DATA SHEET

NEC

MOS INTEGRATED CIRCUIT

μPD70108

V20™
16-/8-BIT MICROPROCESSOR

a

The μPD70108 (V20) is a CMOS 16-/8-bit microprocessor. It has a 16-bit architecture and is equipped with a 8-bit data bus. The μPD70108 has a powerful instruction set which includes bit processing and packed BCD operation and high speed multiplication/division instructions, etc. and contains an 8080 emulation function. Further, the μPD70108 contains a standby function which can greatly lower its power consumption. The μPD70108 is software compatible with the 16-bit microprocessor μPD70116 (V30™).

Its functions are described in details in the manual indicated below. Please read this manual before starting design.

- **V20, V30 User's Manual Hardware: IEM-871**
- **16-bit V Series User's Manual Instruction: IEU-804**

FEATURES

- Memory addressing space: 1 M bytes
- Minimum instruction execution time:
  : 400 ns (5 MHz, 5 V; 70108-5)
  250 ns (8 MHz, 5 V; 70108-8)
  200 ns (10 MHz, 5 V; 70108-10)
- High-speed multiplication/division instruction:
  : 3.8 to 11.4 μs (5 MHz, 5 V; 70108-5)
  2.4 to 7.1 μs (8 MHz, 5 V; 70108-8)
  1.9 to 5.7 μs (10 MHz, 5 V; 70108-10)
- High-speed block transfer instruction:
  : 625 K words/second (5 MHz, 5 V; 70108-5)
  1 M words/second (8 MHz, 5 V; 70108-8)
  1.25 M words/second (10 MHz, 5 V; 70108-10)
- Following microprocessors are offered as a dedicated clock pulse generator/driver.
  - μPD71084 : for μPD70108-5 and -8
  - μPD71011 : for μPD70108-5 and -8
  - μPD71011-10 : for μPD70108-10

Document No. IC-1827B
(O.D. No. IC-6659D)
Date Published June 1994 P
Printed in Japan

The mark ★ shows revised points.

★ ORDERING INFORMATION

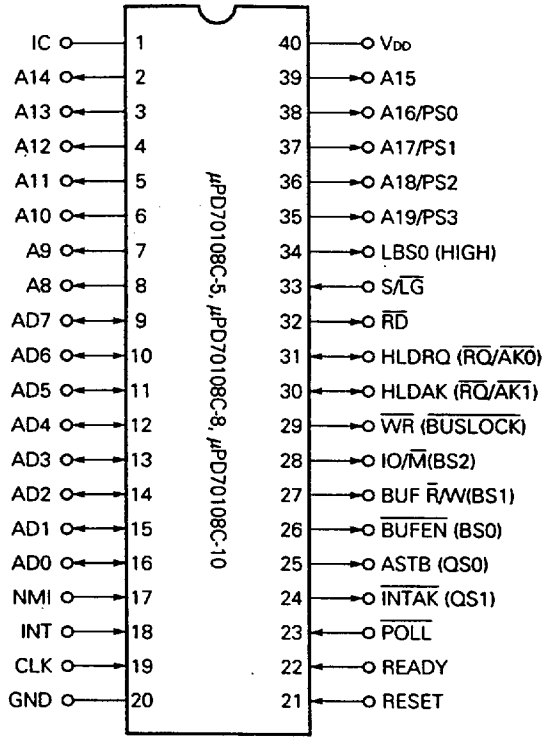
Part Number	Package	Max. operation freq.(MHz)
μPD70108C-5	40-pin plastic DIP (600 mil)	5
μPD70108C-8	40-pin plastic DIP (600 mil)	8
μPD70108C-10	40-pin plastic DIP (600 mil)	10
μPD70108GC-5-3B6	52-pin plastic QFP (□ 14 mm)	5
μPD70108GC-8-3B6	52-pin plastic QFP (□ 14 mm)	8
μPD70108GC-10-3B6	52-pin plastic QFP (□ 14 mm)	10
μPD70108L-5	44-pin plastic QFJ (□ 650 mil)	5
μPD70108L-8	44-pin plastic QFJ (□ 650 mil)	8
μPD70108L-10	44-pin plastic QFJ (□ 650 mil)	10

