USER'S MANUAL
μPD70108
μPD70116

NEC ELECTRONICS (EUROPE) GMBH

USER'S MANUAL

μPD70108
μPD70116

3/85 V1.0

VOLUME OF PAGE

	Pages
i - iv	
Chapter 1	10
2	12
3	8
4	7
5	5
6	11
7	4
8	1
9	4
10	11
11	4
12	i - viii & 329
13	10
A	3
B	5
TOTAL	436

Table of Contents

Chapter 1 General

The uPD70108/70116 is a CMOS 8/16-bit microprocessor provided with 8/16-bit architecture and an 8-bit data bus. The uPD70108/70116 has a powerful instruction set including bit processing, packed multi-digit BCD operations, high-speed multiplication/division operations, and variable bit field operations, realizing flexible processes of various applications at high speeds. As one of the features this microprocessor offers, the hardware incorporates a multiplication/division circuit and an effective address generator. Additionally, an internal dual bus system is employed.

The uPD70108/70116 also has emulation functions of an uPD8080AF and comes with a standby mode that significantly reduces power consumption. The uPD70108 and uPD70116 are software-compatible with one another.

This document introduces a new product, still under development, and its functions. The descriptions are therefore subject to change without advance notice.

1.1 Features

(1) 8-bit bus microprocessor: uPD70108

(2) 16-bit bus microprocessor: uPD70116

(3) Minimum instruction execution time: 400 ns (at 5MHz/5V)

(4) Maximum addressable memory: 1M byte

(5) Abundant memory addressing modes

(6) 14- x 16-bit register set

(7) 101 instructions

(8) Bit field operation instructions: Data transfer between 1- to 16-bit fields of the memory and accumulator

(9) Packed BCD operation instruction: Addition, subtraction, and comparison of 1- to 255-digit BCD strings

(10) High-speed multiplication/division instruction (exclusive hardware is incorporated): 6 to 8 us (at 5MHz/5V)

(11) Inter-memory high-speed block transfer:
uPD70108: 625K bytes/sec (at 5MHz/5V)
uPD70116: 625K words/sec (at 5MHz/5V)

(12) Bit operation instruction: 8- or 16-bit register
Any bit of the memory can be set, cleared, inverted, and tested.

(13) High-speed calculation of effective addresses: Exclusive hardware is incorporated.

(14) Abundant interrupt processing functions:
External interrupt: NMI (non-maskable interrupt),
INT (maskable interrupt)
Interrupt by software: BRK (unconditional)
BRKV (when V=1)
BRKEN (emulation)
CHKIND (array index check)
CALLN (native routine call)

(15) IEEE-796 bus-compatible interface

(16) Two operation modes

- ° Native mode: executes instruction set of uPD70108/70116.
- ° Emulation mode: executes instruction set of uPD8080AF.
- ° The mode can be changed by a mode-change instruction (such as BRKEM and RETEM) and an external interrupt.

(17) Standby function: Program standby (halt function) Results in reduced power consumption (approximately 1/10 of normal operating current consumption.)

(18) CMOS

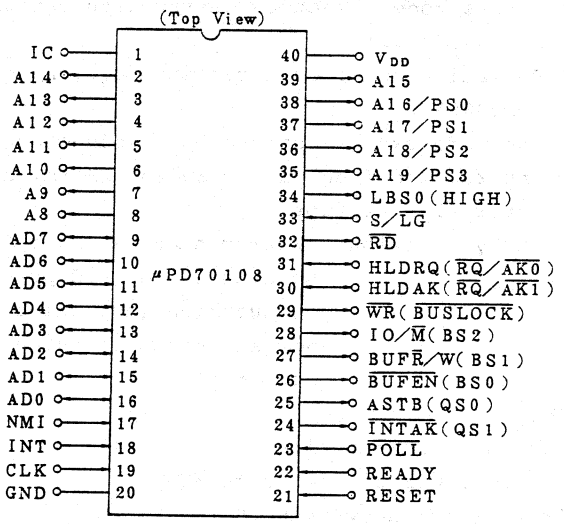
(19) Low power consumption

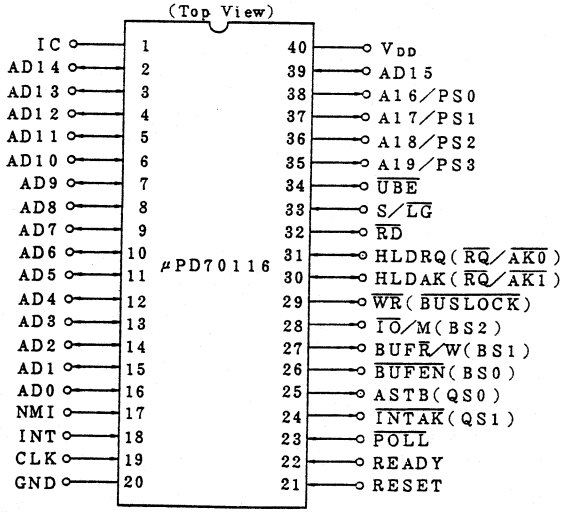
(20) Single power source: 3 to 6V

(21) 5MHz clock: duty cyck 50%

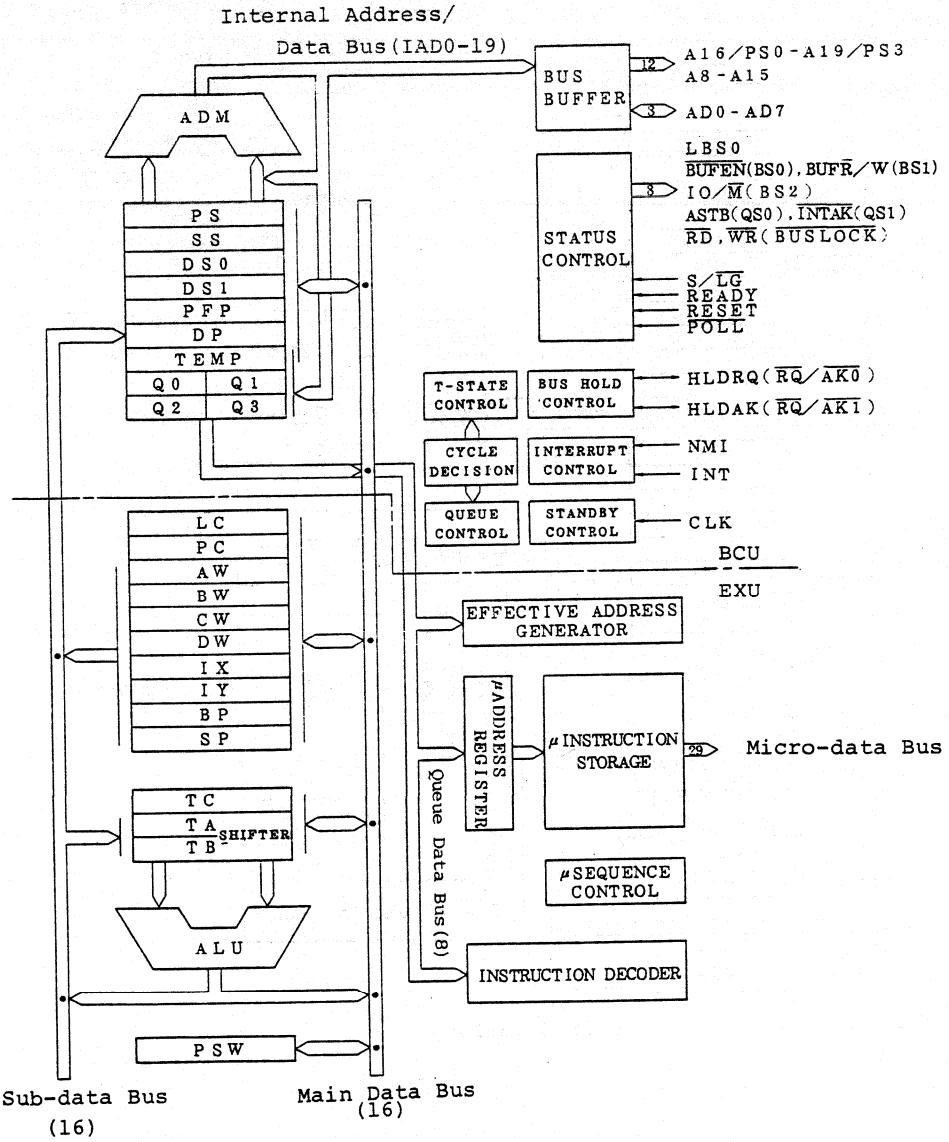
(22) 40-pin plastic DIP

1.2 Pin Configuration of uPD70108/70116

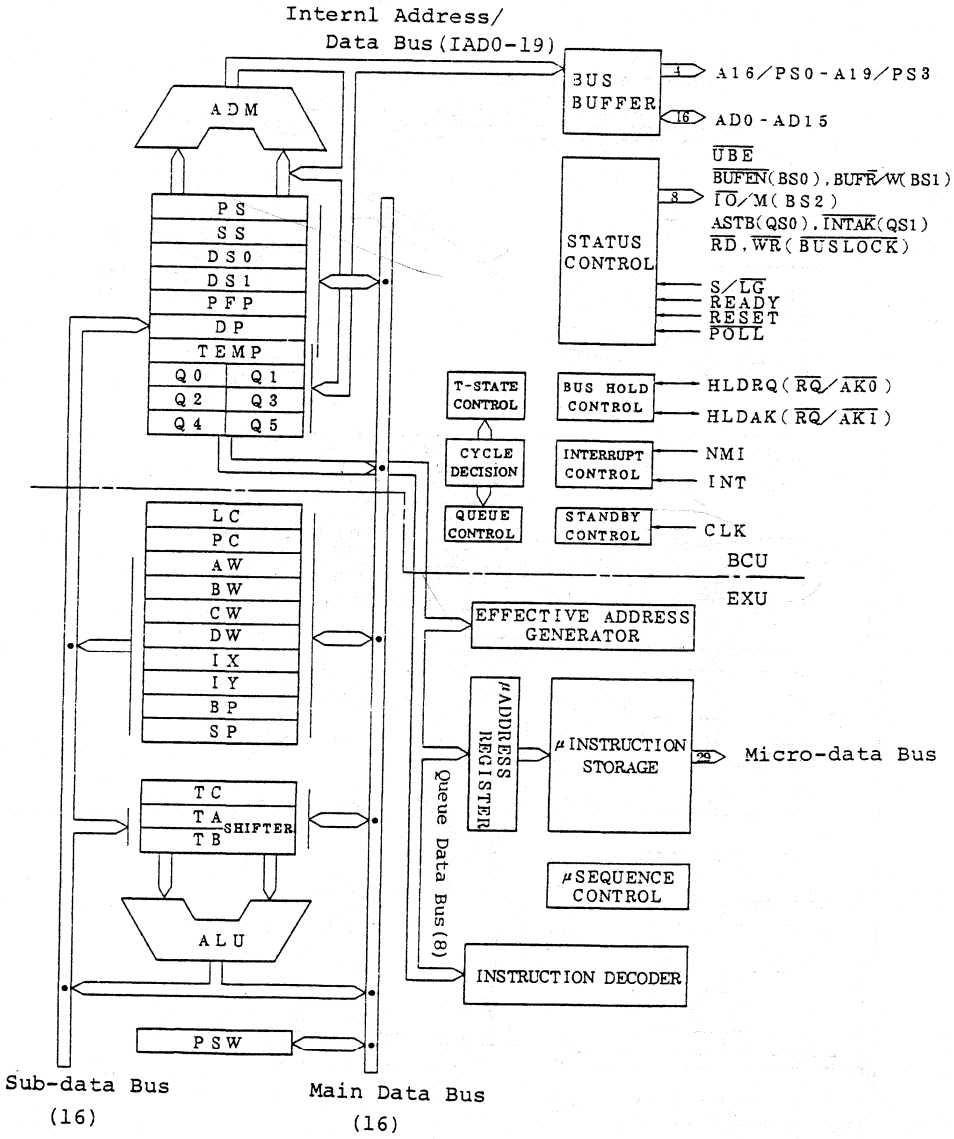




1.3 uPD70108/70116 Block Diagram



uPD70108 Block Diagram



uPD70116 Block Diagram

1.4 uPD70108 Pin Identification

No.	Symbol	Direction	Function
1	IC	-	Internally Connected
2-8	A14-A8	Out	Address Bus, Middle Bits
9-16	AD7-AD0	In/Out	Address/Data Bus
17	NMI	In	Nonmaskable Interrupt Input
18	INT	In	Maskable Interrupt Input
19	CLK	In	Clock Input
20	GND	-	Ground Potential
21	RESET	In	Reset Input
22	READY	In	Ready Input
23	$\overline{\text{POLL}}$	In	Poll Input
24	$\overline{\text{INTAK}}$ (QS1)	Out	Interrupt Acknowledge Output (Queue Status Bit 1 Output)
25	ASTB (QS0)	Out	Address Strobe Output (Queue Status Bit 0 Output)
26	$\overline{\text{BUFEN}}$ (BS0)	Out	Buffer Enable Output (Bus Status Bit 0 Output)
27	BUF$\overline{\text{R}}$/W (BS1)	Out	Buffer Read/Write Output (Bus Status Bit 1 Output)
28	IO/$\overline{\text{M}}$ (BS2)	Out	Access is I/O or Memory (Bus Status Bit 2 Output)

29	$\overline{WR}$ ($\overline{BUSLOCK}$)	Out	Write Strobe Output (Bus Lock Output)
30	HLDAK ($\overline{RQ}$/$\overline{AK1}$)	Out (In/Out)	Hold Acknowledge Output (Bus Hold Request Input /Acknowledge Output 1)
31	HLDRQ ($\overline{RQ}$/$\overline{AK0}$)	In (In/Out)	Hold Request Input (Bus Hold Request Input/Acknowledge Output 0)
32	$\overline{RD}$	Out	Read Strobe output
33	S/$\overline{LG}$	In	Small-scale/Large-scale System Input
34	LBS0 (HIGH)	In	Latched Bus Status Input 0 (Always High in Large-scale Systems)
35-38	A19/PS3 - A16/PS0	Out	Address Bus, High Bits or Processor Status Output
39	A15	Out	Address Bus, Bit 15
40	VDD	-	Power Supply

Note: Where pins have different functions in small- and large-scale systems, the large-scale system pin name and function are in parentneses.

1.5 uPD70116 Pin Identification

No.	Symbol	Direction	Function
1	IC	-	Internally Connected
2-8	AD14-AD8	In/Out	Address/Data Bus
9-16	AD7-AD0	In/Out	Address/Data Bus
17	NMI	In	Nonmaskable Interrupt Input
18	INT	In	Maskable Interrupt Input
19	CLK	In	Clock Input
20	GND	-	Ground Potential
21	RESET	In	Reset Input
22	READY	In	Ready Input
23	$\overline{\text{POLL}}$	In	Poll Input
24	$\overline{\text{INTAK}}$ (QS1)	Out	Interrupt Acknowledge Output (Queue Status Bit 1 Output)
25	ASTB (QS0)	Out	Address Strobe Output (Queue Status Bit 0 Output)
26	$\overline{\text{BUFEN}}$ (BS0)	Out	Buffer Enable Output (Bus Status Bit 0 Output)
27	BUF$\overline{\text{R}}$/W (BS1)	Out	Buffer Read/Write Output (Bus Status Bit 1 Output)
28	$\overline{\text{IO}}$/M (BS2)	Out	Access is I/O or Memory (Bus Status Bit 2 Output)

29	$\overline{WR}$ ($\overline{BUSLOCK}$)	Out	Write Strobe Output (Bus Lock Output)
30	HLDAK ($\overline{RQ}$/$\overline{AK1}$)	Out (In/Out)	Hold Acknowledge Output (Bus Hold Request Input/Acknowledge Output 1)
31	HLDRQ ($\overline{RQ}$/$\overline{AK0}$)	In (In/Out)	Hold Request Input (Bus Hold Request Input/Acknowledge Output 0)
32	$\overline{RD}$	Out	Read Strobe output
33	S/$\overline{LG}$	In	Small-scale/Large-scale System Input
34	$\overline{UBE}$	In	Upper Byte Enable
35-38	A19/PS3-A16/PS0	Out	Address Bus, High Bits or Processor Status Output
39	AD15	In/Out	Address/Data Bus
40	VDD	-	Power Supply

Note: Where pins have different functions in small- and large-scale systems, the large-scale system pin name and function are in parentneses.

Chapter 2 Pin Functions

This chapter describes the function of each uPD70108/70116 pin. Unless otherwise specified, these descriptions are applicable to both the uPD70108 and the uPD70116.
Because the bus width of the uPD70108 and uPD70116 are not the same, each microprocessor uses the bus for both address and data in its own particular way.
Moreover, the memory identification signals for the two microprocessors also are different; the uPD70108 uses an IO/$\overline{M}$ signal while the uPD70116 uses an $\overline{IO}$/M signal. Some pins of both microprocessors are exclusively used depending on whether the microprocessors are mounted in a small- or large-scale system. Other pins function in either system.

2.1 A15-A8 (Address Bus) ... For small- and large-scale systems

(This description applies to the uPD70108 only)
The CPU uses these pins to output the middle 8 bits of the 20-bit address information. They are three-state output and become high impedance during hold acknowledge.

2.2 AD7-AD0 (Address/Data Bus) ... For small- and large-scale systems

(This description applies to the uPD70108 only)
The CPU uses these pins as the time multiplexed address and data bus. They output the lower 8 bits of the 20-bit address information and 8-bit data.
Input/Output operation of 16-bit data is performed in two steps. The low byte is sent first, followed by the high byte. These pins are three-state I/O and become high impedance during hold and interrupt acknowledge.

2.3 AD15-AD0 (Address/Data Bus) ... For small- and large scale systems.

(This description applies to the uPD70116 only)

The AD15-AD0 is a time-multiplexed Address/Data bus. This performs output of lower 16 bits of 20 bits address information and input/output of byte or word data. The uPD70116 locates memory and I/O operand to a byte-data bank to be accessed with even address (AD0=0) and a byte-data bank to be accessed with odd address (AD0=1). The LSB(AD0) has no meaning as a word data address but used for selecting the odd or even address bank. The $\overline{UBE}$ (Upper Byte Enable) signal is provided to access byte/word data besides AD0. This is used as the combination of the following table.

Operand	$\overline{UBE}$	AD0	No. of Bus Cycle
Word of Even Address	0	0	1
Word of Odd Address	0	1*	2
	1	0**	
Byte of Even Address	1	0	1
Byte of Odd Address	0	1	1

*: First Time **: Second Time

A word operand in odd address is performed via two consecutive access of odd-byte bank and even-byte bank. In this case, first AD0=1 showing odd bank address is output, and second AD0=1 showing continuous even bank address is output automatically. These outputs are held to high or low level in the standby mode. These are 3-state output and becomes high impedance in the hold acknowledge and interrupt acknowledge states.

2.4 NMI (Nonmaskable Interrupt) ... For small- and large-scale systems

This pin is used to input nonmaskable interrupt requests. NMI cannot be masked by software. This input is active at the rising edge of a signal and can be sensed during any clock cycle. Actual interrupt processing begins, however, after completion of the instruction in progress.

The contents of interrupt vector 2 determines the starting address for the interrupt-servicing routine. Note that a hold request will be accepted even during NMI acknowledge.

This interrupt can be used to release the standby mode.

2.5 INT (Maskable Interrupt) ... For small- and large-scale systems

This pin is used to input an interrupt request that can be masked by software.

This input is active high level and is sensed during the last clock of the instruction. The interrupt will be accepted if the system is in interrupt enable state (if the interrupt enable flag (IE) is set). The CPU outputs the $\overline{\text{INTAK}}$ signal to inform external devices whether the interrupt request has been acknowledged.

The priority of interrupts is shown below. If NMI and INT interrupts occur at the same time, NMI has priority and INT can not be accepted. A hold request will be accepted even during INT ackowledge.

INT < NMI

This interrupt can be used to release the standby mode.

2.6 CLK (Clock) ... For small- and large-scale systems

This pin is used for external clock input.

2.7 RESET (Reset) ... For small- and large-scale systems

This pin is used to input the CPU reset signal, which is active high level.Input of this signal has priority over all other operations. After the reset signal input returns to low level the CPU begins execution of the program starting at address FFF0H.

In addition to the use during normal CPU start, the RESET input can be used to release the standby mode.

2.8 READY (Ready) ... For small- and large-scale systems

READY indicates that the data transfer is completed. When the signal goes high during read cycle, the data is latched one clock cycle later and the bus cycle terminated.

When the signal goes high during write cycle, the bus cycle is terminated.

2.9 $\overline{\text{POLL}}$ (Poll) . . . For small- and large-scale systems

The CPU checks the input at this pin by the execution of the $\overline{\text{POLL}}$ instruction. If the input is low level, execution continues. If the input is high level, the CPU will check the $\overline{\text{POLL}}$ input every five clock cycles until the input becomes low again.

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

2.10 $\overline{\text{RD}}$ (Read Strobe) ... For small- and large-scale systems

The CPU outputs this strobe signal during data read from an I/O device or memory. The IO/M signal is used to select between I/O and memory. This pin's output is three state and becomes high impedance during hold acknowledge.

2.11 S/$\overline{LG}$ (Small/Large) ... For small- and large-scale systems

This signal determines the operation mode of the CPU. This signal is fixed at either high or low level. When this signal is high level, the CPU will operate in small-scale system mode, and when low level, in large-scale system mode.
Pins 24 to 31 and pin 34 function differently depending on the operating mode of the CPU. Separate nomenclature is adopted for these signals in the two operation mades.

Pin No.	Function S/$\overline{LG}$- high	Function S/$\overline{LG}$-low
24	$\overline{INTAK}$	QS1
25	ASTB	QS0
26	$\overline{BUFEN}$	BS0
27	BUF$\overline{R}$/W	BS1
28	IO/$\overline{M}$,$\overline{IO}$/M	BS2
29	$\overline{WR}$	$\overline{BUSLOCK}$
30	HLDAK	$\overline{RQ}$/$\overline{AK1}$
31	HLDRQ	$\overline{RQ}$/$\overline{AK0}$
34	LBS0	Always high

Note 1. IO/$\overline{M}$ for uPD70108. $\overline{IO}$/M for uPD70116.
2. LBS0 for uPD70108.

2.12 $\overline{INTAK}$ (Interrupt Acknowledge) ... For small-scale systems

The CPU asserts the $\overline{INTAK}$ signal active low when it accepts an INT signal.

The external device synchronizes with this signal and outputs the interrupt vector to the CPU via the data bus (AD7-AD0). This output is held to high level in the standby mode.

2.13 ASTB (Address Strobe) ... For small-scale systems

The CPU outputs this strobe signal to latch address information at an external latch.
This output is held to high level in the standby mode.

2.14 $\overline{\text{BUFEN}}$ (Buffer Enable) ... For small-scale systems

This is used as the output enable signal for an external bidirectional buffer. The CPU outputs this signal during data transfer operations with an external memory or I/O device or during input of a interrupt vector.
This output is held to high level in the standby mode.

2.15 BUF$\overline{\text{R}}$/W (Buffer Read/Write) ... For small-scale systems

The output of this signal determines the direction of data transfer with an external bidirectional buffer. A high output specifies transmission from the CPU to the external device, while a low signal specifies reception from the external device to the CPU.

This output is held to high or low level in the standby mode.

2.16 IO/$\overline{\text{M}}$ (IO/Memory) ... For small-scale systems
(This description applies to the uPD70108 only)
The CPU outputs this signal to specify either I/O access or memory access. A high-level output specifies I/O access and a low-level signal specifies memory access.

This output is held to high or low level in the standby mode.

The output from this pin is three state and becomes high impedance during hold acknowledge.

2.17 $\overline{IO}$/M (IO/Memory) ... For small-scale systems
(This description applies to uPD70116 only)
The CPU outputs this signal to specify either I/O access or memory access. A low-level output specifies I/O access and a high-level signal specifies memory access.

This output is held to high or low level in the standby mode.

The output from this pin is three state and becomes high impedance during hold acknowledge.

2.18 $\overline{WR}$ (Write Strobe) ... For small-scale systems

The CPU outputs this strobe signal during data write to an I/O device or memory. Selection of either I/O or memory is performed by the IO/$\overline{M}$ signal.

This output is held to high level in the standby mode.

This pin's output is three state and becomes high impedance during hold acknowledge.

2.19 HLDAK (Hold Acknowledge) ... For small-scale systems

This pin outputs an acknowledge signal to indicate when the CPU accepts a hold request signal (HLDRQ). While this signal is active (high level), the address bus, address/data bus, and the control lines become high impedance.

2.20 HLDRQ (Hold Request) ... For small-scale systems

This signal is input from external devices to request the CPU to release the address bus, address/data bus, and the control bus.

2.21 LBS0 (Latched Bus Status 0) ... For small-scale system
(This description applies to the uPD70108 only)
The CPU uses this signal along with the IO/$\overline{M}$ and BUF$\overline{R}$/W signals to inform an external device what the current bus cycle is.

IO/$\overline{M}$	BUF$\overline{R}$/W	LBS0	Bus Cycle
	0	0	Program fetch
0		1	Memory read
	1	0	Memory write
		1	Passive state
	0	0	Interrupt acknowledge
1		1	I/O read
	1	0	I/O write
		1	Halt

2.22 $\overline{UBE}$ (Upper Byte Enable) ... For small- and large scale systems
(This description applies to uPD70116 only)
This output indicates the using of the upper 8 bits (AD15-AD8) of the Address/Data bus during T2-T4 of bus cycle. This signal is active low and output during T1-T4 of the bus cycle. Bus cycles in which the $\overline{UBE}$ is active are shown in the following table.

Operand	$\overline{UBE}$	AD0	No.of Bus Cycle
Word of Even Address	0	0	1
Word of Odd Address	0	1*	2
	1	0**	
Byte of Even Address	1	0	1
Byte of Odd Address	0	1	1

*: First Time
**: Second Time

The $\overline{UBE}$ signal goes low level continuously during interrupt acknowledge state (because of necessity of word access of even address for vector read).
This signal is held to high level in the standby mode. The $\overline{UBE}$ is 3-state output and becomes high impedance during hold acknowledge.

2.23 A19/PS3 - A16/PS0 (Address Bus/Processor Status) ... For small- and large-scale systems

These pins are time multiplexed to operate as an address bus and for output of processor status signals.

When used as the address bus, these pins output the high 4 bits of the 20-bit memory address. During I/O access, all 4 bits output data 0.

The processor status signals are output for both memory and I/O access. PS3 is always 0 in the native mode and always 1 in the emulation mode. The contents of the interrupt enable flag (IE) are output to PS2. PS1 and PS0 indicate which memory segment is being accessed.

A17/PS1	A16/PS0	Segment
0	0	Data segment 1
0	1	Stack segment
1	0	Program segment
1	1	Data segment 0

The output of these pins are three state and becomes high impedance during hold acknowledge.

2.24 QS1, QS0 (Queue Status) ... For large-scale systems

The CPU uses these signals to inform external devices, such as the floating point arithmentic processor chip, about the status of the internal CPU instruction queue.

QS1	QS0	Instruction queue status
0	0	NOP (Queue does not operate)
0	1	First byte of instruction
1	0	Queue empty
1	1	Subsequent byte

The instruction queue status indicated by these signals is the status when the execution unit (EXU) accesses the instruction queue. The data output from these pins are therefore valid only for one clock cycle immediately following queue access. These status signals are provided so that the floating point operation chip can monitor the CPU's program execution status and synchronize its operation with the CPU when control is passed to it by the FPO (Floating Point Operation) instruction.

These outputs are held to low level in the standby mode.

2.25 BS2-BS0 (Bus Status) ... For lage-scale systems

The CPU uses these status signals to inform an external bus controller what the current bus cycle is. The external bus controller decodes these signals and generates the control signals required to perform access of the memory or I/O device.

BS2	BS1	BS0	Bus cycle
	0	0	Interrupt acknowledge
0		1	I/O read
	1	0	I/O write
		1	Halt
	0	0	Program fetch
1		1	Memory read
	1	0	Memory write
		1	Passive state

These outputs are held to high level in the standby mode.

These are three-state output and become high impedance during hold acknowledge.

2.26 $\overline{\text{BUSLOCK}}$ (Bus Lock) ... For large-scale systems

The CPU uses this signal to secure the bus during the instruction immediately following the execution of the BUSLOCK prefix instruction. It is a request signal to the other master CPUs in a multiprocessor system inhibiting them from using the system bus during this time.

This output is held to high level in the standby mode, however, it is held to low level if BUSLOCK instruction is executed right before HALT instruction.
The output of this signal is three state and becomes high impedance during hold acknowledge.

2.27 $\overline{RQ/AK1}$, $\overline{RQ/AK0}$ (Hold Request/ Acknowledge) ... For large-scale systems

These pins function as bus hold request inputs ($\overline{RQ}$) and as bus hold acknowledge outputs ($\overline{AK}$). This is the priority for these signals.

$$\overline{RQ/AK1} < \overline{RQ/AK0}$$

These pins have three-state output buffer. They also are provided with on-chip pull-up resistors. They are inactive (high level) when open.

2.28 VDD (Power Supply) ... For small- and large-scale systems

This pin is used for positive power supply.

2.29 GND (Ground) ... For small- and large-scale systems

This pin is used for the ground potential.

2.30 IC (Internally Connected)

This pin is used for tests performed at the factory by NEC. Normally, the uPD70108/70116 is used with this pin at ground potential.

Chapter 3 Functions of Internal Block

The uPD70108/70116 consists of two independent processing unit: a bus control unit (BCU) and an execution unit(EXU).

BCU - Prefetches instruction bytes using instruction queue
(4-byte for the uPD70108, 6-byte for the uPD70116)

EXU - Internal data processing (Execute microprogram)

3.1 Program Counter (PC)

The program counter is a 16-bit binary counter that stores offset data for the next programme memory address the EXU is to execute.

The PC increments each time the microprogram fetches an instruction from the instruction queue. A new location value is loaded into the PC each time a branch, call, return, or break instruction is executed. At this time, the contents of the PC are the same as the Prefetch Pointer (PFP).

3.2 Prefetch Pointer (PFP)

The prefetch pointer (PFP) is a 16-bit binary counter that stores the offset data for the program memory address that the bus control unit (BCU) is about to prefetch in to the instruction queue.
The PFP is incremented each time the BCU prefetches an instruction from the program memory. A new location will be loaded into the PFP whenever a branch, call, return, or break instruction is executed. At that time the contents of the PFP will be the same as those of the PC (Program Counter). The contents of PFP are an offset from the PS Program Segment) register.

3.3 Q0-Q3/Q0-Q5 (Prefetch Queue)

The uPD70108/70116 has 4-/6-byte instruction queue (FIFO), and it can store up to 4/6 instruction byte prefetched by the BCU. The instruction codes stored in the queue are fetched and executed by the EXU. The queue is cleared and prefetched with the instruction code of a new location when branch, call, return, or break instruction has been executed and when external interrupt has been acknowledged. Normally, the uPD70108 prefetches if the queue has one byte or more space and the uPD70116 prefetches if the queue has one word (two bytes) or more space. If the time required to prefetch the instruction code from the external memory is less than the mean

execution time of instructions which are executed sequentially, then the actual instructions cycle will be shortend by this amount of time i.e. the time needed to fetch the instruction code. This is because the instruction code to be next executed by the EXU can be available in the queue immediately after the completion of one instruction. As the result, processing speed is highly upgraded compared with the conventional CPU which fetch and execute instructions one by one. Queuing effect is lowered if there were many instructions which clears queue like the branch instruction or in the case of continuous instructions with too short instruction time.

3.4 DP (Data Pointer)

This is a 16-bit register indicates read/write addresses of variables. Effective address made in the effective address generator and the register contents including memory address offsets are transferred to the DP.

3.5 TEMP (Temporary Communication Register)

This is a 16-bit temporary register used by communications between external data bus and the EXU. The TEMP can be read or written by upper byte or lower byte independently for byte access. Basically, the EXU completes write operation with transferring data to the TEMP and completes read operation with recognizing the data has been transferred to the TEMP from external data bus.

3.6 Segment Registers (PS,SS,DS0, and DS1)

The memory addresses accessed by the uPD70108/70116 are divided into 64K-byte logical segments. The starting (base) address of each segment is specified by a segment register, and the offset from this starting address is specified by the contents of another register or by the effective address.
These are the four types of segment registers used.

Segment register	Default offset
PS (Program Segment)	PFP
SS (Stack Segment)	SP, effective address
DS0 (Data Segment 0)	IX, effective address
DS1 (Data Segment 1)	IY

3.7 ADM (Address Modifier)

The Address Modifier performs the generation of physical address (adding segment register and PFP or DP) and increment of PFP (Prefetch Pointer).

3.8 General-Purpose Registers (AW, BW, CW, and DW)

There are four 16-bit general-purpose registers. These can be accessed as 16-bit registers or as 8-bit registers by dividing them into their high and low 8-bits (AH, AL, BH, BL, CH, CL, DH, DL).

Each register is also used as a default register for processing specific instructions. These are the default assignments.

AW: Word multiplication/division, word I/O, data conversion

AL: Byte multiplication/division, byte I/O, BCD rotation, data conversion, translation

AH: Byte multiplication/division,

BW: Translation

CW: Loop control branch, repeat prefix

CL: Shift instructions, rotation instructions, BCD operations

DW: Word multiplication/division, indirect addressing I/O

3.9 Pointers (SP, BP) and Index Registers (IX, IY)

These registers serve as base pointers or index registers when accessing the memory using based addressing, indexed addressing, or based indexed addressing.

These registers can also be used for data transfer and arithmetic and logical operations in the same manner as the general-purpose registers. They cannot be used this way, however, as 8-bit registers.

Also, each of these registers acts as default registers for spacific operations. The derault assignments are:

SP: Stack operations

IX: Block transfer (source), BCD string operations

IY: Block transfer (destination), BCD string operations

3.10 TA/TB (Temporary Register/Shifter A,B)

The TA/TB are 16-bit temporary register/shifter used with execution of multiply/divide and shift/rotate (including BCD rotate) instructions. When executing multiply or divide instruction TA+TB operates as a 32-bit temporary register/shifter, and TB operates as a 16-bit temporary register/shifter when executing shift/rotate instructions. Both the TA and TB can be read or written to and from the internal bus by upper byte or lower byte independently. The contents of the TA .and TB are input to the ALU.

3.11 TC (Temporary Register C)

The TC is a 16-bit temporary register used with internal processing like the multiply or divide operation, etc. The TC content is output to the ALU.

3.12 ALU (Arithmetic & Logic Unit)

The Arithmetic and Logic Unit is consists of a full adder and logical operation circuit and performs these operations:

- Arithmetic operation (Add, Subtract, Multiply, Divide, increment, decrement, and complement)
- Logical operation (test, And, Or, Xor and bit test, set, clear, and complement)

3.13 Program Status Word (PSW)

The program status word consists of the following six status and four control flags.

Status flags

V (Overflow)

S (Sign)

Z (Zero)

AC (Auxiliary Carry)

P (Parity)

CY (Carry)

Control Flags

MD (Mode)

DIR (Direction)

IE (Interrupt Enable)

BRK (Break)

When the PSW is pushed onto the stack, the word image of the various flags is as shown here.

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

The status flags are set and reset depending on the result (data value) of each type of instruction executed.

Instructions are provided to set, reset, and not (or complement) the CY flag directly.

Other instructions set and reset the control flags and control the operation of the CPU.

3.14 LC (Loop Counter)

The Loop Counter (LC) is a 16-bit register which counts these items.

- Loop number of the primitive block transfer and input/output instructions (MOVBK, OUTM, etc.) controlled with repeat prefix instructions (REP, REPC, etc.).

- Shift number of the multi-bit shift/rotate instructions.

3.15 EAG (Effective Address Generator)

The Effective Address Generator (EAG) performs high-speed effective address calculation necessary for memory access. This completes all the calculations with 2 clocks for every addressing mode.
This fetches the instruction byte (2nd or 3rd byte) which has operand specifying field, if the instruction needs memory access. Then calculates effective address and transfers it to the DP (Data Pointer) and generates control signals relating to handling ALU and corresponding registers. In addition, if it's necessary, the EAG requests to the BCU for starting the bus cycle (memory read).

3.16 Instruction Decoder

The Instruction Decoder classifies 1st byte of an instruction code into some groups with specific function and holds them during macro-instruction execution.

3.17 Microaddress Register

The microaddress register specifies address of an microinstruction ROM to be next executed. At starting of an microinstruction execution, the 1st byte of an instruction byte(s) stored in the queue is fetched in this register and it specifies a start address of the corresponding microinstruction sequence.

3.18 Microinstruction ROM

The Microinstruction ROM has 1024 words by 29 bits of microinstructions.

3.19 Microinstruction Sequencer

The Microinstruction Sequencer controls the microaddress register operation, microinstruction ROM output, and synchronizing the EXU with BCU.

Chapter 4 Configuration of Memory and Input/Output

4.1 Memory Configuration and Accessing

The μPD70108/70116 can access and address up to 1M bytes (512K words) of memory area. The memory area is addressed by the address information output by the 20-bit address bus (A19 to A0).

The memory map is shown in Fig. 4-1.

The 1K bytes from addresses 0H to 3FFH are allocated as an interrupt vector table. This area may be used for other purposes in some systems. The 12 bytes from address FFFF0H to FFFFBH are automatically used by the CPU when it is started or reset and therefore cannot be used for any other purpose. The four bytes from addresses FFFFCH to FFFFFH are reserved for future use and consequently not available to the user.

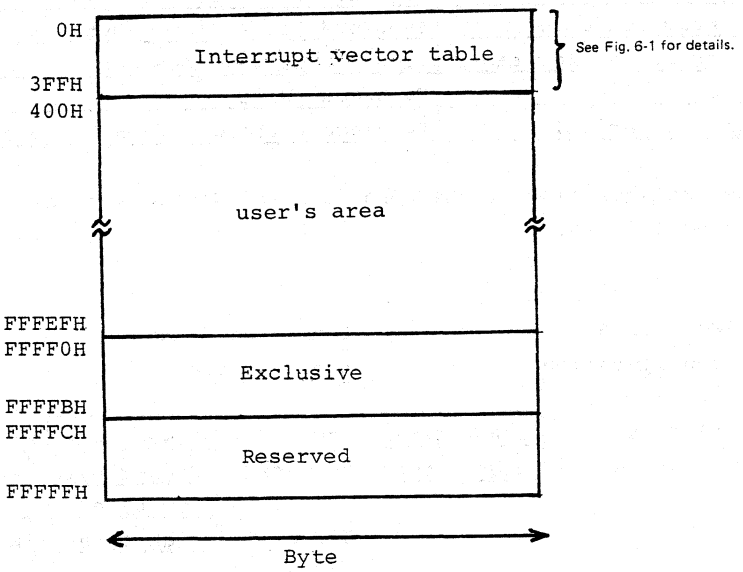


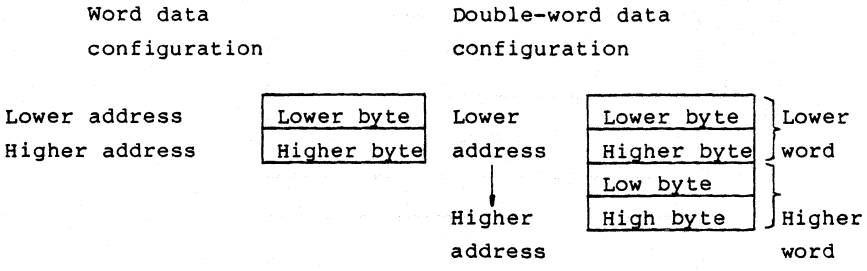
Fig. 4-1 Memory Map

The memory stores such things as instruction codes, interrupt start addresses, stack data, and general variables. Some of these are stored in byte units and the others are in word units.
These data can be stored in both even (A0=0) and odd (A0=1) addresses generated by an instruction. The area where interrupt start addresses (interrupt vector table) are stored is addressed only by using even addresses, however. The μPD70116 is so designed that a word data can be accessed regardless of whether the word data is of an even or an odd address, allowing both the even and odd addresses to be generated by an instruction.
Table 4-1 shows the address and data configuration of each data.

Table 4-1

Data	Address	Data configuration
Instruction code	Even/odd	1/2/3/4/5/6 bytes
Interrupt vector table	Even	2 words/vector
Stack	Even/odd	Word
General variable	Even/odd	Byte/word/double word

The configuration of the word data and double-word data are as follows:



The interface between the memory and the CPU μPD70108 is shown in Fig. 4-2. The interface between the memory and the CPU μPD70116 is shown in Fig. 4-3.

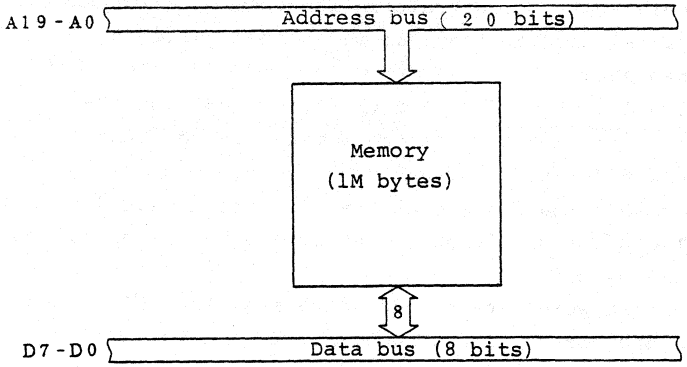


Fig. 4-2 Interface Between Memory and uPD70108

Because the data bus of the μPD70108 is 8 bits wide, a byte data is accessed during one bus cycle and two bus cycles are required to access word data regardless of whether the word data is of an even or odd address.

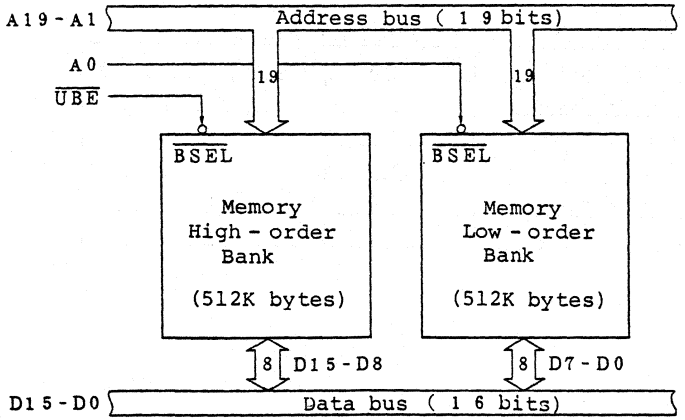


Fig. 4-3 Interface Between Memory and uPD70116

Because the data bus of the μPD70116 is 16 bits wide, in principle the μPD70116 has the capability to transfer a 16-bit word data within one bus cycle. This is true, however, only when an even-address word data (i.e., when A0=0) generated by an instruction is accessed. Two bus cycles are required to transfer odd-address word data (i.e., when A0=1).

The A0 signal in Fig. 4-3 is active low, enabling the byte data in the low bank of the memory. In addition to the address information output from the address bus, a $\overline{UBE}$ (Upper Byte Enable) signal that is also active low is output. The $\overline{UBE}$ signal enables the byte data in the higher bank of the memory.

When accessing word data of odd addresses, the $\overline{UBE}$ signal becomes 0 and the A0 signal becomes 1 during the first bus cycle. Only the higher byte of an address is accessed at this time. Subsequently, the $\overline{UBE}$ signal automatically becomes 1 and the lower 16 bits of the address information (A15-A0) is incremented by one, in other words, the A0 signal becomes 0. Then the lower byte of the address is accessed.

The word data of an even address is accessed during one bus cycle when the $\overline{UBE}$ and A0 signals are 0s.

Table 4-2 summarizes the above description.

Table 4-2 μPD70116 Data Access

Accessed data	$\overline{UBE}$	A0	No. of bus cyles
Word (even address)	0	0	1
Word (odd address)	0	1	2
	1	0	
Byte (even address)	1	0	1
Byte (odd address)	0	1	1

The μPD70116 normally accesses (prefetches) instruction codes in word units. If a branch operation to an odd address takes place, however, only one byte is fetched from that odd address. After that, instruction codes are prefetched in word units again.

When the interrupt vector table is accessed in response to an interrupt, even addresses are always generated from vector numbers (0 to 255) as the addresses of the vector table. Data of even addresses are always accessed as word data when the interrupt vector table is accessed. Consequently, when the vector table is accessed in response to an interrupt, two bus cycles are required because two words (segment base and offset) are accessed. One bus cycle to access the memory requires four clocks. As a result, each time word data from an odd address is accessed, four additional clocks of instruction execution time are necessary compared to that required when accessing an even-address word data. This only applies to instructions that require more than one bus cycle.

When tranferring a word data from one memory area to another, the memory must be accessed twice because the word data must be read from the source first and then written to the destination. If both the source and the destination are odd addresses the execution time will be maximized.

The same holds true during a stack operation. The contents of a register are automatically saved to the stack when an interrupt is serviced. These contents are all word data. If they are processed by using an odd address, therefore, the number of bus cycles required is twice as many as the bus cycles required to process using an even address.

The result is a delay in the interrupt response time. Consideration should be given to this point.

As is apparent from this discussion, it's a good idea to allocate even addresses to word data that can be checked by the program when accessed in the μPD70116.

Example: Execution time of MOV reg,mem instruction (number of clocks for word data accessed one time)

Byte data	11:	μPD70108/70116
Word data	15:	uPD70108/odd address of 70116
	11:	even address of μPD70116

4.2 I/O Configuration and Accessing

The μPD70108/70116 can access up to 64K bytes (32K words) of I/O address area used independently of the memory. The I/O address area is addressed by address information output by the lower 16/8 bits of the address bus. The I/O map is shown in Fig. 4-4.

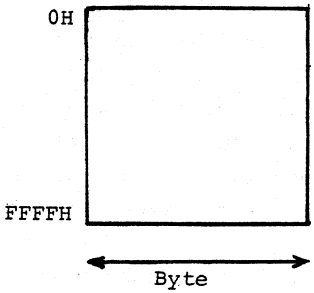


Fig. 4-4 I/O Map

Unlike the memory, the segment method is not applied to the I/O address area.
When I/O addresses are output, 0s are output to the higher four bits (A19 to A16) of the address bus.

