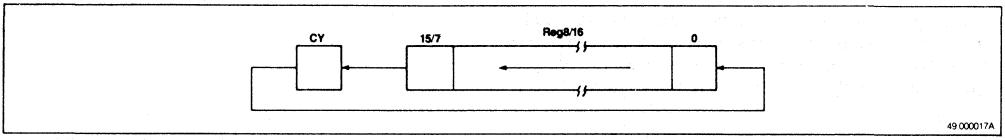
Bytes: 2

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

7							0
1	1	0	1	0	0	1	W

mod	0	1	0	mem

(disp-low)

(disp-high)

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

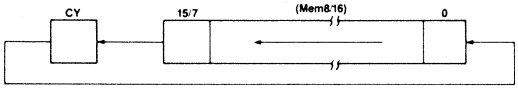
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

Example:

```
ROLC    BYTE PTR [IY],CL
ROLC    WORD_VAR,CL
```



49-000419A

ROLC reg,imm8

Rotate left with carry, register, multibit

7							0
1	1	0	0	0	0	0	W

1	1	0	1	0	reg		

imm8

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

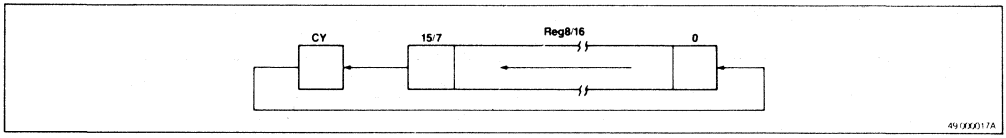
Transfers: None

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

7							0
1	1	0	0	0	0	0	W

mod	0	1	0	mem

(disp-low)

(disp-high)

imm8

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

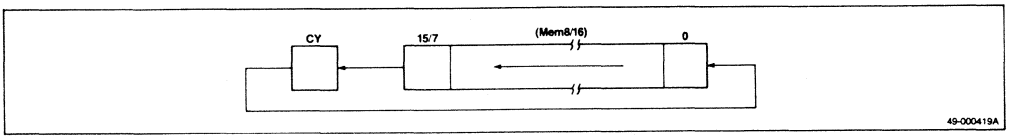
Transfers: 2

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.

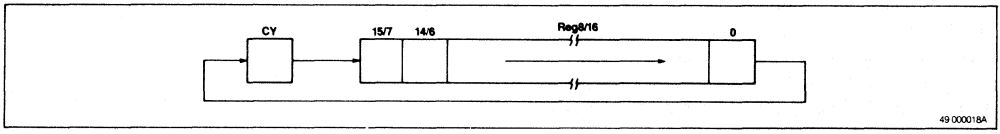
Bytes: 2

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

7							0
1	1	0	1	0	0	0	W

mod	0	1	1	mem

(disp-low)

(disp-high)

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

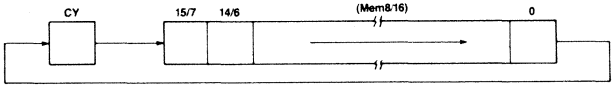
Transfers: 2

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.

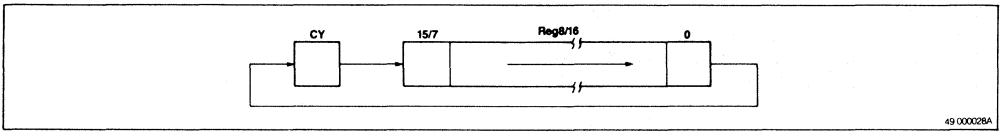
Bytes: 2

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

7							0
1	1	0	1	0	0	1	W
mod		0	1	1	mem		
(disp-low)							
(disp-high)							

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

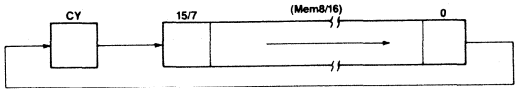
Transfers: 2

Flag operation:

V	S	Z	AC	P	CY
X					X

Example:

```
RORC    BYTE_VAR,CL
RORC    WORD_VAR [BP],CL
```



49-000421A

RORC reg,imm8

Rotate right with carry, register, multibit

7							0
1	1	0	0	0	0	0	W
1	1	0	1	1	reg		
imm8							

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

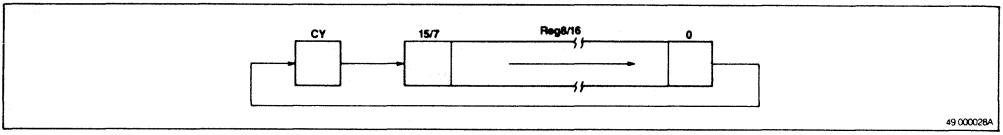
Transfers: None

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

7							0
1	1	0	0	0	0	0	W

mod	0	1	1	mem

(disp-low)