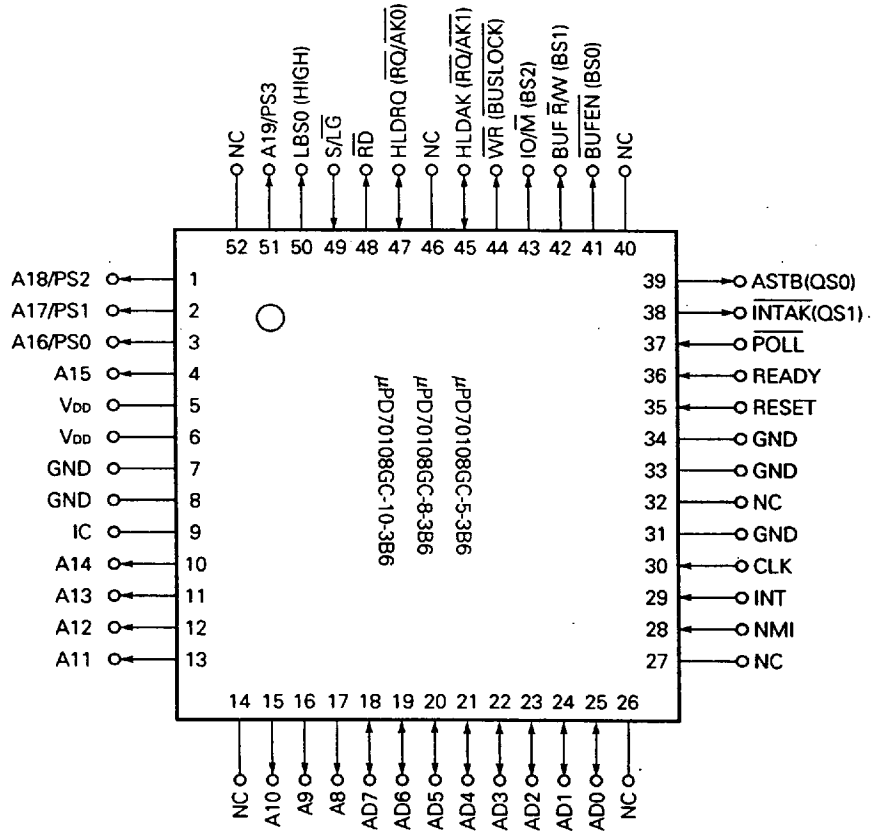
QUALITY GRADE

Standard

Please refer to "Quality grade on NEC Semiconductor Devices" (Document number IEI-1209) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