Since data are transferred between the CPU and the I/O address area in byte or word units, both 8-bit and 16-bit I/O devices can be connected to the uPD70116 and only 8-bit I/O devices can be connected to the uPD70108.

However, in the μPD70116, a word data of an even address, in the same way as when accessing the memory, requires only one bus cycle and two cycles are required to access odd-address word data.

When the μPD70116 accesses an 8-bit I/O device, bit A0 of the I/O address information is only used to select an I/O device. The A1 and higher bits are used to select a device and several registers in that device.

Consequently, only even addresses should be assigned to an I/O device as well as to the internal registers of the I/O devices so that the registers can be selected using only even addresses. Similarly, I/O devices whose internal registers are assigned with odd addresses must be assigned with an odd address. If memory-mapped I/O configuration (allocating a certain area of the memory to an I/O address area) is employed, the I/O address area can be allocated to a 1M-byte memory area. In this way, numerous addressing modes of the memory and operation processes can be directly performed on I/O devices.

For example, if a bit operation instruction for the memory is used, one line of a specific I/O port can be tested (to examine 1 or 0), set (to 1), cleared (to 0), or inverted.

In a memory-mapped I/O configuration, however, control signals output from the CPU are regarded as those for the memory. The I/O device is therefore distinguished from the memory only by address information. For this reason, then, be careful that addresses of variables or the stack do not conflict with the addresses allocated to I/O.

Chapter 5 Read/Write Timing of Memory and Input/Output

Read/write timings of the μPD70108/70116 memory and I/O are shown in Figs. 5-1 to 5-8. One bus cycle is required for each access (read/write) of the memory or I/O address area. A bus cycle is basically made up of four states (clocks): T1 to T4. One state is 200ns when the microprocessor operates at 5MHz.

The μPD70108 and μPD70116 fetch an instruction code and read data using exactly the same timing (see Figs. 5-1, 5-3, 5-5, and 5-7). If a particular instruction calls for a longer internal process time, the EXU fetches the instruction code of that instruction from the instruction queue and executes it. After that, the BCU continues prefetching instructions to the instruction queue until the queue becomes full. Should the EXU not fetch an instruction code from the instruction queue because another instruction is being executed, the BCU will not prefetch the next instruction. Instead, it automatically inserts an idle state (TI) after state T3. More idle states are inserted until the EXU finishes executing the instruction being processed. Then it fetches the next instruction code from the instruction queue.

When the next instruction code is fetched, the BCU advances the state of the bus cycle from state T4 to T1. For a memory or I/O device whose access time is slow, the BCU detects a READY signal sent from the memory or the I/O device. If the READY signal is at low level, BCU will not advance the bus cycle state from T3 to T4. Instead, it will delay by inserting a wait state (TW). When the READY signal becomes high, execution advances the bus cycle state from T4 to T1 so the next instruction can be fetched.

When the wait state TW is inserted, the current level of each signal is retained and the read/write timing is expanded.

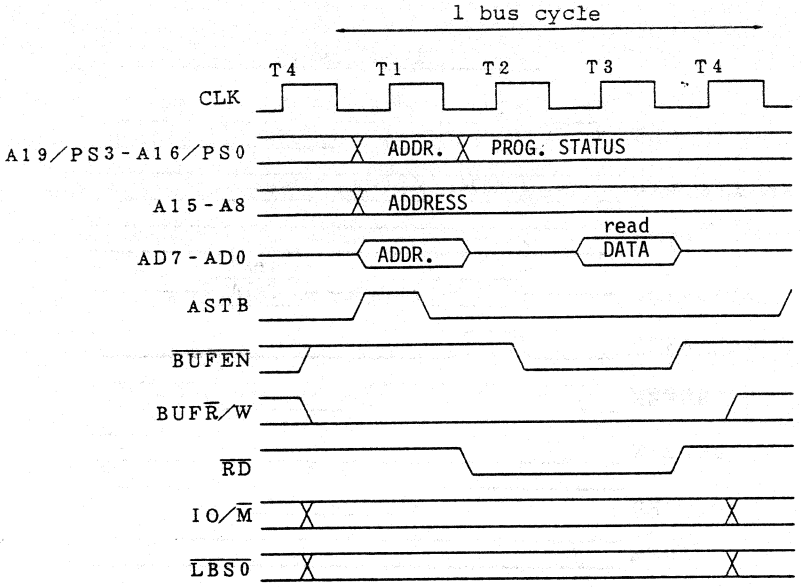


Fig. 5-1 Read Timing of μPD70108 Memory and I/O
(for small-scale systems)

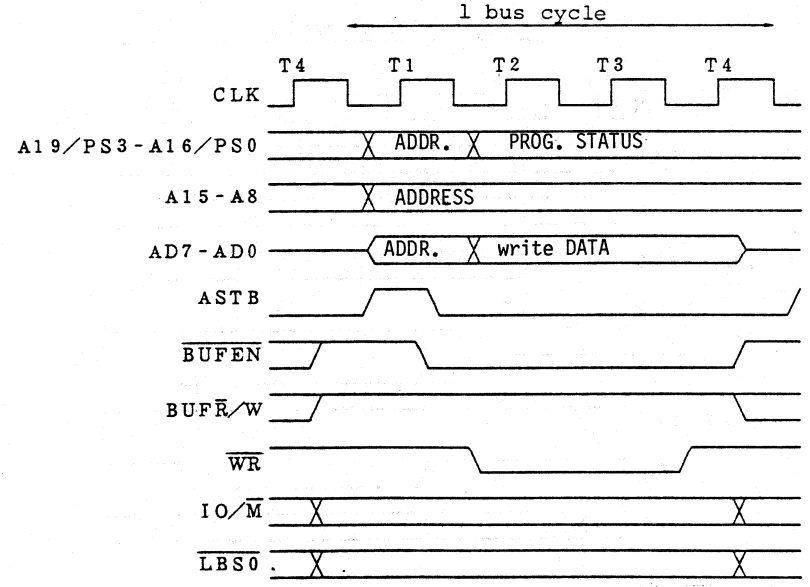


Fig. 5-2 Write Timing of μPD70108 Memory and I/O
(for small-scale systems)

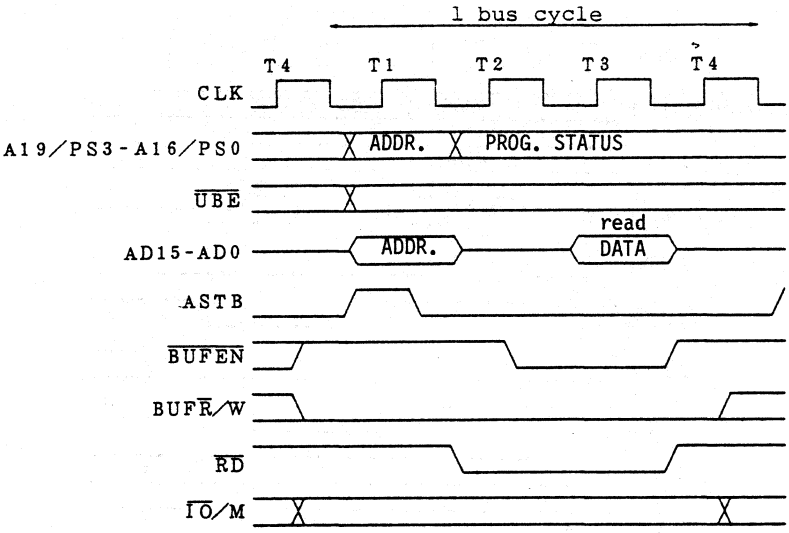


Fig. 5-3 Read Timing for μPD70116 Memory and I/O
(for small-scale systems)

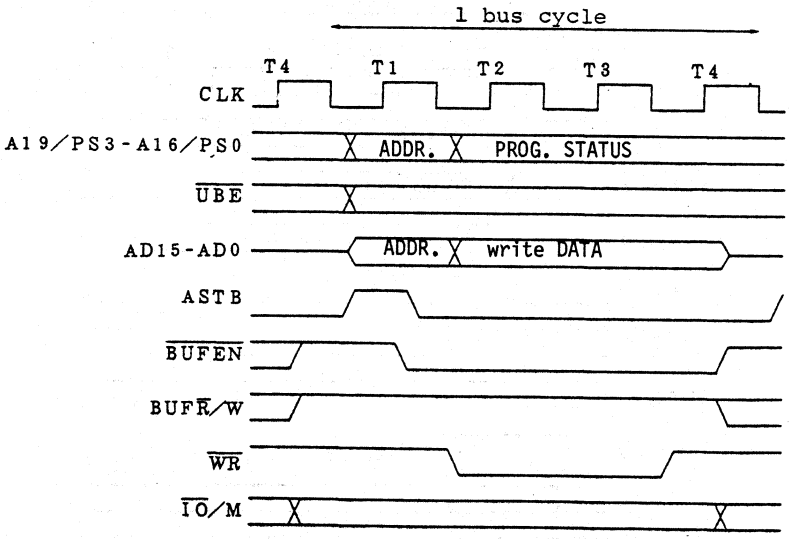


Fig. 5-4 Write Timing for μPD70116 Memory and I/O
(for small-scale systems)

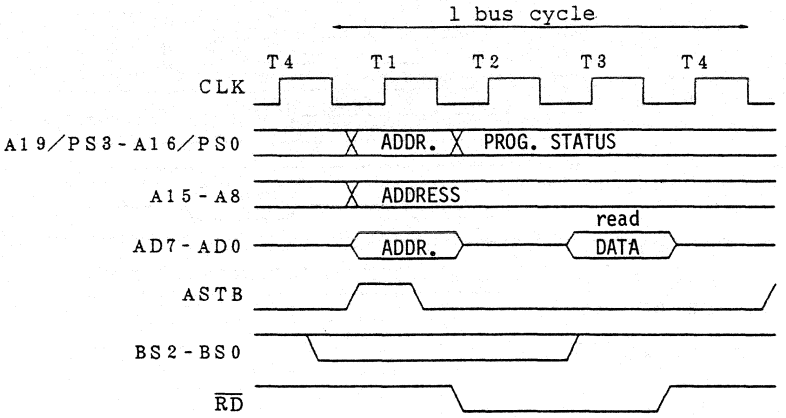


Fig. 5-5 Read Timing for μPD70108 Memory and I/O
(for large-scale systems)

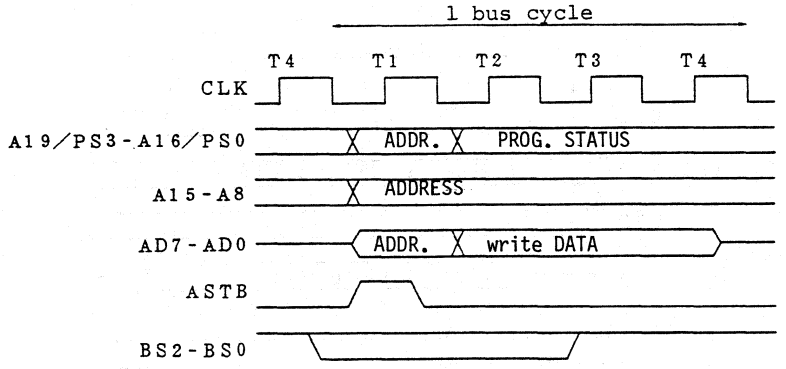


Fig. 5-6 Write Timing for μPD70108 Memory and I/O
(for large-scale systems)

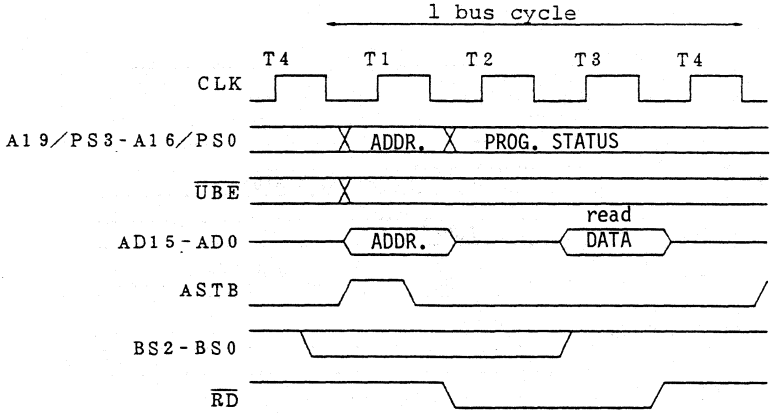


Fig. 5-7 Read Timing for μPD70116 Memory and I/O
(for large-scale systems)

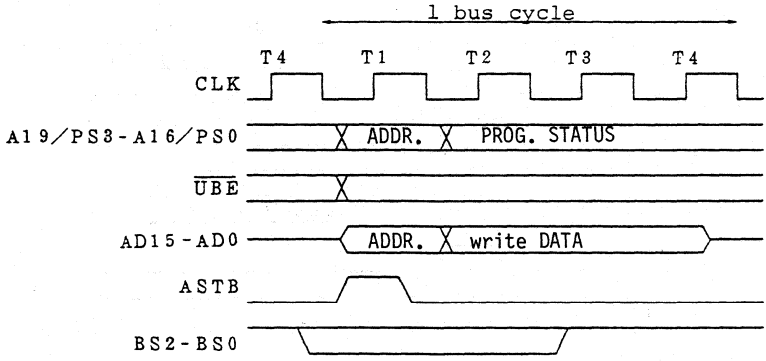


Fig. 5-8 Read Timing for μPD70116 Memory and I/O
(for large-scale systems)

Chapter 6 Interrupts

These interrupts accepted by the μPD70108/70116 are roughly classified into two types. One type is caused by an external interrupt request and the other is caused by a software process. Both types are vectored interrupts.
When an interrupt occurs, therefore, a location in the interrupt vector table that has been prepared beforehand is selected either automatically (a fixed vector) or by specification (a variable vector). This selected location determines the start address of the corresponding interrupt routine.
Table 6-1 shows the vector, priority, and number of clocks required to process each interrupt.

Table 6-1 Interrupt Source

	Interrupt source	No. of clocks	Vector	Priority
External	NMI (rising-edge active)	58/38	2	2
	INT (high-level active)	68/49	32-255	3
Software	DIVU divide error	65/45	0	1
	DIV divide error	65-75/45-55		
	CHKIND boundary over	81-84/53-56	5	
	BRKV	60/40	4	
	BRK3	58/38	3	
	BRK imm8		32-255	
	BRKEM imm8 CALLN imm8			
	BRK flag (single step)		1	4

NOTE: The number of clocks to the left of the slash (/) is for the μPD70108 and the number on the right is for the μPD70116.

The interrupt vector table is shown in Fig. 6-1. This table is allocated to a 1K-byte memory area (from addresses 000H to 3FFH) and can hold up to 256 vectors (four bytes are required per vector).

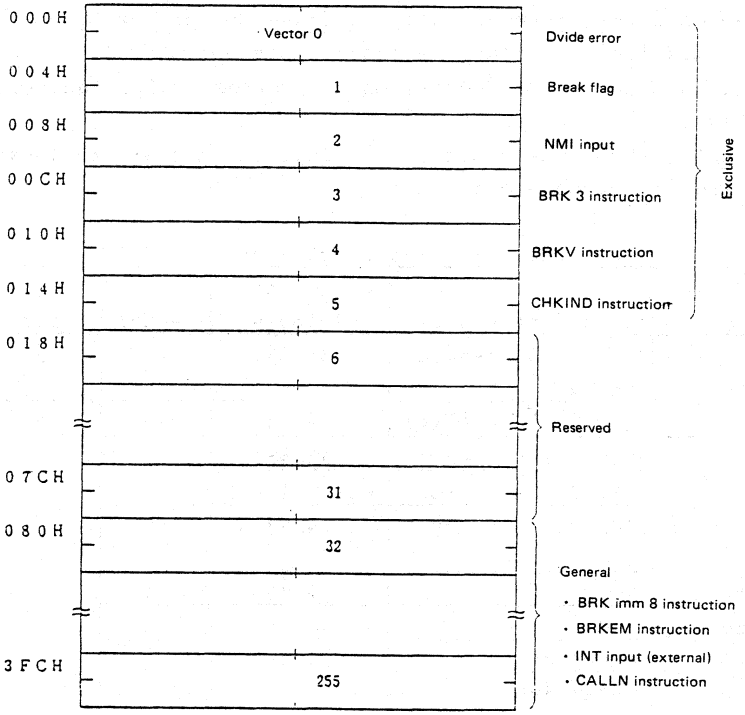


Fig. 6-1 Interrupt Vector Table

The interrupt sources that can use vectors 0 to 5 are specified and vectors 6 to 31 are reserved. These vectors, therefore, are not for general use.
Vectors 32 to 255 are for general use. These vectors are used for four interrupt sources: a 2-byte break, BRKEM, CALLN instructions (during emulation), and INT input.

Four bytes are used for each interrupt vector. The two bytes of the lower address and the two bytes of the higher address are loaded to the programmable counter (PC) as an offset and a base, respectively.

Example: vector 0

000H	001H
002H	003H

PS ← (003H, 002H)
PC ← (001H, 000H)

Based on this format, the contents of each vector are initialized at the beginning of a program.
The basic steps to be followed when the program execution jumps to an interrupt routine are as follows:

(SP-1, SP-2) ← PSW
(SP-3, SP-4) ← PS
(SP-5, SP-6) ← PC
SP ← SP-6
IE ← 0, BRK ← 0, MD ← 1
PS ← higher vector
PC ← lower vector

Because the interrupt enable (IE) and break (BRK) flags are reset when an interrupt routine is started, maskable interrupts (INT) and single-step interrupts are disabled.

6.1 INT Interrupts

If an INT input signal is detected at high level at the end of executing one instruction while interrupt is enabled (IE=1), the INT interrupt request will be acknowledged unless the NMI and hold request signals are active at the same time. The program execution then enters an interrupt acknowledge cycle (see Figs. 6-2 and 6-3).

The interrupt acknowledge cycle consists of two bus cycles. The $\overline{\text{INTAK}}$, ASTB, and $\overline{\text{BUFEN}}$ signals are output during the first cycle. Although the bus cycle is started, initially no data read/write operation is performed and the address/data bus becomes high impedance. At this time the hold request signal is not accepted. If the μPD70108/70116 is in the maximum mode, the $\overline{\text{BUSLOCK}}$ signal is output, inhibiting other devices from using the bus.

The first interrupt acknowledge cycle is necessary to synchronize with the external interrupt controller. When the $\overline{\text{INTAK}}$, ASTB, and $\overline{\text{BUFEN}}$ signals are output during the second interrupt acknowledge cycle, the external interrupt controller sends an interrupt vector to the data bus (AD7 to AD0).

After the second interrupt acknowledge cycle has been completed, the location in the interrupt vector table corresponding to the vector obtained during that acknowledge cycle is accessed. (At the same time, the contents of the PSW, PS, and PC are saved.) The interrupt start address is then output according to the contents of the interrupt vector table and the interrupt process routine is started.

The following provides interrupt acknowledge operations performed by the μPD70108 and μPD70116.

° μPD70108 interrupt acknowledge operation

1. Acknowledge cycle (first)
2. Acknowledge cycle (second)
3. Read lower byte of offset word
4. Read higher byte of offset word
5. Read lower byte of segment word
6. Read higher byte of segment word
7. Save lower byte of PSW
8. Save higher byte of PSW
9. Save lower byte of PS
10. Save higher byte of PS

11. Save lower byte of PC
12. Save higher byte of PC
13. Output interrupt start address.

- μPD70116 interrupt acknowledge cycle
  1. Acknowledge cycle (first)
  2. Acknowledge cycle (second)
  3. Read offset word
  4. Read segment word
  5. Save PSW
  6. Save PS
  7. Save PC
  8. Output interrupt start address

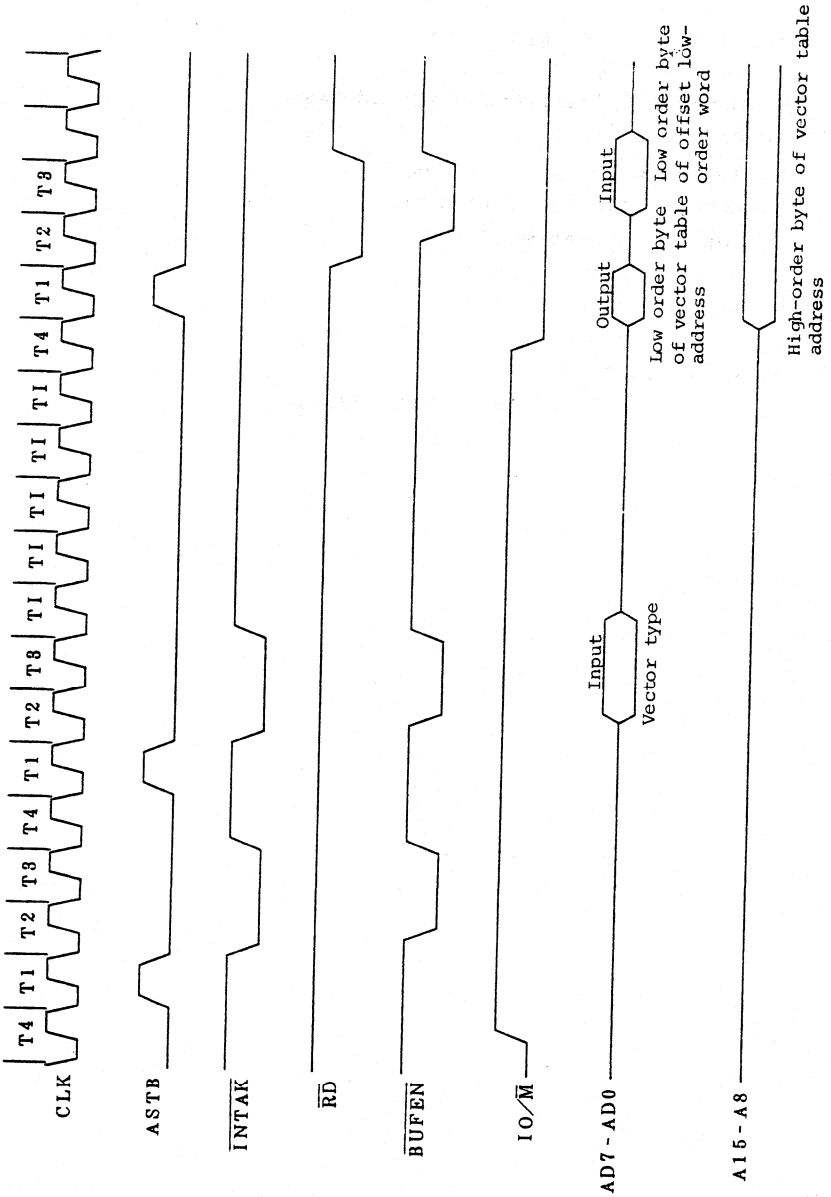


Fig. 6-2 Interrupt Acknowledge Timing (μPD70108)

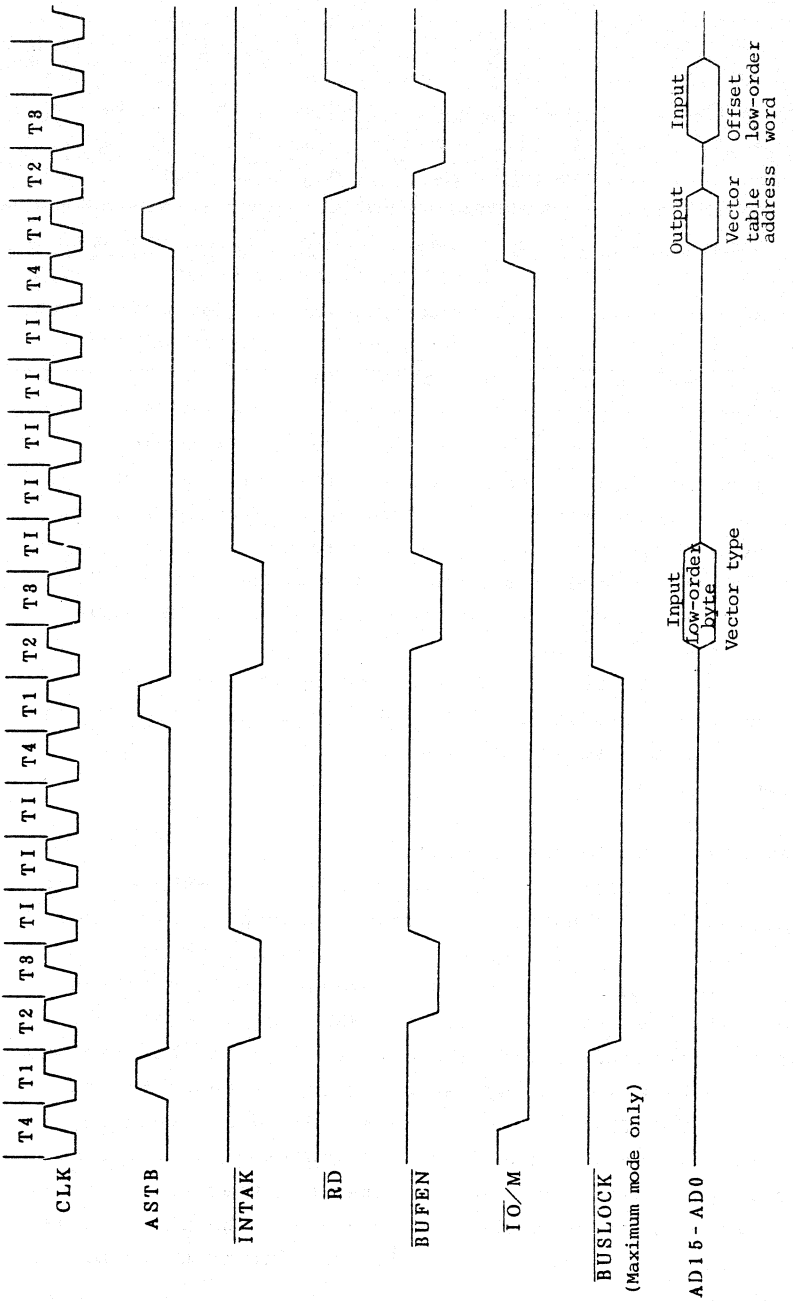


Fig. 6-3 Interrupt Acknowledge Timing (μPD70116)

During the first interrupt acknowledge cycle, no idle state (TI) is inserted in the bus cycle of the μPD70108. In the μPD70116, however, three TI's are inserted. During the second interrupt acknowledge cycle, five TI's are inserted in the bus cycles of both microprocessors. The CPU's of both the μPD70108 and μPD70116 read an 8-bit vector during the second interrupt acknowledge cycle. The number of cycles required to save the contents of the PSW, PS, and PC are different for the two microprocessors.

This is because the width of the μPD70108 data bus is smaller than that of the μPD70116. That is, two bus cycles are required for the μPD70108 to perform each of the necessary operations such as reading an offset word and a segment word, and saving the PSW, PS, and PC. In contrast, the μPD70116 performs each of these operations within one bus cycle.

The $\overline{\text{UBE}}$ signal of the μPD70116 remains at low level during the first and second interrupt acknowledge cycles and during the subsequent access of the offset and segment words.

6.2 BRK Flag (Single-Step) Interrupt

The μPD70108/70116 is provided with a single-step interrupt function that is useful for program dubugging and other operations. This interrupt is controlled by the break (BRK) flag ; bit 8 of the PSW. However, no instruction that directly sets or resets the BRK flag is available and the contents of the PSW must be saved to the stack to control the BRK flag. By restoring the contents to the PSW, the BRK flag is indirectly set or reset. When the BRK flag is set, an interrupt routine (monitor program, etc.) specified by vector 1 is started after the following instruction has been executed. The BRK flag is reset at this point together with the interrupt enable (IE) flag.

While the interrupt routine is being executed, the number of single-steps is checked. If it is judged that the single-step operation may be terminated, a memory operation instruction resets the BRK flag that is now saved in the stack. The CPU operation then returns to the main routine after which the next sequence of instructions is successively carried out.
If the execution returns to the main routine without operation of the BRK flag, the BRK (=1) flag saved to the stack will be restored to the PSW. One instruction of the main routine is then executed and vector 1 interrupt is caused again.

6.3 Timing That Does Not Accept Interrupts

While an instruction that directly sets data in the segment register is being executed or while the program execution is between three types of prefix instructions and the next single instruction or between the EI instruction and the next instruction, neither NMI, INT interrupts, nor single-step break is acknowledged as shown below.

- Between one of these instructions and the next instruction-- MOV sreg, reg16; Mov sreg,mem16; MCV reg16, sreg; MOV mem16,sreg; POP sreg
- Between one of these instructions and the next instruction--
  segment override prefix (PS:, SS:, DS0:, DS:)
- Between one of these instructions and the next instruction--
  repeat prefix (REPC, REPNC, REP, REPE, REPZ, REPNE, REPNZ)
- Between this instruction and the next instruction--
  bus lock prefix (BUSLOCK)
- Between the EI instruction and the next instruction

An NMI request signal generated during the abovementioned interrupt disable timing will be internally retained and acknowledged on completion of the subsequent single instruction.

6.4 Interrupt process During Block Transfer Instruction Execution

Should an external interrupt (NMI or INT when the interrupt is enabled) occur while a primitive block transfer/comparison, or I/O instruction is being executed, the CPU will acknowledge the interrupt and branches the execution to the corresponding interrupt address. At the beginning of the interrupt routine started in this manner, the contents of CW register that has been serving as a counter for block data will be saved to the stack. After the contents of the CW have been restored at the end of the interrupt process routine, the execution of the CPU will be returned to the original routine. In this way, the interrupted block operation will be resumed.

If prefix instructions have existed before the block operation instruction, up to three instructions will be retained.

On returning from the interrupt routine, the execution must return to the address at which the prefix instruction is held. For this purpose, the uPD70108/70116 is so designed that the return address when it is saved (minus one address per prefix instruction) is modified.
To make the best use of these functions, more than three prefix instructions must not be placed before a block operation instruction.

[Example of correct procedure]

```
        BUSLOCK
        REPC
NMI ──► CMPBKB dst-block, SS: src-block
```

In this example, the BUSLOCK, REPC, and SS: instructions are executed when the program execution has been returned from the NMI interrupt process.

[Example of incorrect procedure]

```
      BUSLOCK
      REP
      REPC
NMI → CMPBK dst-block, SS: src-block
```

In this example, it is judged that two types of prefix instructions exist (repeat prefix instruction is the same type) and two return addresses are accordingly modified. In reality, however, the prefix instruction occupies 3 addresses. Consequently, the program execution cannot return from the interrupt routine to the BUSLOCK instruction and, instead, will return to the REP instruction.

Chapter 7 Standby Function

The μPD70108/70116 is provided with standby mode. In standby mode, program execution can be terminated or resumed as required retaining all the statuses in the CPU. Moreover, by making the most of the CMOS characteristics of the microprocessor, the clock is not supplied except to the circuits related to the hold and standby release functions. As a result, the power consumption can be reduced to approximately 1/10 of that required for normal operation in native or emulation mode.

7.1 Setting Standby Mode

The standby mode can be set by executing the HALT instruction in native mode. When the microprocessor is in emulation mode, execute the HLT instruction to set the standby (hold) mode.

7.2 Standby Mode

In standby mode, the clock is not supplied to any circuit except those required to release the standby mode and those related to the hold function. Even if the supply of the clock is stopped, however, all CPU statuses immediately before the standby mode are retained. By stopping the supply of the clock to most of the circuits, power consumption drops to approximately 1/10th that normally required to execute programs.

Although the bus hold function can be used in standby mode, the CPU reenters the standby mode when the hold acknowledge cycle is completed.

The status of each output signal in standby mode is as follows:

Table 7-1 Signal Status in the Standby Mode

	Output signal	Status
Large-scale system mode	QS1,Q	Fixed at low level
	BS2-BS0	Fixed at high level
	$\overline{\text{BUSLOCK}}$	Fixed at high level (however fixed at low level if BUSLOCK instruction exists before HALT instruction.)
Small-scale system mode	$\overline{\text{INTAK}}$ $\overline{\text{BUFEN}}$ $\overline{\text{WR}}$ $\overline{\text{RD}}$	Fixed at high level
	ASTB	Fixed at low level
	BUF$\overline{\text{R}}$/W IO/$\overline{\text{M}}$ (μPD70108) $\overline{\text{IO}}$/M (μPD70116) $\overline{\text{LBS0}}$ (μPD70108)	Fixed at either high or low level
Common	$\overline{\text{UBE}}$ (μPD70116)	Fixed at high level
	A19/PS3-A16/PS0 A15-A8 (μPD70108) AD7-AD0 (μPD70108) AD15-AD0 (μPD70116)	Fixed at either high or low level

As can be seen from this table, the control outputs provided with an active level are fixed at inactive level in standby mode. The other output signals are fixed at either high or low level.

The standby mode is released by input of the RESET signal or an external interrupt (NMI or INT).

7.3 Releasing Standby Mode by External Interrupt Input

The standby mode is released when an input NMI or INT signal becomes active. If the standby mode has been released by the INT signal, the operation the CPU performs is determined depending on whether the interrupt has been enabled (EI) or disabled (DI) when the INT signal is recognized.

7.3.1 Releasing Standby Mode by NMI Input

Regardless of whether the CPU enters the standby mode from the native mode or the emulation mode, the standby mode is unconditionally released and the NMI interrupt routine of the native mode is started. If the RETI instruction is executed at the end of the NMI servicing routine, the CPU can reenter the mode set before the CPU entered the standby mode. The program execution is then resumed starting from the instruction next to the HALT/HLT instruction.

7.3.2 Releasing Standby Mode by INT input

(1) When interrupt is disabled (DI)

When the standby mode has been released, the CPU enters the mode that had been set before the standby mode was set. That is, if the standby mode was set while the CPU was in native mode, the CPU returns to the native mode upon releasing the standby mode. If the standby mode was set when the CPU was in emulation mode, the emulation mode is resumed. The program execution will be resumed starting from the instruction next to the HALT or HLT instruction.

NOTE: When releasing the standby mode by an input INT signal while the interrupt is disabled, the INT signal must be kept at high level until the instruction next to the HALT/HLT instruction is executed. In other words, the INT signal must remain high level for a lapse of 15 clocks. This applies on the assumption that the instruction

queue has become empty after executing the HALT/HLT instruction, however. If wait states are inserted, the number of inserted wait states must be accordingly added to the 15 clocks.

(2) When interrupt is enabled (EI)

The standby mode is released and the INT interrupt routine in the native mode is started regardless of whether the CPU was in native or emulation mode before the standby mode was set. If the RETI instruction is executed at the end of the INT interrupt routine, the CPU can return to the mode that had been set before the standby mode. The program execution will be resumed starting from the instruction next to the HALT/HLT instruction.

7.4 Releasing Standby Mode by RESET Input

The standby mode is unconditionally released when the RESET input signal becomes active regardless of whether the standby mode was set while the CPU was in native or emulation mode. On releasing the standby mode, a normal CPU reset operation is performed in native mode. The statuses of the CPU that have been retained during the standby mode will be consequently reset. The program execution once stopped by the standby mode therefore cannot be resumed.

Chapter 8 Reset Operation

The μPD70108/70116 is reset at the rising edge of the RESET signal and initialized as follows:

(1) The PFP (Prefetch Pointer) and PC (Program Counter) are cleared to 0000H.

(2) The PS (Program Segment) register is set to FFFFH.

(3) The SS (Stack Segment), DS0 (Data Segment 0), and DS1 (Data Segment 1) registers are cleared to 0000H.

(4) The instruction queue is cleared.

(5) Only the MD (Mode) flag of the PSW is set and the other flags are reset.

MD=1 : Specifies the native mode.

DIR=0 : Specifies the direction address modification (during block transfer).

IE=0 : Disables the INT interrupt.

BRK=0 : Disables the single-step interrupt.

(6) Registers other than the above will be undefined.

When the level of the RESET signal is lowered after the signal has been kept at high level for an elapse of four clocks or more, the CPU starts prefetching instructions starting from address FFFF0H.

Chapter 9 Logical and Physical Addresses

The μPD70108/70116 is provided with a 20-bit address bus (the lower 8/16 bits are also used as a data bus) and can access up to 1M bytes of memory area.
However, it is almost impossible for a programmer to write programs using 1M bytes of addresses that directly correspond to hardware (these addresses are referred to as "physical addresses").
To solve this problem, the μPD70108/70116 employs a memory segment method that allows the programmer to treat the 1M-byte memory area as an aggregate of logical address areas. These logical address areas are considered to be small units not directly dependent on physical addresses.
Four types of segments are used: program, stack, data 0, and data 1 segments. The addresses of each segment are determined according to the offset value of the first address of the logical segment that is specified by one of the four 16-bit segment registers (PS, SS, DS0, and DS1) each of which corresponds to each of the four logical segments.

Segment register	Default offset
PS	PFP
SS	SP, effective address
DS0	IX, effective address
DS1	IY

The function of each segment is described next.

(1) Program segment

The first address of the program segment is determined by the program segment (PS) register. The offset from the first address is specified by the prefetch pointer (PFP). This segment is assigned with instruction codes and table data, among others. The data in this segment can be accessed as general variables or source block data by using the segment override prefix (PS:) instruction.

(2) Stack segment

The first address of the stack segment is determined by the stack segment register. The offset from the first address is specified by the stack pointer (SP).

This segment is used as an area that saves the contents of the PC (return address), the PSW, and the general-purpose registers.

The data in this segment can be accessed as general variables or source block data by using the segment override prefix (SS:) instruction.

When addressing general variables, the SS register is automatically used as a segment register if the BP register is specified as the base register. The offset is specified by an effective address and therefore the data in the stack segment can be accessed as if they were general variables.

(3) Data segment 0

The first address of the data segment 0 is determined by the data segment 0 (DS0) register and the offset from the first address is specified by an effective address.

This segment is used to store general variables.

When executing a block transfer or BCD string operation instruction, this segment is used to store the source block data. On this occasion, however, the offset is determined according to the contents of the IX register.

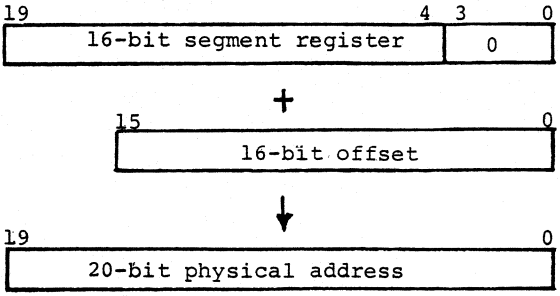
When the BP is specified as base register, default segment register is to be SS. In this case, programmer can override with segment override prefix (DS0:) and the data in the Data Segment 0 can be addressed with DS0 + BP.

(4) Data segment 1

The first address of this segment is determined by the data segment 1 register (DS1) and the offset from the first address is specified by the IY register.

This segment is used to store the destination block data when executing a block transfer or BCD string operation instruction.

The data in this segment can be accessed as general variables (the offset is determined by an effective address) or source block data (the offset is determined according to the contents of the IX register) by using the segment override prefix (DS1:) instruction.
With the segment method, the programmer can write programs paying attention to only the contents of the segment registers and the offset value of the contents. The contents of the segment registers may be a default or specified as an override. If the contents of a segment register constitute address 0, the offsets of the addresses in the segment specified by that segment register can be treated as logical addresses.
A program written as an aggregate of segments specified by logical addresses is compiled, assembled and treated as plural object modules. Each object module has its own segment name, size, partition, and control information. These object modules are processed by the linker and segment bases corresponding to physical addresses are specified. The object modules are then ready to be actually loaded to the memory.
The following figure shows the relation between a segment register, offset, and physical address.



To obtain a physical address, the contents of a segment register are multiplied by 16 and an offset value is added. The result is used as a physical address. Note that the contents of the segment register and the offset value are treated as unsigned data.

Unless a specific program is executing an instruction that modifies the segment base (unless a branch instruction or a variable reference is in the segment), the addresses in the program can be determined by the offset obtained from the contents of a segment register.

The program can be loaded to any memory area only by matching the contents of the segment register with the first physical address of the memory area to which the program is to be loaded.

By using this feature of the segment method, a program stored in an external file such as a floppy disk can be loaded to available buffer memory and run when the program is called by the program currently being executed by the CPU.

In this manner, a program stored in a file or separated into plural files is loaded to an available memory area. This is called "dynamic relocation".

Chapter 10 Addressing

10.1 Instruction Address

The current address of μPD70108/70116 is automatically incremented every time an instruction is executed. In addition, the microprocessor is provided with the following addressing methods for controlling execution procedure of instructions.

10.1.1 Direct addressing

A 2- or 4-byte immediate data in an instruction is directly loaded to the PC alone or to both the PS and PC. The immediate data is then used as a branch address. This addressing method is employed when executing the following instructions.

```
CALL far-proc
CALL memptr16
CALL memptr32
BR   far-label
BR   memptr16
BR   memptr32
```

10.1.2 Relative addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and added to the contents of the PC. The result of this addition is used as a branch address.

The sign bit of an 8-bit displacement value is extended and added to the contents of the PC as a 16-bit data. When the addition is performed, the contents of the PC indicate the first address of the next instruction.

This addressing method is employed when executing the following instructions.

```
CALL near-proc
BR   near-label
BR   short-label
Conditional Branch Instruction   short-label
```

10.1.3 Register addressing

The contents of any 16-bit register specified by the register-specifying field (3 bits) of an instruction are loaded to the PC as a branch address. Unlike when an immediate data is used as a branch address, this addressing method allows all the eight 16-bit registers (AW, BW, CW, DW, IX, IY, SP, and BP) to be used. This addressing method is used when executing the following instructions:

	Example
CALL regptr 16	CALL AW
BR regptr 16	BR BW

10.1.4 Register indirect addressing

A 16-bit register (IX, IY, or BW) is specified by the register-specifying field in an instruction. The specified register then addresses the contents of the memory (word or double word).

The addressed contents are then loaded to the PC (or to both the PC and PS) as a branch address.

```
CALL   memptr16        CALL WORD PTR  [IX]
CALL   memptr32        CALL DWORD PTR [IY]
BR     memptr16        BR   WORD PTR  [BW]
BR     memptr32        BR   DWORD PTR [IX]
```

NOTE: Instruction code memptr 16 and memptr 32 are generated by the assembler in response to keywords WORD PTR and DWORD PTR, respectively.

10.1.5 Indexed addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and is added to the contents of a 16-bit register that serves as an index register (IX or IY).

The result of this addition addresses memory operand (word or double word) and it is loaded to PC as branch address.

		Example	
CALL	memptr 16	CALL	var [IX][2]
CALL	memptr 32	CALL	var [IY]
BR	memptr 16	BR	var [IY]
BR	memptr 32	BR	var [IX + 4]

10.1.6 Based addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and is added to the contents of a 16-bit register (BP or BW) that serves as base register. The contents of the memory addressed by the result of this addition (word or double word) are loaded to the PC as a branch address.

This addressing method is employed when executing the following instructions:

	Example
CALL memptr16	CALL var [BP + 2]
CALL memptr32	CALL var [BP]
BR memptr16	BR var [BW][2]
BR memptr32	BR var [BP]

NOTE: Instruction code memptr 16 is generated by the assembler if variable var has a word attribute. If it has a double word attribute, instruction code memptr 32 is generated.

10.1.7 Based indexed addressing

A 1- or 2-byte immediate data in an instruction byte is treated as a signed displacement value. This value is added to the contents of a 16-bit register that serves as a base register (BP or BW) and to the contents of a 16-bit register that serves as an index register (IX or IY). The result of this addition addresses the contents of the memory (word or double word). The addressed memory contents are loaded to the PC as a branch address. This addressing method is employed when executing the following instructions:

```
CALL memptr16
CALL memptr32
BR   memptr16
BR   memptr32
```

Example

```
CALL var [BP][IX]
CALL var [BW + 2][IY]
BR   var [BW][2][IX]
BR   var [BP + 4] [IY]
```

NOTE: Instruction code memptr 16 is generated by the assembler if variable var has a word attribute. If it has a double word attribute, instruction code memptr 32 is generated.

10.2 Memory Operand Address

This section describes several addressing methods for addressing registers or the memory when executing instructions.

10.2.1 Register addressing

The contents of the register-specifying field (reg=3-bit field, sreg=2-bit field) in an instruction addresses a register. The 3-bit field "reg" is used in pairs with one bit (bit W) that is in the same instruction and indicates whether a word or a byte register is to be specified. Eight types of word registers (AW, BW, CW, DW, BP, SP, IX, and IY) and eight types of byte registers (AL, AH, BL, BH, CL, CH, DL, and DH) are specified.

The 2-bit field "sreg" specifies four types of segment registers (PS, SS, DS0, and DS1).

On some occasions, the operation code of an instruction specifies a register.

This addressing method is employed when executing the following instructions that have the following operand-writing formats:

Format	Items
reg	AW, BW, CW, DW, SP, BP, IX, IY, AL, AH, BL, BH, CL, CH, DL, DH
reg16	AW, BW, CW, DW, SP, BP, IX, IY
reg8	AL, AH, BL, BH, CL, CH, DL, DH
sreg	PS, SS, DS0, DS1
acc	AW, AL

Example

When MOV reg,reg is specified.

```
MOV BP,SP
MOV AL,CL
```

10.2.2 Immediate addressing

A 1-or 2-byte immediate data in an instruction is used as is.

This addressing method is employed when executing the instructions that have the following operand-writing formats:

Format	Items
imm	8/16-bit immediate data
imm16	16-bit immediate data
imm8	8-bit immediate data
pop-value	16-bit immediate data

If imm alone is specified, the assembler checks the value of imm written as an operand or the attribute of other operands that may be written at the same time and judges whether the value of imm is 8 bits or 16 bits. The status of word/byte-specifying bit W is then determined.

Example

When MOV reg,imm is specified

```
MOV AL, 5: byte
```

When MUL reg16,reg16,imm16 is specified

```
MUL AW,BW,1000H
```

10.2.3 Direct addressing

The immediate data in an instruction addresses the memory.