(disp-high)

imm8

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

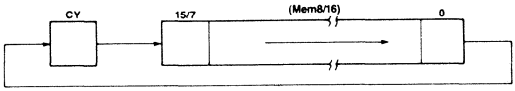
Transfers: 2

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

7							0
1	1	1	0	1	0	0	0

(disp-low)

(disp-high)

(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

7							0
1	1	1	1	1	1	1	1

1	1	0	1	0	reg		

(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

7							0
1	1	1	1	1	1	1	1
mod		0	1	0	mem		
(disp-low)							
(disp-high)							

(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

7							0
1	0	0	1	1	0	1	0
(offset-low)							
(offset-high)							
(seg-low)							
(seg-high)							

(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

7							0
1	1	1	1	1	1	1	1

mod	0	1	1	mem

(disp-low)

(disp-high)

(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

7							0
1	1	0	0	0	0	1	1

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

7							0
1	1	0	0	0	0	1	0

pop-value-low

pop-value-high

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

7							0
1	1	0	0	1	0	1	1

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

7							0
1	1	0	0	1	0	1	0

pop-value-low

pop-value-high

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

7							0
1	1	1	1	1	1	1	1

mod	1	1	0	mem

(disp-low)

(disp-high)

(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

7							0
0	1	0	1	0	reg		

(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

7						0
0	0	0	sreg	1	1	0

(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

7							0
1	0	0	1	1	1	0	0

(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

7							0
0	1	1	0	0	0	0	0

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

7							0
0	1	1	0	1	0	1	0
imm8							

(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

7							0
0	1	1	0	1	0	0	0
imm16-low							
imm16-high							

(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

7							0
1	0	0	0	1	1	1	1
mod		0	0	0	mem		
(disp-low)							
(disp-high)							

(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

7							0
0	1	0	1	1	reg		

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

7						0
0	0	0	sreg	1	1	1

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

7							0
1	0	0	1	1	1	0	1

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

7							0
0	1	1	0	0	0	0	1

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

7							0
1	1	0	0	1	0	0	0
imm16-low							
imm16-high							
imm8							

(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

7							0
1	1	0	0	1	0	0	1

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

7							0
1	1	1	0	1	0	0	1

(disp-low)

(disp-high)

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

7							0
1	1	1	0	1	0	1	1

disp8

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

7							0
1	1	1	1	1	1	1	1
1	1	1	0	0	reg		

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

7							0
1	1	1	1	1	1	1	1
mod		1	0	0	mem		
(disp-low)							
(disp-high)							

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

7							0
1	1	1	0	1	0	1	0
offset-low							
offset-high							
seg-low							
seg-high							

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

7							0
1	1	1	1	1	1	1	1
mod		1	0	1	mem		
(disp-low)							
(disp-high)							

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

7							0
0	1	1	1	0	0	0	0
disp8							

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

7							0
0	1	1	1	0	0	0	1
disp8							

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

7							0
0	1	1	1	0	0	1	0

disp8

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

7							0
0	1	1	1	0	0	1	1

disp8

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

7							0
0	1	1	1	0	1	0	0

disp8

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

7							0
0	1	1	1	0	1	0	1

disp8

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

7							0
0	1	1	1	0	1	1	0
disp8							

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

7							0
0	1	1	1	0	1	1	1
disp8							

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

7							0
0	1	1	1	1	0	0	0
disp8							

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

7							0
0	1	1	1	1	0	0	1
disp8							

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

7							0
0	1	1	1	1	0	1	0

disp8

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

7							0
0	1	1	1	1	0	1	1

disp8

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

7							0
0	1	1	1	1	1	0	0
disp8							

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

7							0
0	1	1	1	1	1	0	1
disp8							

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

7							0
0	1	1	1	1	1	1	0

disp8

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

7							0
0	1	1	1	1	1	1	1

disp8

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

7							0
1	1	1	0	0	0	0	0

disp8

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

7							0
1	1	1	0	0	0	0	1

disp8

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

7							0
1	1	1	0	0	0	1	0

disp8

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

7							0
1	1	1	0	0	0	1	1

disp8

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

7							0
0	0	0	0	1	1	1	1
1	0	0	1	1	1	0	0
mem 8							
imm 3							
disp 8							

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

7							0
1	1	0	0	1	1	0	0

(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

7							0
1	1	0	0	1	1	0	1

imm8

(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

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

Transfers: 3

Flag operation:

V	DIR	IE	BRK	S	Z
R	R	R	R	R	R

AC	P	CY				
R	R	R*				

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

7							0
0	1	1	0	0	0	1	0

mod	reg	mem

(disp-low)

(disp-high)

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:

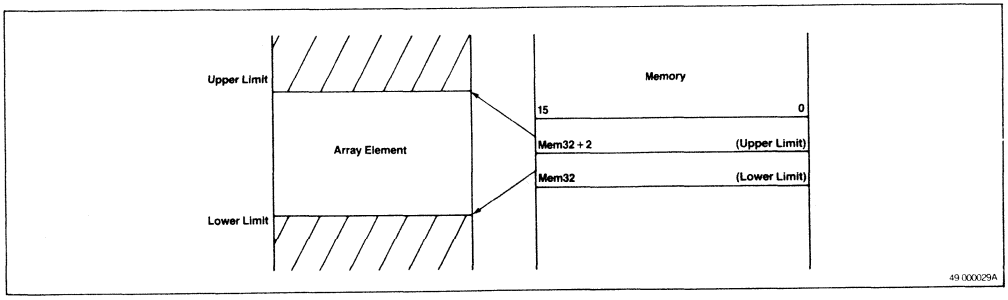
When interrupt condition is fulfilled:

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

7							0
1	1	1	1	0	1	0	0

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

7							0
1	0	0	1	1	0	1	1

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

7							0
0	0	0	0	1	1	1	1

1	0	0	1	1	1	1	0

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

7							0
1	1	1	1	1	0	1	0

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

7							0
1	1	1	1	1	0	1	1

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

7							0
1	1	1	1	0	0	0	0

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

7							0
1	1	0	1	1	X	X	X

1	1	Y	Y	Y	Z	Z	Z

instruction not executed, interrupt takes place

FPO1 fp-op,mem

Floating point operation 1, memory

7							0
1	1	0	1	1	X	X	X
mod		Y	Y	Y	mem		
(disp-low)							
(disp-high)							

Data bus ← (mem)

instruction not executed, interrupt takes place

FPO2 fp-op

Floating point operation 2, register

7							0
0	1	1	0	0	1	1	X
1	1	Y	Y	Y	Z	Z	Z

instruction not executed, interrupt takes place

FPO2 fp-op, mem

Floating point operation 2, memory

7							0
0	1	1	0	0	1	1	X

mod		Y	Y	Y		mem	

(disp-low)

(disp-high)

Data bus ← (mem)

instruction not executed, interrupt takes place

NOP (no operand)

No operation

7							0
1	0	0	1	0	0	0	0

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:

7						0
0	0	1	sreg	1	1	0

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
```

OVERVIEW OF INSTRUCTIONS

(ALPHABETIC ORDER)

INSTRUCTION EXECUTION TIMES

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 11 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 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 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	
			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 field transfer instructions	INS	reg 8, reg 8	0 0 0 0 1 1 1 1	0 0 1 1 0 0 0 1	3				
			1 1 reg reg						
		reg 8, imm 4	0 0 0 0 1 1 1 1	0 0 1 1 1 0 0 1	4				
			1 1 0 0 0 reg						
	EXT	reg 8, reg 8	0 0 0 0 1 1 1 1	0 0 1 1 0 0 1 1	3				
			1 1 reg reg						
		reg 8, imm 4	0 0 0 0 1 1 1 1	0 0 1 1 1 0 1 1	4				
			1 1 0 0 0 reg						
I/O instructions	IN	acc, imm 8	1 1 1 0 0 1 0 W		2	14 + W	14 + W	16 + 2 W	16 + 2 W
		acc, DW	1 1 1 0 1 1 0 W		1	13 + W	13 + W	15 + 2 W	15 + 2 W
	OUT	imm 8, acc	1 1 1 0 0 1 1 W		2	10 + W	10 + W	10 + 2 W	10 + 2 W
		DW, acc	1 1 1 0 1 1 1 W		1	9 + W	9 + W	9 + 2 W	9 + 2 W
Primitive I/O instructions	INM	dst-block, DW	0 1 1 0 1 1 0 W		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	0 1 1 0 1 1 1 W		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 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 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	
			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 operation instructions	ADD4S		0 0 0 0 1 1 1 1	0 0 1 0 0 0 0 0	2	22 + (27 + 3 W) n	22 + (25 + 3 W) n	–	–
	SUB4S		0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 0	2	22 + (27 + 3 W) n	22 + (25 + 3 W) n	–	–
	CMP4S		0 0 0 0 1 1 1 1	0 0 1 0 0 1 1 0	2	22 + (23 + 2 W) n	22 + (23 + 2 W) n	–	–
	ROL4	reg 8	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 0 0	3	17	17	–	–
		mem 8	0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 0 0	3-5	EA + 18 + 2 W	EA + 16 + 2 W	–	–
	ROR4	reg 8	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 1 0	3	21	21		
		mem 8	0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 1 0	3-5	EA + 24 + 2 W	EA + 22 + 2 W		
increase/decrease instruction	INC	reg 8	1 1 1 1 1 1 1 0	1 1 0 0 0 reg	2	5	5	5	5
		mem	1 1 1 1 1 1 1 W	mod 0 0 0 mem	2-4	EA + 11 + 2 W	EA + 9 + 2 W	EA + 15 + 4 W	EA + 11 + 4 W
		reg 16	0 1 0 0 0 reg		1	–	–	2	2
	DEC	reg 8	1 1 1 1 1 1 1 0	1 1 0 0 1 reg	2	5	5	5	5
		mem	1 1 1 1 1 1 1 W	mod 0 0 1 mem	2-4	EA + 11 + 2 W	EA + 9 + 2 W	EA + 15 + 4 W	EA + 11 + 4 W
		reg 16	0 1 0 0 1 reg		1	–	–	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
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 11 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

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
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
Shift instructions	SHR	reg, 1	1 1 0 1 0 0 0 W	1 1 1 0 1 reg	2	8	8	8	8
		mem, 1	1 1 0 1 0 0 0 W	mod 1 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 1 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 1 0 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 1 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 1 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
	SHRA	reg, 1	1 1 0 1 0 0 0 W	1 1 1 1 1 reg	2	8	8	8	8
		mem, 1	1 1 0 1 0 0 0 W	mod 1 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 1 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 1 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 1 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 1 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

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	
			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	ROLC	reg, 1	1 1 0 1 0 0 0 W	1 1 0 1 0 reg	2	8	8	8	8