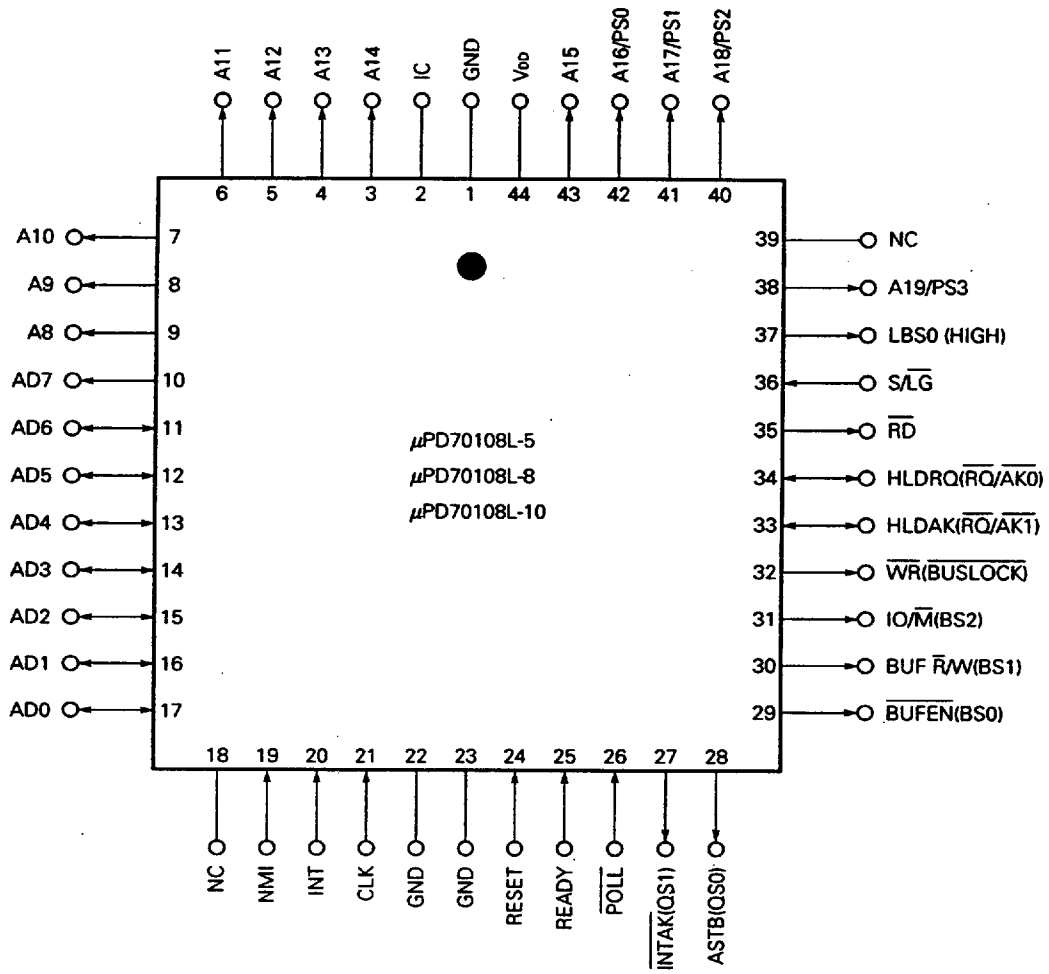
PIN CONFIGURATION (Top View)