This addressing method is employed when executing the instructions that have the following operand-writing formats:

Format	Item
mem	16-bit variable that specifies an 8- or 16-bit memory data
dmem	16-bit variable that specifies an 8- or 16-bit memory data
imm4	4-bit variable that indicates the bit length of the bit field data

Example

When MOV mem,imm is specified

```
MOV WORD-VAR, 2000H
```

When MOV acc,dmem is specified

```
MOV AL,BYTE-VAR
```

10.2.4 Register indirect addressing

A 16-bit register (IX, IY, and BW) specified by the memory-specifying field in an instruction addresses the memory.

This addressing method is employed when executing the instructions that have the following operand-writing formats:

Format	Method
mem	[IX],[IY],[BW]

Example

When SUB mem,reg is specified

```
SUB [IX],AW
```

10.2.5 Autoincrement/-decrement addressing

This addressing method falls into the category of register indirect addressing.

The contents of a default register addresses a register or memory. Then the contents of the default register are automatically incremented/decremented by one if a byte process is performed. If it is a word process, the register contents are incremented/decremented by two. Stated another way, the address is automatically modified by this addressing function for processing the next byte/word operand. This addressing method is always applicable to default registers and is employed when executing the instructions that have the following operand formats:

Format	Default register
dst-block	IY
src-block	IX

This addressing will control block data operations if it is used in combination with a counter (CW) that counts the number of repetitions of a byte/word operand operation.

10.2.6 Indexed addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and is added to the contents of a 16-bit register that serves as an index register (IX or IY). The result of this addition addresses a memory operand.

This addressing is useful when accessing an array of data. The displacement value indicates the start address of the array.

The contents of the index register determines the address of the data to be accessed.

This addressing method is employed when executing the instructions that have the following operand-writing formats:

Format	Method
mem	var [IX], var [IY]
mem16	var [IX]
mem8	var [IX]

Example

When TEST mem,imm is specified

```
TEST BYTE-VAR[IX],7FH
TEST BYTE-VAR[IX+8], 7FH
TEST WORD-VAR[IX][8],7FFFH
```

NOTE: If variable var has a byte attribute, a byte operand is specified. If it has a word attribute, a word operand is specified. An instruction code is generated by the assembler to each operand.

10.2.7 Based addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value and is added to the contents of a 16-bit base register that serves as a base register (BP or BW). The result of this addition addresses a memory operand.

This addressing is useful to access structural data that are stored at separate memory locations. The base register indicates the start address of each structural data and the displacement value selects one piece of data from each structural data.

This addressing method is employed when executing the instructions that have the following operand-writing format:

Format	Method
mem	var[BP],var[BW]
mem16	var[BP]
mem8	var[BP]

Example

When SHL mem,1 is specified

```
SHL BYTE-VAR[BP],1
SHL WORD-VAR[BP+2],1
SHL BYTE-VAR[BP][4],1
```

NOTE: If variable var has a byte attribute, a byte operand is specified. If it has a word attribute, a word operand is specified. An instruction code corresponding to each operand is generated by the assembler.

10.2.8 Based indexed addressing

A 1- or 2-byte immediate data in an instruction is treated as a signed displacement value that is added to the contents of two 16-bit registers.
One of the registers serves as a base register (BP or BW) and the other as an index register (IX or IY). The result of the addition addresses a memory operand.
Since this addressing method allows accessing one data by modifying the contents of both the base and index registers, it is very useful when accessing structural data that are stored at separate memory locations and include data array.
For example, the contents of the base register indicates the first address of each structual data. The displacement value in turn indicates the number of offsets from that first address to the first address of a data array.
Then the index register can indicate a specific data in the data array.
This addressing method is employed when executing instructions that have the following operand-writing format:

Format	Item
mem	var [base register][index register]
mem16	var [base register][index register]
mem8	var [base register][index register]

Example

When PUSH mem16 is specified

```
PUSH WORD-VAR [BP][IX]
PUSH WORD-VAR [BP+2][IX+6]
PUSH WORD-VAR [BP][4][IX][8]
```

10.2.9 Bit addressing

A 3- or 4-bit immediate data in an instruction or lower 3- or 4-bit of the CL register specifies one bit of the 8- or 16-bit register or memory.

By using this addressing method, a specific single bit in a register or the memory can be tested (for 0 or 1), set, cleared, or inverted without affecting the contents of other bits. That is, unlike when setting or resetting a bit by using the AND or OR instruction, a byte or word data does not have to be prepared to operate one bit. This method is employed when executing the instructions that have the following operand-writing formats:

Format	Item
imm4	Bit number of word operand
imm3	Bit number of byte operand
CL	CL

Example

```
TEST1 reg8,CL
TEST1 AL,CL
NOT1  reg8,imm3
NOT1  CL,5
CLR1  mem16,CL
CLR1  WORD-VAR[IX],CL
SET1  mem16,imm4
SET1  WORD-VAR[BP],9
```

Chapter 11 Implementation of Faster Execution

To reduce the time required to execute instructions, the uPD70108 comes with these hardware functions.

Dual data bus in EXU

Effective address generator

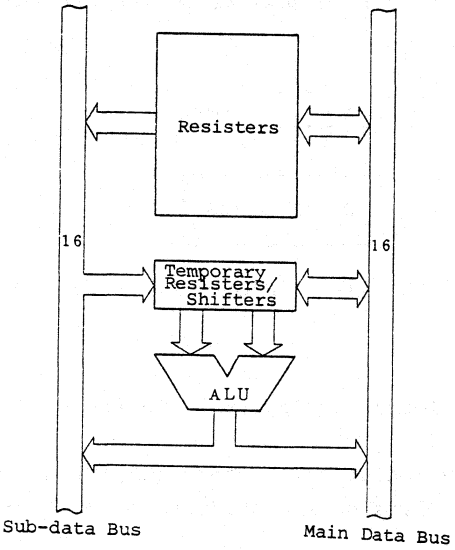
16/32-bit temporary registers/shifters (TA, TB)

16-bit loop counter

PC and PFP

11.1 Dual Data Bus Method

To reduce the number of processing steps for instruction execution, the dual data bus method has been adopted for the uPD70108/70116. The two data buses (the main data bus and the subdata bus) are both 16 bits wide. For addition/subtraction and logical and comparison operations, processing time has been speeded up some 30% over single-bus systems.



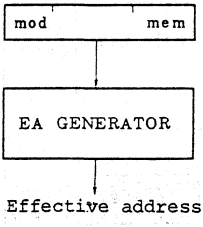
Example

```
ADD   AW, BW    ;AW ← AW + BW

   Single bus          Dual bus

Step 1 ALU ← AW        ALU ← AW, BW

Step 2 ALU ← BW        AW ← ALU

Step 3 AW ← ALU
```

11.2 Effective Address Generator

This circuit performs high-speed processing to calculate effective addresses required when accessing memory.

Calculationg an effective address by the microprogramming method normally requires 5 to 12 clock cycles. This hardware circuit, however, reserved only for calculating effective addresses, requires only two clock cycles for addresses to be generated for any addressing mode. Thus, processing is several times faster with this highly effective hardware assist.



11.3 16/32-Bit Temporary Registers/Shifters (TA,TB)

These temporary registers/shifters (TA,TB) are provided for multiplication/division and shift/rotation instructions.

Adopting these circuits has particularly speeded up execution of mulitplication/division instructions. In fact, these instructions can be executed about four times faster than with the microprogramming method.

TA + TE: 32-bit temporary register/shifter for multiplication and division instructions

TB: 16-bit temporary register/shifter for shift/rotation instructions

11.4 Loop Counter (LC)

This counter is used to count the number of loops for a primitive block transfer instruction controlled by a repeat prefix instruction and the number of shifts that will be performed for a multiple bit shift/rotation instruction.

The processing performed for multiple bit rotation of a register is shown below. The average speed is approximately doubled over the microprogram method.

Example

```
RORC    AW, CL    ;CL=5
```

Microprogram method	LC method
8 + (4 x 5) = 28 clocks	7 + 5 = 12 clocks

11.5 PC and PFP

The uPD70108 microprocessor has a program counter which addresses the program memory location to be executed next, and a prefetch pointer, which addresses the program memory location to be fetehed next. Both functions are provided by hardware. A time savings of several clocks is realized for branch, call, return, and break instruction execution, compared with microprocessors that have only one instruction pointer.

Table 12-1 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 address
mem8	8-bit memory address
mem16	16-bit memory address
mem32	32-bit memory address
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 accumlator
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 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

Table 12-2 Operand Types
(continued from Table 12-1)

Identifier	Description
regptr16	16-bit general purpose register containing offset of call address within current program segment
memptr16	16-bit memory address containing offset of call address within current program segment
memptr32	32-bit memory address containing offset of call address and segment data in 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 flating point processor chip
R	Register set (AW, BW, CW, DW, SP, BP, IX, IY)
DS1-spec	1) DS1 2) Segment or Group name assumed to DS1
Seg-spec	1) Any name of segment register 2) Segment or Group name assumed to segment register
[ ]	May be omitted

Table 12-3 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-sxtention specification bit (1-sign extention,0-no sign extention)

Table 12-4 Instruction Words (continued)

Identifier	Description
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
seg-high	High 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-hihg	High byte of 16-bit data which specifies number of bytes of stack to be discarded
disp 8	8-bit displacement added to PC
X XXX YYY ZZZ	Operation codes for external floating Point processor chip

Table 12-5 Operation description

Identifier	Description
AW	Accumulator (16 bits)
AH	Accumulator (high byte)
AL	Accumulator (low byte)
BW	BW registor (16 bits)
CW	CW register (16 bits)
CL	CW 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)
IY	Index register (destination) (16 bits)
PS	Program segment register (16 bits)
DS1	Data segment 1 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
MD	Mode 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
x	Multiplication

Table 12-6 Operation description(continued)

-	Division
%	Modulo
∧	Logical and
∨	Logical or
⊻	Exclusive or
XXH	Hexadecimal 2-digit data
XXXXH	Hexadecimal 4-digit data

Table 12-7 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 12-8 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 12-9 Selection of 8- and 16-bit Registers

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	SP
1 0 1	CH	BP
1 1 0	DH	IX
1 1 1	BH	IY

Table 12-10 Selection of Segment Registers

sreg	
0 0	DS 1
0 1	PS
1 0	SS
1 1	DS 0

The following pages show the format and descriptions for the different instructions of the instruction set.

No. of clocks includes these times;

- Decoding
- EA Generation
- Operand fetch
- Execution

and assumes the instruction byte (s) have been prefetched.

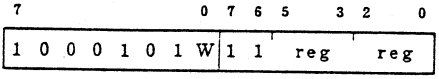
12.1 Data Transfer Instructions

12.1.1 MOV (Move)

(1) Register to register

1) Description format
MOV reg,reg

2) Instruction format



3) Number of bytes
2

4) Number of clocks
2

5) Number of transfers of 16-bit words
None

6) Function
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.

reg ← reg

7) Flag operation
None

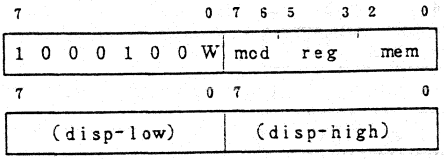
8) Description example
MOV BP,SP

2) Register to memory

1) Description format

MOV mem,reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 9

When W=1, 13: μPD70108

μPD70116 odd addresses

9: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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.

(mem) ← reg

7) Flag operation

None

8) Description example

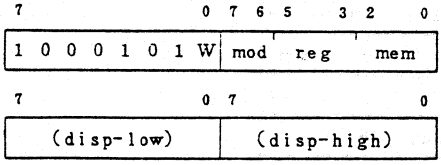
MOV [BP][IX],AW

(3) Memory to register

1) Description format

MOV reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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.

reg ← (mem)

7) Flag operation

None

8) Description example

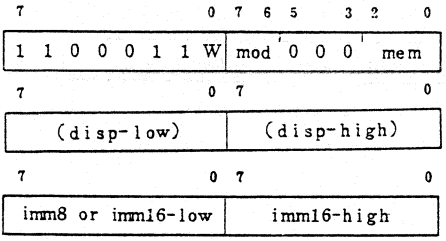
MOV AW, [BW][IY]

(4) Immediate data to memory

1) Description format

MOV mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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.

(mem) ← imm

7) Flag operation

None

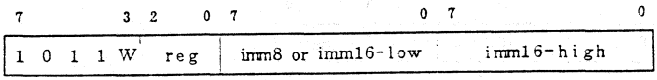
8) Description example

MOV [BP][IX],0000H

(5) Immediate data to register

1) Description format
MOV reg,imm

2) Instruction format



3) Number of bytes
2/3

4) Number of clocks
4

5) Number of transfers of 16-bit words
None

6) Function
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.

reg ← imm

7) Flag operation
None

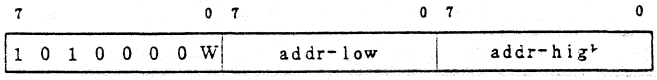
8) Description example
MOV BP, 8000H

(6) Memory to accumulator

1) Description format

MOV acc,dmem

2) Instruction format



3) Number of bytes

3

4) Number of clocks

When W=0, 10

When W=1, 14: μPD70108
μPD70116 odd addresses
10: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

Transfers the memory contents addressed by the second operand to the accumulator (AL or AW) specified by the first operand.

When W=0 AL ← (dmem)

When W=1 AW ← (dmem +1, dmem)

7) Flag operation

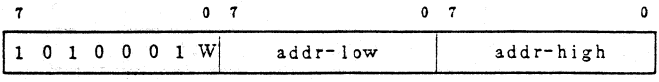
None

(7) Accumulator to memory

1) Description format

MOV dmem,acc

2) Instruction format



3) Number of bytes

3

4) Number of clocks

When W=0, 9

When W=1, 13: μPD70108
μPD70116 odd addresses
9: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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.

When W=0 (dmem) ← AL

When W=1 (dmem +1, dmem) ← AW

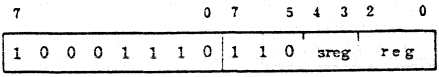
7) Flag operation

None

(8) Register to segment register

1) Description format
MOV sreg,reg16

2) Instruction format



3) Number of bytes
2

4) Number of clocks
2

5) Number of transfers of 16-bit words
None

6) Function

sreg ← reg16

Transfers the contentes of the 16-bit register specified 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 instruction.

7) Flag operation
None

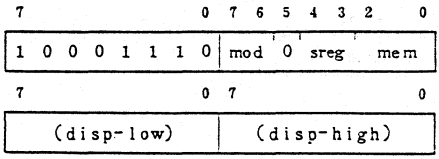
8) Description example
MOV SS,AW

(9) Memory to segment register

1) Description format

MOV sreg,mem16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

sreg ← (mem16)

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 instruction.

7) Flag operation

None

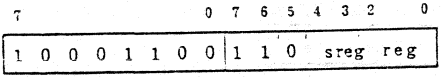
8) Description example

MOV DS0,SEG[BW][IX]

(10) Segment register to register

1) Description format
MOV reg16,sreg

2) Instruction format



3) Number of bytes
2

4) Number of clocks
2

5) Number of transfers of 16-bit words
None

6) Function
Transfers the contents of the segment register specified by the second operand to the 16-bit register specified by the first operand.

reg16 ← sreg

7) Flag operation
None

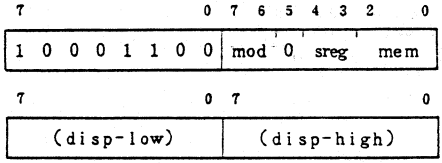
8) Description example
MOV AW,DS1

(11) Segment register to memory

1) Description format

MOV mem16,sreg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 10

When W=1, 14: μPD70108
μPD70116 odd addresses
10: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

Transfers the contents of the segment register specified by the second operand to the 16-bit memory location addressed by the first operand.

(mem16) ← sreg

7) Flag operation

None

8) Description example

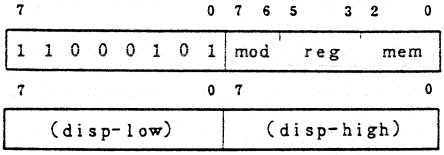
MOV [IX],PS

(12) 32-bit memory to 16-bit register and DS0

1) Description format

MOV DS0,reg16,mem32

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

26: μPD70108

μPD70116 odd addresses

18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

reg16 ← (mem32)

DS0 ← (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 DS0 segment register.

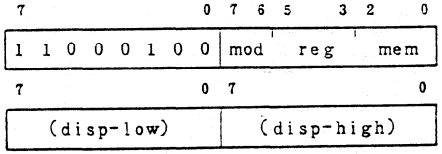
7) Flag operation

None

(13) 32-bit memory to 16-bit register and DS1

1) Description format
   MOV DS1,reg16,mem32

2) Instruction format



3) Number of bytes
   2/3/4

4) Number of clocks
   26: μPD70108
       μPD70116 odd addresses
   18: μPD70116 even addresses

5) Number of transfers of 16-bit words
   2

6) Function
   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 DS1 segment register.

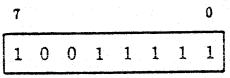
7) Flag operation
   None

(14) PSW to AH

1) Description format

MOV AH,PSW

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

AH ← S,Z,X,AC,X,P,X,CY

Transfers each of flags S, Z, AC, P, and CY of PSW to the AH register. Bits 5, 3, and 1 will be undefined.

7) Flag operation

None

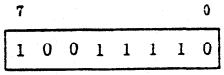
8) Description example

MOV AH,PSW

(15) AH to PSW

1) Description format
MOV PSW,AH

2) Instruction format



3) Number of bytes
1

4) Number of clocks
3

5) Number of transfers of 16-bit words
None

6) Function

S,Z,X,AC,X,P,X,CY ← AH

Transfers bits 7, 6, 4, 2, and 0 of the AH register to each of flags S, Z, AC, P, and CY of PSW.

7) Flag operation

V	S	Z	AC	P	CY
	×	×	×	×	×

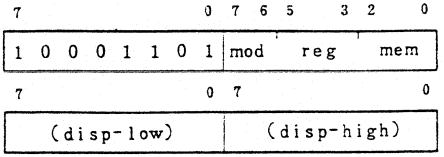
8) Description example
MOV PSW,AH

12.1.2 LDEA (Load Effective Address to register)

1) Discription format

LDEA reg16,mem16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

Loads the effective address (offset) generated by the second operand to the 16-bit general purpose register specified by the first operand.

This is used to set starting address values to the registers used to automatically specify the operand for TRANS or block instructions.

reg16 ← mem16

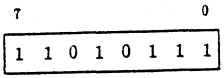
7) Flag operation

None

12.1.3 TRANS/TRANSB (Translate byte)

1) Description format
   TRANS src-table
   TRANS (no operand)
   TRANSB (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   9

5) Number of transfers of 16-bit words
   1

6) Function
   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 from the starting address.

   AL ← (BW + AL)

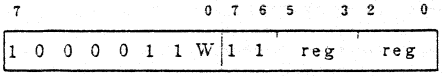
7) Flag operation
   None

12.1.4 XCH (Exchange)

(1) Register with register

1) Description format
   XCH reg,reg

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   3

5) Number of transfers of 16-bit words
   None

6) Function
   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.

   reg ↔ reg

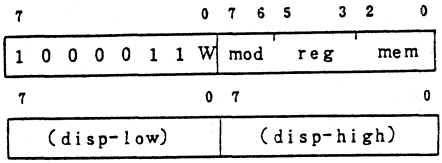
7) Flag operation
   None

8) Description example
   XCH AW,BW

(2) Memory with register

1) Description format
XCH mem,reg or XCH reg,mem

2) Instruction format



3) Number of bytes
2/3/4

4) Number of clocks
When W=0, 16
When W=1, 24: μPD70108
μPD70116 odd addresses
16: μPD70116 even addresses

5) Number of transfers of 16-bit words
2

6) Function
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.

(mem) ↔ reg

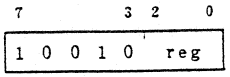
7) Flag operation
None

8) Description example
XCH WORD_VAR[BP], CW

(3) Accumulator with register

1) Description format
XCH AW,reg16 or XCH reg16, AW

2) Instruction format



3) Number of bytes
1

4) Number of clocks
2

5) Number of transfers of 16-bit words
None

6) Function
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.

AW ↔ reg16

7) Flag operation
None

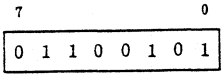
8) Description example
XCH AW, DW

12.2 Repeat Prefix

12.2.1 REPC (Repeat While Carry)

1) Description format
   REPC (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   While CW≠0, the block comparison instruction (CMPBK or CMPM) placed in the following byte is executed after which CW is decremented (-1).
   If the result of the block comparison instruction is CY≠1, program execution exits the loop.
   CW is checked before execution of the block comparison instruction. That is, against the condition immediately before execution of the REPC 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 even once.
   CY check is performed to test the result of the block comparison instruction; the contents of CY immediately before the first execution of the REPC instruction are "don't care".

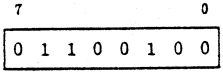
7) Flag operation
None

8) Description example
REPC CMPBKW

12.2.2 REPNC (Repeat While Not Carry)

1) Description format
   REPNC (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   While CW≠0, the block comparison instruction (CMPBK or CMPM) placed in the following byte is executed after which CW is decremented (-1). If the result of the comparison instruction is CY=1 program execution exits the loop. CW is checked before execution of the block comparison instruction. That is, against the condition immediately before execution of the REPNC 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 even once.
   CY check is performed to test the result of the block comparison instruction; the contents of CY immediately before the first execution of the REPNC instruction are "don't care".

7) Flag operation
   None

8) Description example
   REPNC CMPMB

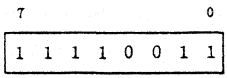
8.2.3 REP/REPE/REPZ (Repeat/Repeat while Equal/Repeat while Zero)

1) Description format

REP (no operand)

REPE/REPZ (no operand)

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

While CW≠0, the block transfer/comparison/input/output instruction is executed and CW is decremented (-1).

REP is used with MOVBK, LDM, STM, OUTM, or INM instruction and performs repeat operation while CW≠0 but disregarding Z flag.

REPZ or REPE is used with 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 before execution of the block instruction. That is, it is done 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 even once.

Z 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."

7) Flag operation

None

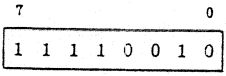
8) Description example

```
REP MOVBKW
REPZ CMPBKW
```

12.2.4 REPNE/REPNZ (Repeat while Not Equal/Repeat while Not Zero)

1) Description format
   REPNE/REPNZ (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   While CW≠0, the block comparison instruction (CMPBK, CMPM) is executed after which CW is decremented (-1).
   If the result of the block comparison instruction is Z≠0, program execution exits the loop.
   CW is checked before execution of the block comparison instruction. That is, against the condition immediately before execution of the REPNE/REPNZ 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 even once.
   Z 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."

7) Flag operation
None

8) Description example
REPNE CMPMB
REPNZ CMPBKW

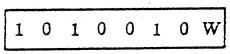
12.3 Primitive Block Transfer Instruction

12.3.1 MOVBK/MOVBKB/MOVBKW (Move Block/Move Block Byte/Move Block Word)

1) Description format

(repeat) MOVBK [DS1-spec:]dst-block,[Seg-spec:]src-block
(repeat) MOVBKB (no operand)
(repeat) MOVBKW (no operand)

2) Instruction format



3) Number of bytes

1

4) Number of clocks

Repeat: When W=0, 11+8/rep
When W=1, 11+16/rep: µPD70108
µPD70116 odd, odd addresses
11+12/rep: µPD70116 odd, even addresses
11+8/rep : µPD70116 even,
even addresses

Single operation:
When W=0, 11
When W=1, 19 : µPD70108
µPD70116 odd, odd addresses
15 : µPD70116 odd, even addresses
11 : µPD70116 even,
even addresses

5) Number of transfers of 16-bit words

Repeat : 2/rep
Single operation: 2

6) Function

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 to the block addressed by the IY by repeating byte or word data. 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 1 byte or 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, it is specified directly to the byte or word type.

The destination block must always be located within the segment specified by the DS1 segment register; segment override is disabled. On the other hand, the default segment for the source block register is DS0, and segment override is possible. The source block may be located in the segment specified by any of the segment registers.

7) Flag operation

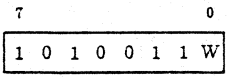
None

12.3.2 CMPBK/COMPBKB/CMPBKW (Compare Block/Compare Block Byte/Compare Block Word)

1) Description format

(repeat) CMPBK [DSl-spec:]dst-block, [Seg-spec:] src-block
(repeat) CMPBKB (no operand)
(repeat) CMPBKW (no operand)

2) Instruction format



3) Number of bytes

1

4) Number of clocks

Repeat: When W=0, 7+14/rep
When W=1, 7+22/rep: μPD70108
μPD70116 odd, odd addresses
7+18/rep: μPD70116 odd, even addresses
7+14/rep: μPD70116 even, even addresses

Single operation:
When W=0, 13
When W=1, 21 : μPD70108
μPD70116 odd, odd addresses
17 : μPD70116 odd, even addresses
13 : μPD70116 even, even addresses

5) Number of transfers of 16-bit words

Repeat : 2/rep
Single operation: 2

6) Function

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

Subtracts the block addressed by the IY from the block addressed by the IX repeatedly byte by byte or word by word, and the result is shown by the flag.
The IX or IY is automatically incremented (+1 or +2) or decremented (-1 or -2) each time 1 byte or word is processed to process the next byte or word. 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 CMPBK is used. If the CMPBKB or CMPBKW is used, it is specified directly to the byte or word type.
The destination block must always be located within the segment specified by the DS1 register; segment override is disabled. On the other hand, default segment register for the source block is DS0, and segment override is possible. The source block may be located in the segment specified by any (optional) segment register.

7) Flag operation

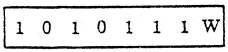
V	S	Z	AC	P	CY
x	x	x	x	x	x

12.3.3 CMPM/CMPMB/CMPMW (Compare Multiple/Compare Multiple Byte/Compare Multiple Word)

1) Description format

(repeat) CMPM [DS1-spec:]dst-block
(repeat) CMPMB (no operand)
(repeat) CMPMW (no operand)

2) Instruction format



3) Number of bytes

1

4) Number of clocks

Repeat: When W=0, 7+10/rep
When W=1, 7+14/rep: μPD70108
μPD70116 odd addresses
7+10/rep: μPD70116 even addresses

Single operation:
When W=0, 7
When W=1, 11 : μPD70108
: μPD70116 odd addresses
7 : μPD70116 even addresses

5) Number of transfers of 16-bit words

Repeat : 1/rep
Single operation: 1

6) Function

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

Subtracts the block addressed by the IY from the accumulator (AL or AW) repeatedly byte by byte or word by word, and the result is shown by the flag.
To process the next byte or word, the IY is automatically incremented (+1 or +2) or decremented (-1 or -2) each time 1 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 the CMPM is used. If the CMPMB or the CMPMW is used, it is specified directly to the byte or word type.
The destination block must always be located within the segment specified by the DS1 segment register and segment override is disabled.

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

8) Description example

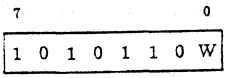
```
REPC CMPM BYTE_VAR,BYTE_VAR
REPNC CMPMW
REPZ CMPMB
```

12.3.4 LDM /LDMB/LDMW (Load Multiple/Load Multiple Byte/ Load Multiple Word)

1) Description format

(repeat) LDM [Seg-spec:]src-block
(repeat) LDMB (no operand)
(repeat) LDMW (no operand)

2) Instruction format



3) Number of bytes

1

4) Number of clocks

Repeat: When W=0, 7+9/rep
When W=1, 7+13/rep: μPD70108
μPD70116 odd addresses
7+9/rep : μPD70116 even addresses

Single operation:
When W=0, 7
When W=1, 11 : μPD70108
: μPD70116 odd addresses
7 : μPD70116 even addresses

5) Number of transfers of 16-bit words

Repeat : 1/rep
Single operation: 1

6) Function

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 to the accumulator (AL or AW) repeatedly byte by byte or word by word.
To process the next byte or word the IX is automatically incremented (+1 or +2) or decremented (-1 or -2) each time 1 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 the LDM is used; if the LDMB or LDMW is used, it is specified directly to the byte or word type.
The default segment register for the source block is DS0, and segment override is possible. The source block may be located within the segment specified by any (optional) segment register.

7) Flag operation
None

8) Description example

```
REP LDM DS1: BYTE_VAR  ; DS1 segment
REP LDMB               ; DS0 segment
```

12.3.5 STM/STMB/STMW (Store Multiple/Store Multiple Byte/ Store Multiple Word)

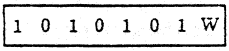
1) Description format

(repeat) STM [DS1-spec:]dst-block

(repeat) STMB (no operand)

(repeat) STMW (no operand)

2) Instruction format



3) Number of bytes

1

4) Number of clocks

Repeat: When W=0, 7+4/rep

When W=1, 7+8/rep: μPD70108

μPD70116 odd addresses

7+4/rep: μPD70116 even addresses

Single operation:

When W=0, 7

When W=1, 11 : μPD70108

μPD70116 odd addresses

7 : μPD70116 even addresses

5) Number of transfers of 16-bit words

Repeat : 1/rep

Single operation: 1

6) Function

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 AL or AW to the block addressed by IY repeatedly byte by byte or word by word.
To process the next byte or word IY is automatically incremented (+1 or +2) or decremented (-1 or -2) each time 1 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 the STM is used; if the STMB or the STMW is used, it is specified directly to the byte or word type.
The destination block must always be located within the segment specified by the DS1 segment register, and segment override is disabled.

7) Flag operation
None

8) Description example

```
REP STM  DS1:WORD_VAR   ;segment,register DS1
REP STMB                ;segment,register DS1
```

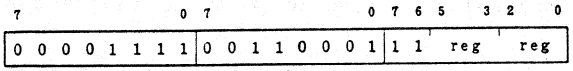
12.4 Bit Field Manipulation Instructions

12.4.1 INS (Insert Bit Field)

(1) Register

1) INS reg8,reg8

2) Instruction format



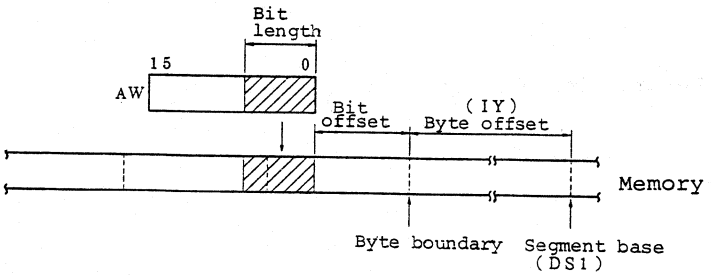
3) Number of bytes
3

4) Number of clocks

75 -103 : uPD70108
uPD70116 odd addresses
67 - 87 : uPD70116 even addresses

5) Number of transfers of 16-bit words
2 or 4

6) Function



Transfer the lower bits data 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 DS1 segment register and the IY index register) and bit offset (specified by the 8-bit register of the first operand).
After the transfer is completed, the IY register and the 8-bit register specified by the first operand are automatically updated to point to the next bit field.
For the 8-bit register of the first operand that specifies the bit offset (maximum length: 15 bits), only the lower 4 bits (0-15) will be valid. Also, for the 8-bit register of the second operand that specifies the bit length (maximum length: 16 bits), only the lower 4 bits (0-15) will be valid. 0 specifies 1-bit length, and 15 specifies 16-bit length.
Bit field data may overlap the byte boundary of memory.

7) Flag operation

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

8) Description example

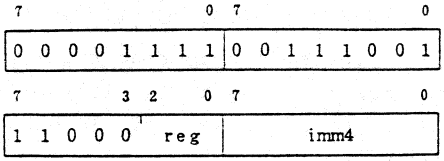
```
INS  DL, CL
```

(2) Immediate data

1) Description format

INS reg8,imm4

2) Instruction format



3) Number of bytes

4

4) Number of clocks

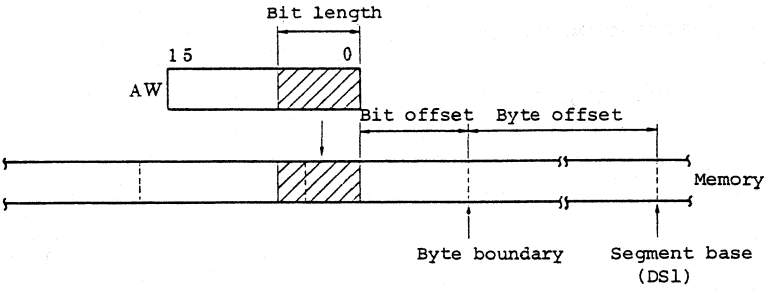
75-103 : uPD70108
uPD70116 odd addresses

67-87 : uPD70116 even addresses

5) Number of transfers of 16-bit words

2 or 4

6) Function



Transfer the lower bits data of the 16-bit AW register (bit length is specified by the 4-bit immediate data of the second operand) to the memory location determined by the byte offset (addressed by the DS1 segment register and the IY register) and bit offset (specified by the 8-bit register of the first operand).

After the transfer is completed, the IY register and the 8-bit register specified by the first operand are automatically updated to point to the next bit field.
For the 8-bit register of the first operand that specifies the bit offset (maximum length: 15 bits), only the lower 4 bits (0-15) will be valid. The immediate data value of the second operand that specifies bit length (maximum length: 16 bits) will be valid only from 0-15. 0 specifies 1-bit length, and 15 specifies 16-bit length.
Bit field data may overlap the byte boundary of memory.

7) Flag operation

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

8) Description example

```
INS  DL, 12
```

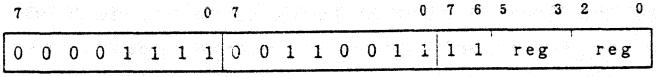
12.4.2 EXT (Extract Bit Field)

(1) Register

1) Description format

EXT reg8,reg8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

25-52 : uPD70108

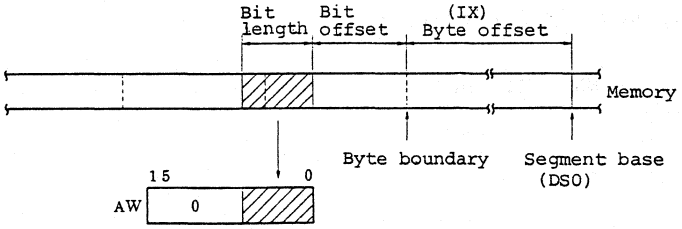
: uPD70116 odd addresses

21-44 : uPD70116 even addresses

5) Number of transfers of 16-bit words

1 or 2

6) Function



Loads to the AW register the bit field data whose bit length is specified by the 8-bit register specified as the second operand. The segment base of the memory location of the bit field is specified by the DS0 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, 0s are loaded to the remaining upper bits of the AW register.
After transfer is completed, the IX register and the 8-bit register specified by the first operand are automatically updated to point to the next bit field.
For the 8-bit register of the first operand that specifies the bit offset (maximum length: 15 bits), only the lower 4 bits (0-15) will be valid. Also for the 8-bit register of the second operand that specifies bit length (maximum length: 16 bits), only the lower 4 bits will be valid.
0 specifies 1-bit length, and 15 specifies 16-bit length. Bit field data may overlap the byte boundary of memory.

7) Flag operation

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

8) Description example

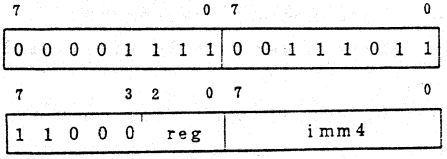
```
EXT  CL, DL
```

(2) Immediate data

1) Description format

EXT reg8,imm4

2) Instruction format



3) Number of bytes

4

4) Number of clocks

25-52 : uPD70108

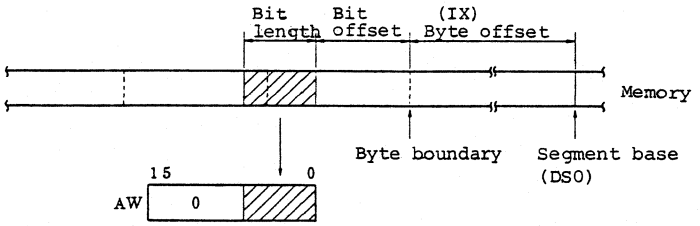
: uPD70116 odd addresses

21-44 : uPD70116 even addresses

5) Number of transfers of 16-bit words

1 or 2

6) Function



Loads to the AW register bit field data (whose bit length is specified by the 4-bit immediate data of the second operand) from the memory location specified by the byte offset (addressed by the DS0 segment register and the IX index register) and the bit offset (specified by the 8-bit register of the first operand).

After transfer is completed,the IX register and the 8-bit register specified by the first operand are automatically updated to point to the next bit field. For the 8-bit register of the first operand that specifies the bit length (maximum length: 15 bits), only the lower 4 bits (0-15) will be valid. The immediate data value of the second operand that specifies bit length (maximum length: 16 bits) will be valid only from 0-15.
0 specifies 1-bit length, and 15 specifies 16-bit length. Bit field data may overlap the byte boundary of memory.

7) Flag operation

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

8) Description example

```
EXT  CL, 8
```

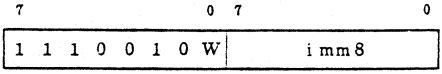
12.5 Input/Output Instruction

12.5.1 IN (Input)

(1) Directly specified I/O device

1) Description format
   IN acc,imm8

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When W=0, 9
   When W=1, 13: μPD70108
   μPD70116 odd addresses
   9: μPD70116 even addresses

5) Number of transfers of 16-bit words
   1

6) Function
   Inputs the contents of the I/O device specified by the second operand to the accumulator (AL or AW) specified by the first operand.

   When W=0 AL ← (imm8)
   When W=1 AW ← (imm8+1, imm8)

7) Flag operation
   None

8) Description example
   IN AL, 20H

(2) Indirectly specified (by DW) I/O device

1) Description format

IN acc,DW

2) Instruction format

7							0
1	1	1	0	1	1	0	W

3) Number of bytes

1

4) Number of clocks

When W=0, 8

When W=1, 12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

Inputs the contents of the I/O device specified by the DW register to the accumulator (AL or AW) specified by the first operand.

When W=0 AL ← (DW)

When W=1 AW ← (DW+1,DW)

7) Flag operation

None

8) Description example

IN AL, DW

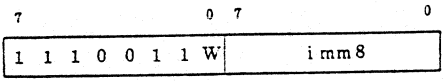
12.5.2 OUT (Output)

(1) Directly specified I/O device

1) Description format

OUT imm8,acc

2) Instruction format



3) Number of bytes

2

4) Number of clocks

When W=0, 8

When W=1, 12: μPD70108

μPD70116 odd addresses

8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

Outputs the contents of the accumulator (AL or AW) specified by the second operand to the I/O device specified by the first operand.

When W=0, (imm8) ← AL

When W=1, (imm8+1,imm8) ← AW

7) Flag operation

None

8) Description example

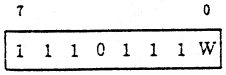
OUT 30H, AW

(2) Indirectly specified (by DW) I/O device

1) Description format

OUT DW,acc

2) Instruction format



3) Number of bytes

1

4) Number of clocks

When W=0, 8

When W=1, 12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

Outputs the contents of the accumulator (AL or AW) specified by the second operand to the I/O device specified by the first operand.

When W=0, (DW) ← AL

When W=1, (DW+1, DW) ← AW

7) Flag operation

None

8) Description example

```
OUT DW, AW
```

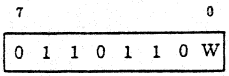
12.6 Primitive Input/Output Instruction

12.6.1 INM (Input Multiple)

1) Description format

(repeat) INM [DS1-spec:]dst-block,DW

2) Instruction format



3) Number of bytes

1

4) Number of clocks

Repeat: When W=0, 9+8/rep
When W=1, 9+16/rep: uPD70108
uPD70116 odd-odd addresses
9+12/rep: uPD70116 odd-even addresses
9+8/rep : uPD70116 even-even addresses

Single operation:
When W=0, 10
When W=1, 18 : uPD70108
uPD70116 odd-odd addresses
14 : uPD70116 odd-even addresses
10 : uPD70116 even-even addresses

5) Number of transfers of 16-bit words

Repeat : 2/rep
Single operation: 2

6) Function

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 used as a pair with a repeat prefix (REP) instruction, the REP instruction controls the number of times 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 IY index register is automatically incremented (+1 or +2) or decremented (-1 or -2) each time 1 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 INM instructions are used with the REP instruction of the repeat prefix. The destination block must always be located within the segment specified by the DS1 segment register, and the segment override is prohibited.

7) Flag operation
None

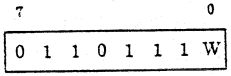
8) Description example
```
REP INM BYTE_VAR,DW
```

12.6.2 OUTM (Output Multiple)

1) Description format

   OUTM DW,[seg-spec:]src-block

2) Instruction format



3) Number of bytes

   1

4) Number of clocks

   Repeat: When W=0, 9+8/rep
   When W=1, 9+16/rep: uPD70108
   uPD70116 odd-odd addresses
   9+12/rep: uPD70116 odd-even addresses
   9+8/rep : uPD70116 even-even addresses

   Single operation:
   When W=0, 10
   When W=1, 18 : uPD70108
   uPD70116 odd-odd addresses
   14 : uPD70116 odd-even addresses
   10 : uPD70116 even-even addresses

5) Number of transfers of 16-bit words

   Repeat : 2/rep
   Single operation: 2

6) Function

   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 used as a pair with a repeat prefix (REP) instruction, the 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 1 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 OUTM instructions are used with the REP instruction of the repeat prefix. The default segment register for the source block is DS0, and segment override is possible. The source block may be located within the segment specified by any (optional) segment register.

7) Flag operation
None

8) Description example
```
REP OUTM DS1: BYTE_VAR
REP OUTMB
```

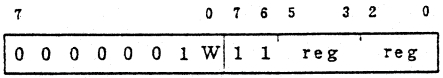
12.7 Addition/Subtraction Instructions

12.7.1 ADD (Add)

(1) Register with register to register

1) Description format
   ADD reg,reg

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   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, and stores the result in the register specified by the first operand.

   reg ← reg + reg

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

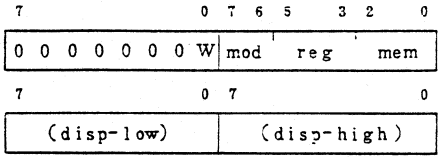
8) Description example
   ADD AW, BW

(2) Memory with register to memory

1) Description format

ADD mem, reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

2

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand.

(mem) ← (mem) + reg

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

8) Description example

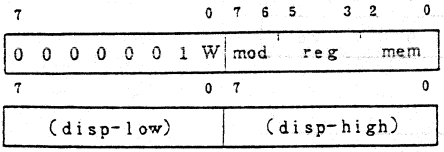
ADD WORD_VAR, AW

(3) Register with memory to register

1) Description format

ADD reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108
μPD70116 odd addresses
11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg + (mem)

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

8) Description example

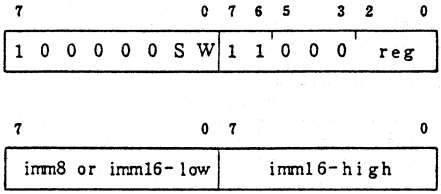
ADD AW, WORD_VAR

(4) Register with immediate data to register

1) Description format

ADD reg,imm

2) Instruction format



3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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.

reg ← reg + imm

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

8) Description example

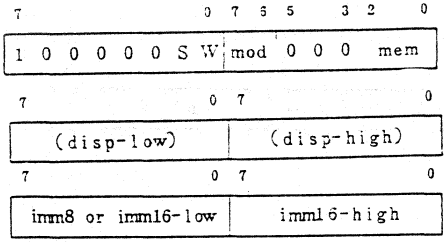
ADD AL, 10

(5) Register with immediate data to memory

1) Description format

ADD mem, imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 18

When W=1, 26: μPD70108

μPD70116 odd addresses

18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand.

(mem) ← (mem) + imm

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

8) Description example

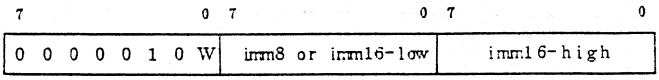
ADD BYTE_VAR[BP], 100

(6) Accumulator with immediate data to accumulator

1) Description format

ADD acc,imm

2) Instruction format



3) Number of bytes

2/3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the accumulator specified by the first operand.

When W=0 AL ← AL + imm8
When W=1 AW ← AW + imm16

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

8) Description example

ADD AL, 3

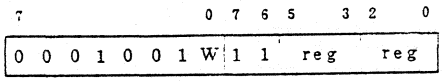
12.7.2 ADDC (Add with Carry)

(1) Register with register to register

1) Description format

ADDC reg,reg

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg + reg + CY

7) Flag operation

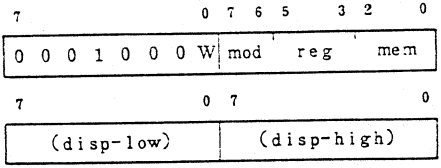
V	S	Z	AC	P	CY
x	x	x	x	x	x

(2) Memory with register to memory

1) Description format

ADDC mem,reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand.

(mem) ← (mem) + reg + CY

7) Flag operation

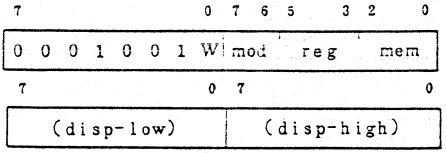
V	S	Z	AC	P	CY
x	x	x	x	x	x

(3) Register with memory to register

1) Description format

ADDC reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg + (mem) + CY

7) Flag operation

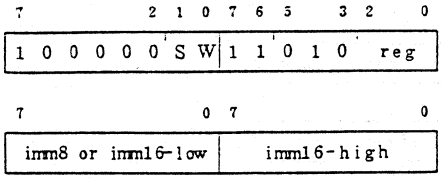
V	S	Z	AC	P	CY
x	x	x	x	x	x

(4) Register with immediate data to register

1) Description format

ADDC reg,imm

2) Instruction format



3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg + imm + CY

7) Flag operation

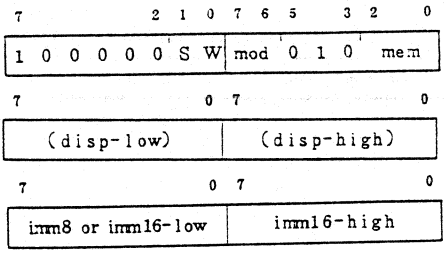
V	S	Z	AC	P	CY
x	x	x	x	x	x

(5) Memory with immediate data to memory

1) Description format

ADDC mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 18

When W=1, 26: μPD70108

μPD70116 odd addresses

18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand.