		mem, 1	1 1 0 1 0 0 0 W	mod 0 1 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 0 1 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 1 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 1 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 1 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

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
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

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	20 + 2 W
		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	30 + 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
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 + 2 W	6
		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	12 + 2 W
		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	–	–	–	–
	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
Interrupt instructions	BRK	3	1 1 0 0 1 1 0 0		1	–	–	55 + 10 W	43 + 10 W
		imm 8 (÷3)	1 1 0 0 1 1 0 1		2	–	–	56 + 10 W	44 + 10 W
	BRKV		1 1 0 0 1 1 1 0		1	–	–	55 + 10 W	43 + 10 W
	RETI		1 1 0 0 1 1 1 1		1	–	–	43 + 6 W	35 + 2 W
	RETRBI *		0 0 0 0 1 1 1 1	1 0 0 1 0 0 0 1	2	–	–	12	12
	FINT *		0 0 0 0 1 1 1 1	1 0 0 1 0 0 1 0	2	2	2	2	2
	CHKIND	reg 16, mem 32	0 1 1 0 0 0 1 0	mod reg mem	2-4	–	–	–	–

*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	–	–	–	–
	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	27	27	27	27

*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.

EUROPEAN DISTRIBUTORS

AUSTRIA
A & D
ABRAHAMCZIK & DEMEL
GES. MBH. & CO KG
EICHENSTRASSE 58-64/1
1120 WIEN
TEL.: (222) 857661
TLX.: 134273

BELGIUM
CN ROOD
DE JAMBLINNE DE MEUXPLEIN 37
1040 BRUSSEL
TEL.: (02) 7352135
TLX.: 22846

MALCHUS ELECTRONICS PVBA
PLANTIN EN MORETUSLEI 172
2000 ANTWERPEN
TEL.: (032) 353256
TLX.: 33637

DENMARK
MER-EL A/S
VED KLAEDEBO 18
2970 HOERSHOLM
TEL.: (2) 571000
TLX.: 37360

FINLAND
OY FERRADO A/B
P.O. BOX 54
VALIMONTIE 1
00380 HELSINKI 38
TEL.: (0) 550002
TLX.: 122214

FRANCE
ASAP
MONSIEUR LEGRIS
42, RUE HENRI MATISSE
59930 LA CHAPELLE D'ARMENTIERES
TEL.: 20 35 11 10

ASAP
RUE DE TROIS PEUPLES
78190 MONTIGNY LE BRETONNEUX
TEL.: (1) 30 43 82 33
TLX.: 69 88 87

CCI
5, RUE MARCELIN BERTHELOT
BP 92
92164 ANTONY
TEL.: (1) 46 66 21 82
TLX.: 203881

CCI
5, RUE BATAILLE
69008 LYON
TEL.: 78 74 44 56

CEDIS (TOURS)
1, RUE DU DANEMARK
37100 TOURS
TEL.: 47 41 76 46

CELTI
Z.I. DE COURTABŒUF
9, AVENUE DU QUEBEC
91940 LES ULIS
TEL.: (1) 64 46 09 09

DIM INTER
65-67, RUE DES CITES
93300 AUBERVILLIERS
TEL.: (1) 48 34 93 70
TLX.: 230524

DIM INTER (COLMAR)
27, RUE KLEBER
68000 COLMAR
TEL.: 89 41 15 43

DIM INTER (VILLEURBANNE)
101, RUE DEDIEU
69100 VILLEURBANNE
TEL.: 78 68 32 29

EALING
BATIMENT AUVIDULIS
AVENUE D'OCEANIE
Z.A. D'ORSAY COURTABŒUF
BP 90
91943 LES ULIS CEDEX
TEL.: (1) 69 28 01 31

GEDIS
352, AVENUE G. CLEMENCEAU
92000 NANTERRE
TEL.: (1) 42 04 04 04

GEDIS (AIX)
MERCURE C
Z.I. D'AIX EN PROVENCE
13763 LES MILLES CEDEX
TEL.: 42 60 01 77

GEDIS (ALPES)
21, RUE DES GLAISONS
38400 ST. MARTIN D'HERES
TEL.: 75 51 23 32

SERTRONIQUE (LILLE)
20, RUE CABANIS
BP 35
59007 LILLE CEDEX
TEL.: 20 47 70 70

SERTRONIQUE (MANS)
60, RUE SAGEBIEN
CEDEX 43
72040 LE MANS
TEL.: 43 84 24 60
TLX.: 720019

TEKELEC
RUE CARLE VERNET
CITE DES BRUYERES
92310 SEVRES
TEL.: (1) 45 34 75 35

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
BAHNHOFSTRASSE 55-1
7250 LEONBERG
TEL.: (0 71 52) 2 60 31
TLX.: 17 715 218

GLYN GMBH
SCHÖNE AUSSICHT 30
6272 NIEDERNHAUSEN
TEL.: (0 61 27) 80 77
TLX.: 4 186 911