NC : No Connection

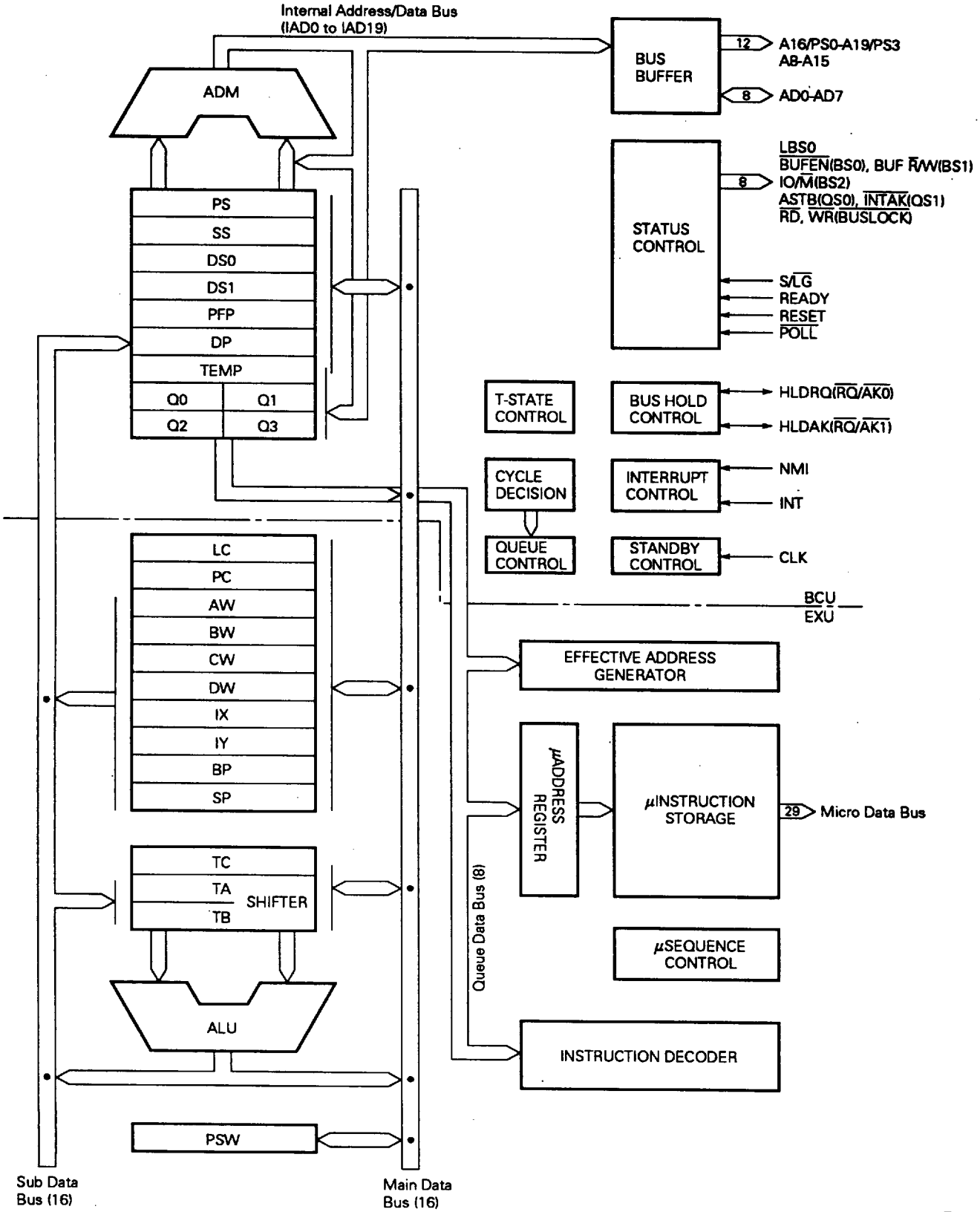
IC : Internally Connected (Connect to GND.)



NC : No Connection

IC : Internally Connected (Connect to GND.)

μPD70108 BLOCK DIAGRAM



CONTENTS

1. PIN FUNCTIONS

There are some pins which work for either a small system or a large system, and others for both small and large systems.

(1) A15 to A8 (Address Bus) ... small/large

These pins output the middle 8 bits of the 20-bit address information. They are 3-state outputs and become high-impedance during hold-acknowledge.

(2) AD7 to AD0 (Address/Data Bus) ... small/large

These are buses for both address and data. They output the lower 8 bits of 20-bit address information and input/output data using the time-division method.
The 16-bit data input/output is divided into two times. The 1st byte is lower and the 2nd byte is higher.
These pins are 3-state inputs/outputs and they become high-impedance during hold-acknowledge and interruption.

(3) NMI (Non-Maskable Interrupt) ... small/large

This is an interrupt request input which is non-maskable by software.
This input is active at the rising edge, and it can be detected at any clock cycle. The actual interrupt servicing begins after the completion of executing instruction.
Interrupt start address for the above interrupt is decided by the interrupt vector 2.
After the rising edge, NMI signal must be kept at the high level of the minimum 5 clock cycles.
Its priority is shown below. Hold request can be accepted during the NMI acknowledge.

INT < NMI < HLDRQ (small) or $\overline{RQ}$ (large)

This interrupt can be used for the release of a standby mode.