(mem) ← (mem) + imm + CY

7) Flag operation

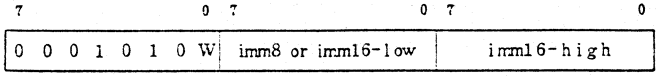
V	S	Z	AC	P	CY
x	x	x	x	x	x

(6) Accumulator with immediate data to accumulator

1) Description format

ADDC acc,imm

2) Instruction format



3) Number of bytes

2/3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the accumulator specified by the first operand.

When W=0, AL ← AL + imm8 + CY
when W=1, AW ← AW + imm16 + CY

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

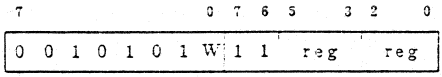
12.7.3 SUB (Subtract)

(1) Register from register to register

1) Description format

SUB reg,reg

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg - reg

7) Flag operation

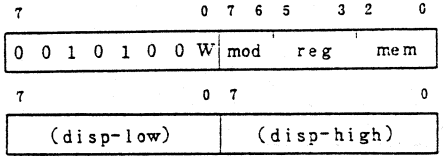
V	S	Z	AC	P	CY
x	x	x	x	x	x

(2) Register from memory to memory

1) Description format

SUB mem,reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand.

(mem) ← (mem) - reg

7) Flag operation

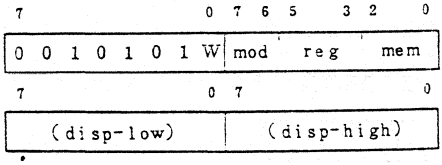
V	S	Z	AC	P	CY
x	x	x	x	x	x

(3) Memory from register to register

1) Description format

SUB reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg - (mem)

7) Flag operation

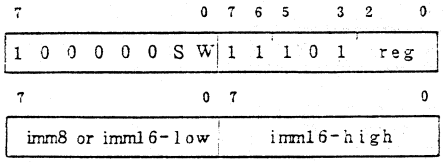
V	S	Z	AC	P	CY
x	x	x	x	x	x

(4) Immediate from register to register

1) Description format

SUB reg,imm

2) Instruction format



3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg - imm

7) Flag operation

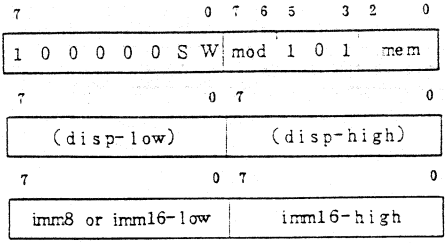
V	S	Z	AC	P	CY
x	x	x	x	x	x

(5) Immediate data from memory to memory

1) Description format

SUB mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 18

When W=1, 26: μPD70108

μPD70116 odd addresses

18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand.

(mem) ← (mem) - imm

7) Flag operation

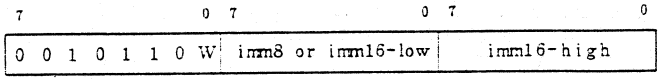
V	S	Z	AC	P	CY
x	x	x	x	x	x

(6) Immediate data from accumulator to accumulator

1) Description format

SUB acc,imm

2) Instruction format



3) Number of bytes

2/3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

Subtracts the 8- or 16-bit immediate data specified by the second operand from the accumulator (AL or AW) specified by the first operand, and stores the result in the accumulator specified by the first operand.

When W=0, AL ← AL - imm8

when W=1, AW ← AW - imm16

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

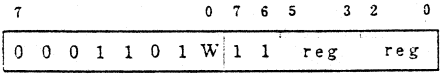
12.7.4 SUBC (Subtract with Carry)

(1) Register from register to register

1) Description format

SUBC reg,reg

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

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.

reg ← reg - reg - CY

7) Flag operation

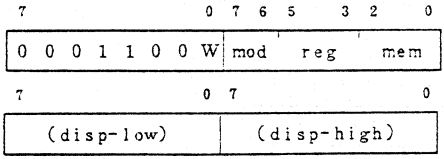
V	S	Z	AC	P	CY
x	x	x	x	x	x

(2) Register from memory to memory

1) Description format

SUBC mem,reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand.

(mem) ← (mem) - reg - CY

7) Flag operation

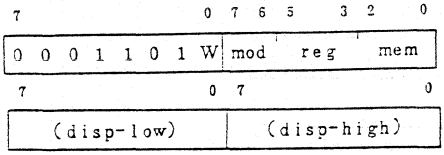
V	S	Z	AC	P	CY
x	x	x	x	x	x

(3) Memory from register to register

1) Description format

SUBC reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg - (mem) - CY

7) Flag operation

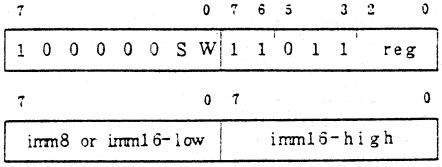
V	S	Z	AC	P	CY
x	x	x	x	x	x

(4) Immediate data from register to register

1) Description format

SUBC reg,imm

2) Instruction format



3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand.

reg ← reg - imm - CY

7) Flag operation

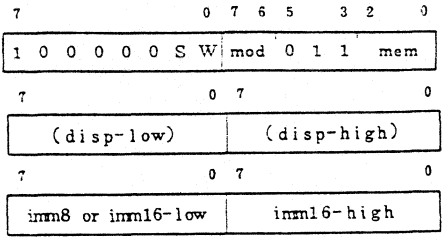
V	S	Z	AC	P	CY
x	x	x	x	x	x

(5) Immediate data from memory to memory

1) Description format

SUBC mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 18

When W=1, 26: μPD70108
μPD70116 odd addresses
18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand.

(mem) ← (mem) - imm - CY

7) Flag operation

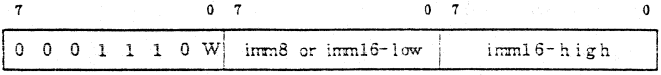
V	S	Z	AC	P	CY
x	x	x	x	x	x

(6) Immediate data from accumulator to accumulator

1) Description format

SUBC acc,imm

2) Instruction format



3) Number of bytes

2/3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the accumulator specified by the first operand.

When W=0 AL ← AL - imm8 - CY
When W=1 AW ← AW - imm16 - CY

7) Flag operation

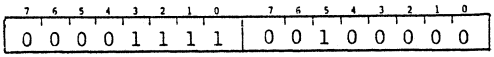
V	S	Z	AC	P	CY
x	x	x	x	x	x

12.8 BCD Operation Instructions

12.8.1 ADD4S (Add Nibble String)

1) Description format
   ADD4S [DS1-spec:]dst-string, [seg-spec:]src-string
   ADD4S (no operand)

2) Instruction format



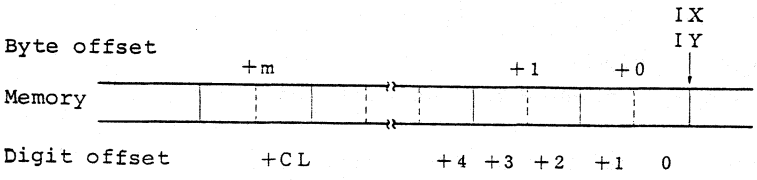
3) Number of bytes
   2

4) Number of clocks
   19 x n + 7 n: one-half the number of BCD digits

5) Number of transfers of 16-bit words
   None

6) Function
   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, and 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 (if the contents of the CL register is d, d digits), and can contain from 1 to 255 digits.
   When the number of digits is odd, the lower 4 bits of the most significant byte becomes the most significant digit of the BCD. Due to the result of this instruction, the contents of the upper 4 bits of the most significant byte are not assured.

The destination string must always be located within the segment specified by the DS1 segment register, and the segment override is prohibited.
The default segment register for the source string is DS0, 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 on the next page.



7) Flag operation

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

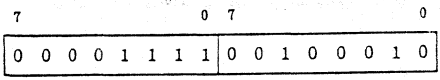
12.8.2 SUB4S (Subtract Nibble String)

1) Description format

SUB4S [DS1-spec:]dst-string,[seg-spec:]src-string

SUB4S (no operand)

2) Instruction format



3) Number of bytes

2

4) Number of clocks

19 x n + 7 n: one-half the number of BCD digits

5) Number of transfers of 16-bit words

None

6) Function

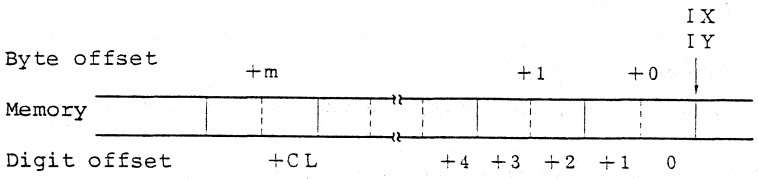
BCD string (IY,CL) ← BCD string (IY,CL) - BCD string (IX, CL)

Subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY index register, and 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 (if the contents of the CL register is d, d digits), and can contain from 1 to 255 digits.

When the number of digits is odd, the lower 4 bits of the most significant byte becomes the most significant digit of the BCD. The contents of the upper 4 bits of the most significant byte are not assured due to the result of the operation.

The destination string must always be located within the segment specified by the DS1 segment register, and the segment override is prohibited.
The default segment register for the source string is DS0, 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 on the next page.



7) Flag operation

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

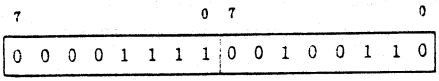
12.8.3 CMP4S (Compare Nibble String)

1) Description format

CMP4S [DS1-spec:]dst-string,[seg-spec:]src-string

CMP4S (no operand)

2) Instruction format



3) Number of bytes

2

4) Number of clocks

19 x n + 7 n: one-half the number of BCD digits

5) Number of transfers of 16-bit words

None

6) Function

BCD string (IY,CL) - BCD string (IX,CL)

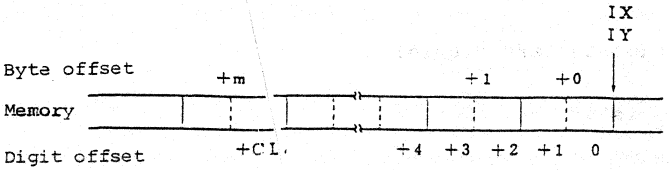
Subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY index register. The result is not stored and only the flags are affected. The length of the string (number of BCD digits) is specified by the CL register (if the contents of the CL register is d, d digits), and can contain from 1 to 255 digits.

When the number of digits is odd, the lower 4 bits of the most significant byte becomes the most significant digit of the BCD.

The destination string must always be located within the segment specified by the DS1 segment register, and the segment override is prohibited. The default segment register for the source string is DS0 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.



7) Flag operation

V	S	Z	AC	P	CY
x	U	x	U	U	x

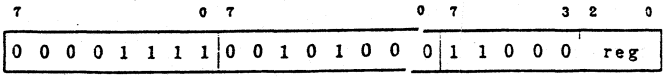
12.8.4 ROL4 (Rotate Left Nibble)

(1) 8-bit register

1) Description format

ROL4 reg8

2) Instruction format



3) Number of bytes

3

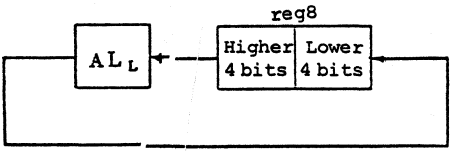
4) Number of clocks

25

5) Number of transfers of 16-bit words

None

6) Function



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.

7) Flag operation

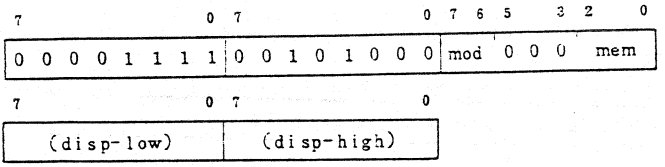
None

(2) 8-bit memory

1) Description format

ROL4 mem8

2) Instruction format



3) Number of bytes

3/4/5

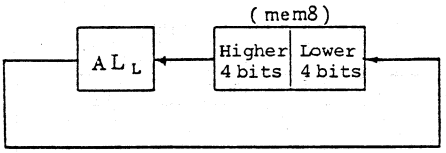
4) Number of clocks

28

5) Number of transfers of 16-bit words

None

6) Function



Treats the byte data of the 8-bit memory location addressed by the operand as 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.

7) Flag operation

None

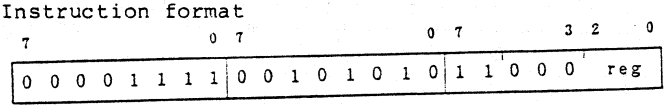
12.8.5 ROR4 (Rotate Right Nibble)

(1) 8-bit register

1) Description format

ROR4 reg8

2) Instruction format



3) Number of bytes

3

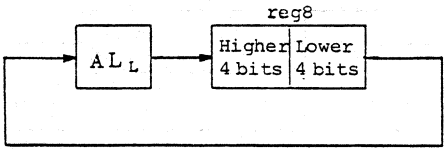
4) Number of clocks

29

5) Number of transfers of 16-bit words

None

6) Function



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 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.

7) Flag operation

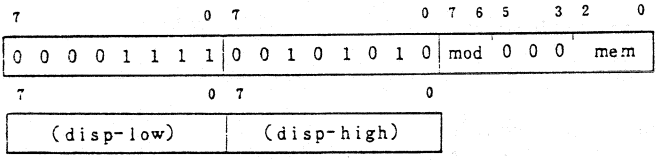
None

(2) 8-bit memory

1) Description format

ROR4 mem8

2) Instruction format



3) Number of bytes

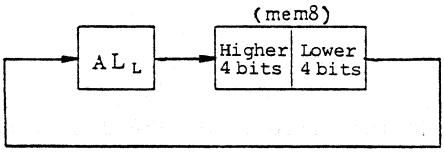
3/4/5

4) Number of clocks

33

5) Number of transfers of 16-bit words

6) Function



Treats the byte data of the 8-bit memory location addressed by the operand as 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.

7) Flag operation

None

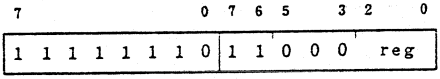
12.9 Increment/Decrement Instruction

12.9.1 INC (Increment)

(1) 8-bit register

1) Description format
   INC reg8

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   Increments (+1) the contents of the 8-bit register specified by the operand.

   reg8 ← reg8 + 1

7) Flag operation

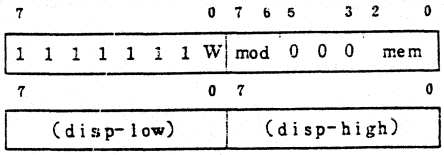
V	S	Z	AC	P	CY
x	x	x	x	x	

(2) Memory

1) Description format

INC mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Increments (+1) the contents of the 8- or 16-bit memory contents specified by the operand.

(mem) ← (mem) +1

7) Flag operation

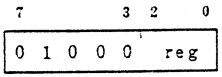
V	S	Z	AC	P	CY
x	x	x	x	x	

(3) 16-bit register

1) Description format

INC reg16

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

Increments (+1) the contents of the 16-bit register specified by the operand.

reg16 ← reg16 + 1

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	

12.9.2 DEC (Decrement)

(1) 8-bit register

1 Description format

DEC reg8

2 Instruction format

7							0	7	6	5			3	2		0
1	1	1	1	1	1	1	0	1	1	0	0	1		reg		

3 Number of bytes

2

4 Number of clocks

2

5 Number of transfers of 16-bit words

None

6 Function

Decrements (-1) the contents of the 8-bit register specified by the operand.

reg8 ← reg8 - 1

7 Flag operation

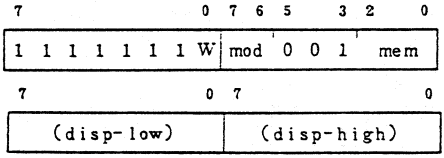
V	S	Z	AC	P	CY
x	x	x	x	x	

(2) Memory

1) Description format

DEC mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Decrements (-1) the 8- or 16-bit memory contents addressed by the operand.

(mem) ← (mem) - 1

7) Flag operation

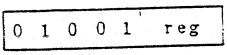
V	S	Z	AC	P	CY
x	x	x	x	x	

(3) 16-bit register

1) Description format

DEC reg16

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

Decrements (-1) the contents of the 16-bit register specified by the operand.

reg16 ← reg16 - 1

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	

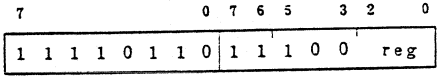
12.10 Multiplication Instructions

12.10.1 MULU (Multiply Unsigned)

(1) 8-bit register

1) Description format
   MULU reg8

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   21 or 22 (according to data)

5) Number of transfers of 16-bit words
   None

6) Function
   Performs unsigned multiplication of the contents of the AL register and the contents of the 8-bit register specified by the operand, and 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.

   AW ← AW x reg8
   When AH=0 CY ← 0, V ← 0
   When AH≠0 CY ← 1, V ← 1

7) Flag operation

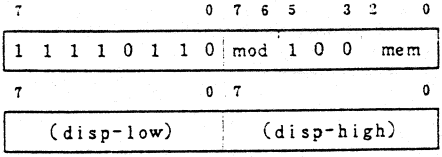
V	S	Z	AC	P	CY
x	U	U	U	U	x

(2) 8-bit memory

1) Description format

MULU mem8

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

27 or 28 (according to data)

5) Number of transfers of 16-bit words

1

6) Function

Performs unsigned multiplication of the contents of the AL register and the 8-bit memory contents addressed by the operand, and 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. The AH register is the expansion register.

AW ← AL x (mem8)
When AH=0 CY ← 0, V ← 0
When AH≠0 CY ← 1, V ← 1

7) Flag operation

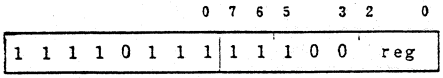
V	S	Z	AC	P	CY
x	U	U	U	U	x

(3) 16-bit register

1) Description format

MULU reg16

2) Instruction format



3) Number of bytes

2

4) Number of clocks

29 or 30 (according to data)

5) Number of transfers of 16-bit words

None

6) Function

Performs unsigned multiplication of the contents of the AW register and the contents of the 16-bit register specified by the operand, and 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. The DW register is the expansion register.

DW, AW ← AW x reg16
When DW=0 CY ← 0, V ← 0
When DW≠0 CY ← 1, V ← 1

7) Flag operation

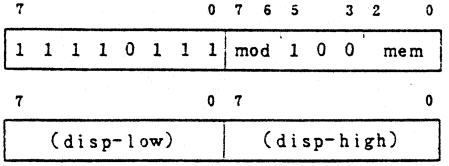
V	S	Z	AC	P	CY
x	U	U	U	U	x

(4) 16-bit memory

1) Description format

MULU mem16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

39 or 40: μPD70108
μPD70116 odd addresses
35 or 36: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

Performs unsigned multiplication of the contents of the AW register and the 16-bit memory contents addressed by the operand, and 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. The DW register is the expansion register.

DW, AW ← AW x (mem16)
When DW=0 CY ← 0, V ← 0
When DW≠0 CY ← 1, V ← 1

7) Flag operation

V	S	Z	AC	P	CY
x	U	U	U	U	x

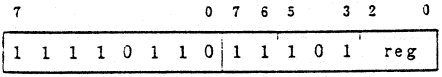
12.10.2 MUL (Multiply Signed)

(1) 8-bit register

1) Description format

   MUL reg8

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   33 to 39 (according to data)

5) Number of transfers of 16-bit words

   None

6) Function

   Performs signed multiplication of the contents of the AL register and the contents of the 8-bit register specified by the operand, and stores the double-length 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. The AH register is the expansion register.

   AW ← AL x reg8
   When AH=sign extension of AL CY ← 0, V ← 0
   When AH≠sign extension of AH CY ← 1, V ← 1

7) Flag operation

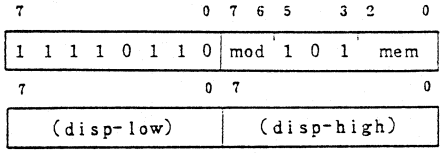
V	S	Z	AC	P	CY
x	U	U	U	U	x

(2) 8-bit memory

1) Description format

MUL mem8

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

39 to 45 (according to data)

5) Number of transfers of 16-bit words

None

6) Function

Performs signed multiplication of the contents of the AL register and the 8-bit memory contents addressed by the operand, and stores the double-length 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. The AH register is the expansion register.

AW ← AL x (mem8)
When AH=sign extension of AL CY ← 0, V ← 0
When AH≠sign extension of AH CY ← 1, V ← 1

7) Flag operation

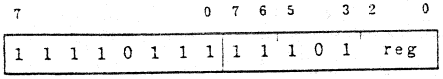
V	S	Z	AC	P	CY
x	U	U	U	U	x

(3) 16-bit register

1) Description format

MUL reg16

2) Instruction format



3) Number of bytes

2

4) Number of clocks

41 to 47 (according to data)

5) Number of transfers of 16-bit words

None

6) Function

Performs signed multiplication of the contents of the AW register and the contents of the 16-bit register specified by the operand, and 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. The DW register is the expansion register.

DW, AW ← AW x reg16
When DW=sign extension of AW CY ← 0, V ← 0
When DW≠sign extension of AH CY ← 1, V ← 1

7) Flag operation

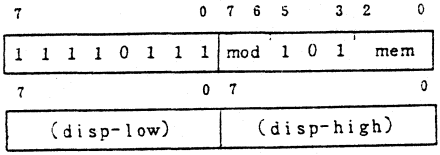
V	S	Z	AC	P	CY
x	U	U	U	U	x

(4) 16-bit memory

1) Description format

MUL mem16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

51 to 57: μPD70108
μPD70116 odd addresses
47 to 53: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

Performs signed multiplication of the contents of the AW register and the 16-bit memory contents addressed by the operand, and stores the double-word result in the AW and DW register. 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. The DW register is the expansion register.

DW, AW ← AW x (mem16)
When DW=sign extension of AW CY ← 0, V ← 0
When DW≠sign extension of AW CY ← 1, V ← 1

7) Flag operation

V	S	Z	AC	P	CY
x	U	U	U	U	x

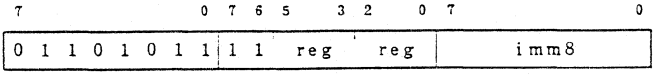
(5) 16-bit register x 8-bit immediate data to 16-bit register

1) Description format

MUL reg16,reg16,imm8

MUL reg16,imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

28 to 34 (according to data)

5) Number of transfers of 16-bit words

None

6) Function

reg16 ← reg16 x 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 (when a two-operand description, by the first operand) and the 8-bit immediate data specified by the third operand (when a two-operand description, by the second operand), and stores the result in the 16-bit register specified by the first operand.

When the source register and the destination register can be the same, a two-operand description is feasible.

7) Flag operation

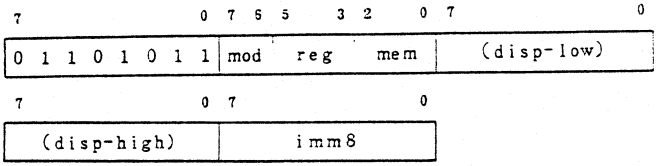
V	S	Z	AC	P	CY
x	U	U	U	U	x

(6) 16-bit memory x 8-bit immediate data to 16-bit register

1) Description format

MUL reg16,mem16,imm8

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

38 to 44: μPD70108
μPD70116 odd addresses

34 to 40: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

reg16 ← (mem16) x imm8

If the product ≤ 16 bits: CY ← 0, V ← 0

If the product > 16 bits: CY ← 1, V ← 1

Performs signed multiplication of the 16-bit memory contents addressed by the second operand and the 8-bit immediate data specified by the third operand, and stores the result in the 16-bit register specified by the first operand.

7) Flag operation

V	S	Z	AC	P	CY
x	U	U	U	U	x

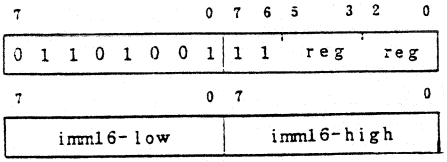
(7) 16-bit register x 16-bit immediate data to 16-bit register

1) Description format

MUL reg16,reg16,imm16

MUL reg16,imm16

2) Instruction format



3) Number of bytes

4

4) Number of clocks

36 to 42 (according to data)

5) Number of transfers of 16-bit words

None

6) Function

reg16 ← reg16 x imm16

If product $\leq$ 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 (or by the first operand, in the case of a two-operand description) and the 16-bit immediate data specified by the third (second) operand and stores the result in the 16-bit register specified by the first operand.

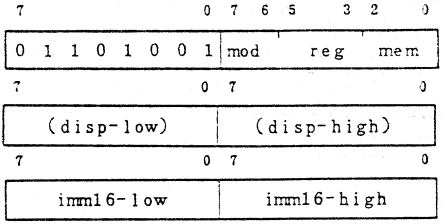
When the source register and the destination register can be the same, a two-operand description is possible.

(8) 16-bit memory x 16-bit immediate data to 16-bit register

1) Description format

MUL reg16,mem16,imm16

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

46 to 52: μPD70108
μPD70116 odd addresses
42 to 48: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

reg16 ← (mem16) x imm16
If product $\leq$ 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, and stores the result in the 16-bit register specified by the first operand.

7) Flag operation

V	S	Z	AC	P	CY
x	U	U	U	U	x

12.11 Division Instructions

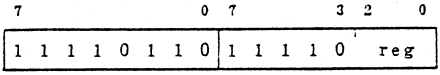
12.11.1 DIVU (Divide Unsigned)

(1) 8-bit register

1) Description format

DIVU reg8

2) Instruction format



3) Number of bytes

2

4) Number of clocks

19

5) Number of transfers of 16-bit words

None

6) Function

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 ← (003H, 002H)
  PC ← (001H, 000H)

Divides (by 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, and any remainder is stored in the AH register.
When the quotient exceeds FFH, that is the capacity of the AL destination register, the vector 0 interrupt is generated. When this occurs, the quotient and remainder become undefined. This particularly occurs when divided by 0. The fractional quotient is rounded off.

7) Flag operation

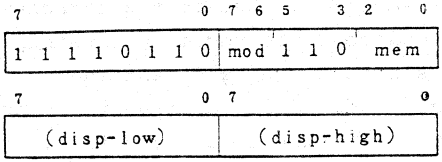
V	S	Z	AC	P	CY
U	U	U	U	U	U

(2) 8-bit memory

1) Description format

DIVU mem8

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

25

5) Number of transfers of 16-bit words

None

6) Function

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 (with 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, that is the capacity of the AL destination register, the vector 0 interrupt is generated. When this occurs, the quotient and remainder become undefined. This particularly occurs when divided by 0. The fractional quotient is rounded off.

7) Flag operation

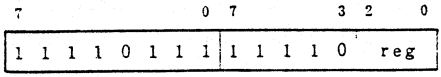
V	S	Z	AC	P	CY
U	U	U	U	U	U

(3) 16-bit register

1) Description format

DIVU reg16

2) Instruction format



3) Number of bytes

2

4) Number of clocks

25

5) Number of transfers of 16-bit words

None

6) Function

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 ← (003H, 002H)
    PC ← (001H, 000H)

Divides (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, and the remainder, if any, is

stored in the DW register. When the quotient exceeds FFFFH, that is the capacity of the AW destination register, the vector 0 interrupt is generated, and the quotient and remainder become undefined. This occurs particularly when divided by 0. A fractional quotient is rounded off.

7) Flag operation

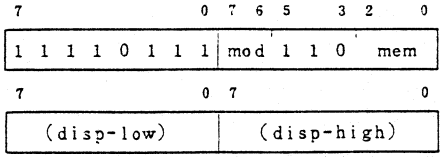
V	S	Z	AC	P	CY
U	U	U	U	U	U

(4) 16-bit memory

1) Description format

DIVU mem16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

35: μPD70108
μPD70116 odd addresses
31: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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 ← (003H, 002H)
PC ← (001H, 000H)

Divides (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, while the remainder, if any, is stored in the DW register.
When the quotient exceeds FFFFH, that is the capacity of the AW destination register, the vector 0 interrupt is generated and the quotient and remainder become undefined. This occurs particularly when divided by 0.
A fractional quotient is rounded off.

7) Flag operation

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

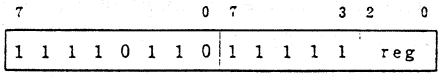
12.11.2 DIV (Divide Signed)

(1) 8-bit register

1) Description format

   DIV reg8

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   29 to 34 (according to data)

5) Number of transfers of 16-bit words

   None

6) Function

```
temp ← AW
When temp ÷ reg8 > 0 and temp ÷ reg8 ≤ 7FH
     or
     temp ÷ reg8 < 0 and temp ÷ reg8 > 0-7FH-1
        AH ← temp % reg8
        AL ← temp ÷ reg8
When temp ÷ reg8 > 0 and temp ÷ reg8 > 7FH
     or
     temp ÷ reg8 < 0 and temp ÷ reg8 ≤ 0-7FH-1
        quotient and remainder are undefined
        (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 (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, while the remainder, if any, is stored in the AH register. The maximum value of a positive quotient is +127 (7FH), while the minimum value of a negative quotient is -127 (81H). When a quotient is greater than the maximum value or less than the minimum value, the quotient and remainder become undefined, and the vector 0 interrupt is generated.
This occurs particularly when divided by 0. A. fractional quotient is rounded off. The remainder will have the same sign as the dividend.

7) Flag operation

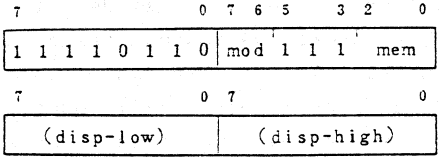
V	S	Z	AC	P	CY
U	U	U	U	U	U

(2) 8-bit memory

1) Description format

DIV mem8

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

35 to 40 (according to data)

5) Number of transfers of 16-bit words

None

6) Function

temp ← AW

When temp ÷ (mem8) > 0 and temp ÷ (mem8) ≤ 7FH

or

temp ÷ (mem8) < 0 and temp ÷ (mem8) > 0-7FH-1

AH ← temp % (mem8)

AL ← temp ÷ (mem8)

When temp ÷ (mem8) > 0 and temp ÷ (mem8) > 7FH

or

temp ÷ (mem8) < 0 and temp ÷ (mem8) ≤ 0-7FH-1

quotient and remainder are undefined

(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 (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, while the remainder, if any, is stored in the AH register. The maximum value of a positive quotient is +127 (7FH), while the minimum value of a negative quotient is -127 (81H). When a quotient is greater than the maximum value or less than the minimum value, the quotient and remainder become undefined, and the vector 0 interrupt is generated.
This occurs particularly when divided by 0. A fractional quotient is rounded off. The remainder will have the same sign as the dividend.

7) Flag operation

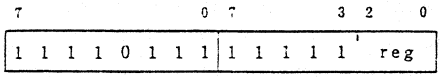
V	S	Z	AC	P	CY
U	U	U	U	U	U

(3) 16-bit register

1) Description format

DIV reg16

2) Instruction format



3) Number of bytes

2

4) Number of clocks

38 to 43 (according to the data)

5) Number of transfers of 16-bit words

1

6) Function

temp ← DW, AW
When temp ÷ reg16 > 0 and temp ÷ reg16 ≤ 7FFFH
or
temp ÷ reg16 < 0 and temp ÷ reg16 > 0-7FFFH-1
DW ← temp % reg16
AW ← temp ÷ reg16
When temp ÷ reg16 > 0 and temp ÷ reg16 > 7FFFH
or
temp ÷ reg16 < 0 and temp ÷ reg16 ≤ 0-7FFFH-1
quotient and remainder are undefined
(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 (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), while the minimum value of a negative quotient is -32,767 (8001H). When the quotient is greater than the maximum value or less than the minimum value, the quotient and remainder become undefined, and the vector 0 interrupt is generated.

This occurs particularly when divided by 0. A fractional quotient is rounded off. The remainder will have the same sign as the dividend.

7) Flag operation

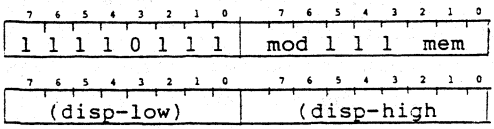
V	S	Z	AC	P	CY
U	U	U	U	U	U

(4) 16-bit memory

1) Description format

DIV mem16

2) Instruction format



3) Number of bytes

2 / 3 / 4

4) Number of clocks

48 to 53: μPD70108
μPD70116 odd addresses

44 to 49: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

temp ← DW, AW
When temp ÷ (mem16) > 0 and temp ÷ (mem16) ≤ 7FFFH
or
temp ÷ (mem16) < 0 and temp ÷ (mem16) > 0-7FFFH-1
DW ← temp % (mem16)
AW ← temp ÷ (mem16)
When temp ÷ (mem16) > 0 and temp ÷ (mem16) > 7FFFH
or
temp ÷ (mem16) < 0 and temp ÷ (mem16) ≤ 0-7FFFH-1
quotient and remainder are undefined
(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 (signed division) the contents of the DW and the 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), while the minimum value of a negative quotient is -32,767 (8001H). When quotient is greater than the maximum value or less than the minimum value, the quotient and remainder become undefined, and the vector 0 interrupt is generated.

This occurs particularly when divided by 0. A fractional quotient is rounded off. The remainder will have the same sign as the dividend.

7) Flag operation

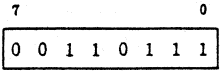
V	S	Z	AC	P	CY
U	U	U	U	U	U

12.12 BCD Adjust Instructions

12.12.1 ADJBA (Adjust Byte Add)

1) Description format
   ADJBA (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   3

5) Number of transfers of 16-bit words
   None

6) Function
   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 ∧ 0FH >9 or AC=1
   AL ← AL + 6
   AH ← AH + 1
   AC ← 1
   CY ← AC
   AL ← AL ∧ 0FH

7) Flag operation

V	S	Z	AC	P	CY
U	U	U	x	U	x

8) Description example
   ADJBA

12.12.2 ADJ4A (Adjust Nibble Add)

1) Description format
   ADJ4A (no operand)

2) Instruction format

7							0
0	0	1	0	0	1	1	1

3) Number of bytes
   1

4) Number of clocks
   3

5) Number of transfers of 16-bit words
   None

6) Function
   Adjusts the result of packed decimal addition stored in the AL register into a single packed decimal number.