H3W ELEKTRONIK VERTRIEB 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) 63 20 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 VERTRIEB GMBH
HEESFELD 4
3300 BRAUNSCHWEIG
TEL.: (05 31) 31 40 95
TLX.: 9 52 351

ULTRATRONIK GMBH
MÜNCHENER STRASSE 6
8031 SEEFELD
TEL.: (0 81 52) 70 90
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.: (051) 532119

CLAITRON S.P.A.
VIA GALLARATE, 211
20151 MILANO
TEL.: (02) 3010091

MELCHIONI S.P.A.
VIA COLETTA, 37
20135 MILANO
TEL.: (02) 57941

PANTRONIC S.R.L.
VIA MATTIA BATTISTINI, 212/a
00167 ROMA
TEL.: (06) 6273909

NETHERLANDS
CN ROOD
CORT V.D. LINDENSTRAAT 11-13
2288 EV RIJSWIJK
TEL.: (070) 996360
TLX.: 31238

INNOCIRCUIT
MALCHUS ELECTRONICA
ADVIESGROEP
MALCHUS B.V.
FOKKERSTRAAT 511-513
3125 BD SCHIEDAM
TEL.: (010) 373777
TLX.: 21598

INTRA ELECTRONICS BV
DUIVENDIJK 5C
P.O. BOX 424
5672 AD NUENEN
TEL.: (0031) 40 838 009
TLX.: (0044) 59 418 INTR NL

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
AMITRON S.A.
AVENIDA DE VALLADOLID 47 A
28008 MADRID
TEL.: (1) 247 93 13
TLX.: 45550

COMELTA S.A.
EMILIO MUNOZ 41, NAVE 1-1-2
MADRID 17
TEL.: (1) 754 30 01
TLX.: 42007

LOBER S.A.
MONTE ESQUINZA 28
MADRID 4
TEL.: (1) 442 11 00
TLX.: 49533

SWEDEN
NORDQVIST & BERG
BOX 9145
AARSTAAENGS VAEGEN 19
10272 STOCKHOLM
TEL.: (0) 8690400
TLX.: 10407

TH'S ELEKTRONIK
BOX 3027
16303 SPAANGA
TEL.: (0) 8362970
TLX.: 11145

SWITZERLAND
MEMOTEC AG
GASWERKSTRASSE 32
4901 LANGENTHAL
TEL.: (63) 28 11 22
TLX.: 9 82 550

TURKEY
BURC ELEKTRONIK
VE MAKINA
SANAYI VE TICARET A.S.
BANKATCHI-SOKAK 15/2
KÜCÜKESAT
ANKARA
TEL.: (0090) 41250300
TLX.: 43430

UNITED KINGDOM
ANZAC COMPONENTS LTD
BURNHAM LANE
SLOUGH SL1 6LN
ENGLAND
TEL.: (06286) 4701

DIALOGUE DISTRIBUTION LTD
WATCHMOOR ROAD
CAMBERLER
SURREY GU15 3AQ
ENGLAND
TEL.: (0276) 688001

FARNEHL ELECTRONIC COMPONENTS LTD
CANAL ROAD
LEEDS LS12 2TU
ENGLAND
TEL.: (0532) 636311

IMPULSE ELECTRONICS LTD
HAMMOND HOUSE
CATERHAM
SURREY CR3 6XG
TEL.: (0883) 46433

STC MULTI COMPONENTS
EDINBURGH WAY
HARLOW
CM20 2DF
ENGLAND
TEL.: (0279) 442971

VSI ELECTRONICS LTD
ROYDOMBURY INDUSTRIAL PARK
HORSECROFT ROAD 9
HARLOW, 5
ESSEX CM19 5BYQM
TEL.: (0279) 29666

NEC OFFICES

NEC Electronics (Europe) GmbH, Oberrather Str. 4, 4000 Düsseldorf 30, W. Germany,
Tel. (0211) 650301, Telex 858996-0

NEC Electronics (Germany) GmbH, Oberrather Str. 4, 4000 Düsseldorf 30,
Tel. (0211) 650302, Telex 858996-0
- Hindenburgstr. 28/29, 3000 Hannover 1, Tel. (0511) 881013–16, Telex 9230109
- Arabellastr. 17, 8000 München 2, Tel. (089) 4160020, Telex 522971
- Heilbronner Str. 314, 7000 Stuttgart 30, Tel. (0711) 890910, Telex 7252220

NEC Electronics (BNL) – Boschdijk 187a, NL-5612 HB Eindhoven, Tel. (040) 445845,
Telex 51923

NEC Electronics (Scandinavia) – Box 4039, S-18304 Täby, Tel. (08) 7328200,
Telex 13839

NEC Electronics (France) S.A., 9, rue Paul Dautier, B.P. 187,
F-78142 Velizy Villacoublay Cedex, Tél. (1) 39469617, Télex 699499

NEC Electronics (France) S.A., Representacion en Espana, Edificio «La Caixa»,
Paseo de la Castellana 51, E-28046 Madrid, Tél. (1) 4194150, Télex 41316

NEC Electronics Italiana S.R.L., Via Fabio Filzi, 25A, I-20124 Milano, Tel. (02) 6709108,
Telex 315355
- Rome Office, International Business Center, P. Le Di Porta Pia, I-00189 Rome,
Tel. (06) 816051 oder 868854, Telex 613689

NEC Electronics (UK) Ltd., Cygnus House, Sunrise Park Way, Milton Keynes, MK14 6NP,
Tel. (0908) 691133, Telex 777565
- Birmingham Office, 9th Floor, Swan Office Centre, 1508 Coventry Road, Yardley,
Birmingham B 25 8 VL, Tel. (021) 7081500, Telex 333014
- Dublin Office, 34/35 South William Street, Dublin 2, Ireland, Tel. (0001) 710200

9/86 V1.0