(4) INT (Maskable Interrupt) ... small/large

This is an interrupt request input which is maskable by software.
This input is active at the high level and can be detected at the last clock cycle of an instruction, then accepted if this input is interrupt enable status (if interrupt enable flag IE is set). The external device checks if the INT interrupt request has been accepted or not by $\overline{INTAK}$ signal output from the CPU.
INT signal must be kept at a high level until the first $\overline{INTAK}$ signal is output.
The priority is shown below. If a NMI arises simultaneously, the NMI takes priority over the INT.
Hold request can be accepted during the INT acknowledge.

INT < NMI < HLDRQ (small) or $\overline{RQ}$ (large)

This interrupt can be used for the release of a standby mode.

(5) CLK (Clock) ... small/large

This is an external clock input.

(6) **RESET (Reset) ... small/large**

This is a CPU reset input which is active at the high level. It takes priority over all operations. After RESET is released, the CPU starts a program from FFFF0H.
RESET input is used not only for usual CPU start, but also for the release of a standby mode.

(7) **READY (Ready) ... small/large**

When memory or I/O cannot end the read/write operation within the basic access time of the CPU, this signal is requested to be inactivated (at the low level) to generate wait state (TW) in the CPU, and to extend the read/write cycle.
If the READY signal is active (at the high level) at T3 or TW state, the CPU won't generate any wait state.
Since this signal cannot guarantee correct operation unless it satisfies setup/hold time, it should be synchronized with an external device.

(8) **$\overline{\text{POLL}}$ (Poll) ... small/large**

$\overline{\text{POLL}}$ input is checked by a POLL instruction. If the signal is at the low level, the next instruction is executed. If it is at the high level, $\overline{\text{POLL}}$ input is checked every 5-clock cycle which continues until the signal is at the low level.
These functions are utilized to synchronize the CPU's program with external device operations.

(9) **$\overline{\text{INTAK}}$ (Interrupt Acknowledge) ... small**

This pin outputs when it receives INT signal. An external device inputs the interrupt vector in synchronization with this signal to the CPU through data buses (AD7 to AD0).
This output is fixed at the high level in a standby mode.

(10) **ASTB (Address Strobe) ... small**

This is a strobe signal which is output to latch address information into an external latch.
Once this output gets at the high level (for about 1/2 clock) in a standby mode, then it is fixed at the low level.

(11) **$\overline{\text{BUFEN}}$ (Buffer Enable) ... small**

This is a signal used as an output enable signal of external bi-directional buffers. It is output when data is exchanged with memory or I/O, or an interrupt vector is input.
This output is fixed at the high level in a standby mode.
This pin is a 3-state output, its impedance is high during the hold acknowledge.

(12) **BUF $\overline{\text{R}}$/W (Buffer Read/Write) ... small**

This signal is output to decide the data transfer direction of external bi-directional buffers. It shows the sending direction from the CPU to an external device at a high level, and the receiving direction from an external device to the CPU at the low level.
This output is fixed at the high or low level in a standby mode.
This pin is a 3-state output, its impedance is high during the hold acknowledge.

(13) **IO/$\overline{\text{M}}$ (IO/Memory) ... small**

The signal is output to differentiate I/O access from memory access. It shows the I/O at the high level, and memory at the low level.
This output is fixed at the high or low level in a standby mode.
This pin is a 3-state output, its impedance is high during the hold acknowledge.

(14) $\overline{WR}$ (Write Strobe) ... small

The signal is output when data is written to I/O or memory, the distinction between I/O and memory is executed by the IO/$\overline{M}$ signal.
This output is fixed at the high level in a standby mode.
This pin is a 3-state output, its impedance is high during the hold acknowledge.

(15) HLDAK (Hold Acknowledge) ... small

An acknowledge signal is output, which shows that the CPU received a hold request signal (HLDRQ).
While this signal is active (at the high level), address bus, address/data bus, and control bus of 3-state output is high-impedance.