```
When AL ∧ 0FH >9 or AC=1
     AL ← AL + 6
     CY ← CY V AC
     AC ← 1
When AL >9FH  or CY=1
     AL ← AL + 60H
     CY ← 1
```

7) Flag operation

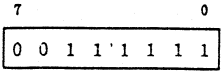
V	S	Z	AC	P	CY
x	x	x	x	x	x

8) Description example
   ADJ4A

12.12.3 ADJBS (Adjust Byte Subtract)

1) Description format
   ADJBS (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   7

5) Number of transfers of 16-bit words
   None

6) Function
   Adjusts the result of unpacked decimal subtraction stored in the AL register into a single unpacked decimal number. The higher 4 bits become zero.

   When AL ∧ 0FH >9 or AC=1
   AL ← AL - 6
   AH ← AH - 1
   AC ← 1
   CY ← AC
   AL ← AL ∧ 0FH

7) Flag operation

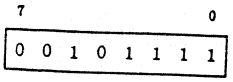
V	S	Z	AC	P	CY
U	U	U	x	U	x

8) Description example
   ADJBS

12.12.4 ADJ4S (Adjust Nibble Subtract)

1) Description format
   ADJ4S (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   7

5) Number of transfers of 16-bit words
   None

6) Function
   Adjusts the result of packed decimal subtraction stored in the AL register into a single packed decimal number.

   When AL ∧ 0FH >9 or AC=1
   AL ← AL - 6
   CY ← AC V CY
   AC ← 1
   When AL >9FH or CY=1
   AL ← AL - 60H
   CY ← 1

7) Flag operation

V	S	Z	AC	P	CY
U	x	x	x	x	x

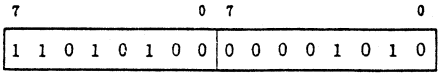
8) Description example
   ADJ4S

12.13 Data Conversion Instructions

12.13.1 CVTBD (Convert Binary to Decimal)

1) Description format
   CVTBD (no operand)

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   15

5) Number of transfers of 16-bit words
   None

6) Function
   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, if any, is then stored in the AL register.

7) Flag operation

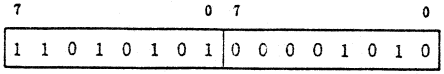
V	S	Z	AC	P	CY
U	x	x	U	x	U

8) Description example
   CVTBD

12.13.2 CVTDB (Convert Decimal to Binary)

1) Description format
   CVTDB (no operand)

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   7

5) Number of transfers of 16-bit words
   None

6) Function
   AL ← AH x 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.

7) Flag operation

V	S	Z	AC	P	CY
U	x	x	U	x	U

8) Description example
   CVTDB

12.13.3 CVTBW (Convert Byte to Word)

1) Description format
   CVTBW (no operand)

2) Instruction format

7							0
1	0	0	1	1	0	0	0

3) Number of bytes
   1

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function

   Expands the sign of the byte in the AL register to the AH register. This is effective to derive a double-length (word) dividend from a byte before byte division is performed.

   when AL < 80H AH ← 0
   when AL $\geq$ 80H AH ← FFH

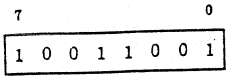
7) Flag operation
   None

8) Description example
   CVTBW

12.13.4 CVTWL (Convert Word to Long Word)

1) Description format
   CVTWL (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   4 or 5 (according to data)

5) Number of transfers of 16-bit words
   None

6) Function

   Expands the sign of the word in the AW register to the DW register. This is effective to derive a double-length (double word) dividend from a word before word division is performed.

   when AW < 8000H DW ← 0
   when AW $\geq$ 8000H DW ← FFFFH

7) Flag operation
   None

8) Description example
   CVTWL

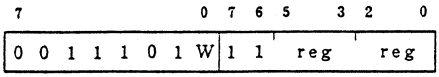
12.14 Comparison Instructions

12.14.1 CMP (Compare)

(1) Register and register

1) Description format
CMP reg,reg

2) Instruction format



3) Number of bytes
2

4) Number of clocks
2

5) Number of transfers of 16-bit words
None

6) Function
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, only the flags are affected.

reg - reg

7) Flag operation

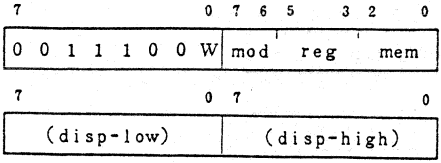
V	S	Z	AC	P	CY
x	x	x	x	x	x

(2) Memory and register

1) Description format

CMP mem,reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, only the flags are affected.

(mem) - reg

7) Flag operation

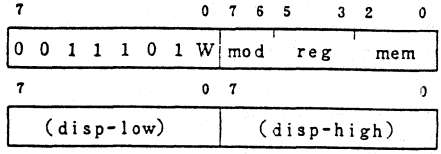
V	S	Z	AC	P	CY
x	x	x	x	x	x

(3) Register and memory

1) Description format

CMP reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108
μPD70116 odd addresses
11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, only the flags are affected.

reg - (mem)

7) Flag operation

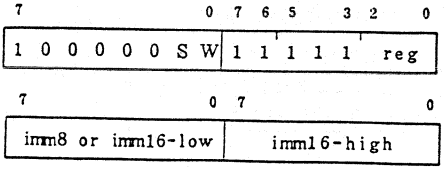
V	S	Z	AC	P	CY
x	x	x	x	x	x

(4) Register and immediate data

1) Description format

CMP reg,imm

2) Instruction format



3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, only the flags are affected.

reg - imm

7) Flag operation

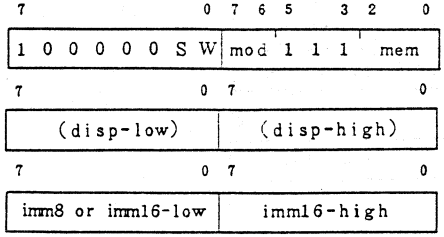
V	S	Z	AC	P	CY
x	x	x	x	x	x

(5) Memory and immediate data

1) Description format

CMP mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 13

When W=1, 17: μPD70108
μPD70116 odd addresses
13: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

Subtracts the 8- or 16-bit immediate data specified by second operand from the 8- or 16-bit memory contents addressed by the first operand. The result is not stored, only the flags are affected.

(mem) - imm

7) Flag operation

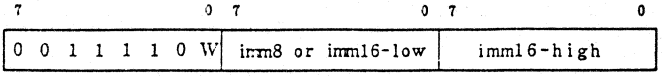
V	S	Z	AC	P	CY
x	x	x	x	x	x

(6) Accumulator and immediate data

1) Description format

CMP acc,imm

2) Instruction format



3) Number of bytes

2/3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, only the flags are affected.

When W=0 AL - imm8

When W=1 AW - imm16

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	x

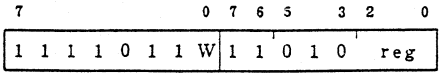
12.15 Complement Operation Instructions

12.15.1 NOT (Not)

(1) Register

1) Description format
NOT reg

2) Instruction format



3) Number of bytes
2

4) Number of clocks
2

5) Number of transfers of 16-bit words
None

6) Function
Inverts (by performing a one's complement) each bit of the 8- or 16-bit register specified by the operand and stores the result in the specified register.

reg ← $\overline{\text{reg}}$

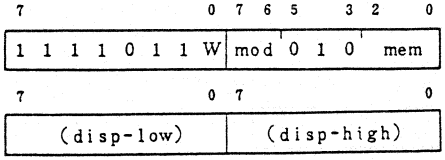
7) Flag operation
None

(2) Memory

1) Description format

NOT mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Inverts (by performing a one'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.

(mem) ← ($\overline{\text{mem}}$)

7) Flag operation

None

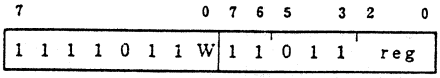
12.15.2 NEG (Negate)

(1) Register

1) Description format

NEG reg

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

reg ← $\overline{\text{reg}}$ + 1

Takes the two's complement of the contents of the 8- or 16-bit register specified by the operand.

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	1*

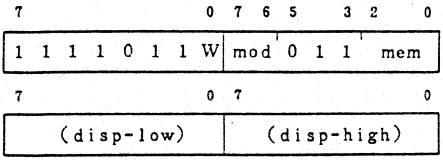
*: 0 if the contents of the register to be operated is 0.

(2) Memory

1) Description format

NEG mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

(mem) ← $\overline{\text{(mem)}}$ + 1

Takes the two's complement of the 8- or 16-bit memory contents addressed by the operand.

7) Flag operation

V	S	Z	AC	P	CY
x	x	x	x	x	1*

*: 0 if the contents of the memory to be operated is 0.

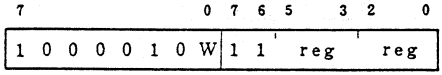
12.16 Logical Operation Instructions

12.16.1 TEST (Test)

(1) Register and register

1) Description format
   TEST reg,reg

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   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, only the flags are affected. The CY and V flags are cleared, while the AC flag becomes undefined.

   reg ∧ reg

7) Flag operation

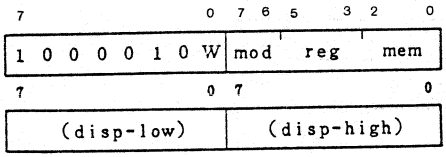
V	S	Z	AC	P	CY
0	x	x	U	x	0

(2) Register and memory

1) Description format

TEST mem,reg or TEST reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 10

When W=1, 14: μPD70108

μPD70116 odd addresses

10: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

ANDs the contents of the 8- or 16-bit register specified by the second operand and the 8- or 16-bit memory contents addressed by the first operand. The result is not stored, only the flags are affected. The CY flag and V flags are cleared, while the AC flag becomes undefined.

(mem) ∧ reg

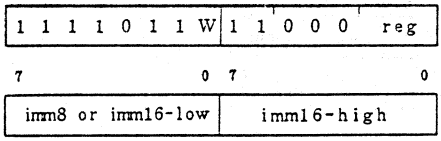
7) Flag operation

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

(3) Immediate data and register

1) Description format

TEST reg,imm



3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

ANDs the contents of the 8- or 16-bit register specified by the first operand and the 8- or 16-bit data specified by the second operand. The result is not stored, only the flags are affected. The CY flag and V flags are cleared, while the AC flag becomes undefined.

reg ∧ imm

7) Flag operation

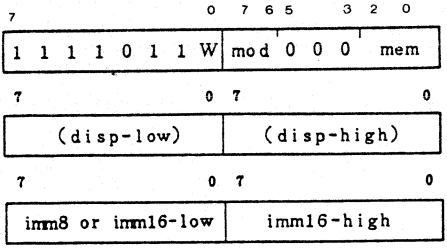
V	S	Z	AC	P	CY
0	x	x	U	x	0

(4) Immediate data and memory

1) Description format

TEST mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, only the flags are affected. The CY and V flags are cleared, while the AC flag becomes undefined.

(mem) $\wedge$ imm

7) Flag operation

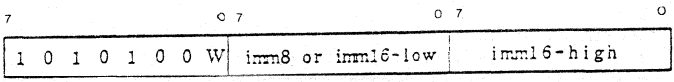
V	S	Z	AC	P	CY
0	x	x	U	x	0

8) Description example

(5) Immediate data and accumulator

1) Description format
TEST acc,imm

2) Instruction format



3) Number of bytes
2/3

4) Number of clocks

5) Number of transfers of 16-bit words
None

6) Function
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, only the flags are affected. The CY and V flags are cleared, while the AC flag becomes undefined.

When W=0 AL ∧ imm8
When W=1 AW ∧ imm16

7) Flag operation

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

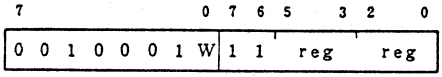
12.16.2 AND (And)

(1) Register with register to register

1) Description format

AND reg,reg

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand. The CY and V flags are cleared, while AC flag becomes undefined.

reg ← reg ∧ reg

7) Flag operation

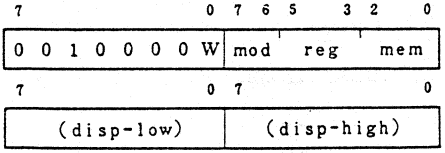
V	S	Z	AC	P	CY
0	x	x	U	x	0

(2) Memory with register to memory

1) Description format

AND mem,reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108
μPD70116 odd addresses
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

(mem) ← (mem) ∧ reg

7) Flag operation

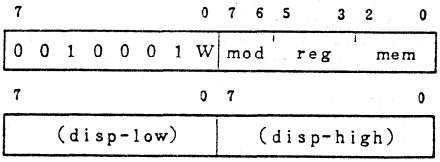
V	S	Z	AC	P	CY
0	x	x	U	x	0

(3) Register with memory to register

1) Description format

AND reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, and stores the result in the register specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

reg ← reg ∧ (mem)

7) Flag operation

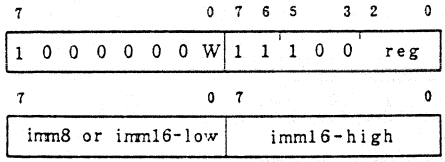
V	S	Z	AC	P	CY
0	x	x	U	x	0

(4) Register with immediate data to register

1) Description format

AND reg,imm

2) Instruction format



3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

reg ← reg ∧ imm

7) Flag operation

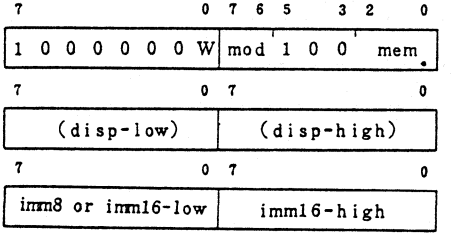
V	S	Z	AC	P	CY
0	x	x	U	x	0

(5) Memory with immediate data to memory

1) Description format

AND mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 18

When W=1, 26: μPD70108

μPD70116 odd addresses

18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

(mem) ← (mem) ∧ imm

7) Flag operation

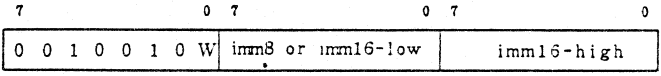
V	S	Z	AC	P	CY
0	x	x	U	x	0

(6) Accumulator with immediate data to accumulator

1) Description format

AND acc,imm

2) Instruction format



3) Number of bytes

2/3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the accumulator specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

When W=0 AL ← AL ∧ imm8
When W=1 AW ← AW ∧ imm16

7) Flag operation

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

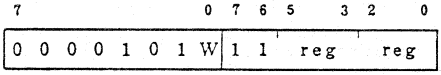
12.16.3 OR (Or)

(1) Register and register to register

1) Description format

OR reg,reg

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

reg ← reg V reg

7) Flag operation

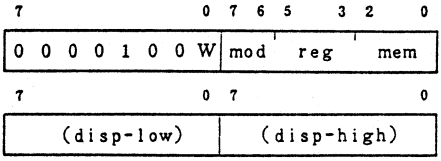
V	S	Z	AC	P	CY
0	x	x	U	x	0

(2) Memory and register to memory

1) Description format

OR mem,reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108
μPD70116 odd addresses
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

(mem) ← (mem) V reg

7) Flag operation

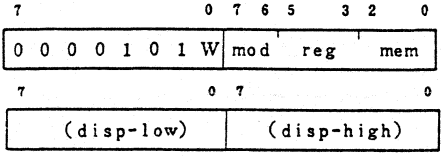
V	S	Z	AC	P	CY
0	x	x	U	x	0

(3) Register and memory to register

1) Description format

OR reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, and stores the result in the register specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

reg ← reg V (mem)

7) Flag operation

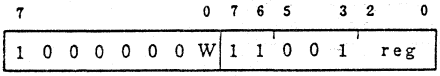
V	S	Z	AC	P	CY
0	x	x	U	x	0

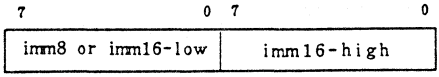
(4) Register with immediate data to register

1) Description format

OR reg,imm

2) Instruction format





3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand. The CY flag and the V flag are cleared, while the AC flag becomes undefined.

reg ← reg V imm

7) Flag operation

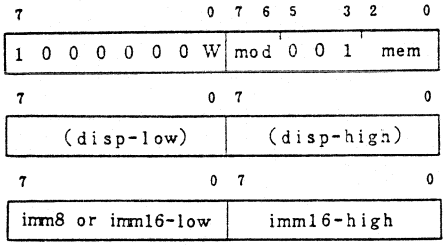
V	S	Z	AC	P	CY
0	x	x	U	x	0

(5) Memory with immediate data to memory

1) Description format

OR mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 18

When W=1, 26: μPD70108

μPD70116 odd addresses

18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

(mem) ← (mem) V imm

7) Flag operation

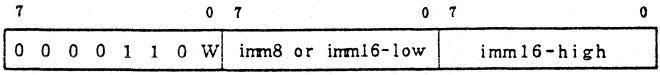
V	S	Z	AC	P	CY
0	x	x	U	x	0

(6) Accumulator with immediate data to accumulator

1) Description format

OR acc,imm

2) Instruction format



3) Number of bytes

2/3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the accumulator specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

When W=0 AL ← AL V imm8

When W=1 AW ← AW V imm16

7) Flag operation

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

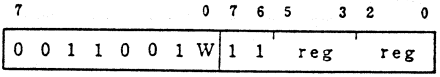
12.16.4 XOR (Exclusive Or)

(1) Register and register to register

1) Description format

XOR reg,reg

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

reg ← reg $\forall$ reg

7) Flag operation

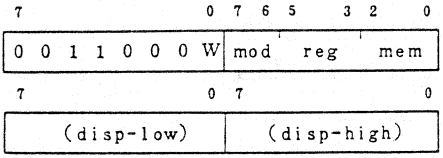
V	S	Z	AC	P	CY
0	x	x	U	x	0

(2) Memory and register to memory

1) Description format

XOR mem,reg

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

(mem) ← (mem) $\forall$ reg

7) Flag operation

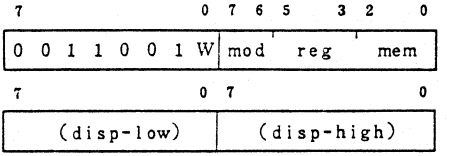
V	S	Z	AC	P	CY
0	x	x	U	x	0

(3) Register and memory to register

1) Description format

XOR reg,mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 11

When W=1, 15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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, and stores the result in the register specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

reg ← reg ⊻ (mem)

7) Flag operation

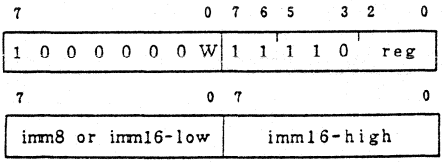
V	S	Z	AC	P	CY
0	x	x	U	x	0

(4) Register with immediate data to register

1) Description format

XOR reg,imm

2) Instruction format



3) Number of bytes

3/4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the register specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

reg ← reg $\forall$ imm

7) Flag operation

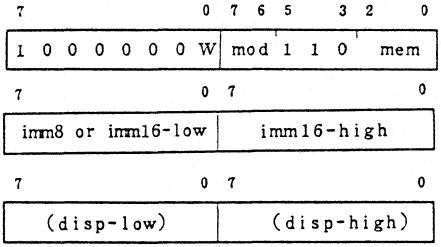
V	S	Z	AC	P	CY
0	x	x	U	x	0

(5) Memory with immedate data to memory

1) Description format

XOR mem,imm

2) Instruction format



3) Number of bytes

3/4/5/6

4) Number of clocks

When W=0, 18

When W=1, 26: μPD70108

μPD70116 odd addresses

18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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, and stores the result in the memory location addressed by the first operand. The CY flag and the V flag are cleared, while the AC flag becomes undefined.

(mem) ← (mem) $\forall$ imm

7) Flag operation

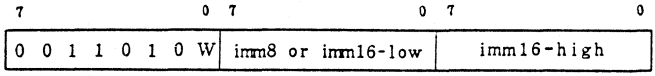
V	S	Z	AC	P	CY
0	x	x	U	x	0

(6) Accumulator with immediate data to accumulator

1) Description format

XOR acc,imm

2) Instruction format



3) Number of bytes

2/3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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, and stores the result in the accumulator specified by the first operand. The CY and V flags are cleared, while the AC flag becomes undefined.

When W=0 AL ← AL $\forall$ imm8

When W=1 AW ← AW $\forall$ imm16

7) Flag operation

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

12.17 Bit Manipulation Instruction

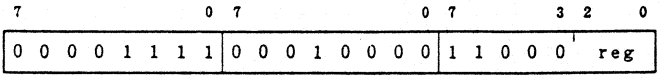
12.17.1 TEST1 (Test Bit)

(1) bit CL of the 8-bit register

1) Description format

TEST1 reg8,CL

2) Instruction format



3) Number of bytes

3

4) Number of clocks

3

5) Number of transfers of 16-bit words

None

6) Function

When bit CL of reg8 = 0, Z ← 1
When bit CL of reg8 = 1, Z ← 0

Sets the Z flag (to 1) when the bit specified by the CL register of the 8-bit register specified by the first operand is 0 and resets the Z flag (to 0) when the bit is 1. The CL value is valid for the lower 3 bits (0 to 7) only.

7) Flag operation

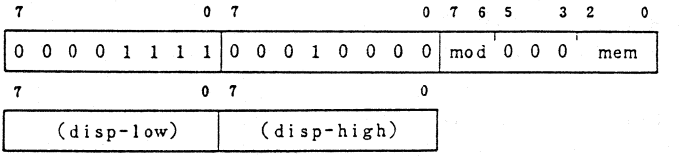
V	S	Z	AC	P	CY
0	U	x	U	U	0

(2) bit CL of the 8-bit memory

1) Description format

TEST1 mem8,CL

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

12

5) Number of transfers of 16-bit words

None

6) Function

When bit CL of (mem8) = 0, Z ← 1

When bit CL of (mem8) = 1, Z ← 0

Sets the Z flag (to 1) when the bit specified by the CL register of the 8-bit memory addressed by the first operand is 0 and resets the Z flag (to 0) when the bit is 1. The CL value is valid for the lower 3 bits (0 to 7) only.

7) Flag operation

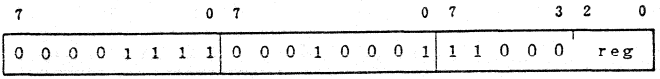
V	S	Z	AC	P	CY
0	U	x	U	U	0

(3) bit CL of the 16-bit register

1) Description format

TEST1 reg16,CL

2) Instruction format



3) Number of bytes

3

4) Number of clocks

3

5) Number of transfers of 16-bit words

None

6) Function

When bit CL of reg16 = 0, Z ← 1

When bit CL of reg16 = 1, Z ← 0

Sets the Z flag (to 1) when the bit specified by the CL register of the 16-bit register specified by the first operand is 0 and resets the Z flag (to 0) when the bit is 1. The CL value is valid for the lower 4 bits (0 to 15) only.

7) Flag operation

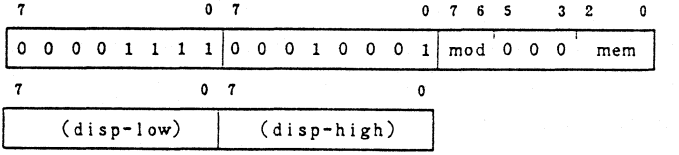
V	S	Z	AC	P	CY
0	U	x	U	U	0

(4) bit CL of the 16-bit memory

1) Description format

TEST1 mem16,CL

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

16: μPD70108

μPD70116 odd addresses

12: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

When bit CL of (mem16) = 0, Z ← 1

When bit CL of (mem16) = 0, 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. CL is valid only for the lower 4 bits (0 to 15).

7) Flag operation

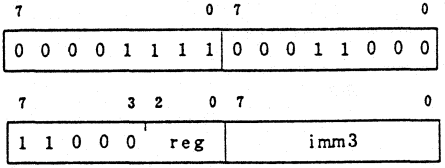
V	S	Z	AC	P	CY
0	U	x	U	U	0

(5) bit imm3 of the 8-bit register

1) Description format

TEST1 reg8,imm3

2) Instruction format



3) Number of bytes

4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

When bit imm3 of reg8 = 0, Z ← 1
When bit imm3 of reg8 = 1, Z ← 0

Sets the Z flag (to 1) when the bit specified by the imm3 bit of the 8-bit register specified by the first operand is 0 and resets the Z flag (to 0) when the bit is 1. Only the lower 3 bits of the immediated data are valid in the 4th byte of the instruction.

7) Flag operation

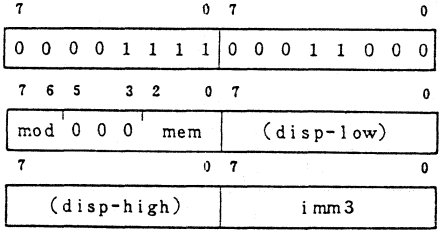
V	S	Z	AC	P	CY
0	U	x	U	U	0

(6) bit imm3 of the 8-bit memory

1) Description format

TEST1 mem8,imm3

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

13

5) Number of transfers of 16-bit words

None

6) Function

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. The immediate data at the last byte of the instruction is valid for the lower 3 bits only.

7) Flag operation

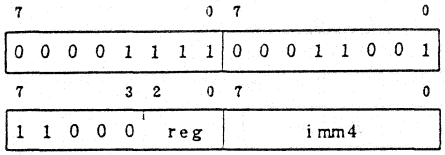
V	S	Z	AC	P	CY
0	U	x	U	U	0

(7) bit imm4 of the 16-bit register

1) Description format

TEST1 reg16,imm4

2) Instruction format



3) Number of bytes

4

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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. The immediate data at 4th byte of the instruction is valid for the lower 4 bits only.

7) Flag operation

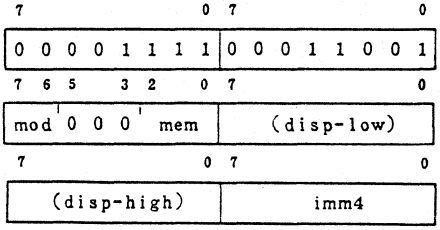
V	S	Z	AC	P	CY
0	U	x	U	U	0

(8) bit imm4 of the 16-bit memory

1) Description format

TEST1 mem16,imm4

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

17: μPD70108
μPD70116 odd addresses

13: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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 for the lower 4 bits only.

7) Flag operation

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

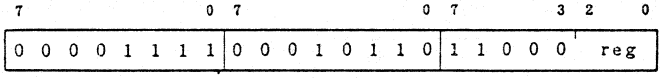
12.17.2 NOT1 (Not Bit)

(1) bit CL of the 8-bit register

1) Description format

NOT1 reg8,CL

2) Instruction format



3) Number of bytes

3

4) Number of clocks

4

5) Number of transfers of 16-bit words

None

6) Function

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 valid.

7) Flag operation

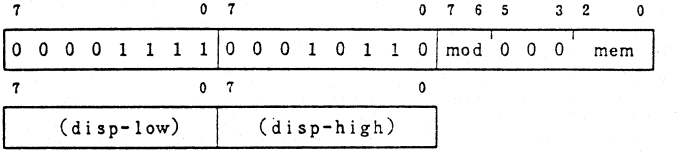
None

(2) bit CL of the 8-bit memory

1) Description format

NOT1 mem8,CL

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

18

5) Number of transfers of 16-bit words

None

6) Function

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 valid.

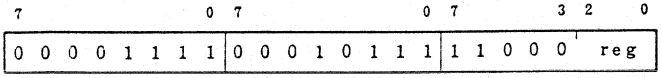
7) Flag operation

None

(3) bit CL of the 16-bit register

1) Description format
   NOT1 reg16, CL

2) Instruction format



3) Number of bytes
   3

4) Number of clocks
   4

5) Number of transfers of 16-bit words
   None

6) Function
   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 valid.

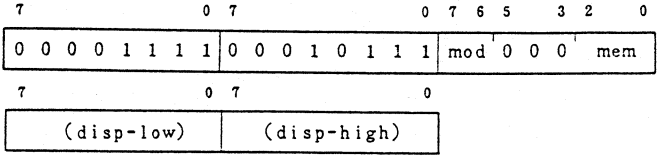
7) Flag operation
   None

(4) bit CL of the 16-bit memory

1) Description format

NOT1 mem16,CL

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

26: μPD70108
μPD70116 odd addresses
18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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 valid.

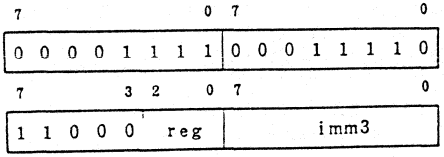
7) Flag operation

None

(5) bit imm3 of the 8-bit register

1) Description format
   NOT1 reg8,imm3

2) Instruction format



3) Number of bytes
   4

4) Number of clocks
   5

5) Number of transfers of 16-bit words
   None

6) Function
   bit imm3 of reg8 ← $\overline{\text{bit imm3 of reg8}}$

   The bit imm3 (second operand) specifies which bit of the 8-bit register specified by the first operand is to be inverted. Only the lower 3 bits of the immediate data at 4th byte of the instruction are valid.

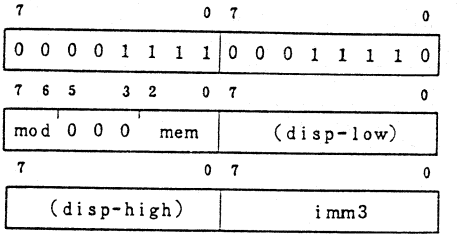
7) Flag operation
   None

(6) bit imm3 of 8-bit memory

1) Description format

NOT1 mem8,imm3

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

19

5) Number of transfers of 16-bit words

None

6) Function

bit imm3 of mem8 ← $\overline{\text{bit imm3 of mem8}}$

The 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 valid in the last byte of the instruction.

7) Flag operation

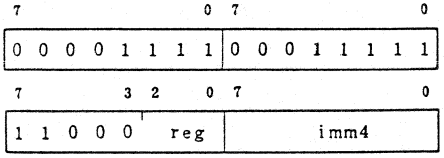
None

(7) bit imm4 of the 16-bit register

1) Description format

NOT1 reg16,imm4

2) Instruction format



3) Number of bytes

4

4) Number of clocks

5

5) Number of transfers of 16-bit words

None

6) Function

bit imm4 of reg16 ← $\overline{\text{bit imm4 of reg16}}$

The 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 valid in the 4th byte of the instruction.

7) Flag operation

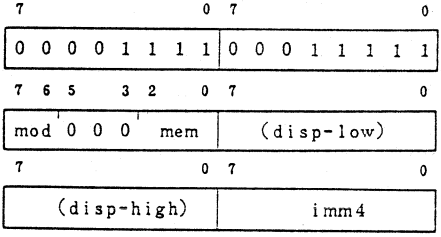
None

(8) bit imm4 of the 16-bit memory

1) Description format

NOT1 mem16,imm4

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

27: μPD70108

μPD70116 odd addresses

19: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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 valid in the last byte of the instruction.

7 Flag operation

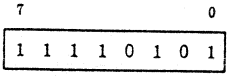
None

(9) Carry flag

1) Description format

NOTl CY

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

$CY \leftarrow \overline{CY}$

Inverts the CY flag.

7) Flag operation

V	S	Z	AC	P	CY
					X

8) Description example

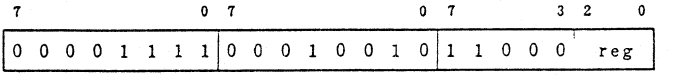
NOTl CY

12.17.3 CLR1 (Clear Bit)

(1) bit CL of the 8-bit register

1) Description format
   CLR1 reg8,CL

2) Instruction format



3) Number of bytes
   3

4) Number of clocks
   5

5) Number of transfers of 16-bit words
   None

6) Function

   bit CL of reg8 ← 0

   Clears the bit (to 0) specified by the CL register of the 8-bit register specified by the first operand. Only the lower three bits are valid for CL.

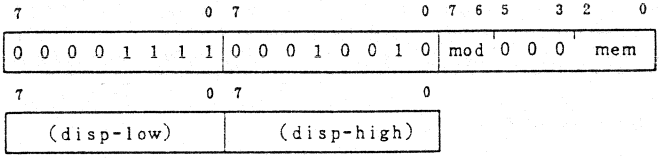
7) Flag operation
   None

(2) bit CL of the 8-bit memory

1) Description format

CLR1 mem8,CL

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

14

5) Number of transfers of 16-bit words

None

6) Function

bit CL of (mem8) ← 0

Clears the bit (to 0) specified by the CL register of the 8-bit memory location addressed by the first operand. Only the lower three bits are valid for CL.

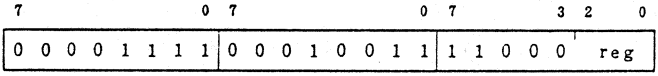
7) Flag operation

None

(3) bit CL of the 16-bit register

1) Description format
   CLR1 reg16,CL

2) Instruction format



3) Number of bytes
   3

4) Number of clocks
   5

5) Number of transfers of 16-bit words
   None

6) Function
   bit CL of reg16 ← 0

   Clears the bit (to 0) specified by the CL register of the 16-bit register specified by the first operand. Only the lower four bits are valid for CL.

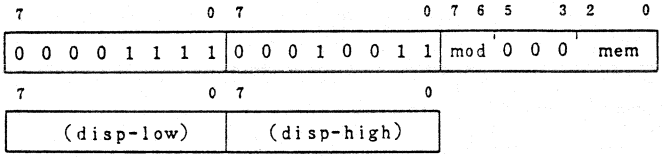
7) Flag operation
   None

(4) bit CL of the 16-bit memory

1) Description format

CLR1 mem16,CL

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

22: μPD70108
μPD70116 odd addresses
14: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

bit CL of (mem16) ← 0

Clears the bit (to 0) specified by the CL register of the 16-bit memory location addressed by the first operand. Only the lower 4 bits are valid for CL.

7) Flag operation

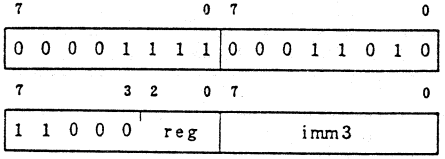
None

(5) bit imm3 of the 8-bit register

1) Description format

CLR1 reg8,imm3

2) Instruction format



3) Number of bytes

4

4) Number of clocks

6

5) Number of transfers of 16-bit words

None

6) Function

bit imm3 of reg8 ← 0

Clears (to 0) the bit specified by the 3-bit immediate data (second operand) of the 8-bit register specified by the first operand. Only the lower 3 bits of the immediate data are valid in the 4th byte of the instruction.

7) Flag operation

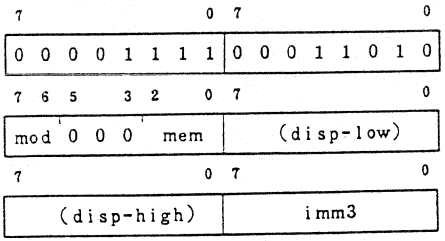
None

(6) bit imm3 of the 8-bit memory

1) Description format

CLR1 mem8,imm3

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

15

5) Number of transfers of 16-bit words

None

6) Function

bit imm3 of (mem8) ← 0

Clears (to 0) the bit specified by the 3-bit immediate data (second operand) of the 8-bit memory location addressed by the first operand. Only the lower 3 bits of the immediate data are valid in the last byte of the instruction.

7) Flag operation

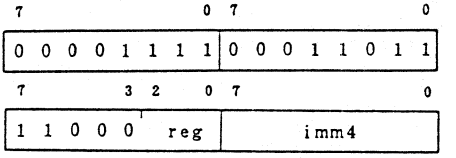
None

(7) bit imm4 of the 16-bit register

1) Description format

CLR1 reg16,imm4

2) Instruction format



3) Number of bytes

4

4) Number of clocks

6

5) Number of transfers of 16-bit words

None

6) Function

bit imm4 of reg16 ← 0

Clears (to 0) the bit specified by the 4-bit immediate data (second operand) of the 16-bit register specified by the first operand. Only the lower 4 bits of the immediate data are valid in the 4th byte of the instruction.

7) Flag operation

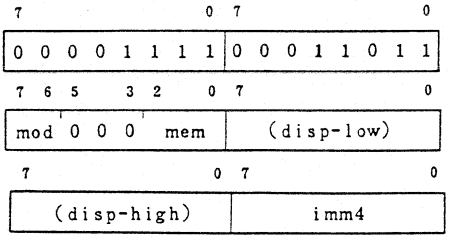
None

(8) bit imm4 of the 16-bit memory

1) Description format

CLR1 mem16,imm4

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

27: μPD70108
μPD70116 odd addresses
15: μPD70116 even addresses

5) Number of transfers of 16-bit words

3

6) Function

bit imm4 of (mem16) ← 0

Clears (to 0) the bit specified by the 4-bit immediate data (second operand) of the 16-bit memory location addressed by the first operand. Only the lower 4 bits of immediate data are valid in the last byte of the instruction.

7) Flag operation

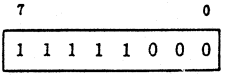
None

(9) Carry flag

1) Description format

CLR1 CY

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

CY ← 0

Clears the CY flag.

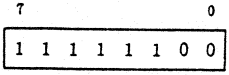
7) Flag operation

V	S	Z	AC	P	CY
					0

(10) Direction flag

1) Description format
CLR1 DIR

2) Instruction format



3) Number of bytes
1

4) Number of clocks
2

5) Number of transfers of 16-bit words
None

6) Function

DIR ← 0

Clears the DIR flag.
Sets index registers IX and IY to autoincrement at the execution of MOVBK, CMPBK, CMPM, LDM, STM, INM, and OUTM instructions.

7) Flag operation

DIR
0

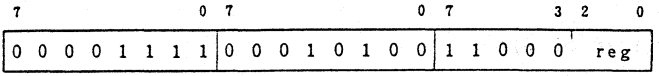
8) Description example
CLR1 DIR

12.17.4 SET1 (Set Bit)

(1) bit CL of the 8-bit register

1) Description format
SET1 reg8,CL

2) Instruction format



3) Number of bytes
3

4) Number of clocks
4

5) Number of transfers of 16-bit words
None

6) Function
bit CL of reg8 ← 1

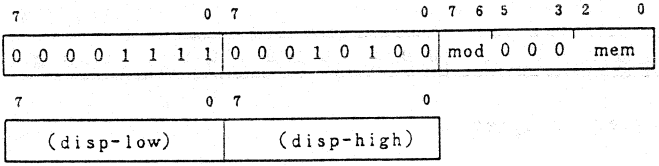
Sets(to 1)the bit specified by the CL register (second operand) of the 8-bit register specified by the first operand. Only the lower 3 bits are valid for CL.

7) Flag operation
None

(2) bit CL of the 8-bit memory

1) Description format
SET1 mem8,CL

2) Instruction format



3) Number of bytes
3/4/5

4) Number of clocks
13

5) Number of transfers of 16-bit words
None

6) Function
bit CL of (mem8) ← 1

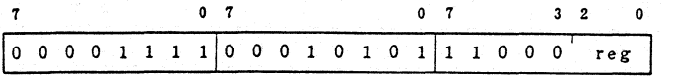
Sets(to 1)the bit specified by the CL register (second operand) of the 8-bit memory location addressed by the first operand. Only the lower 3 bits are valid for CL.

7) Flag operation
None

(3) bit CL of the 16-bit register

1) Description format
   SET1 reg16,CL

2) Instruction format



3) Number of bytes
   3

4) Number of clocks
   4

5) Number of transfers of 16-bit words
   None

6) Function
   bit CL of reg16 ← 1

   Sets(to 1) the bit specified by the CL register (second operand) of the 16-bit register specified by the first operand. Only the lower 4 bits are valid for CL.

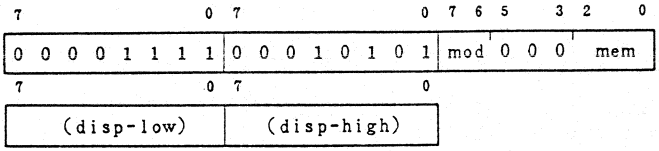
7) Flag operation
   None

(4) bit CL of 16-bit memory

1) Description format

SET1 mem16,CL

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

21: μPD70108
μPD70116 odd addresses
13: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

bit CL of (mem16) ← 1

Sets(to 1)the bit specified by the CL register (second operand) of the 16-bit memory location addressed by the first operand. Only the lower 4 bits are valid for CL.

7) Flag operation

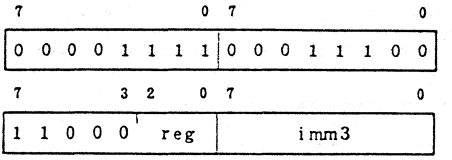
None

(5) bit imm3 of the 8-bit register

1) Description format

SET1 reg8,imm3

2) Instruction format



3) Number of bytes

4

4) Number of clocks

5

5) Number of transfers of 16-bit words

None

6) Function

bit imm3 of reg8 ← 1

Sets(to 1)the bit specified by the 3-bit immediate data (second operand) of the 8-bit register specified by the first operand. Only the lower 3 bits of the immediate data are valid in the 4th byte of the instruction.

7) Flag operation

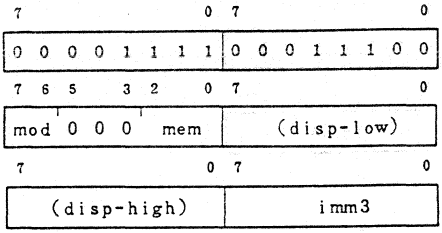
None

(6) bit imm3 of the 8-bit memory

1) Description format

SET1 mem8,imm3

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

14

5) Number of transfers of 16-bit words

None

6) Function

bit imm3 of (mem8) ← 1

Sets(to 1) the bit specified by the 3-bit immediate data (second operand) of the 8-bit memory location addressed by the first operand. Only the lower 3 bits of immediate data are valid in the last byte of the instruction.

7) Flag operation

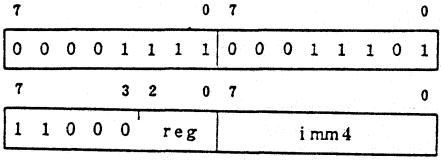
None

(7) bit imm4 of the 16-bit register

1) Description format

SET1 reg16,imm4

2) Instruction format



3) Number of bytes

4

4) Number of clocks

5

5) Number of transfers of 16-bit words

None

6) Function

bit imm4 of reg16 ← 1

Sets (to 1) the bit specified by the 4-bit immediate data (second operand) of the 16-bit register specified by the first operand. Only the lower 4 bits of the immediate data are valid in the 4th byte of the instruction.

7) Flag operation

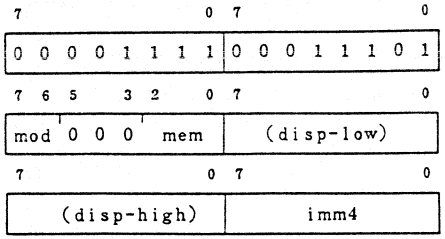
None

(8) bit imm4 of the 16-bit memory

1) Description format

SET1 mem16,imm4

2) Instruction format



3) Number of bytes

4/5/6

4) Number of clocks

22: μPD70108
μPD70116 odd addresses
14: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

bit imm4 of (mem16) ← 1

Sets(to 1)the bit specified by the 4-bit immediate data (second operand) of the 16-bit memory location addressed by the first operand. Only the lower 4 bits of immediate data are valid in the last byte of the instruction.

7) Flag operation

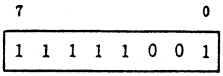
None

(9) Carry flag

1) Description format

SETl CY

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

CY ← 1

Sets the CY flag.

7) Flag operation

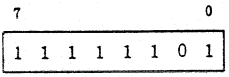
V	S	Z	AC	P	CY
					1

(10) Direction flag

1) Description format

SET1 DIR

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

DIR ← 1

Sets the direction flag.

Sets index registers IX and IY to autodecrement at the execution of the MOVBK, CMPBK, CMPM, LDM, STM, INM, and OUTM instructions.

7) Flag operation

DIR
1

8) Description example

SET1 DIR

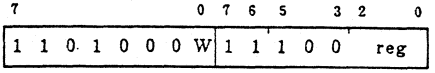
12.18 Shift Instructions

12.18.1 SHL (Shift Left)

(1) Register single-bit

1) Description format
SHL reg,1

2) Instruction format

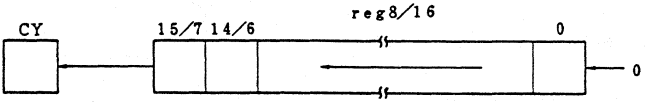


3) Number of bytes
2

4 Number of clocks
2

5) Number of transfers of 16-bit words
None

6) Function
Performs 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.



7) Flag operation

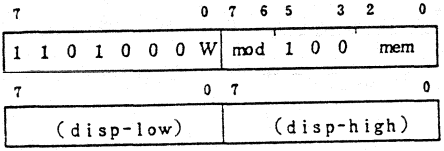
V	S	Z	AC	P	CY
x	x	x	U	x	x

(2) Memory single-bit

1) Description format

SHL mem,1

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

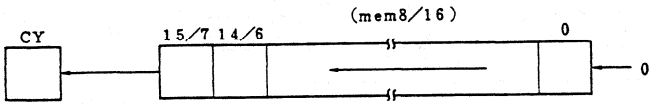
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Performs 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.



7) Flag operation

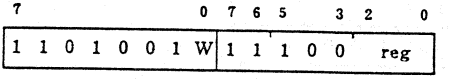
V	S	Z	AC	P	CY
x	x	x	U	x	x

(3) Register variable-bit

1) Description format

SHL reg,CL

2) Instruction format



3) Number of bytes

2

4) Number of clocks

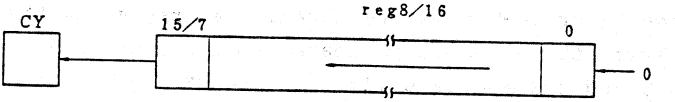
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Performs shift left (by the number of bits equal to the number of bits in the CL register) of the 8- or 16-bit register specified by the first operand. Zero is loaded to the specified register's LSB and the MSB is shifted to the CY flag.



7) Flag operation

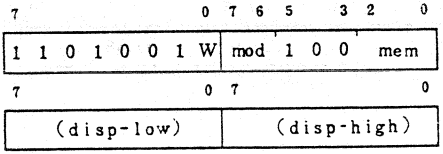
V	S	Z	AC	P	CY
U	x	x	U	x	x

(4) Memory variable-bit

1) Description format

SHL mem,CL

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: μPD70108

μPD70116 odd addresses

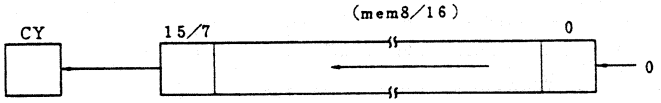
19 + n: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Performs shift left (by the) number of bits equal to the number of bits in the CL register) 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.



7) Flag operation

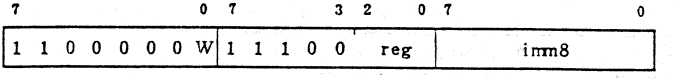
V	S	Z	AC	P	CY
U	x	x	U	x	x

(5) Register multi-bit

1) Description format

SHL reg,imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

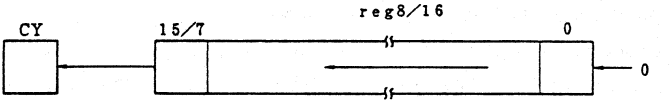
7 + n n: number of shifts

5) Number of transfers of 16-bit words

None

6) Function

Performs shift left (by the bit numbers specified by the 8-bit immediate data, second operand) of the 8- or 16-bit register specified by the first operand. Zero is loaded to the specified register's LSB and the MSB is shifted to the CY flag.



7) Flag operation

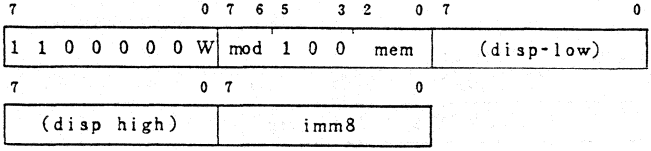
V	S	Z	AC	P	CY
U	x	x	U	x	x

(6) Memory multi-bit

1) Description format

SHL mem,imm8

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: µPD70108
µPD70116 odd addresses

19 + n: µPD70116 even addresses

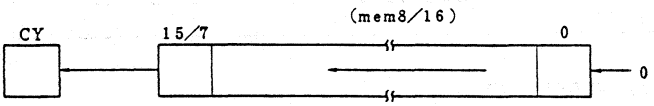
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Performs shift left (by the bit numbers specified by the 8-bit immediate data, second operand) of the 8- or 16-bit memory location addressed by the first operand. Zero is loaded to the specified memory location's LSB and the MSB is shifted to the CY flag.



7) Flag operation

V	S	Z	AC	P	CY
U	x	x	U	x	x

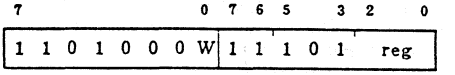
12.18.2 SHR (Shift Right)

(1) Register single-bit

1) Description format

SHR reg,1

2) Instruction format



3) Number of bytes

2

4) Number of clocks

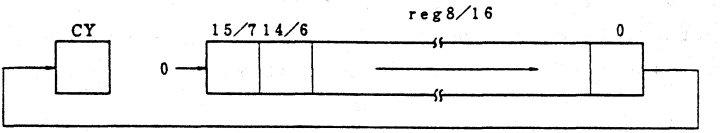
2

5) Number of transfers of 16-bit words

None

6) Function

Performs logical shift right (1 bit) of the 8- or 16-bit register specified by the first operand. Zero is loaded to the specified register'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.



7) Flag operation

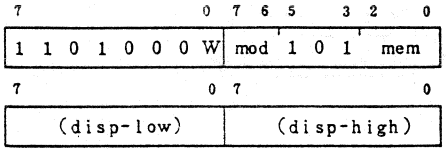
V	S	Z	AC	P	CY
x	x	x	U	x	x

(2) Memory single-bit

1) Description format

SHR mem,1

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

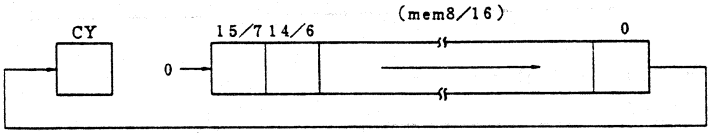
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Performs 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.



7) Flag operation

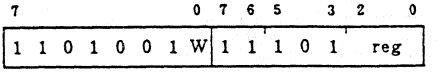
V	S	Z	AC	P	CY
x	x	x	U	x	x

(3) Register variable-bit

1) Description format

SHR reg,CL

2) Instruction format



3) Number of bytes

2

4) Number of clocks

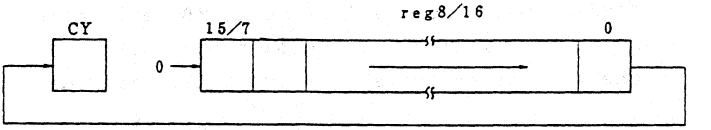
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Performs logical shift right (by the number of bits equal to the value of the CL register) of the 8- or 16-bit register specified by the first operand. Zero is loaded to the specified register's MSB and the LSB is shifted to the CY flag.



7) Flag operation

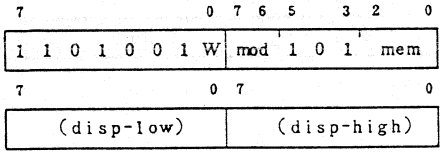
V	S	Z	AC	P	CY
U	x	x	U	x	x

(4) Memory variable-bit

1) Description format

SHR mem,CL

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: µPD70108
µPD70116 odd addresses

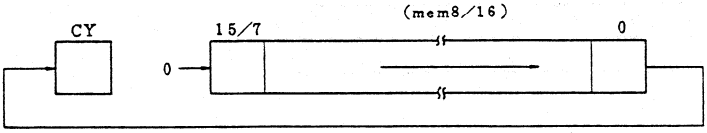
19 + n: µPD70116 even addresses
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Performs logical shift right (by the number of bits equal to the value of the CL register) 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.



7) Flag operation

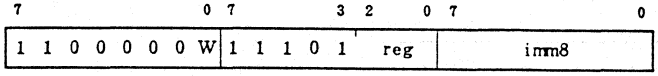
V	S	Z	AC	P	CY
U	x	x	U	x	x

(5) Register multi-bit

1) Description format

SHR reg,imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

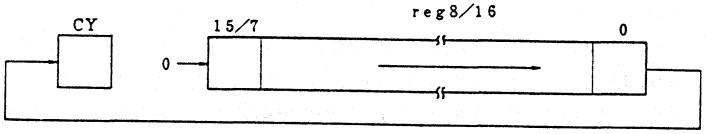
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Performs shift right (by the number of bits specified by the 8-bit immediate data of the second operand) of the 8- or 16-bit register specified by the first operand. Zero is loaded to the specified register's MSB and the LSB is shifted to the CY flag.



7) Flag operation

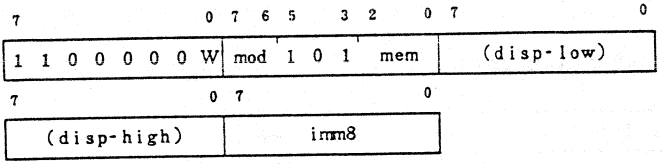
V	S	Z	AC	P	CY
U	x	x	U	x	x

(6) Memory multi-bit

1) Description format

SHR mem,imm8

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: μPD70108
μPD70116 odd addresses

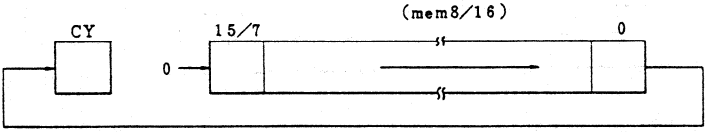
19 + n: μPD70116 even addresses
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Performs shift right (by the number of bits specified by the 8-bit immediate data of the second operand) 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.



7) Flag operation

V	S	Z	AC	P	CY
U	x	x	U	x	x

12.18.3 SHRA (Shift Right Arithmetic)

(1) Register single-bit

1) Description format

SHRA reg,1

2) Instruction format

7							0	7	6	5		3	2	0
1	1	0	1	0	0	0	W	1	1	1	1	1	reg	

3) Number of bytes

2

4) Number of clocks

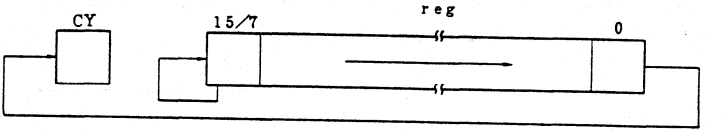
2

5) Number of transfers of 16-bit words

None

6) Function

Performs 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 and the LSB is shifted to the CY flag. The sign remains unchanged after the shift.



7) Flag operation

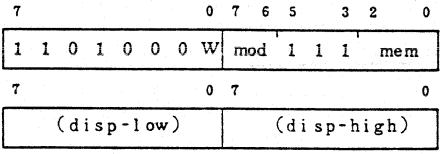
V	S	Z	AC	P	CY
0	x	x	U	x	x

(2) Memory single-bit

1) Description format

SHRA mem,1

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

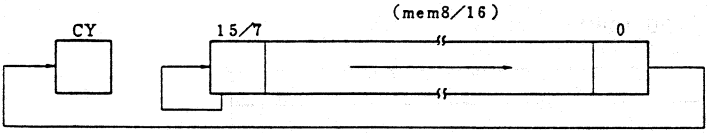
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Performs 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 and the LSB is shifted to the CY flag. The sign remains unchanged after the shift.



7) Flag operation

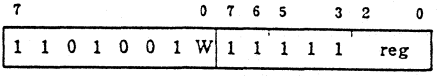
V	S	Z	AC	P	CY
0	x	x	U	x	x

(3) Register variable-bit

1) Description format

SHRA reg,CL

2) Instruction format



3) Number of bytes

2

4) Number of clocks

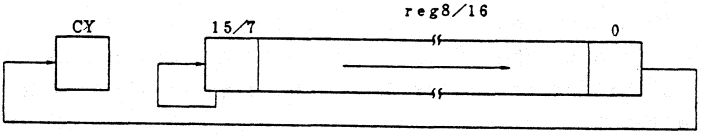
7 + n (n= number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Performs arithmetic shift right (by the number of bits equal to the number of bits in the CL register) 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 register's MSB and the LSB is shifted to the CY flag. The sign remains unchanged after the shift.



7) Flag operation

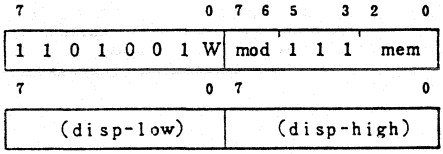
V	S	Z	AC	P	CY
U	x	x	U	x	x

(4) Memory variable-bit

1) Description format

SHRA mem,CL

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: µPD70108

µPD70116 odd addresses

19 + n: µPD70116 even addresses

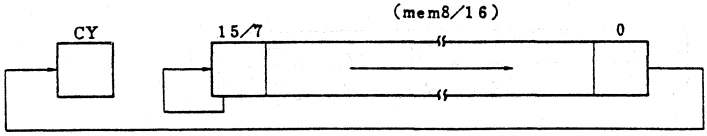
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Performs arithmetic shift right (by the number of bits equal to the number of bits in the CL register) 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 and the LSB is shifted to the CY flag. The sign remains unchanged after the shift.



7) Flag operation

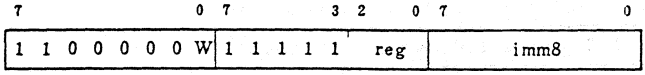
V	S	Z	AC	P	CY
U	x	x	U	x	x

(5) Register multi-bit

1) Description format

SHRA reg,imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

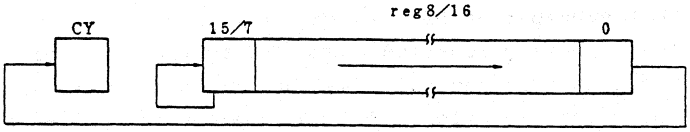
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Performs arithmetic shift right (by the number of bits specified by the 8-bit immediate data in the second operand) 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 register's MSB and the LSB is shifted to the CY flag. The sign remains unchanged after the shift.



7) Flag operation

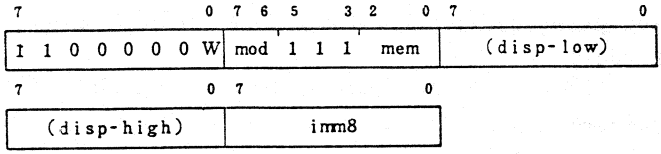
V	S	Z	AC	P	CY
U	x	x	U	x	x

(6) Memory multi-bit

1) Description format

SHRA mem,imm8

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: μPD70108
μPD70116 odd addresses

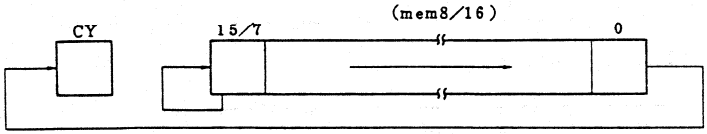
19 + n: μPD70116 even addresses
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Performs arithmetic shift right (by the number of bits specified by the 8-bit immediate data in the second operand) 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 register's MSB and the LSB is shifted to the CY flag. The sign remains unchanged after the shift.



7) Flag operation

V	S	Z	AC	P	CY
U	x	x	U	x	x

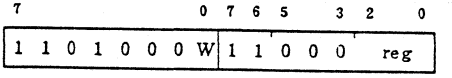
12.19 Rotate Instructions

12.19.1 ROL (Rotate Left)

(1) Register single-bit

1) Description format
   ROL reg,1

2) Instruction format

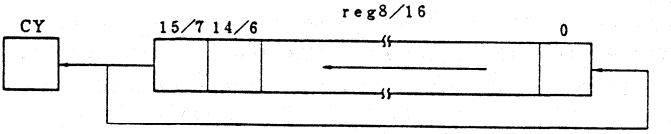


3) Number of bytes
   2

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   Rotates left by 1 bit the 8- or 16-bit register specified by the first operand. If the MSB changes, the V flag is set; if it stays the same, the V flag is cleared.



7) Flag operation

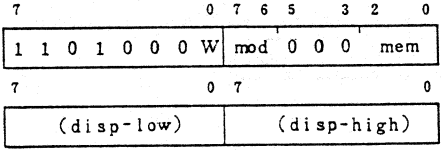
V	S	Z	AC	P	CY
x					x

(2) Memory single-bit

1) Description format

ROL mem,1

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

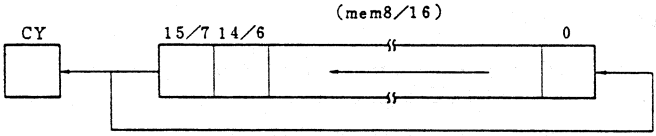
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Rotates left by 1 bit the 8- or 16-bit memory location addressed by the first operand. If the MSB changes, the V flag is set; if it stays the same, the V flag is cleared.



7) Flag operation

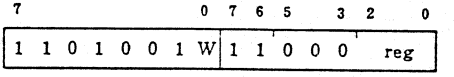
V	S	Z	AC	P	CY
x					x

(3) Register variable-bit

1) Description format

ROL reg,CL

2) Instruction format



3) Number of bytes

2

4) Number of clocks

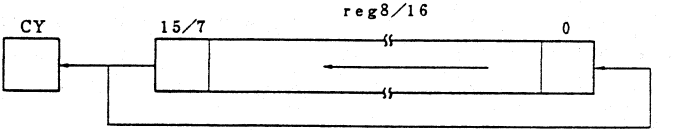
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Rotates left (by the number of bits equal to the number of bits in the CL register) the 8- or 16-bit register specified by the first operand.



7) Flag operation

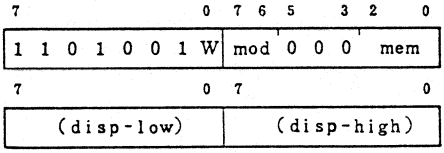
V	S	Z	AC	P	CY
U					x

(4) Memory variable-bit

1) Description format

ROL mem,CL

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: μPD70108

μPD70116 odd addresses

19 + n: μPD70116 even addresses

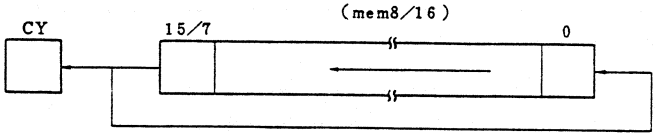
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Rotates left (by the number of bits equal to the number of bits in the CL register) the 8- or 16-bit memory location addressed by the first operand.



7) Flag operation

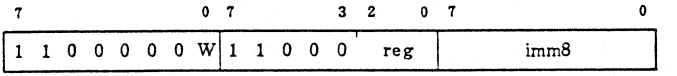
V	S	Z	AC	P	CY
U					x

(5) Register multi-bit

1) Description format

ROL reg,imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

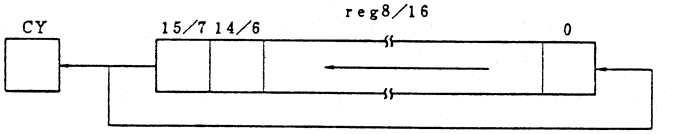
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Rotates left (by the number of bits specified by the 8-bit immediate data in the second operand) the 8- or 16-bit register specified by the first operand. The register's MSB is shifted to the CY flag and also to the LSB.



7) Flag operation

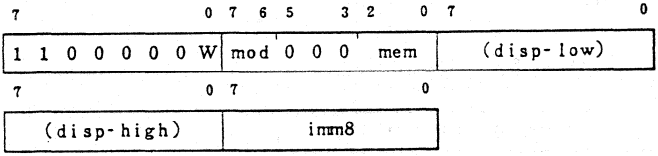
V	S	Z	AC	P	CY
U					X

(6) Memory multi-bit

1) Description format

ROL mem,imm8

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: μPD70108

μPD70116 odd addresses

19 + n: μPD70116 even addresses

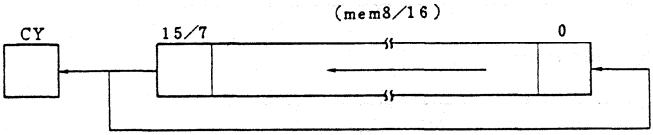
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Rotates left (by the number of bits specified by the 8-bit immediate data in the second operand) the 8- or 16-bit memory location addressed by the first operand. The memory location's MSB is shifted to the CY flag and also to the LSB.