(16) HLDRQ (Hold Request) ... small

A signal is input, which allows an external device to request the CPU to release address bus, address/data bus, and control bus.
Since this signal cannot guarantee correct operation unless it satisfies setup time, it should be synchronized with external device.

★ **(17) $\overline{RD}$ (Read Strobe) ... small/large**

This signal is output when reading data from I/O or memory. The distinction between the I/O and the memory is executed by the IO/$\overline{M}$.
This signal exists originally for a small mode, but it may be output at the same timing in a large mode.
This output is fixed at the high level in a standby mode.
This pin is a 3-state output, its impedance is high during the hold acknowledge.

★ **(18) S/$\overline{LG}$ (Small/Large) ... small/large**

This is a pin to decide the CPU operation mode. This pin is used fixed at the high or low level. This pin operates at the high level in a small mode, and at the low level in a large mode.
The pin numbers indicated differentiate their functions depending on the mode to be operated, then each pin has its own name.

Pin Number[Note]			Function	
DIP	QFP	QFJ	S/$\overline{LG}$ = High	S/$\overline{LG}$ = Low
24	38	27	$\overline{INTAK}$	QS1
25	39	28	ASTB	QS0
26	41	29	$\overline{BUFEN}$	BS0
27	42	30	BUF $\overline{R}$/W	BS1
28	43	31	IO/$\overline{M}$	BS2
29	44	32	$\overline{WR}$	$\overline{BUSLOCK}$
30	45	33	HLDAK	$\overline{RQ}$/$\overline{AK1}$
31	47	34	HLDRQ	$\overline{RQ}$/$\overline{AK0}$
34	50	37	LBS0	Always high

Note Pin number is different from package.

(19) LBS0 (Latched Bus Status 0) ... small

This signal is used with the IO/$\overline{M}$ signal and BUF$\overline{R}$/W signal and informs external what the current bus cycle is.

IO/$\overline{M}$	BUF$\overline{R}$/W	LBS0	Bus cycle
0	0	0	Program fetch
		1	Memory read
	1	0	Memory write
		1	Receiving state
1	0	0	Interruption acknowledge
		1	I/O read
	1	0	I/O write
		1	Hold

(20) A19/PS3 to A16/PS0 (Address Bus/Processor Status) ... small/large

This is a dual-function output pin for address bus and processor status signal, the contents of each pin are output by time multiplexing.

As an address bus, the upper 4 bits are output out of the 20-bit memory address. 0 is output to all bits during the I/O access.

Processor status signal is output to both memory and I/O accesses. PS3 is always 0 in native mode, and always 1 in an emulation mode. The contents of interrupt enable flag (IE) is output to PS2. PS1 or PS0 shows which segment is currently used.

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

These outputs are fixed at the high or low level in a standby mode.

The A19/PS3 to A16/PS0 pins are 3-state outputs and impedance is high during the hold acknowledge.

(21) QS1, QS0 (Queue Status) ... large

This signal notifies an external device (floating-point operation coprocessor) of the instruction queue status in the CPU.

QS1	QS0	Status of Instruction Queue
0	0	No operation (no change in the queue)
0	1	First byte of instructions
1	0	Empty
1	1	After 2nd byte of instructions

This status of instruction queue represents the status when EXU accesses an instruction queue. The contents which are output to the QS1 and QS0 pins are effective only in the 1 clock cycle immediately after this queue access.
This status signal is offered so that the coprocessor for floating-point operation can monitor the CPU program execution state and process when the control is shifted to the coprocessor itself (by FPO: Floating-Point Operation instruction).
These outputs are fixed at the low level in standby mode.

(22) BS2 to BS0 (Bus Status) ... large

This is a status signal to inform an external bus controller what the current bus cycle is.
The external bus controller decodes these signals, and generates control signals to access memory or I/O.