7) Flag operation

V	S	Z	AC	P	CY
U					x

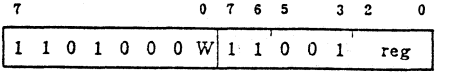
12.19.2 ROR (Rotate Right)

(1) Register single-bit

1) Description format

ROR reg,1

2) Instruction format



3) Number of bytes

2

4) Number of clocks

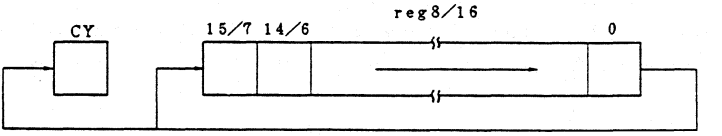
2

5) Number of transfers of 16-bit words

None

6) Function

Rotates right by 1 bit the 8- or 16-bit register specified by the first operand. If the MSB of the specified register changes, the overflow flag is set. If the MSB stays the same, the overflow flag is cleared.



7) Flag operation

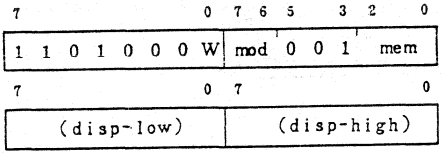
V	S	Z	AC	P	CY
x					x

(2) Memory single-bit

1) Description format

ROR mem,1

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

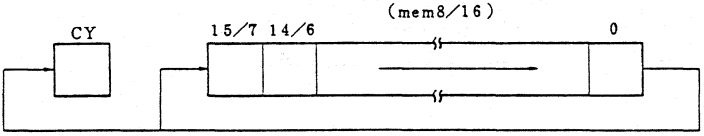
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Rotates right by 1 bit the 8- or 16-bit memory location addressed by the first operand. If the MSB of the addressed memory changes, the overflow flag is set. If the MSB stays the same, the overflow flag is cleared.



7) Flag operation

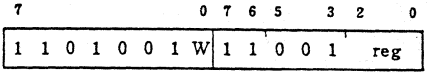
V	S	Z	AC	P	CY
x					x

(3) Register variable-bit

1) Description format

ROR reg,CL

2) Instruction format



3) Number of bytes

2

4) Number of clocks

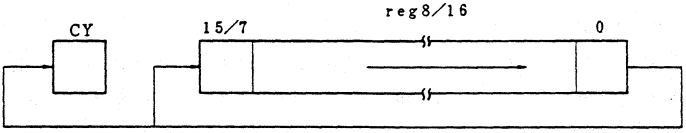
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Rotates right (by the number of bits equal to the number of bits in the CL register) the 8- or 16-bit register specified by the first operand.



7) Flag operation

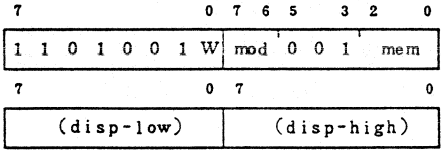
V	S	Z	AC	P	CY
U					x

(4) Memory variable-bit

1) Description format

ROR mem,CL

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: µPD70108
µPD70116 odd addresses

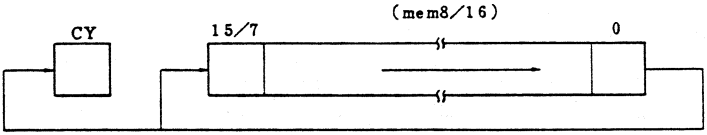
19 + n: µPD70116 even addresses
(n: number of shifts)

5) Number of transfers of 16-bit words

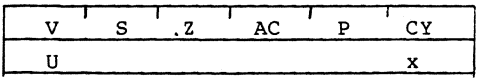
2

6) Function

Rotates right (by the number of bits equal to the number of bits in the CL register) the 8- or 16-bit memory location specified by the first operand.



7) Flag operation

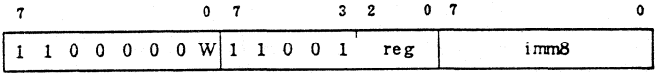


(5) Register multi-bit

1) Description format

ROR reg,imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

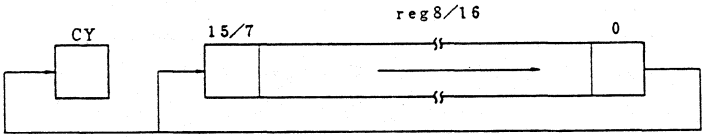
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Rotates right (by the number of bits specified by the 8-bit immediate data in the second operand) the 8- or 16-bit register specified by the first operand. The register's LSB is shifted to the MSB as well as to the CY flag.



7) Flag operation

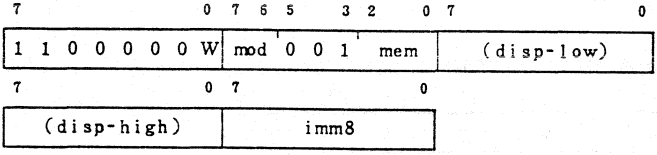
V	S	Z	AC	P	CY
U					x

(6) Memory multi-bit

1) Description format

ROR mem,imm8

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: µPD70108
µPD70116 odd addresses

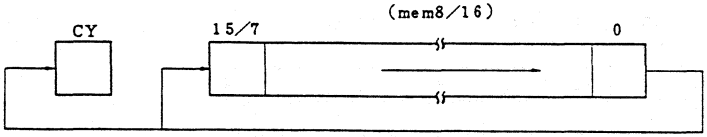
19 + n: µPD70116 even addresses
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Rotates right (by the number of bits specified by the 8-bit immediate data in the second operand) the 8- or 16-bit memory location addressed by the first operand. The memory location's LSB is shifted to the MSB as well as to the CY flag.



7) Flag operation

V	S	Z	AC	P	CY
U					x

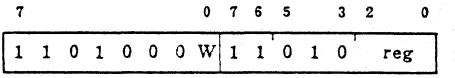
12.19.3 ROLC (Rotate Left with Carry)

(1) Register single-bit

1) Description format

ROLC reg,1

2) Instruction format



3) Number of bytes

2

4) Number of clocks

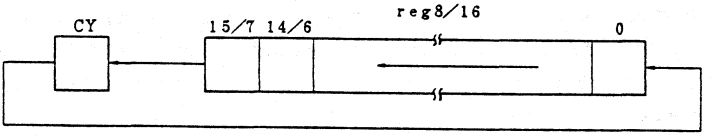
2

5) Number of transfers of 16-bit words

None

6) Function

Rotates left by one bit (including the CY flag) the 8- or 16-bit register specified by the first operand. If the register's MSB changes, the V flag is set. If it stays the same, the V flag is cleared.



7) Flag operation

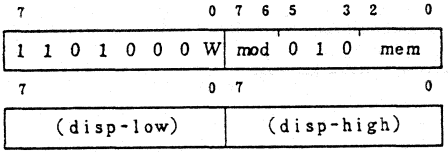
V	S	Z	AC	P	CY
x					x

(2) Memory singl-bit

1) Description format

ROLC mem,1

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

When W=1, 24: μPD70108

μPD70116 odd addresses

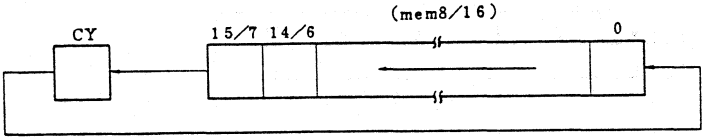
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Rotates left by one bit (including the CY flag) the 8- or 16-bit memory location addressed by the first operand. If the memory location's MSB changes, the V flag is set. If it stays the same, the V flag is cleared.



7) Flag operation

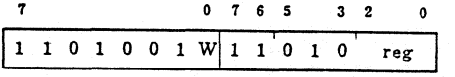
V	S	Z	AC	P	CY
x					x

(3) Register variable-bit

1) Description format

ROLC reg,CL

2) Instruction format



3) Number of bytes

2

4) Number of clocks

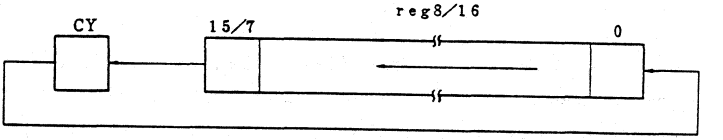
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Rotates left (including the CY flag) the 8- or 16-bit register specified by the first operand by the number of bits equal to the number of bits in the CL register.



7) Flag operation

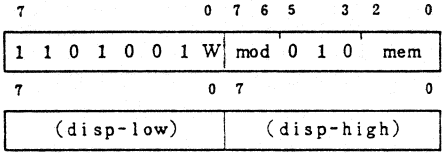
V	S	Z	AC	P	CY
U					x

(4) Memory variable-bit

1) Description format

ROLC mem,CL

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: µPD70108
µPD70116 odd addresses

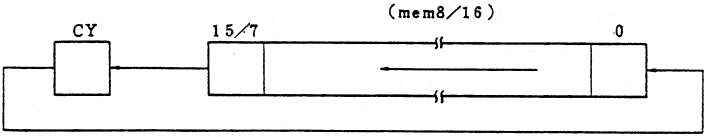
19 + n: µPD70116 even addresses
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Rotates left (including the CY flag) the 8- or 16-bit memory location addressed by the first operand by the number of bits equal to the number of bits in the CL register.



7) Flag operation

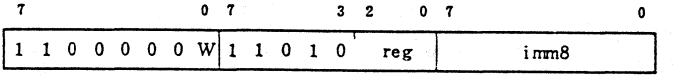
V	S	Z	AC	P	CY
U					X

(5) Register multi-bit

1) Description format

ROLC reg,imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

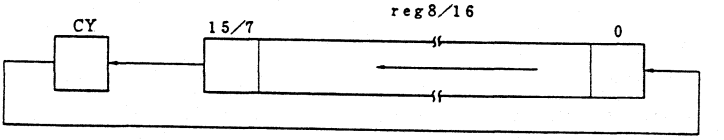
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Rotates left (including the CY flag) the 8- or 16-bit register specified by the first operand by the number of bits specified by the 8-bit immediate data of the second operand.



7) Flag operation

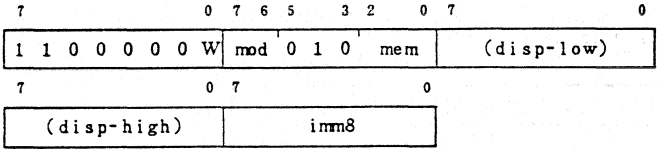
V	S	Z	AC	P	CY
U					x

(6) Memory multi-bit

1) Description format

ROLC mem,imm8

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: μPD70108

μPD70116 odd addresses

19 + n: μPD70116 even addresses

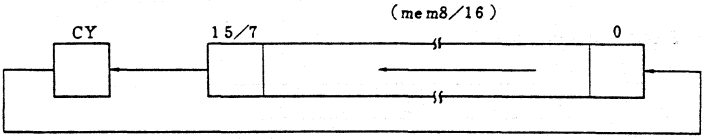
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Rotates left (including the CY flag) the 8- or 16-bit memory location addressed by the first operand by the number of bits specified by the 8-bit immediate data of the second operand.



7) Flag operation

V	S	Z	AC	P	CY
U					x

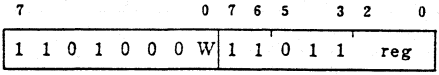
12.19.4 RORC (Rotate Right with Carry)

(1) Register single-bit

1) Description format

RORC reg,1

2) Instruction format



3) Number of bytes

2

4) Number of clocks

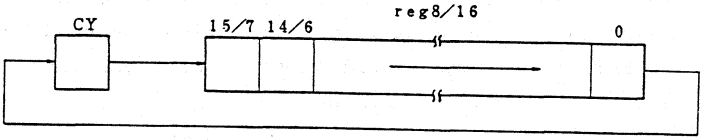
2

5) Number of transfers of 16-bit words

None

6) Function

Rotates right (including the CY flag) by one bit the 8- or 16-bit register specified by the first operand. If the MSB changes, the V flag is set. If it remains unchanged, the V flag is cleared.



7) Flag operation

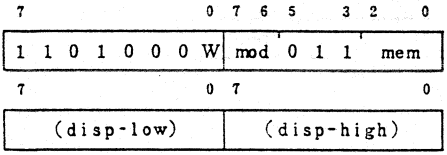
V	S	Z	AC	P	CY
x					x

(2) Memory single-bit

1) Description format

RORC mem,1

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 16

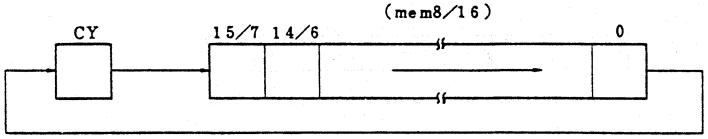
When W=1, 24: μPD70108
μPD70116 odd addresses
16: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

Rotates right (including the CY flag) by one bit the 8- or 16-bit memory location addressed by the first operand. If the MSB changes, the V flag is set. If it remains unchanged, the V flag is cleared.



7) Flag operation

V	S	Z	AC	P	CY
x					x

(3) Register variable-bit

1) Description format

RORC reg,CL

2) Instruction format

7							0	7	6	5			3	2		0
1	1	0	1	0	0	1	W	1	1	0	1	1			reg	

3) Number of bytes

2

4) Number of clocks

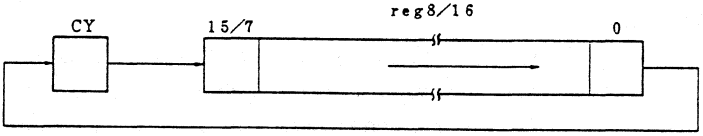
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Rotates right (including the CY flag) by the number of bits equal to the number of bits in the CL register the 8- or 16-bit register specified by the first operand.



7) Flag operation

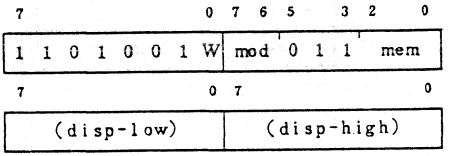
V	S	Z	AC	P	CY
U					x

(4) Memory variable-bit

1) Description format

RORC mem,CL

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: μPD70108
μPD70116 odd addresses

19 + n: μPD70116 even addresses

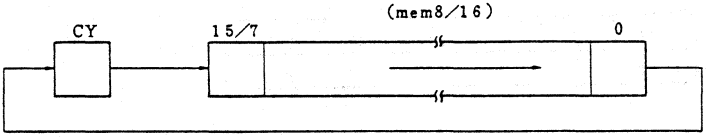
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Rotates right (including the CY flag) by the number of bits equal to the number of bits in the CL register the 8- or 16-bit memory location specified by the first operand.



7) Flag operation

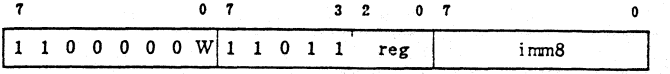
V	S	Z	AC	P	CY
U					x

(5) Register multi-bit

1) Description format

RORC reg,imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

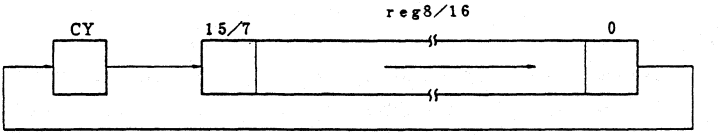
7 + n (n: number of shifts)

5) Number of transfers of 16-bit words

None

6) Function

Rotates right (including the CY flag) the 8- or 16-bit register specified by the first operand by the number of bits specified by the 8-bit immediate data of the second operand.



7) Flag operation

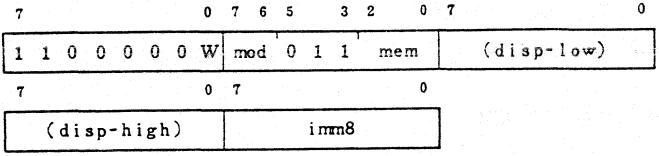
V	S	Z	AC	P	CY
U					x

(6) Memory multi-bit

1) Description format

RORC mem,imm8

2) Instruction format



3) Number of bytes

3/4/5

4) Number of clocks

When W=0, 19 + n

When W=1, 27 + n: μPD70108
μPD70116 odd addresses

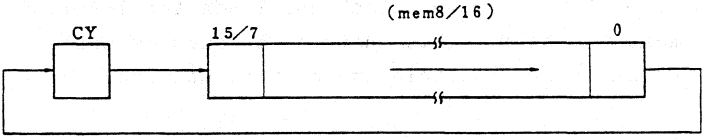
19 + n: μPD70116 even addresses
(n: number of shifts)

5) Number of transfers of 16-bit words

2

6) Function

Rotates right (including the CY flag) the 8- or 16-bit memory location addressed by the first operand by the number of bits specified by the 8-bit immediate data of the second operand.



7) Flag operation

V	S	Z	AC	P	CY
U					x

12.20 Subroutine Control Instructions

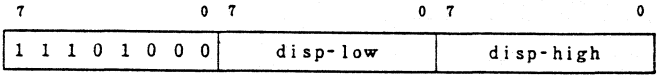
12.20.1 CALL (Call)

(1) Relative (same segment)

1) Description format

CALL near-proc

2) Instruction format



3) Number of bytes

3

4) Number of clocks

20: µPD70108

µPD70116 odd addresses

16: µPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

(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. This instruction enables calls to any address within the current segment.

7) Flag operation

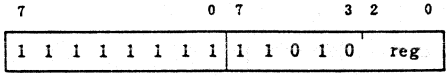
None

(2) Register (same segment)

1) Description format

CALL regptr16

2) Instruction format



3) Number of bytes

2

4) Number of clocks

18: μPD70108
μPD70116 odd addresses
14: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

(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. This instruction enables calls to any address within the current segment.

7) Flag operation

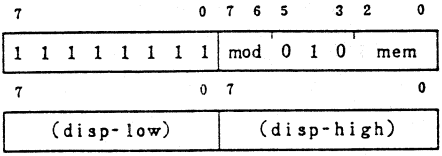
None

(3) Memory (same segment)

1) Description format

CALL memptr16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

31: μPD70108
μPD70116 odd addresses

23: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

(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 (offset) addressed by the operand to the PC. This instruction enables calls to any address within the current segment.

7) Flag operation

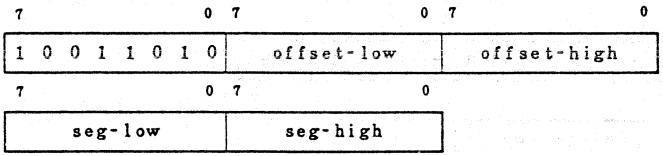
None

(4) Direct (external segment)

1) Description format

CALL far-proc

2) Instruction format



3) Number of bytes

5

4) Number of clocks

29: μPD70108

μPD70116 odd addresses

21: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

(SP - 1, SP - 2) ← PS

(SP - 3, SP - 4) ← PC

SP ← SP - 4

SP ← 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. This instruction enables calls to any address in any segment.

7) Flag operation

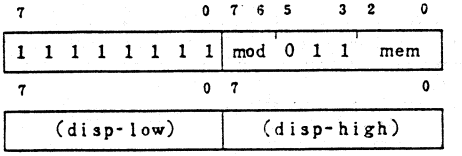
None

(5) Memory (external segment)

1) Description format

CALL memptr32

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

47: μPD70108

μPD70116 odd addresses

31: μPD70116 even addresses

5) Number of transfers of 16-bit words

4

6) Function

(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 and the lower two bytes to the PC. This instruction enables calls to any address in any segment.

7) Flag operation

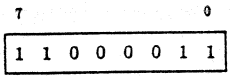
None

12.20.2 RET (Return from Procedure)

(1) Same segment

1) Description format
   RET (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   19: μPD70108
       μPD70116 odd addresses
   15: μPD70116 even addresses

5) Number of transfers of 16-bit words
   1

6) Function
   PC ← (SP + 1, SP)
   SP ← SP + 2

   Restores the PC from the stack and is used for returning from intra-segment calls. The assembler automatically distinguishes this instruction from the RET instruction of (3).

7) Flag operation
   None

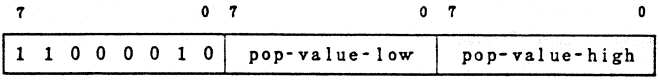
8) Description example
   RET

(2) SP jump (same segment)

1) Description format

RET pop-value

2) Instruction format



3) Number of bytes

3

4) Number of clocks

24: μPD70108
μPD70116 odd addresses
20: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

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. This instruction is effective for jumping a desired number of parameters when the parameters saved in the stack subsequently to the PC become unnecessary. This instruction is used for returning from intra-segment calls. The assembler automatically distinguishes this instruction from the RET pop-value instruction of (4).

7) Flag operation

None

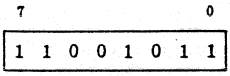
8) Description example

RET 8

(3) External segment

1) Description format
RET (no operand)

2) Instruction format



3) Number of bytes
1

4) Number of clocks
29: μPD70108
μPD70116 odd addresses
21: μPD70116 even addresses

5) Number of transfers of 16-bit words
2

6) Function
PC ← (SP + 1, SP)
PS ← (SP + 3, SP + 2)
SP ← SP + 4

Restores the PC and PS from the stack and is used for returning from inter-segment calls. The assembler automatically distinguishes this instruction from the RET instruction of (1).

7) Flag operation
None

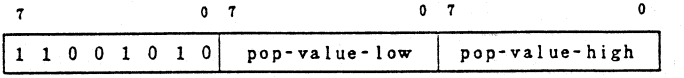
8) Description example
RET

(4) SP jump (inter-segment)

1) Description format

RET pop-value

2) Instruction format



3) Number of bytes

3

4) Number of clocks

32: μPD70108

μPD70116 odd addresses

24: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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 to the PC and PS become unnecessary. This instruction is used for returning from inter-segment calls. The assembler automatically distinguishes this instruction from the RET pop-value instruction of (2).

7) Flag operation

None

12.21 Stack operation Instruction

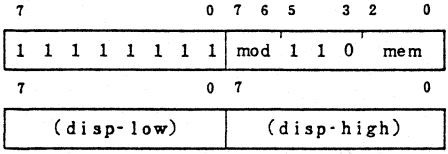
12.21.1 PUSH (Push)

(1) 16-bit memory

1) Description format

PUSH mem16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

26: μPD70108

μPD70116 odd addresses

18: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

(SP - 1, SP - 2) ← (mem16)

SP ← SP - 2

Saves the contents of the 16-bit memory location addressed by the operand to the stack.

7) Flag operation

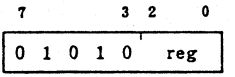
None

(2) 16-bit register

1) Description format

PUSH reg16

2) Instruction format



3) Number of bytes

1

4) Number of clocks

12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

(SP - 1, SP - 2) ← reg16
SP ← SP - 2

Saves the 16-bit register specified by the operand to the stack.

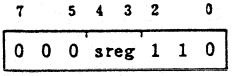
7) Flag operation

None

(3) Segment register

1) Description format
PUSH sreg

2) Instruction format



3) Number of bytes
1

4) Number of clocks
12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words
1

6) Function
(SP - 1, SP - 2) ← sreg
SP ← SP - 2

Saves the segment register specified by the operand to the stack.

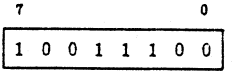
7) Flag operation
None

(4) Program status word

1) Description format

PUSH PSW

2) Instruction format



3) Number of bytes

1

4) Number of clocks

12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

(SP - 1, SP - 2) ← PSW
SP ← SP - 2

Saves the PSW to the stack.

7) Flag operation

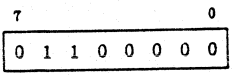
None

(5) Register set

1) Description format

PUSH R

2) Instruction format



3) Number of bytes

1

4) Number of clocks

67: μPD70108

μPD70116 odd addresses

35: μPD70116 even addresses

5) Number of transfers of 16-bit words

8

6) Function

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.

7) Flag operation

None

8) Description example

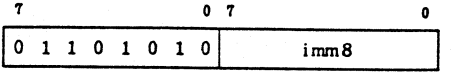
PUSH R

(6) 8-bit immediate data sign expansion

1) Description format

PUSH imm8

2) Instruction format



3) Number of bytes

2

4) Number of clocks

11: μPD70108
μPD70116 odd addresses
7: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

(SP-1, SP-2) ← Sign expansion of imm8
SP ← SP-2

Expands the sign of the 8-bit immediate data described by the operand and saves it as 16-bit data to the stack addressed by the SP.

7) Flag operation

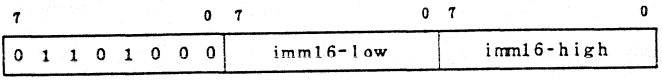
None

(7) 16-bit immediate data

1) Description format

PUSH imm16

2) Instruction format



3) Number of bytes

3

4) Number of clocks

12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

(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.

7) Flag operation

None

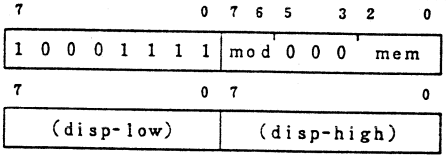
12.21.2 POP (Pop)

(1) 16-bit memory

1) Description format

POP mem16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

25: μPD70108

μPD70116 odd addresses

17: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

(mem16) ← (SP + 1, SP)

SP ← SP + 2

Transfers the contents of the stack to the 16-bit memory location addressed by the operand.

7) Flag operation

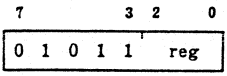
None

(2) 16-bit register

1) Description format

POP reg16

2) Instruction format



3) Number of bytes

1

4) Number of clocks

12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

reg16 ← (SP + 1, SP)
SP ← SP + 2

Transfers the contents of the stack to the 16-bit register specified by the operand.

7) Flag operation

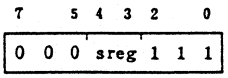
None

(3) Segment register

1) Description format

POP sreg

2) Instruction format



3) Number of bytes

1

4) Number of clocks

12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

sreg ← (SP + 1, SP)
SP ← SP + 2

Transfers the contents of the stack to the segment register (except for PS) specified by the operand. External interrupts (NMI, INT) and single-step break will not be acknowledged between this instruction and the next.

7) Flag operation

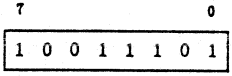
None

(4) Program status word

1) Description format

POP PSW

2) Instruction format



3) Number of bytes

1

4) Number of clocks

12: μPD70108
μPD70116 odd addresses
8: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

PSW ← (SP +1, SP)
SP ← SP + 2

Transfers the contents of the stack to the PSW.

7) Flag operation

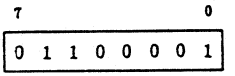
MD	V	DIR	IE	BRK	S	Z	AC	P	CY
R	R	R	R	R	R	R	R	R	R

(5) Register set

1) Description format

POP R

2) Instruction format



3) Number of bytes

1

4) Number of clocks

75: μPD70108

μPD70116 odd addresses

43: μPD70116 even addresses

5) Number of transfers of 16-bit words

7

6) Function

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

The contents of the stack are restored to the following eight 16-bit registers: AW, BW, CW, DW, BP, SP, IX and IY.

7) Flag operation

None

8) Description example

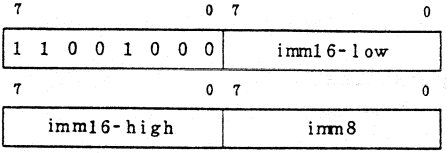
POP R

12.21.3 PREPARE (Prepare New Stack Frame)

1) Description format

PREPARE imm16,imm8

2) Instruction format



3) Number of bytes

4

4) Number of clocks

When imm8=0, 13

When imm8>1, 22 + 20(imm8-1): μPD70108
μPD70116 odd addresses

18 + 12(imm8-1): μPD70116 even addresses

5) Number of transfers of 16-bit words

When imm8=0, none

When imm8>1, 1 + 2(imm8-1)

6) Function

(SP - 1, SP - 2) ← BP

SP ← SP - 2

temp ← SP

When imm8 > 0, repeat these operations "imm8-1" times : } *1

(SP - 1, SP - 2) ← (BP - 1, BP - 2)

SP ← SP - 2

BP ← BP - 2

and perform these operations:

(SP - 1, SP - 2) ← temp } *2

SP ← SP - 2

Then performs these operations:

BP ← temp

SP ← SP - imm16

When imm8=1, *1 is not performed,

When imm8=0, *1 and *2 are not performed.

This instruction is 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 the frame containing the variables that can be referenced from the current procedure.

This instruction copies frame-pointers to reserve the local variable area and to enable global variable references. The first operand (16-bit immediate data) specifies (in bytes) the size of the local variable area. The second operation (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 save to the stack. This is done 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

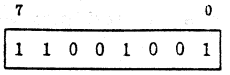
range is always a value one less than the lexical level of the procedure.
If the lexical level of a procedure is greater than 1, the pointers of that procedure will also be saved on the stack. This is so that the frame pointer of the calling procedure can also be copied when frame pointer copy is performed within the called procedure.
Next the new frame pointer value is set in 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.

7) Flag operation
None

12.21.4 DISPOSE (Dispose a Stack Frame)

1) Description format
   DISPOSE (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   10: μPD70108
       μPD70116 odd addresses
    6: μPD70116 even addresses

5) Number of transfers of 16-bit words
   1

6) Function
   SP ← BP
   BP ← (SP + 1, SP)
   SP ← SP + 2

   Releases one frame of the stack frame generated by the PREPARE instruction. A value that points to the preceding frame is loaded in BP and the bottom of the frame value is loaded in SP.

7) Flag operation
   None

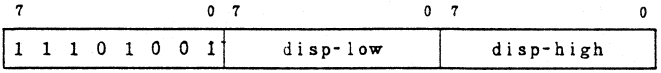
12.22 Branch Instructions

12.22.1 BR (Branch)

(1) Relative (same segment)

1) Description format
   BR near-label

2) Instruction format



3) Number of bytes
   3

4) Number of clocks
   12

5) Number of transfers of 16-bit words
   None

6) Function
   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.

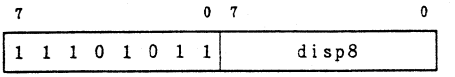
7) Flag operation
   None

(2) Short relative (same segment)

1) Description format

BR short-label

2) Instruction format



3) Number of bytes

2

4) Number of clocks

12

5) Number of transfers of 16-bit words

None

6) Function

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.

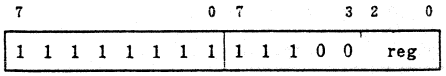
7) Flag operation

None

(3) Register (same segment)

1) Description format
BR regptrl6

2) Instruction format



3) Number of bytes
2

4) Number of clocks
11

5) Number of transfers of 16-bit words
None

6) Function
PC ← regptrl6

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.

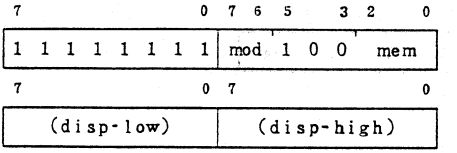
7) Flag operation
None

(4) Memory (same segment)

1) Description format

BR memptr16

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

24: μPD70108
μPD70116 odd addresses
20: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

PC ← (memptr16)

Loads the contents of the 16-bit memory location addressed by the operand to the PC. It can branch to any address in the current segment.

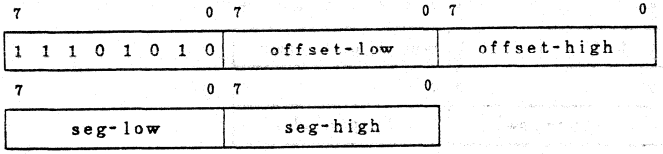
7) Flag operation

None

(5) Direct (external segment)

1) Description format
   BR far-label

2) Instruction format



3) Number of bytes
   5

4) Number of clocks
   15

5) Number of transfers of 16-bit words
   None

6) Function
   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. It can branch to any address in any segment.

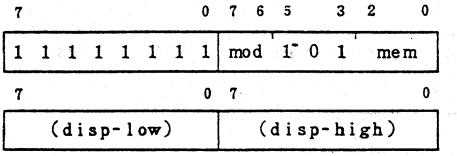
7) Flag operation
   None

(6) Memory (external segment)

1) Description format

BR memptr32

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

35: μPD70108

μPD70116 odd addresses

27: μPD70116 even addresses

5) Number of transfers of 16-bit words

2

6) Function

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. It can branch to any address in any segment.

7) Flag operation

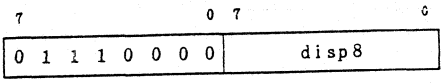
None

12.23 Conditional Branch Instructions

12.23.1 BV (Branch if Overflow)

1) Description format
   BV short-label

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When V=1, 14
   When V=0, 4

5) Number of transfers of 16-bit words
   None

6) Function
   When V=1, PC ← PC + ext-disp8

   When the V 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.

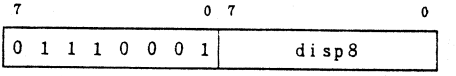
7) Flag operation
   None

12.23.2 BNV (Branch if Not Overflow)

1) Description format

BNV short-label

2) Instruction format



3) Number of bytes

2

4) Number of clocks

When V=0, 14

When V=1, 4

5) Number of transfers of 16-bit words

None

6) Function

When V=0, PC ← PC + ext-disp8

When the V flag is 0, 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.

7) Flag operation

None

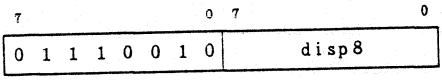
12.23.3 BC/BL (Branch if Carry/Lower)

1) Description format

   BC short-label

   BL short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When CY=1, 14

   When CY=0, 4

5) Number of transfers of 16-bit words

   None

6) Function

   When CY=1, PC ← PC + ext-disp8

   When the CY 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.

7) Flag operation

   None

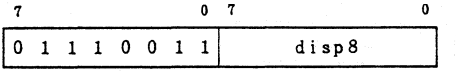
12.23.4 BNC/BNL (Branch if Not Carry/Not Lower)

1) Description format

   BNC short-label

   BNL short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When CY=0, 14

   When CY=1, 4

5) Number of transfers of 16-bit words

   None

6) Function

   When CY=0, PC ← PC + ext-disp8

   When the CY flag is 0, 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.

7) Flag operation

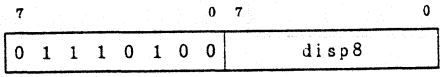
   None

12.23.5 BE/BZ (Branch if Equal/Zero)

1) Description format

BE short-label or BZ short-label

2) Instruction format



3) Number of bytes

2

4) Number of clocks

When Z=1, 14
When Z=0, 4

5) Number of transfers of 16-bit words

None

6) Function

When Z=1, PC ← PC + ext -disp8

When 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.

7) Flag operation

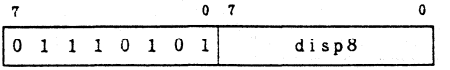
None

12.23.6 BNE/BNZ (Branch if Not Equal/Not Zero)

1) Description format

   BNE short-label or BNZ short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When Z=0, 14

   When Z=1, 4

5) Number of transfers of 16-bit words

   None

6) Function

   When Z=0, PC ← PC + ext-disp8

   When the Z flag is 0, 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.

7) Flag operation

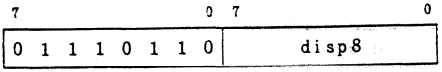
   None

12.23.7 BNH (Branch if Not Higher)

1) Description format

   BNH short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When CY V Z=1, 14

   When CY V Z=0, 4

5) Number of transfers of 16-bit words

   None

6) Function

   When CY V Z=1, PC ← PC + ext-disp8

   When the logical sum of the CY and Z flags 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.

7) Flag operation

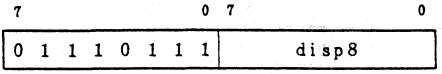
   None

12.23.8 BH (Branch if Higher)

1) Description format

   BH short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When CY V Z=0, 14

   When CY V Z=1, 4

5) Number of transfers of 16-bit words

   None

6) Function

   When CY V Z=0, PC ← PC + ext-disp8

   When the logical sum of the CY and Z flags is 0, 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.

7) Flag operation

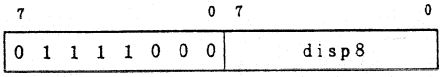
   None

12.23.9 BN (Branch if Negative)

1) Description format

   BN short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When S=1, 14

   When S=0, 4

5) Number of transfers of 16-bit words

   None

6) Function

   When S=1, PC ← PC + ext-disp8

   When the S 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.

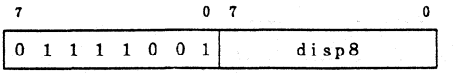
7) Flag operation

   None

12.23.10 BP (Branch if Positive)

1) Description format
   BP short-label

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When S=0, 14
   When S=1, 4

5) Number of transfers of 16-bit words
   None

6) Function
   When S=0, PC ← PC + ext-disp8

   When the S flag is 0, 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.

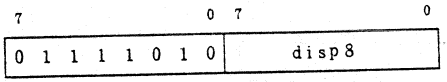
7) Flag operation
   None

12.23.11 BPE (Branch if Parity Even)

1) Description format

BPE short-label

2) Instruction format



3) Number of bytes

2

4) Number of clocks

When P=1, 14

When P=0, 4

5) Number of transfers of 16-bit words

None

6) Function

When P=1, PC ← PC + ext-disp8

When the P 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.

7) Flag operation

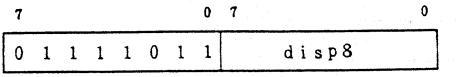
None

12.23.12 BPO (Branch if Parity Odd)

1) Description format

   BPO short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When P=0, 14

   When P=0, 4

5) Number of transfers of 16-bit words

   None

6) Function

   When P=0, PC ← PC + ext-disp8

   When the P flag is 0, 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.

7) Flag operation

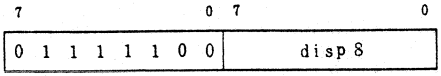
   None

12.23.13 BLT (Branch if Less Than)

1) Description format

   BLT short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When S ⊻ V=1, 14

   When S ⊻ V=0, 4

5) Number of transfers of 16-bit words

   None

6) Function

   When S ⊻ V=1, PC ← PC + ext-disp8

   When the exclusive-or of the S and V flags 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.

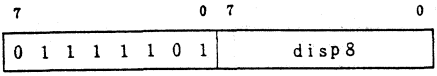
7) Flag operation

   None

12.23.14 BGE (Branch if Greater Than or Equal)

1) Description format
   BGE short-label

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When S ⊻ V=0, 14
   When S ⊻ V=1, 4

5) Number of transfers of 16-bit words
   None

6) Function
   When S ⊻ V=0, PC ← PC + ext-disp8

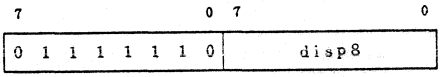
   When the exclusive-or of the S and V flags is 0, loads the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment. when the conditions are unsatisfied, proceeds to the next instruction.

7) Flag operation
   None

12.23.15 BLE (Branch if Less Than or Equal)

1) Description format
   BLE short-label

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When (S ⊻ V) V Z=1, 14
   When (S ⊻ V) V Z=0, 4

5) Number of transfers of 16-bit words
   None

6) Function
   When (S ⊻ V) V Z=1, PC ← PC + ext-disp8

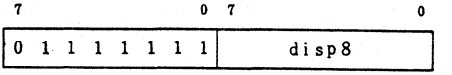
   When the exclusive-or of the S and V flags and the logical sum of that result and the Z flag is 1, loads the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment. When the conditions are unsatisfied, proceeds to the next instruction.

7) Flag operation
   None

12.23.16 BGT (Branch if Greater Than)

1) Description format
   BGT short-label

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When (S ⊻ V)V Z=0, 14
   When (S ⊻ V)V Z=1, 4

5) Number of transfers of 16-bit words
   None

6) Function
   When (S ⊻ V) V Z=0, PC ← PC + ext-disp8

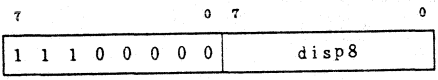
   When the exclusive-or of the S and V flags and the logical sum of that result and the Z flag is 0, loads the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment. When the conditions are unsatisfied, proceeds to the next instruction.

7) Flag operation
   None

12.23.17 DBNZNE (Decrement and Branch if Not Zero and Not Equal)

1) Description format
   DBNZNE short-label

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When CW≠0 and Z=0, 14
   When others, 5

5) Number of transfers of 16-bit words
   None

6) Function
   CW ← CW - 1
   When CW≠0 and Z=0, PC ← PC + ext-disp8

   When the 16-bit register CW is decremented (-1) and the resultant CW value is not 0 and the Z flag is cleared, 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.

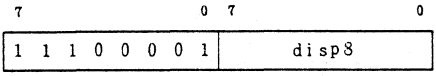
7) Flag operation
   None

12.23.18 DBNZE (Decrement and Branch if Not Zero and Equal)

1) Description format

   DBNZE short-label

2) Instruction format



3) Number of bytes

   2

4) Number of clocks

   When CW≠0 and Z=1, 14

   When others 5

5) Number of transfers of 16-bit words

   None

6) Function

   CW ← CW - 1

   When CW≠0 and Z=1, PC ← PC + ext -disp8

   When the 16-bit register CW is decremented (-1) and the CW is not zero and the Z flag is set, 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.

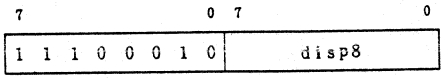
7) Flag operation

   None

12.23.19 DBNZ (Decrement and Branch if Not Zero)

1) Description format
   DBNZ short-label

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When CW≠0, 13
   When CW=0, 5

5) Number of transfers of 16-bit words
   None

6) Function
   CW ← CW - 1
   When CW≠0, PC ← PC + ext-disp8

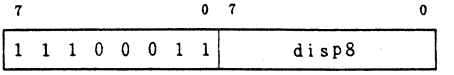
   When the 16-bit register CW is decremented (-1) and the CW value is not zero, loads the current PC value plus the 8-bit (actually, sign-extended 16-bit) displacement value to the PC. This instruction can branch to any address within ±127 bytes of the instruction in the current segment. When the conditions are unsatisfied, proceeds to the next instruction.

7) Flag operation
   None

12.23.20 BCWZ (Branch if CW equals Zero)

1) Description format
   BCWZ short-label

2) Instruction format



3) Number of bytes
   2

4) Number of clocks
   When CW=0, 13
   When CW≠0, 5

5) Number of transfers of 16-bit words
   None

6) Function
   When CW=0, PC ← PC + ext-disp8

   When the 16-bit register CW is 0, 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.

7) Flag operation
   None

12.24 Break Instructions

12.24.1 BRK (Break)

(1) Vector 3

1) Description format

BRK 3

2) Instruction format

7							0
1	1	0	0	1	1	0	0

3) Number of bytes

1

4) Number of clocks

58: μPD70108
μPD70116 odd addresses
38: μPD70116 even addresses

5) Number of transfers of 16-bit words

5

6) Function

(SP-1, SP-2) ← PSW
(SP-3, SP-4) ← PS
(SP-5, SP-6) ← PC
SP ← SP - 6
IE ← 0
BRK ← 0
PC ← (00DH, 00CH)
PS ← (00FH, 00EH)

Saves the PSW, PS, and PC to the stack and resets (to 0) the IE and BRK flags. Then loads the lower two bytes and higher two bytes of vector 3 of the interrupt vector table to the PC and PS, respectively.

7) Flag operation

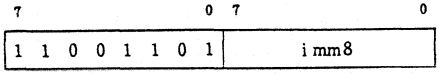
IE	BRK
0	0

(2) Immediate data

1) Description format

BRK imm8 (≠3)

2) Instruction format



3) Number of bytes

1

4) Number of clocks

58: μPD70108
μPD70116 odd addresses
38: μPD70116 even addresses

5) Number of transfers of 16-bit words

5

6) Function

(SP-1, SP-2) ← PSW
(SP-3, SP-4) ← PS
(SP-5, SP-6) ← PC
SP ← SP - 6
IE ← 0
BRK ← 0
PC ← (imm8 x 4 + 1, imm8 x 4)
PS ← (imm8 x 4 + 3, imm8 x 4 + 2)

Saves the PSW, PS, and PC to the stack and resets (to 0) the IE and BRK flags. 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.

7) Flag operation

IE	BRK
0	0

12.24.2 BRKV (Break if Overflow)

1) Description format

   BRKV (no operand)

2) Instruction format

7							0
1	1	0	0	1	1	1	0

3) Number of bytes

   1

4) Number of clocks

   When V=1, 60: μPD70108

   μPD70116 odd addresses

   40: μPD70116 even addresses

   When V=0, 3

5) Number of transfers of 16-bit words

   5

6) Function

   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 (to 0) the IE and BRK flags. 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.

7) Flag operation

IE	BRK
0	0

8) Description example

BRKV

12.24.3 RETI (Return from Interrupt)

1) Description format

   RETI (no operand)

2) Instruction format

7							0
1	1	0	0	1	1	1	1

3) Number of bytes

   1

4) Number of clocks

   39: μPD70108
   μPD70116 odd addresses
   27: μPD70116 even addresses

5) Number of transfers of 16-bit words

   3

6) Function

   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 processinq.

7) Flag operation

MD	V	DIR	IE	BRK	S	Z	AC	P	CY
R	R	R	R	R	R	R	R	R	R

8) Description example

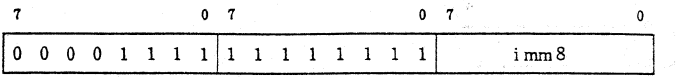
   RETI

12.24.4 BRKEM (Break for Emulation)

1) Description format

BRKEM imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

58: μPD70108

μPD70116 odd addresses

38: μPD70116 even addresses

5) Number of transfers of 16-bit words

5

6) Function

(SP-1, SP-2) ← PSW

(SP-3, SP-4) ← PS

(SP-5, SP-6) ← PC

SP ← SP - 6

MD ← 0

PS ← (imm8 x 4 + 3, imm8 x 4 + 2)

PC ← (imm8 x 4 + 1, imm8 x 4)

This instruction starts the emulation mode. Saves the PSW, PS, and PC and resets (to 0) the MD.

Then, it jumps to the emulation address addressed by the interrupt vector specified by the 8-bit immediate data described by the operand. After fetching the instruction code of the jumped interrupt service routine (for emulation), the CPU interprets and executes the code as the instruction of the

μPD8080AF. Either the RETEM or CALLN instruction is used to return from the emulation mode to the native mode (μPD70108/70116).

7) Flag operation

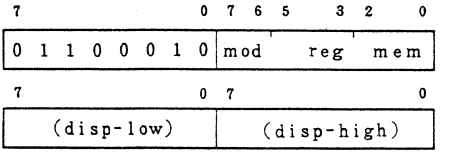
BRK
0

12.24.5 CHKIND (Check Index)

1) Description format

   CHKIND reg16, mem32

2) Instruction format



3) Number of bytes

   2/3/4

4) Number of clocks

   When interrupt condition is fulfilled,

   81-84: μPD70108

          μPD70116 odd addresses

   53-56: μPD70116 even addresses

   When interrupt condition is not fulfilled,

   26: μPD70108

       μPD70116 odd addresses

   18: μPD70116 even addresses

5) Number of transfers of 16-bit words

   When interrupt condition is fulfilled, 7

   When interrupt condition is not fulfilled, 2

6) Function

When (mem32) > reg16 or (mem32 + 2) < reg16,

(SP-1, SP-2) ← PSW

(SP-3, SP-4) ← PS

(SP-5, PS-6) ← PC

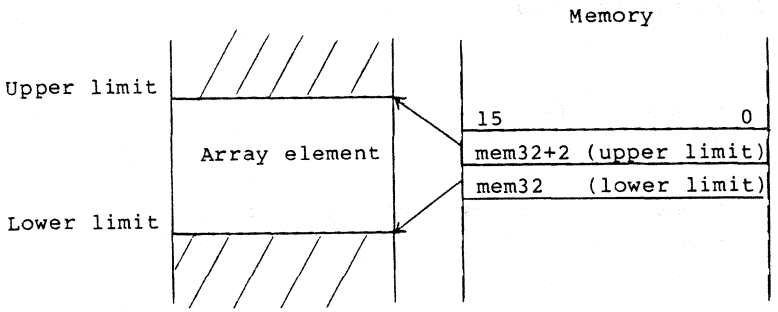
SP ← SP - 6

IE ← 0

BRK ← 0

PS ← (23, 22)

PC ← (21, 20)



This instruction is used to check whether the index value that specifies the data element is in the definition region of the data array. It initiates BRK 5 when the index exceeds the definition region. 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 the memory. The object of the index value is the register (random 16-bit register) used by the array handling program.

7) Flag operation

When interrupt condition is fulfilled,

IE	BRK
0	0

When interrupt condition is not fulfilled,

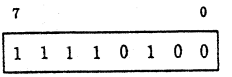
None

12.25 CPU Control Instructions

12.25.1 HALT (Halt)

1) Description format
   HALT (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   Sets at the halt state.
   The halt state is released by either of the following three sources:

   RESET input
   NMI input
   INT input

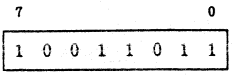
7) Flag operation
   None

8) Description example
   HALT

12.25.2 POLL (Poll and wait)

1) Description format
   POLL (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   2 + 5 x n
   n: Number of $\overline{\text{POLL}}$ pin sampling.

5) Number of transfers of 16-bit words
   None

6) Function
   Keeps the CPU in the wait state until the $\overline{\text{POLL}}$ pin becomes active (low level).

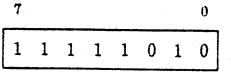
7) Flag operation
   None

8) Description example
   POLL

12.25.3 DI (Disable Interrupt)

1) Description format
   DI (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   IE ← 0

   Resets the IE flag and disables the external maskable interrupt input (INT). This instruction does not disable external non-maskable interrupt input (NMI) and software interrupt instruction.

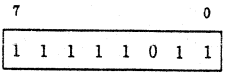
7) Flag operation

IE
0

8) Description example
   DI

12.25.4 EI (Enable Interrupt)

1) Description format
   EI (no operand)



3) Number of bytes
   1

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   IE ← 1

   Sets the IE flag and enables external maskable interrupt input (INT). The system does not enter the interrupt-enable state until executing the instruction immediately after the EI.

7) Flag operation

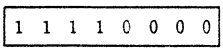
IE
1

8) Description example
   EI

12.25.5 BUSLOCK (Bus Lock Prefix)

1) Description format
   BUSLOCK (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   2

5) Number of transfers of 16-bit words
   None

6) Function
   Outputs the buslock signal ($\overline{\text{BUSLOCK}}$) for the μPD70108/70116 of a large scale mode (S/$\overline{\text{LG}}$=0) configuration while the instruction immediately after the BUSLOCK instruction is being executed.
   When BUSLOCK instruction is used for a block operation instruction with repeat prefix, $\overline{\text{BUSLOCK}}$ signal is kept active (low level) until the end of the block operation instruction when BUSLOCK instruction is executed in the small-scale mode (S/$\overline{\text{LG}}$=1), then $\overline{\text{BUSLOCK}}$ signal is not output.
   In the large-scale mode, hold request is inhibited during $\overline{\text{BUSLOCK}}$ signal is active.
   Accordingly BUSLOCK instruction is effective for the case when program doesn't like to acknowledge a hold request during block operations.

7) Flag operation
   None

8) Description example
```
BUSLOCK
REP
MOVBKB
```

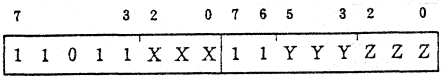
12.25.6 FPO1 (Floating Point Operation 1)

(1) Register

1) Description format

FPO1 fp-op

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

This instruction is used for the floating point arithmetic chip to be connected externally. When the CPU fetches this instruction, it leaves arithmetic processing to the operation chip. When the operation chip (a co-processor) monitors this instruction, it treats the instruction as one it has been given and executes it.

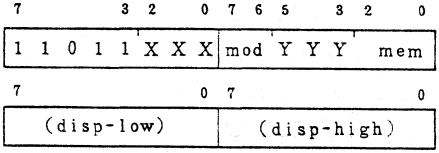
7) Flag operation

None

(2) Memory

1) Description format

FPO1 fp-op,mem



3) Number of bytes

2/3/4

4) Number of clocks

15: μPD70108

μPD70116 odd addresses

11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

data bus ← (mem)

This is the instruction for the externally connected, floating point arithmetic chip. When the CPU fetches this instruction, it leaves arithmetic processing to the operating chip and carries out auxiliary processing (calculation of effective address, generation of physical address and start of memory read cycle) as necessary.

When the operating chip (acting as co-processor) monitors this instruction, it treats the instruction as intended for itself and executes it. In this case, depending on the type of instruction, the operating chip selects either the address information of the memory read cycle started by the CPU or both the address and read data. The CPU does not use the read data on the data bus in the memory read cycle Which has been initiated by the CPU.

7) Flag operation

None

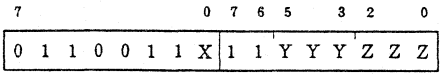
12.25.7 FPO2 (Floating Point Operation 2)

(1) Register

1) Description format

FPO2 fp-op

2) Instruction format



3) Number of bytes

2

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

This instruction is used for an externally connected, floating point arithmetic chip. When the CPU fetches this instruction, it leaves processing to the operating chip. When the operating chip (acting as co-processor) monitors this instruction, it interprets the instruction as intended for itself and executes it.

7) Flag operation

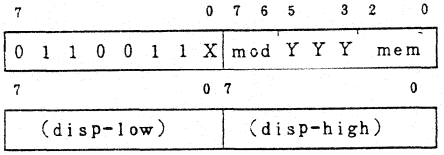
None

(2) Memory

1) Description format

FPO2 fp-op, mem

2) Instruction format



3) Number of bytes

2/3/4

4) Number of clocks

15: μPD70108
μPD70116 odd addresses
11: μPD70116 even addresses

5) Number of transfers of 16-bit words

1

6) Function

data bus ← (mem)

This is the instruction for an externally connected, floating point arithmetic chip. When the CPU fetches this instruction, it leaves arithmetic processing to the arithmetic chip and carries out auxiliary processing (calculation of effective address, generation of physical address and start of memory read cycle) as necessary.
When the operating chip (acting as co-processor) monitors this instruction, it treats the instruction as intended for itself and executes it. In this case, depending on the type of instruction, the operating chip selects either the address information of the memory read cycle started by the CPU or both the address and read data.

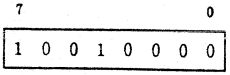
7) Flag operation

None

12.25.8 NOP (No Operation)

1) Description format
   NOP (no operand)

2) Instruction format



3) Number of bytes
   1

4) Number of clocks
   3

5) Number of transfers of 16-bit words
   None

6) Function
   Does nothing for three clocks.

7) Flag operation
   None

8) Description example
   NOP

12.26 Segment override prefix

1) Description format

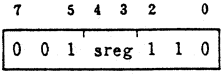
DS0:

DS1:

PS:

SS:

2) Instruction format



3) Number of bytes

1

4) Number of clocks

2

5) Number of transfers of 16-bit words

None

6) Function

It appended to the operand and specifies the segment register to be used for access of a memory operand expecting segment-override.

Programmer can define the segment override by assembler directive "ASSUME" without describing the segment override prefix directly (See Assembler Operating Manual).

7) Flag operation

None

8) Description example

```
REP MOVBKB BYTE_VAR, SS : BYTE_VAR
```

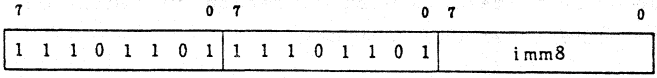
12.27 Emulation Mode Instructions

12.27.1 CALLN (Call Native)

1) Description format

CALLN imm8

2) Instruction format



3) Number of bytes

3

4) Number of clocks

58: μPD70108
μPD70116 odd addresses
38: μPD70116 even addresses

5) Number of transfers of 16-bit words

5

6) Function

(SP-1, SP-2) ← PSW
(SP-3, SP-4) ← PS
(SP-5, SP-6) ← PC
SP ← PS - 6
MD ← 1
PS ← (imm8 x 4 + 3, imm8 x 4 + 2)
PC ← (imm8 x 4 + 1, imm8 x 4)

When this command is executed in the emulation mode (it is interpreted as an uPD8080AF command), the CPU saves the PS, PC, and PSW to the stack (MD=0 is also saved). Then the MD flag is set to 1. The interrupt vector specified by the 8-bit immediate data of the operand is loaded into PS and PC. This command allows the user to call a native mode interrupt routine from the emulation mode.
The RETI command is used to return to emulation mode from the interrupt routine.

7) Flag operation

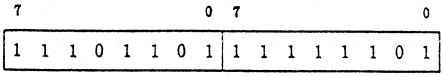
MD
1

12.27.2 RETEM (Return from Emulation)

1) Description format

RETEM

2) Instruction format



3) Number of bytes

2

4) Number of clocks

39: μPD70108
μPD70116 odd addresses
27: μPD70116 even addresses

5) Number of transfers of 16-bit words

3

6) Function

PC ← (SP + 1, SP)
PS ← (SP + 3, SP + 2)
PSW ← (SP + 5, SP + 4)
SP ← SP + 6

When the RETEM command is executed in the emulation mode (it is interpreted as a μPD8080AF command), the CPU restores the PS, PC and PSW saved by the BRKEM command, in the same manner as when returning from interrupt processing. The state of the MD flag (i.e., 1) save by the BRKEM command are also restored, causing the CPU to return to native mode ((μPD70108/70116).

7) Flag operation

MD	V	DIR	IE	BRK	S	Z	AC	P	CY
R	R	R	R	R	R	R	R	R	R

Chapter 13 μPD8080AF Emulation

The μPD70108/70116 is provided with two CPU operation modes: native and emulation. In native mode, the μPD70108/70116 executes instructions specifically intended for the two microprocessors. In emulation mode, the microprocessors execute the instruction set for the μPD8080AF.
These modes are selected by an instruction specially provided for the microprocessors or by using an interrupt. The most significant bit of the PSW is mode (MD) flag that controls the mode selection.

13.1 From Native to Emulation Mode

Two instructions cause the mode to be changed from the native to the emulation mode: BRKEM (Break for Emulation) and RETI (Return from Interrupt).

13.1.1 BRKEM imm8 instruction

This is a basic instruction that starts the emulation mode. It performs the same operation as a normal interrupt instruction (BRK) except that the BRKEM imm8 instruction resets the mode flag.
Therefore, the BRKEM imm8 instruction saves the contents of the PSW, PS, and PC and resets (to 0) the interrupt enable (IE), break (BRK), and MD flags. The segment base of an interrupt vector specified by an operand is loaded to a segment register (PS) and the offset is loaded to the program counter (PC).
The emulation mode started in this way to process an interrupt (when MD=0) executes the program in the 64K-byte segment area specified by the contents of the PS, starting from the address indicated by the contents of the PC. The instruction code fetched at this point is interpreted as the instruction of the μPD8080AF and executed (see Fig. 13-1).

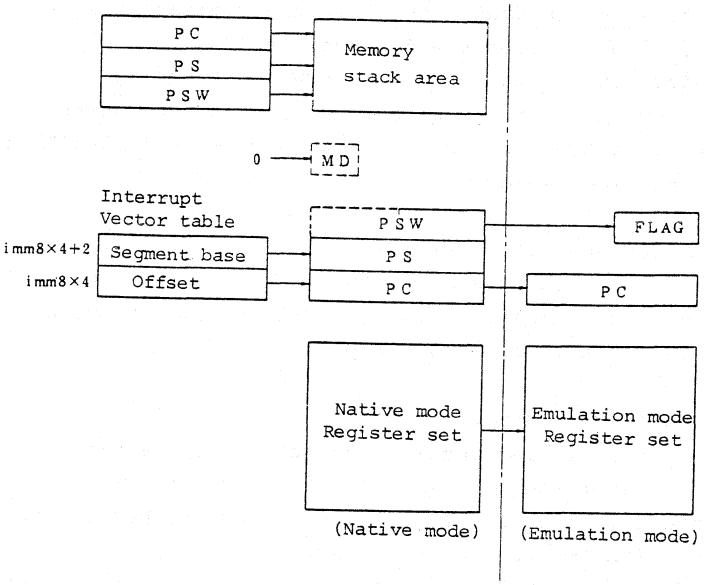


Fig. 13-1 Mode Shift by BRKEM Instruction

13.1.2 RETI instruction

This instruction is generally used when returning program execution of the CPU to the main routine from an interrupt routine started by an external interrupt or BRK instruction. When the contents of the PSW are restored along with the contents of the PS and PC by the RETI instruction, the status of the mode (MD) flag (i.e., 0), which was also saved when the mode was changed from emulation to native, is also restored. This restored MD flag allows the CPU to be set in emulation mode again (see Fig. 13-2). For this reason, if the RETI instruction is executed in native mode at the end of the interrupt routine that has been started by an interrupt instruction (CALLN) or by an external interrupt while the CPU is in emulation mode, the CPU can reenter the emulation mode.

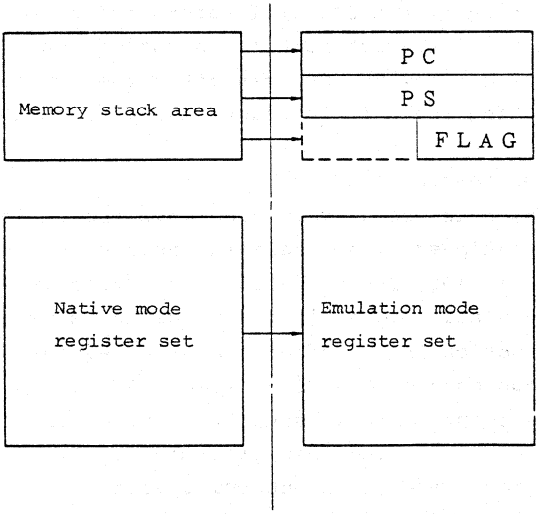


Fig. 13-2 Mode Shift by RETI Instruction

13.2 From Emulation Mode to Native Mode

The following signals and instructions are used to change the mode from emulation to native.

1) RESET input
2) NMI or INT input
3) CALLN (Call Native) instruction
4) RETEM (Return from Emulation) instruction

13.2.1 RESET input

When the RESET signal is input, a reset operation is performed on the CPU the same as in native mode. The emulation in progress will be aborted.

13.2.2 NMI or INT input

When the NMI or INT signal is input, the interrupt process is performed the same as in native mode. Program execution of the CPU will return to the main routine from the interrupt routine in native mode. From this native mode, the CPU can enter the emulation mode by executing the RETI instruction (see Fig. 13-3).

13.2.3 CALLN imm8 instruction

This instruction is exclusively used when the emulation mode that is assigned with undefined instruction code of the μPD8080AF is set. If the CALLN instruction is executed in emulation mode, it functions the same as the BRK instruction in native mode and causes the CPU to save the contents of the PS, PC, and PSW to the stack area (at this point, MD=0 is saved), to reset the IE and BRK flags to 0, to set the mode flag (MD) to 1, and to load the setgment base of an interrupt vector to the segment register (PS) and the offset to the program counter (PC) (see Fig. 13-3).

When the RETI instruction is executed at the end of the interrupt routine, program execution of the CPU can be returned to the main routine in emulation mode from the interrupt routine in native mode started by the CALLN instruction.
Using this function, the program for the μPD70108/70116 can be run while the program of the μPD8080AF is being executed and the result of the program execution of the μPD70108/70116 can be used for a required purpose.

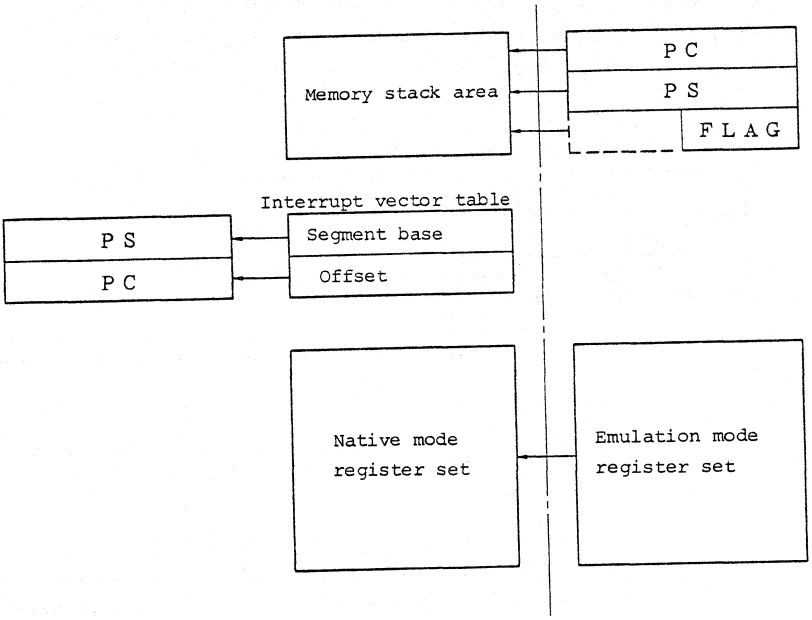


Fig. 13-3 Mode Shift by NMI, INT Input and CALLN Instruction

13.2.4 RETEM instruction

This instruction is exclusively used when the emulation mode that is assigned with the undefined instruction code of the μPD8080AF is set. If the RETEM instruction is executed in emulation mode, program execution of the CPU will return from the interrupt routine to the main routine. Consequently, the contents of the PS, PC, and PSW are restored and the CPU reenters the native mode. At this time, the MD flag contents (i.e., 1), which have been saved to the stack by the BRKEM instruction, are restored. This restoration causes the CPU to be set in native mode (see Fig. 13-4).

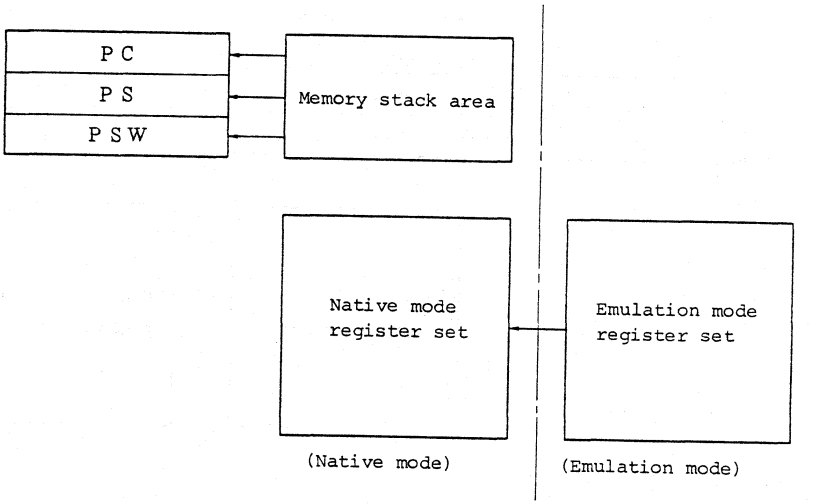


Fig. 13-4 Mode shift by RETEM Instruction

13.3 Emulation Mode

When the mode is changed from native to emulation or vice versa, the statuses in the CPU are passed from one mode to another as shown in Fig. 13.5.

The lower 8 bits of the AW register and both the lower and higher 8 bits of the BW, CW, and DW registers of the μPD70108/70116 play the roles of the accumulator and six general-purpose registers of the μPD8080AF.

The lower 8 bits of the PSW of the μPD70708/70116 play the roles of the μPD8080AF flags. These flags correspond to the lower 8 bits of the PSW.

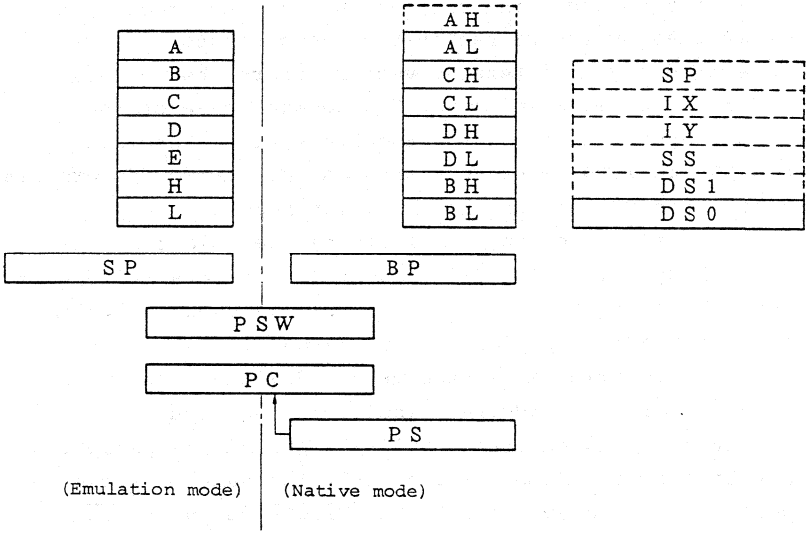


Fig. 13-5 Correspondence of Registers

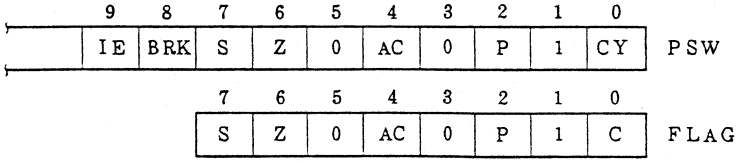


Fig. 13-6 Correspondence Between the PSW and Flags

The SP register serves as the stack pointer of the μPD8080AF in native mode while the BP register acts as the stack pointer in emulation mode. In this way, the μPD70108/70116 employs independent stack pointers and secures stack areas respectively used in each mode. Using independent stack pointers this way prevents accidents such as destruction of the contents of a stack pointer in one mode due to misoperation of the stack pointer in the other mode.

The AH, SP, IX, and IY registers and the four segment registers (PS, SS, DS0, and DS1) are not affected by the emulation mode when they operate in native mode.

In emulation mode, the segment base of the program is determined by the PS register whose contents have been determined by an interrupt vector before the CPU has entered the emulation mode. The segment base of the memory operands (including the stack) is determined by the DS0 register whose contents the programmer determines before the CPU enters the emulation mode.

The bus hold function (available by the hold request/acknowledge/signal) and standby function (available when the HLT instruction is executed) can be used in emulation mode in the same way as the native mode.

Even in emulation mode, the uPD70108/70116 does operate in terms of its hardware. Therefore the input/output operations between the uPD70108/70116 and the peripheral circuits or the memory are exactly the same as those performed in native mode. The $\overline{\text{BUSLOCK}}$ output or $\overline{\text{POLL}}$ input signal that is operated by a uPD70108/70116 instruction is not performed,however.

Whether the external circuits are in emulation mode can be checked by confirming that the processor status PS3 signal output during a μPD70108/70116 bus cycle has become high (this signal is always at low level in the native mode).

Fig. 13-7 shows mode shift operation of the CPU.

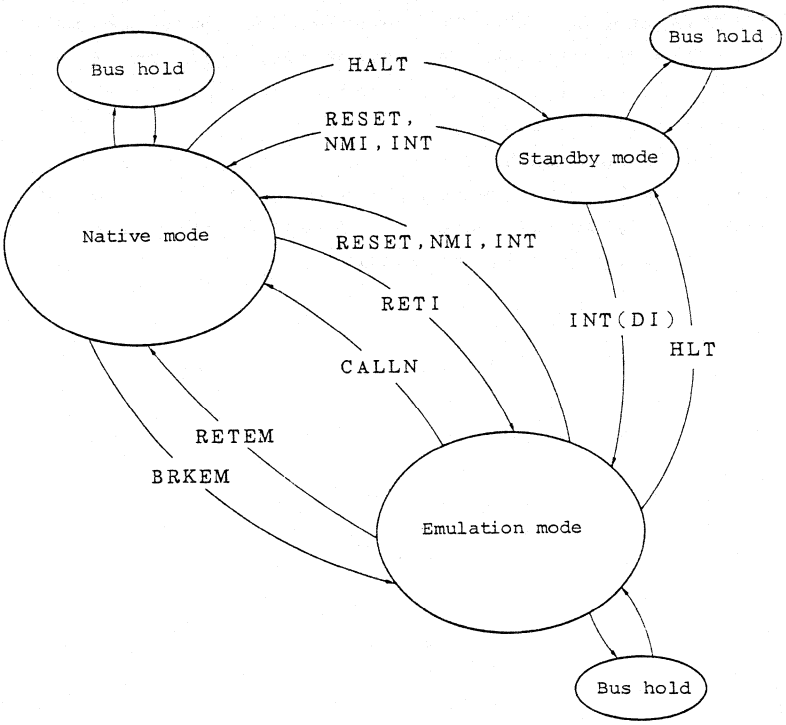


Fig. 13-7 Mode Shift of CPU

Suppose the CPU has entered the standby mode from the emulation mode. The CPU can reenter the emulation mode when the INT signal is input while the interrupt is disabled and restart program execution beginning with the instruction next to the HLT instruction. If the RESET or NMI signal is input instead of the INT signal, or if the INT signal is input while the interrupt is enabled, the CPU will enter the native mode from the standby mode. If this happens, the CPU can enter the emulation mode from the native mode, in other words, from the NMI or INT interrupt routine in native mode, through execution of the RETI instruction.

If the CPU has entered the standby mode from the native mode, the CPU can reenter the native mode by input of the RESET, NMI, or INT signal regardless of whether the interrupt is disabled or enabled.

Appendix A List of μPD70108/70116 Instruction Mnemonics

ADD	Add
ADD4S	Add Nibble String
ADDC	Add with Carry
ADJ4A	Adjust Nibble Add
ADJ4S	Adjust Nibble Subtract
ADJBA	Adjust Byte Add
ADJBS	Adjust Byte Subtract
AND	And
BC	Branch if Carry
BCWZ	Branch if CW equals Zero
BE	Branch if Equal
BGE	Branch if Greater Than or Equal
BGT	Branch if Greater Than
BH	Branch if Higher
BL	Branch if Lower
BLE	Branch if Less Than or Equal
BLT	Branch if Less Than
BN	Branch if Negative
BNC	Branch if Not Carry
BNE	Branch if Not Equal
BNH	Branch if Not Higher
BNL	Branch if Not Lower
BNV	Branch if Not Overflow
BNZ	Branch if Not Zero
BP	Branch if Positive
BPE	Branch if Parity Even
BPO	Branch if Parity Odd
BR	Branch
BRK	Break
BRKEM	Break for Emulation
BRKV	Break for Overflow
BUSLOCK	Bus Lock
BV	Branch if Overflow
BZ	Branch if Zero
CALL	Call
CALLN	Call Native mode
CHKIND	Check Index
CLR1	Clear Bit
CMP	Compare
CMP4S	Compare Nibble String
CMPBK	Compare Block
CMPBKB	Compare Block Byte
CMPBKW	Compare Block Word
CMPM	Compare Multiple
CMPMB	Compare Multiple Byte
CMPMW	Compare Multiple Word
CVTBD	Convert Binary to Decimal
CVTBW	Convert Byte to Word
CVTDB	Convert Decimal to Binary
CVTWL	Convert Word to Long Word
DBNZ	Decrement and Branch if Not Zero
DBNZE	Decrement and Branch if Not Zero and Equal
DBNZNE	Decrement and Branch if Not Zero and Not Equal
DEC	Decrement

DI	Disable Interrupt
DISPOSE	Dispose
DIV	Divide Signed
DIVU	Divide Unsigned
DS0:	Data Segment 0
DS1:	Data Segment 1
EI	Enable Interrupt
EXT	Extract Bit Field
FPO1	Floating Point Operation 1
FPO2	Floating Point Operation 2
HALT	Halt
IN	Input
INC	Increment
INM	Input Multiple
INS	Insert Bit Field
LDEA	Load Effective Address
LDM	Load Multiple
LDMB	Load Multiple Byte
LDMW	Load Multiple Word
MOV	Move
MOVBK	Move Block
MOVBKB	Move Block Byte
MOVBKW	Move Block Word
MUL	Multiply Signed
MULU	Multiply Unsigned
NEG	Negate
NOP	No Operation
NOT	Not
NOT1	Not Bit
OR	Or
OUT	Output
OUTM	Output Multiple
POLL	Poll and wait
POP	Pop
PREPARE	Prepare
PS:	Program Segment
PUSH	Push
REP	Repeat Block Operation
REPC	Repeat Block Operation While Carry
REPE	Repeat Block Operation While Equal
REPNC	Repeat Block Operation While Not Carry
REPNE	Repeat Block Operation While Not Equal
REPNZ	Repeat Block Operation While Not Zero
REPZ	Repeat Block Operation While Zero
RET	Return
RETEM	Return from Emulation
RETI	Return from Interrupt
ROL	Rotate Left
ROL4	Rotate Left Nibble
ROLC	Rotate Left with Carry
ROR	Rotate Right
ROR4	Rotate Right Nibble
RORC	Rotate Right with Carry
SET1	Set Bit
SHL	Shift Left Logical
SHR	Shift Right Logical
SHRA	Shift Right Arithmetic

SS:	Stack Segment
STM	Store Multiple
STMB	Store Multiple Byte
STMW	Store Multiple Word
SUB	Subtract
SUB4S	Subtract Nibble String
SUBC	Subtract with Carry
TEST	Test
TEST1	Test Bit
TRANS	Translate
TRANSB	Translate Byte
XCH	Exchange
XOR	Exclusive-Or

Appendix B Index of μPD70108/70116 Instructions

(in Alphabetical order)

Instruction		page
TEST1	mem8,CL	12-173
	reg16,CL	12-174
	mem16,CL	12-175
	reg8,imm3	12-176
	mem8,imm3	12-177
	reg16,imm4	12-178
	mem16,imm4	12-179
TRANS	src-table	12-17
TRANSB		12-17
XCH	reg,reg	12-18
	mem,reg	12-19
	AW,reg16	12-20
XOR	reg,reg	12-165
	mem,reg	12-166
	reg,mem	12-167
	reg,imm	12-168
	mem,imm	12-169
	acc,imm	12-171
MOV	DS0, reg16,mem32	12-12
	DS1, reg16,mem32	12-13
	AH, PSW	12-14
	PSW, AH	12-15
PUSH	imm8	12-265

INSTRUCTION SET

μPD 70108

μPD 70116

Instruction Set

Table 1 Operation Types

Identifier	Description
reg	8- or 16-Bit general-purpose register
reg8	8-Bit general-purpose register
reg16	16-Bit general-purpose register
dmem	8- or 16-Bit direct memory location
mem	8- or 16-Bit memory location
mem8	8-Bit memory location
mem16	16-Bit memory location
mem32	32-Bit memory location
imm	Constant [0 to FFFFH]
imm16	Constant [0 to FFFFH]
imm8	Constant [0 to FFH]
imm4	Constant [0 to FH]
imm3	Constant [0 to 7]
acc	AW or AL register
sreg	Segment register
src-table	Name of 256-byte translation table
src-block	Name of block addressed by the IX register
dst-block	Name of block addressed by the IY register
near-proc	Procedure within the current program segment
far-proce	Procedure located in another program segment
near-label	Label in the current program segment
short-label	Between −128 and +127 bytes from the end of instruction
far-label	Label in another program segment
memptr16	Word containing the offset of the memory location within the current program segment to which control is to be transferred
memptr32	Double word containing the offset and segment base address of the memory location within the current program segment to which control is to be transferred
regptr16	16-Bit register containing the offset of the memory location within the program segment to which control is to be transferred
pop-value	Number of bytes of the stack to be discarded [0 to 64K bytes, usually odd addresses]
fp-op	Immediate data to identify the instruction code of the external floating point operation

Table 2 Operation Types (Cont'd)

Identifier	Description
R	Register set
W	Word/byte field [0 to 1]
reg	Register field [000 to 111]
mem	Memory field [000 to 111]
mod	Mode field [00 to 10]
S:W	When S:W = 01, data = 16 bits. At all other times, data = 8 bits. When S:W = 11, the sign of the byte data is expanded to make a 16-bit operand
X, XXX, YYY, ZZZ	Data to identify the instruction code of the external floating point arithmetic chip

Table 3

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	CW 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]
IY	Index register [destination] [16 bits]
PS	Program segment register [16 bits]
SS	Stack Segment Register [16 bits]
DS_0	Data segment 0 register [16 bits]
DS_1	Data segment 1 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
MD	Mode flag
[...]	Values in parentheses are memory contents

Table 4

Identifier	Description
disp	Displacement [8- or 16-bits]
ext-disp8	16-Bit displacement [sign-extension byte + 8-bit displacement]
temp	Temporary register [8-/16-/32-bits]
tmpcy	Temporary carry flag [1-bit]
seg	Immediate segment data [16-bits]
offset	Immediate offset data [16-bits]
<—	Transfer direction
+	Addition
−	Subtraction
x	Multiplication
÷	Division
%	Modulo
AND	Logical product
OR	Logical sum
XOR	Exclusive logical sum
XXH	Two-digit hexadecimal value
XXXXH	Four-digit hexadecimal value

Table 5 *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 6 *Memory Addressing*

mod/ mem	00	01	10
000	BW + IX	BW + IX +disp8	BW + IX + disp16
001	BW + IY	BW + IY + disp8	BW + IY + dips16
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 7

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 8 *Selection of Segment Registers*

sreg	
00	DS_1
01	PS
10	SS
11	DS_0

The tables on the following pages show the instruction set.

At "No. of Clocks", the figure on the left side of the slash (/) shows the clocks for a byte operation (W-bit = 0) and the figure on the right side shows the clocks for a word operation (W-bit = 1).

"No. of Clocks" includes these times:

- Decoding
- Effective address generation
- Operand fetch
- Execution

It assumes that the instruction bytes have been pre-fetched.

Data Transfer Instructions

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags (AC CY V P S Z)
MOV	reg, reg	1 0 0 0 1 0 1 W / 1 1 reg reg	2	2	reg ← reg	
	mem, reg	1 0 0 0 1 0 0 W / mod reg mem	2-4	9/13	(mem) ← reg	
	reg, mem	1 0 0 0 1 0 1 W / mod reg mem	2-4	11/15	reg ← (mem)	
	mem, imm	1 1 0 0 0 1 1 W / mod 0 0 0 mem	3-6	11/15	(mem) ← imm	
	reg, imm	1 0 1 1 W reg	2-3	4	reg ← imm	
	acc, dmem	1 0 1 0 0 0 0 W	3	10/14	When W = 0 AL ← (dmem) When W = 1 AH ← (dmem + 1), AL ← (dmem)	
	dmem, acc	1 0 1 0 0 0 1 W	3	9/13	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	2	sreg ← reg16 srseg : SS, DS0, DS1	
	sreg, mem16	1 0 0 0 1 1 1 0 / mod 0 sreg mem	2-4	11/15	sreg ← (mem16) sreg : SS, DS0, DS1	
	reg16, sreg	1 0 0 0 1 1 0 0 / 1 1 0 sreg reg	2	2	reg16 ← sreg	
	mem16, sreg	1 0 0 0 1 1 0 0 / mod 0 sreg mem	2-4	10/14	(mem16) ← sreg	
	DS0, reg16, mem32	1 1 0 0 0 1 0 1 / mod reg mem	2-4	18-26	reg16 ← (mem32) DS0 ← (mem32 + 2)	
	DS1, reg16, mem32	1 1 0 0 0 1 0 0 / mod reg mem	2-4	18/26	reg16 ← (mem32) DS1 ← (mem32 + 2)	
	AH, PSW	1 0 0 1 1 1 1 1	1	2	AH ← S, Z, x, AC, x, P, x, CY	
	PSW, AH	1 0 0 1 1 1 1 0	1	3	S, Z, x, AC, x, P, x, CY ← AH	x x x x x
LDEA	reg16, mem16	1 0 0 0 1 1 0 1 / mod reg mem	2-4	4	reg16 ← mem16	
TRANS	src-table	1 1 0 1 0 1 1 1	1	9	AL ← (BW + AL)	
XCH	reg, reg	1 0 0 0 0 1 1 W / 1 1 reg reg	2	3	reg ↔ reg	
	mem, reg reg, mem	1 0 0 0 0 1 1 W / mod reg mem	2-4	16/24	(mem) ↔ reg	
	AW, reg16 reg16, AW	1 0 0 1 0 reg	1	2	AW ↔ reg16	

Repeat Prefix

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags (AC CY V P S Z)
REPC		0 1 1 0 0 1 0 1	1	2	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	2	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≠0, exit the loop.	
REP REPE REPZ		1 1 1 1 0 0 1 1	1	2	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	2	While CW≠0, the next byte of the primitive block tansfer instruction is executed and CW is decremented (-1). If there is a waiting interrupt, it is processed. If the primitive block transfer instruction is CMPBX or CMPM and Z≠0, exit the loop.	

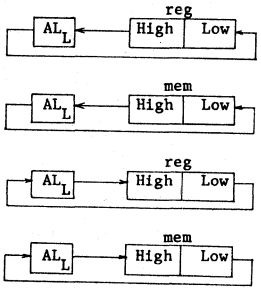
Primitive block transfer instructions

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags (AC CY V P S Z)
MOVBK	dst-block, src-block	1 0 1 0 0 1 0 W	1	See Table 7-8	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 (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	
CMPM	dst-block	1 0 1 0 1 1 1 W	1	See Table 7-9	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	

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags (AC CY V P S Z)
LDM	src-block	1 0 1 0 1 1 0 W	1	See Table 7-9	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 + 2; DIR = 1 : IX ← IX - 2	
STM	dst-block	1 0 1 0 1 0 1 W	1	"	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	
Bit field operation instructions						
INS	reg8, reg8	0 0 0 0 1 1 1 1 / 0 0 1 1 0 0 0 1 1 1 reg reg	3	67-87 /75-103	16bit field ← AW	
	reg8, imm4	0 0 0 0 1 1 1 1 / 0 0 1 1 1 0 0 1 1 1 0 0 0 reg	4	"	16bit field ← AW	
EXT	reg8, reg8	0 0 0 0 1 1 1 1 / 0 0 1 1 0 0 1 1 1 1 reg reg	3	21-44 /25-52	AW ← 16bit field	
	reg8, imm4	0 0 0 0 1 1 1 1 / 0 0 1 1 1 0 1 1 1 1 0 0 0 reg	4	"	AW ← 16bit field	
Input/Output instructions						
IN	acc, imm8	1 1 1 0 0 1 0 W	2	9/13	When W = 0 AL ← (imm8) When W = 1 AH ← (imm8 + 1), AL ← (imm8)	
	acc, DW	1 1 1 0 1 1 0 W	1	8/12	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	8/12	When W = 0 (imm8) ← AL When W = 1 (imm8 + 1) ← AH, (imm8) ← AL	
	DW, acc	1 1 1 0 1 1 1 W	1	8/12	When W = 0 (DW) ← AL When W = 1 (DW + 1) ← AH, (DW) ← AL	
Primitive Input/Output instructions						
INM	dst-block, DW	0 1 1 0 1 1 0 W	1	See Table 7-9	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	

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags AC	CY	V	P	S	Z
OUTM	DW, src-block	0 1 1 0 1 1 1 W	1	See Table 7-9	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						
Addition/subtraction instructions											
ADD	reg, reg	0 0 0 0 0 0 1 W 1 1 reg reg	2	2	reg ← reg + reg	x	x	x	x	x	x
	mem, reg	0 0 0 0 0 0 0 W mod reg mem	2-4	16/24	(mem) ← (mem) + reg	x	x	x	x	x	x
	reg, mem	0 0 0 0 0 0 1 W mod reg mem	2-4	11/15	reg ← reg + (mem)	x	x	x	x	x	x
	reg, imm	1 0 0 0 0 0 S W 1 1 0 0 0 reg	3-4	4	reg ← reg + imm	x	x	x	x	x	x
	mem, imm	1 0 0 0 0 0 S W mod 0 0 0 mem	3-6	18/26	(mem) ← (mem) + imm	x	x	x	x	x	x
	acc, imm	0 0 0 0 0 1 0 W	2-3	4	When W = 0 AL ← AL + imm When W = 1 AW ← AW + imm	x	x	x	x	x	x
ADDC	reg, reg	0 0 0 1 0 0 1 W 1 1 reg reg	2	2	reg ← reg + reg + CY	x	x	x	x	x	x
	mem, reg	0 0 0 1 0 0 0 W mod reg mem	2-4	16/24	(mem) ← (mem) + reg + CY	x	x	x	x	x	x
	reg, mem	0 0 0 1 0 0 1 W mod reg mem	2-4	11/15	reg ← reg + (mem) + CY	x	x	x	x	x	x
	reg, imm	1 0 0 0 0 0 S W 1 1 0 1 0 reg	3-4	4	reg ← reg + imm + CY	x	x	x	x	x	x
	mem, imm	1 0 0 0 0 0 S W mod 0 1 0 mem	3-6	18/26	(mem) ← (mem) + imm + CY	x	x	x	x	x	x
	acc, imm	0 0 0 1 0 1 0 W	2-3	4	When W = 0 AL ← AL + imm + CY When W = 1 AW ← AW + imm + CY	x	x	x	x	x	x
SUB	reg, reg	0 0 1 0 1 0 1 W 1 1 reg reg	2	2	reg ← reg - reg	x	x	x	x	x	x
	mem, reg	0 0 1 0 1 0 0 W mod reg mem	2-4	16/24	(mem) ← (mem) - reg	x	x	x	x	x	x
	reg, mem	0 0 1 0 1 0 1 W mod reg mem	2-4	11/15	reg ← reg - (mem)	x	x	x	x	x	x
	reg, imm	1 0 0 0 0 0 S W 1 1 1 0 1 reg	3-4	4	reg ← reg - imm	x	x	x	x	x	x
	mem, imm	1 0 0 0 0 0 S W mod 1 0 1 mem	3-6	18/26	(mem) ← (mem) - imm	x	x	x	x	x	x
	acc, imm	0 0 1 0 1 1 0 W	2-3	4	When W = 0 AL ← AL - imm When W = 1 AW ← AW - imm	x	x	x	x	x	x

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags AC	CY	V	P	S	Z
SUBC	reg, reg	0 0 0 1 1 0 1 W / 1 1 reg reg	2	2	reg ← reg - reg - CY	x	x	x	x	x	x
	mem, reg	0 0 0 1 1 0 0 W / mod reg mem	2-4	16/24	(mem) ← (mem) - reg - CY	x	x	x	x	x	x
	reg, mem	0 0 0 1 1 0 1 W / mod reg mem	2-4	11/15	reg ← reg - (mem) - CY	x	x	x	x	x	x
	reg, imm	1 0 0 0 0 0 S W / 1 1 0 1 1 reg	3-4	4	reg ← reg - imm - CY	x	x	x	x	x	x
	mem, imm	1 0 0 0 0 0 S W / mod 0 1 1 mem	3-6	18/26	(mem) ← (mem) - imm - CY	x	x	x	x	x	x
	acc, imm	0 0 0 1 1 1 0 W	2-3	4	When W = 0 AL ← AL - imm - CY When W = 1 AW ← AW - imm - CY	x	x	x	x	x	x
BCD operation instructions											
ADD4S		0 0 0 0 1 1 1 1 / 0 0 1 0 0 0 0 0	2	19 x n +7	dst BCD string ← dst BCD string + src BCD string	U	x	x	U	U	x
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	x	x	U	U	x
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	x	x	U	U	x
ROL4	reg8	0 0 0 0 1 1 1 1 / 0 0 1 0 1 0 0 0 1 1 0 0 0 reg	3	25							
	mem8	0 0 0 0 1 1 1 1 / 0 0 1 0 1 0 0 0 mod 0 0 0 mem	3-5	28							
ROR4	reg8	0 0 0 0 1 1 1 1 / 0 0 1 0 1 0 1 0 1 1 0 0 0 reg	3	29							
	mem8	0 0 0 0 1 1 1 1 / 0 0 1 0 1 0 1 0 mod 0 0 0 mem	3-5	33							
Increment/Decrement instructions											
INC	reg8	1 1 1 1 1 1 1 0 / 1 1 0 0 0 reg	2	2	reg8 ← reg8 + 1	x		x	x	x	x
	mem	1 1 1 1 1 1 1 W / mod 0 0 0 mem	2-4	16/24	(mem) ← (mem) + 1	x		x	x	x	x
	reg16	0 1 0 0 0 reg	1	2	reg16 ← reg16 + 1	x		x	x	x	x



n: one half the number of BCD digits

Mnemonic	Operand	Operation code	Bytes	Clocks	Operation	Flags AC	CY	V	P	S	Z
		7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0									
DEC	reg8	1 1 1 1 1 1 1 0 1 1 0 0 1 reg	2	2	reg8 ← reg8 - 1	x		x	x	x	x
	mem	1 1 1 1 1 1 1 W mod 0 0 1 mem	2-4	16/24	(mem) ← (mem) - 1	x		x	x	x	x
	reg16	0 1 0 0 1 reg	1	2	reg16 ← reg16 - 1	x		x	x	x	x
Miltiplication instructions											
MULU	reg8	1 1 1 1 0 1 1 0 1 1 1 0 0 reg	2	21-22	AW ← AL x reg8 AH = 0 : CY ← 0, V ← 0 AH ≠ 0 : CY ← 1, V ← 1	U	x	x	U	U	U
	mem8	1 1 1 1 0 1 1 0 mod 1 0 0 mem	2-4	27-28	AW ← AL x (mem8) AH = 0 : CY ← 0, V ← 0 AH ≠ 0 : CY ← 1, V ← 1	U	x	x	U	U	U
	reg16	1 1 1 1 0 1 1 1 1 1 1 0 0 reg	2	29-30	DW, AW ← AW x reg16 DW = 0 : CY ← 0, V ← 0 DW = 1 : CY ← 1, V ← 1	U	x	x	U	U	U
	mem16	1 1 1 1 0 1 1 1 mod 1 0 0 mem	2-4	35-36 /39-40	DW, AW ← AW x (mem16) DW = 0 : CY ← 0, V ← 0 DW = 1 : CY ← 1, V ← 1	U	x	x	U	U	U
MUL	reg8	1 1 1 1 0 1 1 0 1 1 1 0 1 reg	2	33-39	AW ← AL x reg8 AH = AL sign expansion : CY ← 0, V ← 0 AH ≠ AL sign expansion : CY ← 1, V ← 1	U	x	x	U	U	U
	mem8	1 1 1 1 0 1 1 0 mod 1 0 1 mem	2-4	39-45	AW ← AL x (mem8) AH = AL sign expansion : CY ← 0, V ← 0 AH ≠ AL sign expansion : CY ← 1, V ← 1	U	x	x	U	U	U
	reg16	1 1 1 1 0 1 1 1 1 1 1 0 1 reg	2	41-47	DW, AW ← AW x reg16 DW = AW sign expansion : CY ← 0, V ← 0 DW ≠ AW sign expansion : CY ← 1, V ← 1	U	x	x	U	U	U
	mem16	1 1 1 1 0 1 1 1 mod 1 0 1 mem	2-4	47-53 /53-57	DW, AW ← AW x (mem16) DW = AW sign expansion : CY ← 0, V ← 0 DW ≠ AW sign expansion : CY ← 1, V ← 1	U	x	x	U	U	U

Mnemonic	Operand	Operation code	Bytes	Clocks	Operation	Flags AC CY V P S Z
MUL	reg16, (reg16,) imm8	0 1 1 0 1 0 1 1 1 1 reg reg	3	28-34	reg16 ← reg16 x imm8 Product ≦ 16 bits : CY ← 0, V ← 0 Product > 16 bits : CY ← 1, V ← 1	U x x U U U
	reg16, mem16, imm8	0 1 1 0 1 0 1 1 mod reg mem	3-5	34-40 /38-44	reg16 ← (mem16) x imm8 Product ≦ 16 bits : CY ← 0, V ← 0 Product > 16 bits : CY ← 1, V ← 1	U x x U U U
	reg16, (reg16,) imm16	0 1 1 0 1 0 0 1 1 1 reg reg	4	36-42	reg16 ← reg16 x imm16 Product ≦ 16 bits : CY ← 0, V ← 0 Product > 16 bits : CY ← 1, V ← 1	U x x U U U
	reg16, mem16, imm16	0 1 1 0 1 0 0 1 mod reg mem	4-6	42-48 /46-52	reg16 ← (mem16) x imm16 Product ≦ 16 bits : CY ← 0, V ← 0 Product > 16 bits : CY ← 1, V ← 1	U x x U U U
Unsigned Divide instructions						
DIVU	reg8	1 1 1 1 0 1 1 0 1 1 1 1 1 0 reg	2	19	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	25	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 1 0 reg	2	25	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

Mnemonic	Operand	Operation code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	Bytes	Clocks	Operation	Flags AC CY V P S Z
DIVU	mem16	1 1 1 1 0 1 1 1 mod 1 1 0 mem	2-4	31/35	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
DIV	reg8	1 1 1 1 0 1 1 0 1 1 1 1 1 reg	2	29-34	temp ← AW When temp ÷ reg8 > 0 and temp ÷ reg8 ≦ 7FH or temp ÷ reg8 < 0 and temp ÷ reg8 > 0 - 7FH - 1 AH ← temp % reg8, AL ← temp ÷ reg8 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 ← (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	35-40	temp ← AW When temp ÷ (mem8) > 0 and temp ÷ (mem8) ≦ 7FH or temp ÷ (mem8) < 0 and temp ÷ (mem8) > 0 - 7FH - 1 AH ← temp % (mem8), AL ← temp ÷ (mem8) When temp ÷ (mem8) > 0 and temp ÷ (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 ← (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	38-43	temp ← DW, AW When temp ÷ reg16 > 0 and temp ÷ reg16 ≦ 7FFFH or temp ÷ reg16 < 0 and temp ÷ reg16 > 0 - 7FFFH - 1 DW ← temp % reg16, AW ← temp ÷ reg16 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 ← (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	44-49 /48-53	temp ← DW, AW When temp ÷ (mem16) > 0 and temp ÷ (mem16) ≦ 7FFFH or temp ÷ (mem16) < 0 and temp ÷ (mem16) > 0 - 7FFFH - 1 DW ← temp % (mem16), AW ← temp ÷ (mem16) 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 ← (3,2), PC ←(1,0)	U U U U U U

BCD Adjust instruction

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags AC CY V P S Z
ADJBA		0 0 1 1 0 1 1 1	1	3	When AL ∧ 0FH > 9 or AC = 1, AL ← AL + 6 AH ← AH + 1, AC ← 1, CY ← AC, AL ← AL ∧ 0FH	x x U U U U
ADJ4A		0 0 1 0 0 1 1 1	1	3	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	x x U x x x
ADJBS		0 0 1 1 1 1 1 1	1	7	When AL ∧ 0FH > 9 or AC = 1, AL ← AL - 6, AH ← AH - 1, AC ← 1, CY ← AC, AL ← AL 0FH	x x U U U U
ADJ4S		0 0 1 0 1 1 1 1	1	7	When AL < 0 FH > 9 or AC = 1, AL ← AL - 6, CY ← CY ∨ AC, AC ← 1 When AL > 9FH or CY = 1 AL ← AL - 60H, CY ← 1	x x U x x x

Convert instructions

CVTBD		1 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0	2	15	AH ← AL ÷ 0AH, AL ← AL % 0AH	U U U x x x
CVTDB		1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0	2	7	AH ← 0, AL ← AH x 0AH + AL	U U U x x x
CVTBW		1 0 0 1 1 0 0 0	1	2	When AL < 80H, AH ← 0, other wise AH ← FFH	
CVTWL		1 0 0 1 1 0 0 1	1	4-5	When AL < 8000H, DW ← 0, other wise DW ← FFFFH	

Compare instructions

CMP	reg, reg	0 0 1 1 1 0 1 W 1 1 reg reg	2	2	reg - reg	x x x x x x
	mem, reg	0 0 1 1 1 0 0 W mod reg mem	2-4	11/15	(mem) - reg	x x x x x x
	reg, mem	0 0 1 1 1 0 1 W mod reg mem	2-4	11/15	reg - (mem)	x x x x x x
	reg, imm	1 0 0 0 0 0 S W 1 1 1 1 1 reg	3-4	4	reg - imm	x x x x x x
	mem, imm	1 0 0 0 0 0 S W mod 1 1 1 mem	3-6	13/17	(mem) - imm	x x x x x x
	acc, imm	0 0 1 1 1 1 0 W	2-3	4	When W = 0, AL - imm When W = 1, AW - imm	x x x x x x

Complement instructions

Mnemonic	Operand	Operation code	Bytes	Clocks	Operation	Flags AC	CY	V	P	S	Z
NOT	reg	1 1 1 1 0 1 1 W 1 1 0 1 0 reg	2	2	reg ← $\overline{\text{reg}}$						
	mem	1 1 1 1 0 1 1 W mod 0 1 0 mem	2-4	16/24	(mem) ← ($\overline{\text{mem}}$)						
NEG	reg	1 1 1 1 0 1 1 W 1 1 0 1 1 reg	2	2	reg ← $\overline{\text{reg}}$ + 1	x	x	x	x	x	x
	mem	1 1 1 1 0 1 1 W mod 0 1 1 mem	2-4	16/24	(mem) ← ($\overline{\text{mem}}$) + 1	x	x	x	x	x	x

Logical operation instructions

TEST	reg, reg	1 0 0 0 0 1 0 W 1 1 reg reg	2	2	reg ∧ reg	U	0	0	x	x	x
	mem, reg reg, mem	1 0 0 0 0 1 0 W mod reg mem	2-4	10/14	(mem) ∧ reg	U	0	0	x	x	x
	reg, imm	1 1 1 1 0 1 1 W 1 1 0 0 0 reg	3-4	4	reg ∧ imm	U	0	0	x	x	x
	mem, imm	1 1 1 1 0 1 1 W mod 0 0 0 mem	3-6	11/15	(mem) ∧ imm	U	0	0	x	x	x
	acc, imm	1 0 1 0 1 0 0 W	2-3	4	When W = 0, AL ∧ imm8 When W = 1, AW ∧ imm8	U	0	0	x	x	x
AND	reg, reg	0 0 1 0 0 0 1 W 1 1 reg reg	2	2	reg ← reg ∧ reg	U	0	0	x	x	x
	mem, reg	0 0 1 0 0 0 0 W mod reg mem	2-4	16/24	(mem) ← (mem) ∧ reg	U	0	0	x	x	x
	reg, mem	0 0 1 0 0 0 1 W mod reg mem	2-4	11/15	reg ← reg ∧ (mem)	U	0	0	x	x	x
	reg, imm	1 0 0 0 0 0 0 W 1 1 1 0 0 reg	3-4	4	reg ← reg ∧ imm	U	0	0	x	x	x
	mem, imm	1 0 0 0 0 0 0 W mod 1 0 0 mem	3-6	18/26	(mem) ← (mem) ∧ imm	U	0	0	x	x	x
	acc, imm	0 0 1 0 0 1 0 W	2-3	4	When W = 0, AL ← AL ∧ imm8 When W = 1, AW ← AW ∧ imm16	U	0	0	x	x	x
OR	reg, reg	0 0 0 0 1 0 1 W 1 1 reg reg	2	2	reg ← reg ∨ reg	U	0	0	x	x	x
	mem, reg	0 0 0 0 1 0 0 W mod reg mem	2-4	16/24	(mem) ← (mem) ∨ reg	U	0	0	x	x	x
	reg, mem	0 0 0 0 1 0 1 W mod reg mem	2-4	11/15	reg ← reg ∨ (mem)	U	0	0	x	x	x

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags AC	CY	V	P	S	Z
OR	reg, imm	1 0 0 0 0 0 0 W / 1 1 0 0 1 reg	3-4	4	reg ← reg ∨ imm	U	0	0	x	x	x
	mem, imm	1 0 0 0 0 0 0 W / mod 0 0 1 mem	3-6	18/26	(mem) ← (mem) ∨ imm	U	0	0	x	x	x
	acc, imm	0 0 0 0 1 1 0 W	2-3	4	When W = 0, AL ← AL ∨ imm8 When W = 1, AW ← AW ∨ imm16	U	0	0	x	x	x
XOR	reg, reg	0 0 1 1 0 0 1 W / 1 1 reg reg	2	2	reg ← reg ∀ reg	U	0	0	x	x	x
	mem, reg	0 0 1 1 0 0 0 W / mod reg mem	2-4	16/24	(mem) ← (mem) ∀ reg	U	0	0	x	x	x
	reg, mem	0 0 1 1 0 0 1 W / mod reg mem	2-4	11/15	reg ← reg ∀ (mem)	U	0	0	x	x	x
	reg, imm	1 0 0 0 0 0 0 W / 1 1 1 1 0 reg	3-4	4	reg ← reg ∀ imm	U	0	0	x	x	x
	mem, imm	1 0 0 0 0 0 0 W / mod 1 1 0 mem	3-6	18/26	(mem) ← (mem) ∀ imm	U	0	0	x	x	x
	acc, imm	0 0 1 1 0 1 0 W	2-3	4	When W = 0, AL ← AL ∀ imm8 When W = 1, AW ← AW ∀ imm16	U	0	0	x	x	x
Bit operation instructions											
TEST1	reg8, CL	0 0 0 1 0 0 0 0 / 1 1 0 0 0 reg	3	3	bit No. CL of reg8 = 0 : Z ← 1 " = 1 : Z ← 0	U	0	0	U	U	x
	mem8, CL	0 0 0 1 0 0 0 0 / mod 0 0 0 mem	3-5	12	bit No. CL of (mem8) = 0 : Z ← 1 " = 1 : Z ← 0	U	0	0	U	U	x
	reg16, CL	0 0 0 1 0 0 0 1 / 1 1 0 0 0 reg	3	3	bit No. CL of reg16 = 0 : Z ← 1 " = 1 : Z ← 0	U	0	0	U	U	x
	mem16, CL	0 0 0 1 0 0 0 1 / mod 0 0 0 mem	3-5	12/16	bit No. CL of (mem16) = 0 : Z ← 1 " = 1 : Z ← 0	U	0	0	U	U	x
	reg8, imm3	0 0 0 1 1 0 0 0 / 1 1 0 0 0 reg	4	4	bit No. imm3 of reg8 = 0 : Z ← 1 " = 1 : Z ← 0	U	0	0	U	U	x
	mem8, imm3	0 0 0 1 1 0 0 0 / mod 0 0 0 mem	4-6	13	bit No. imm3 of (mem8) = 0 : Z ← 1 " = 1 : Z ← 0	U	0	0	U	U	x
	reg16, imm4	0 0 0 1 1 0 0 1 / 1 1 0 0 0 reg	4	4	bit No. imm4 of reg16 = 0 : Z ← 1 " = 1 : Z ← 0	U	0	0	U	U	x
	mem16, imm4	0 0 0 1 1 0 0 1 (2nd byte*) / mod 0 0 0 mem (3rd byte*)	4-6	13/17	bit No. imm4 of (mem16) = 0 : Z ← 1 " = 1 : Z ← 0	U	0	0	U	U	x

* : 1st byte = 0FH

Mnemonic	Operand	Operation code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	Bytes	Clocks	Operation	Flags AC CY V P S Z
NOT1	reg8, CL	0 0 0 1 0 1 1 0	1 1 0 0 0 reg	3	4	bit No. CL of reg8 ← $\overline{\text{bit No. CL of reg8}}$	
	mem8, CL	0 0 0 1 0 1 1 0	mod 0 0 0 mem	3-5	18	bit No. CL of (mem8) ← $\overline{\text{bit No. CL of (mem8)}}$	
	reg16, CL	0 0 0 1 0 1 1 1	1 1 0 0 0 reg	3	4	bit No. CL of reg16 ← $\overline{\text{bit No. CL of reg16}}$	
	mem16, CL	0 0 0 1 0 1 1 1	mod 0 0 0 mem	3-5	18/26	bit No. CL of (mem16) ← $\overline{\text{bit No. CL of (mem16)}}$	
	reg8, imm3	0 0 0 1 1 1 1 0	1 1 0 0 0 reg	4	5	bit No. imm3 of reg8 ← $\overline{\text{bit No. imm3 of reg8}}$	
	mem8, imm3	0 0 0 1 1 1 1 0	mod 0 0 0 mem	4-6	19	bit No. imm3 of (mem8) ← $\overline{\text{bit No. imm3 of (mem8)}}$	
	reg16, imm4	0 0 0 1 1 1 1 1	1 1 0 0 0 reg	4	5	bit No. imm4 of reg16 ← $\overline{\text{bit No. imm4 of reg16}}$	
	mem16, imm4	0 0 0 1 1 1 1 1	mod 0 0 0 mem	4-6	19/27	bit No. imm4 of (mem16) ← $\overline{\text{bit No. imm4 of (mem16)}}$	
		2nd byte*	3rd byte*	* : 1st byte = 0FH			
NOT1	CY	1 1 1 1 0 1 0 1		1	2	CY ← $\overline{\text{CY}}$	x
CLR1	reg8, CL	0 0 0 1 0 0 1 0	1 1 0 0 0 reg	3	5	bit No. CL of reg8 ← 0	
	mem8, CL	0 0 0 1 0 0 1 0	mod 0 0 0 mem	3-5	14	bit No. CL of (mem8) ← 0	
	reg16, CL	0 0 0 1 0 0 1 1	1 1 0 0 0 reg	3	5	bit No. CL of reg16 ← 0	
	mem16, CL	0 0 0 1 0 0 1 1	mod 0 0 0 mem	3-5	14/22	bit No. CL of (mem16) ← 0	
	reg8, imm3	0 0 0 1 1 0 1 0	1 1 0 0 0 reg	4	6	bit No. imm3 of reg8 ← 0	
	mem8, imm3	0 0 0 1 1 0 1 0	mod 0 0 0 mem	4-6	15	bit No. imm3 of (mem8) ← 0	
	reg16, imm4	0 0 0 1 1 0 1 1	1 1 0 0 0 reg	4	6	bit No. imm4 of reg16 ← 0	
	mem16, imm4	0 0 0 1 1 0 1 1	mod 0 0 0 mem	4-6	15/27	bit No. imm4 of (mem16) ← 0	
SET1	reg8, CL	0 0 0 1 0 1 0 0	1 1 0 0 0 reg	3	4	bit No. CL of reg8 ← 1	
	mem8, CL	0 0 0 1 0 1 0 0	mod 0 0 0 mem	3-5	13	bit No. CL of (mem8) ← 1	
	reg16, CL	0 0 0 1 0 1 0 1	1 1 0 0 0 reg	3	4	bit No. CL of reg16 ← 1	
	mem16, CL	0 0 0 1 0 1 0 1	mod 0 0 0 mem	3-5	13/21	bit No. CL of (mem16) ← 1	
	reg8, imm3	0 0 0 1 1 1 0 0	1 1 0 0 0 reg	4	5	bit No. imm3 of reg8 ← 1	
		2nd byte*	3rd byte*	* 1st byte = 0FH			

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags AC CY V P S Z
SET1	mem8, imm3	0 0 0 1 1 1 0 0 / mod 0 0 0 mem	4-6	14	bit No. imm3 of (mem8) ← 1	
	reg16, imm4	0 0 0 1 1 1 0 1 / 1 1 0 0 0 reg	4	5	bit No. imm4 of reg16 ← 1	
	mem16, imm4	0 0 0 1 1 1 0 1 (2nd byte*) / mod 0 0 0 mem (3rd byte*)	4-6	14/22	bit No. imm4 of (mem16) ← 1	
		* : 1st byte = 0FH				
CLR1	CY	1 1 1 1 1 0 0 0	1	2	CY ← 0	0
	DIR	1 1 1 1 1 1 0 0	1	2	DIR ← 0	
SET1	CY	1 1 1 1 1 0 0 1	1	2	CY ← 1	1
	DIR	1 1 1 1 1 1 0 1	1	2	DIR ← 1	
Shift instructions						
SHL	reg, 1	1 1 0 1 0 0 0 W / 1 1 1 0 0 reg	2	2	CY ← MSB of reg, reg ← reg x 2 When MSB of reg ≠ CY V ← 1 When MSB of reg = CY V ← 0	U x x x x x
	mem, 1	1 1 0 1 0 0 0 W / mod 1 0 0 mem	2-4	16/24	CY ← MSB of (mem), (mem) ← (mem) x 2 When MSB of (mem) ≠ CY V ← 1 When MSB of (mem) = CY V ← 0	U x x x x x
	reg, CL	1 1 0 1 0 0 1 W / 1 1 1 0 0 reg	2	7 + n	temp ← CL, while temp ≠ 0, repeat this operation CY ← MSB of reg, reg ← reg x 2 temp ← temp - 1	U x U x x x
	mem, CL	1 1 0 1 0 0 1 W / mod 1 0 0 mem	2-4	19/27 +n	temp ← CL, while temp ≠ 0, repeat this operation CY ← MSB of (mem), (mem) ← (mem) x 2 temp ← temp - 1	U x U x x x
	reg, imm8	1 1 0 0 0 0 0 W / 1 1 1 0 0 reg	3	7 + n	temp ← imm8, while temp ≠ 0, repeat this operation CY ← MSB of reg, reg ← reg x 2 temp ← temp - 1	U x U x x x

n: number of shifts

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags AC	CY	V	P	S	Z
SHL	mem, imm8	1 1 0 0 0 0 0 W mod 1 0 0 mem	3-5	19/27 +n	temp ← imm8, while temp ≠ 0, repeat this operation CY ← MSB of (mem), (mem) ← (mem) x 2 temp ← temp - 1	U	x	U	x	x	x
SHR	reg, 1	1 1 0 1 0 0 0 W 1 1 1 0 1 reg	2	2	CY ← MSG 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 ← 1	U	x	x	x	x	x
	mem, 1	1 1 0 1 0 0 0 W mod 1 0 1 mem	2-4	16/24	CY ← MSG 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 ← 1	U	x	x	x	x	x
	reg, CL	1 1 0 1 0 0 1 W 1 1 1 0 1 reg	2	7 + n	temp ← CL, while temp ≠ 0, repeat this operation CY ← MSB of reg, reg ← reg + 2 temp ← temp - 1	U	x	U	x	x	x
	mem, CL	1 1 0 1 0 0 1 W mod 1 0 1 mem	2-4	19/27 +n	temp ← CL, while temp ≠ 0, repeat this operation CY ← MSB of (mem), (mem) ← (mem) + 2 temp ← temp - 1	U	x	U	x	x	x
	reg, imm8	1 1 0 0 0 0 0 W 1 1 1 0 1 reg	3	7 + n	temp ← imm8, while temp ≠ 0, repeat this operation CY ← MSB of reg, reg ← reg + 2 temp ← temp - 1	U	x	U	x	x	x
	mem, imm8	1 1 0 0 0 0 0 W mod 1 0 1 mem	3-5	19/27 +n	temp ← imm8, while temp ≠ 0, repeat this operation CY ← MSB of (mem), (mwm) ← (mem) + 2 temp ← temp - 1	U	x	U	x	x	x
SHRA	reg, 1	1 1 0 1 0 0 0 W 1 1 1 1 1 regg	2	2	CY ← LSB of reg, reg ← reg + 2, V ← 0 MSB of operand does not change.	U	x	0	x	x	x

n: number of shifts

Mnemonic	Operand	Operation code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	Bytes	Clocks	Operation	Flags AC CY V P S Z
SHRA	mem, 1	1 1 0 1 0 0 0 W mod 1 1 1 mem	2-4	16/24	CY ← LSB of (mem), (mem) ← (mem) + 2, V ← 0 MSB of operand does not change.	U x 0 x x x
	reg, CL	1 1 0 1 0 0 1 W 1 1 1 1 1 reg	2	7 + n	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 x U x x x
	mem, CL	1 1 0 1 0 0 1 W mod 1 1 1 mem	2-4	19/27 +n	temp ← CL, while temp ≠ 0, repeat this operation. CL ← LSB of (mem), (mem) ← (mem) + 2 temp ← temp - 1, MSB of operand does not change.	U x U x x x
	reg, imm8	1 1 0 0 0 0 0 W 1 1 1 1 1 reg	3	7 + n	temp ← imm8, while temp ≠ 0, repeat this operation. CY ← LSB of reg, reg ← reg + 2 temp ← temp - 1, MSB of operand does not change.	U x U x x x
	mem, imm8	1 1 0 0 0 0 0 W mod 1 1 1 mem	3-5	19/27 +n	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 x U x x x
Rotate instruction						
ROL	reg, 1	1 1 0 1 0 0 0 W 1 1 0 0 0 reg	2	2	CY ← MSB of reg, reg ← reg x 2 - CY MSB of reg ≠ CY: V ← 1 MSB of reg = CY: V ← 0	x x
	mem, 1	1 1 0 1 0 0 0 W mod 0 0 0 mem	2-4	16/24	CY ← MSB of (mem), (mem) ← (mem) x 2 - CY MSB of (mem) ≠ CY: V ← 1 MSB of (mem) = CY: V ← 0	x x
	reg, CL	1 1 0 1 0 0 1 W 1 1 0 0 0 reg	2	7 + n	temp ← CL, while temp ≠ 0, repeat this operation. CY ← MSB of reg, reg ← reg x 2 - CY temp ← temp - 1	x U
	mem, CL	1 1 0 1 0 0 1 W mod 0 0 0 mem	2-4	19/27 +n	temp ← CL, while temp ≠ 0, repeat this operation. CY ← MSB of (mem), (mem) ← (mem) x 2 - CY temp ← temp - 1	x U
	reg, imm8	1 1 0 0 0 0 0 W 1 1 0 0 0 reg	3	7 + n	temp ← imm8, while temp ≠ 0, repeat this operation. CY ← MSB of reg, reg ← reg x 2 - CY temp ← temp - 1	x U
	mem, imm8	1 1 0 0 0 0 0 W mod 0 0 0 mem	3-5	19/27 +n	temp ← imm8, while temp ≠ 0, repeat this operation. CY ← MSB of (mem), (mem) ← (mem) x 2 - CY temp ← temp - 1	x U

n: number of shifts

Mnemonic	Operand	Operation code	Bytes	Clocks	Operation	Flags AC	CY	V	P	S	Z
ROR	reg, 1	1 1 0 1 0 0 0 W 1 1 0 0 1 reg	2	2	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 ← 1		x	x			
	mem, 1	1 1 0 1 0 0 0 W mod 0 0 1 mem	2-4	16/24	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 ← 1		x	x			
	reg, CL	1 1 0 1 0 0 1 W 1 1 0 0 1 reg	2	7 + n	temp ← CL, while CL ≠ 0, repeat this operation. CY ← LSB Of reg, reg ← reg + 2 MSB of reg ← CY temp ← temp - 1		x	U			
	mem, CL	1 1 0 1 0 0 1 W mod 0 0 1 mem	2-4	19/27 +n	temp ← CL, while CL ≠ 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) + 2 MSB of (mem) ← CY temp ← temp - 1		x	U			
	reg, imm8	1 1 0 0 0 0 0 W 1 1 0 0 1 reg	3	7 + n	temp ← imm8, while CL ≠ 0, repeat this operation. CY ← LSB of reg, reg ← reg + 2 MSB of reg ← CY temp ← temp - 1		x	U			
	mem, imm8	1 1 0 0 0 0 0 W mod 0 0 1 mem	3-5	19/27 +n	temp ← imm8, while CL ≠ 0, repeat this operation. CY ← LSB of (mem), (mem) ← (mem) + 2 temp ← temp - 1		x	U			
ROLC	reg, 1	1 1 0 1 0 0 0 W 1 1 0 1 0 reg	2	2	tmpcy ← CY, CY ← MSB of reg reg ← reg x 2 + tmpcy MSB of reg = CY: V ← 1 MSB of reg ≠ CY: V ← 0		x	x			
	mem, 1	1 1 0 1 0 0 0 W mod 0 1 0 mem	2-4	16/24	tmpcy ← CY, CY ← MSB of (mem) (mem) ← (mem) x 2 + tmpcy MSB of (mem) = CY: V ← 1 MSB of (mem) = CY: V ← 0		x	x			

n: number of shifts

Mnemonic	Operand	Operation code (7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0)	Bytes	Clocks	Operation	Flags AC CY V P S Z
ROLC	reg, CL	1 1 0 1 0 0 1 W 1 1 0 1 0 reg	2	7 + n	temp ← CL, while CY ≠ 0, repeat this operation tmpcy ← CY, CY ← MSB of reg reg ← reg x 2 + tmpcy temp ← temp - 1	x U
	mem, CL	1 1 0 1 0 0 1 W mod 0 1 0 mem	2-4	19/27 +n	temp ← CL, while CY ≠ 0, repeat this operation tmpcy ← CY, CY ← MSB of (mem) (mem) ← (mem) x 2 + tmpcy temp ← temp - 1	x U
	reg, imm8	1 1 0 0 0 0 0 W 1 1 0 1 0 reg	3	7 + n	temp ← imm8, while CL ≠ 0, repeat this operation tmpcy ← CY, CY ← MSB of reg reg ← reg x 2 + tmpcy temp ← temp - 1	x U
	mem, imm8	1 1 0 0 0 0 0 W mod 0 1 0 mem	3-5	19/27 +n	temp ← imm8, while CL ≠ 0, repeat this operation tmpcy ← CY, CY ← MSB of (mem) (mem) ← (mem) x 2 + tmpcy temp ← temp - 1	x U
		n : number of shifts				
RORC	reg, 1	1 1 0 1 0 0 0 W 1 1 0 1 1 reg	2	2	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	x x
	mem, 1	1 1 0 1 0 0 0 W mod 0 1 1 mem	2-4	16/24	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	x x
	reg, CL	1 1 0 1 0 0 1 W 1 1 0 1 1 reg	2	7 + n	temp ← CL, while CL = 0, repeat this operation tmpcy ← CY, CY ← LSB of reg reg ← reg ÷ 2 MSB of reg ← tmpcy temp ← temp - 1	x U
		n: number of shifts				

Mnemonic	Operand	Operation code	Bytes	Clocks	Operation	Flags AC CY V P S Z
RORC	mem, CL	1 1 0 1 0 0 1 W mod 0 1 1 mem	2-4	19/27 +n	temp ← CL, while CL ≠ 0, repeat this operation tmpcy ← CY, CY ← LSB of (mem) reg ← (mem) ÷ 2 MSB of (mem) ← tmpcy temp ← temp - 1	x U
	reg, imm8	1 1 0 0 0 0 0 W 1 1 0 1 1 reg	3	7 + n	temp ← imm8, while CL ≠ 0 repeat this operation tmpcy ← CY, CY ← LSB of reg reg ← reg ÷ 2 MSB of reg ← tmpcy temp ← temp - 1	x U
	mem, imm8	1 1 0 0 0 0 0 W mod 0 1 1 mem	3-5	19/27 +n	temp ← imm8, while CL ≠ 0 repeat this operation tmpcy ← CY, CY ← LSB of (mem) (mem) ← (mem) ÷ 2 MSB of (mem) ← tmpcy temp ← temp - 1	x U

n : number of shifts

Subroutine control instructions

CALL	near-proc	1 1 1 0 1 0 0 0	3	16/20	(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	14/18	(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	23/31	(SP - 1, SP - 2) ← PC, SP ← SP - 2 PC ← (memptr16)
	far-proc	1 0 0 1 1 0 1 0	5	21/29	(SP - 1, SP - 2) ← PC, (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	31/47	(SP - 1, SP - 2) ← PC, (SP - 3, SP - 4) ← PC SP ← SP - 4 PS ← (memptr32 + 2), PC ← (memptr32)
RET		1 1 0 0 0 0 1 1	1	15/19	PC ← (SP + 1, SP) SP ← SP + 2
	pop-value	1 1 0 0 0 0 1 0	3	20/24	PC ← (SP + 1, SP) SP ← SP + 2, SP ← SP + pop-value
		1 1 0 0 1 0 1 1	1	21/29	PC ← (SP + 1, SP) PS ← (SP + 3, SP + 2) SP ← SP + 4

Mnemonic	Operand	Operation code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	Bytes	Clocks	Operation	Flags AC CY V P S Z
	pop-value	1 1 0 0 1 0 1 0	3	24/32	PC ← (SP + 1, SP) PS ← (SP + 3, SP + 2) SP ← SP + 4, SP ← SP + pop-value	

Stack operation instructions

PUSH	mem16	1 1 1 1 1 1 1 1 mod 1 1 0 mem	2-4	18/26	(SP - 1, SP - 2) ← (mem16) SP ← SP - 2	
	reg16	0 1 0 1 0 reg	1	8/12	(SP - 1, SP - 2) ← reg16 SP ← SP - 2	
	sreg	0 0 0 sreg1 1 0	1	8/12	(SP - 1, SP - 2) ← sreg SP ← SP - 2	
	PSW	1 0 0 1 1 1 0 0	1	8/12	(SP - 1, SP - 2) ← PSW SP ← SP - 2	
	R	0 1 1 0 0 0 0 0	1	35/67	Push registers on the stack	
	imm	0 1 1 0 1 0 S 0	2-3	7/11 or 8/12	(SP - 1, SP - 2) ← imm SP ← SP - 2, When s = 1, sign extension	
POP	mem16	1 0 0 0 1 1 1 1 mod 0 0 0 mem	2-4	17/25	(mem16) ← (SP + 1, SP) SP ← SP + 2	
	reg16	0 1 0 1 1 reg	1	8/12	reg16 ← (SP + 1, SP) SP ← SP + 2	
	sreg	0 0 0 sreg1 1 1	1	8/12	sreg ← (SP + 1, SP) sreg : SS,DS0,DS1 SP ← SP + 2	
	PSW	1 0 0 1 1 1 0 1	1	8/12	PSW ← (SP + 1, SP) SP ← SP + 2	R R R R R
	R	0 1 1 0 0 0 0 1	1	43/75	Pop regsters 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	6/10	Dispose of Stack Frame	

* : When imm8 = 0 : 13
When imm8 1 : 22 + 20 (imm8 - 1) : odd address
18 + 12 (imm8 - 1) : even address

Branch instructions

Mnemonic	Operand	Operation code	Bytes	Clocks	Operation	Flags AC CY V P S Z
BR	near-label	1 1 1 0 1 0 0 1	3	13	PC ← PC + disp	
	short-label	1 1 1 0 1 0 1 1	2	12	PC ← PC + ext-disp8	
	regptr16	1 1 1 1 1 1 1 1 | 1 1 1 0 0 | reg	2	11	PC ← regptr16	
	memptr16	1 1 1 1 1 1 1 1 | mod 1 0 0 | mem	2-4	20/24	PC ← (memptr16)	
	far-label	1 1 1 0 1 0 1 0	5	15	PS ← seg PC ← offset	
	memptr32	1 1 1 1 1 1 1 1 | mod 1 0 1 | mem	2-4	27/35	PS ← (memptr32 + 2) PC ← (memptr32)	

Conditional branch instructions

BV	short-label	0 1 1 1 0 0 0 0	2	14/4	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 "	

Mnemonic	Operand	Operation code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	Bytes	Clocks	Operation	Flags AC CY V P S Z
BGE	short-label	0 1 1 1 1 1 0 1	2	14/4	if S ∀ V = 0 PC ← PC + ext-disp8	
BLE	"	0 1 1 1 1 1 1 0	"	"	if (S ∀ V) ∨ Z = 1 "	
BGT	"	0 1 1 1 1 1 1 1	"	"	if (S ∀ V) ∨ Z = 0 "	
DBNZNE	"	1 1 1 0 0 0 0 0	"	14/5	CW = CW - 1 if Z = 0 and CW ≠ 0 "	
DBNZE	"	1 1 1 0 0 0 0 1	"	"	CW = CW - 1 if Z = 1 and CW ≠ 0 "	
DBNZ	"	1 1 1 0 0 0 1 0	"	13/5	CW = CW - 1 if CW ≠ 0 "	
BCWZ	"	1 1 1 0 0 0 1 1	"	"	if CW = 0 "	
Interrupt instructions						
BRK	3	1 1 0 0 1 1 0 0	1	38/58	(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	38/58	(SP - 1, SP - 2) ← PSW, (SP - 3, SP - 4) ← PS (SP - 5, SP - 6) ← PC, SP ← SP - 1 IE ← 0, BRK ← 0 PS ← (n x 4 + 3, n x 4 + 2), PC ← (n x 4 + 1, n x 4) n = imm8	
BRKV		1 1 0 0 1 1 1 0	1	40/60 /3	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	27/39	PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2), PSW ← (SP + 5, SP + 4), SP ← SP + 6	R R R R R R
BRKEM	imm8	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1	3	38/58	(SP - 1, SP - 2) ← PSW, (SP - 3, SP - 4) ← PS, (SP - 5, SP - 6) ← PC, SP ← SP - 6 MD ← 0 PS ← (n x 4 + 3, n x 4 + 2), PC ← (n x 4 + 1, n x 4) n = imm8	

Mnemonic	Operand	Operation code	Bytes	Clocks	Operation	Flags AC CY V P S Z
CHKIND	reg16, mem32	0 1 1 0 0 0 1 0 mod reg mem	2-4	53-56 /81-84 18/26	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)	
CPU control instructions						
HALT		1 1 1 1 0 1 0 0	1	2	CPU Halt	
POLL		1 0 0 1 1 0 1 1	1	2 + 5n	Poll and wait n : $\overline{\text{POLL}}$ pin sampling number	
DI		1 1 1 1 1 0 1 0	1	2	IE ← 0	
EI		1 1 1 1 1 0 1 1	1	2	IE ← 1	
BUSLOCK		1 1 1 1 0 0 0 0	1	2	Bus Lock Prefix	
FPO1	fp-op	1 1 0 1 1 X X X 1 1 Y Y Y Z Z Z	2	2	No Operation	
	fp-op, mem	1 1 0 1 1 X X X mod Y Y Y mem	2-4	11/15	data bus ← (mem)	
FPO2	fp-op	0 1 1 0 0 1 1 X 1 1 Y Y Y Z Z Z	2	2	No Operation	
	fp-op, mem	0 1 1 0 0 1 1 X mod Y Y Y mem	2-4	11/15	data bus ← (mem)	
NOP		1 0 0 1 0 0 0 0	1	3	No Operation	
*		0 0 1 sreg1 1 0	1	2	Segment override prefix	
* : DS0:, DS1:, PS:, SS:						
8080						
RETEM		1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1	2	27/39	PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2), PSW ← (SP + 5, SP + 4), SP ← SP + 6	R R R R R R
CALLN	imm8	1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1	3	38/58	(SP - 1, SP - 2) ← PSW, (SP - 3, SP - 4) ← PS, (SP - 5, SP - 6) ← PC, SP ← SP - 6 MD ← 1 PS ← (n x 4 + 3, n x 4 + 2), PC ← (n x 4 + 1, n x 4)	

NEC OFFICES

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

NEC Electronics (Germany) GmbH, Oberrather Str. 4, D-4000 Düsseldorf 30, Tel. (0211) 650302, Telex 858996-0
- Hindenburgstr. 28/29, D-3000 Hannover 1, Tel. (0511) 881013–16, Telex 9230109
- Arabellastr. 17, D-8000 München 81, Tel. (089) 4160020, Telex 522971
- Heilbronner Str. 314, D-7000 Stuttgart 30, Tel. (0711) 890910, Telex 7252220

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

NEC Electronics (Scandinavia) – Box 4039, S-18304 Täby, Tel. (08) 756 7245, Telex 13839

NEC Electronics (France) S.A., Tour Chenonceaux, 204, Rond Point du Pont de Sèvres, F-92516 Boulogne Billancourt, Tel. (01) 609 90 04, Telex 203544

NEC Electronics Italiana s.r.l., Via Cardano 3, I-20124 Milano, Tel. (02) 670 9108, Telex 315355

NEC Electronics (UK) Ltd., Block 3 Carfin Industrial Estate, Motherwell ML1 4UL, Scotland, Tel. (0698) 732221, Telex 777565