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

These outputs are fixed at the high level in a standby mode.
These pins are 3-state outputs, and impedance is high during the hold acknowledge.
These signals become high when the clock rises immediately after RESET is activated and remain high until the next rise of the clock. After this 1 clock cycle, the signals become high-impedance.

(23) $\overline{\text{BUSLOCK}}$ (Bus Lock) ... large

This is the signal to request the other master CPUs in a multiprocessor system not to use system bus, when 1 instruction following the BUSLOCK front end instruction is being executed.
This output is fixed at the high level in a standby mode (however, it is fixed at the low level if BUSLOCK instruction exists before HALT instruction).
This pin is a 3-state output and impedance is high during the hold acknowledge.

(24) $\overline{\text{RQ}}/\overline{\text{AK1}}$, $\overline{\text{RQ}}/\overline{\text{AK0}}$ (Hold Request/Acknowledge) ... large

$\overline{\text{RQ}}/\overline{\text{AK1}}$ and $\overline{\text{RQ}}/\overline{\text{AK0}}$ are common pins for both bus hold request input ($\overline{\text{RQ}}$) and bus hold acknowledge signal output ($\overline{\text{AK}}$). Their priority is as follows:

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

These pins are 3-state inputs/outputs. They incorporate a pull-up resistor and are set inactive (at the high level) in the open (float) status.
When this signal is used as a bus hold request input ($\overline{\text{RQ}}$), it cannot guarantee correct operation unless it satisfies setup/hold time. Therefore, it should be synchronized with an external device.

(25) V_{DD} (Power Supply) ... small/large
This is a positive power supply pin.

(26) GND (Ground) ... small/large
This is a GND potential.

(27) IC (Internally Connected)
Set this to a GND potential.

2. REGISTER CONFIGURATION

2.1 PFP (Prefetch Pointer)

Prefetch pointer is a 16-bit binary counter holding offset information of the program memory address which BCU is to prefetch to an instruction queue.

The PFP is incremented every time BCU prefetches instruction bytes from a program memory. Also, a new location is loaded when branch, call, return, or break instruction is executed. The contents of PFP at this point are same as that of the PC (Program Counter).

PFP is always used together with PS (Program Segment) register.

2.2 Q0 to Q3 (Prefetch Queue)

The μPD70108 has a 4-byte instruction queue (FIFO). It can store the maximum instruction code of 4 bytes which BCU prefetches.

The instruction codes stored in the queue are fetched and executed by EXU.

When branch, call, return, or break instruction is executed, or external interrupt is processed, the queue contents are cleared, and an instruction of a new location is prefetched.

Usually, the μPD70108 executes prefetch if the queue has blank of 1 byte or more.

If the average execution time of several sequential instructions exceeds the number of clocks, to some extent, which is necessary for prefetching the instruction codes of each instruction, and when EXU ends the execution of one instruction, then the instruction codes which EXU can execute consecutively will be ready in a queue, and the fetch time from external memory may be deducted from the instruction execution time. Therefore, it is possible to increase the processing speed compared with the CPU which fetches and executes in each instruction.

The effect of queue will be reduced, in inverse proportion to the number of instructions whose queue is cleared like the above-mentioned execution of branch instruction, or if the instructions with short execution time continue.

2.3 DP (Data Pointer)

Data pointer is a 16-bit register which specifies the address for reading/writing variables.

The contents of register including the offset of the effective and memory addresses which are created in EA generator are transferred to this data pointer.

2.4 TEMP (Temporary Communication Register)

This is a 16-bit temporary communication register between an external data bus and EXU.

For the purpose of byte access, TEMP can read/write upper and lower bytes independently.

Basically, EXU terminates write operations by transferring data to TEMP, then confirms the data transfers from an external bus to TEMP and terminates read operations.

2.5 Segment Register (PS, SS, DS0, DS1)

In the μPD70108, memory address is divided into logical segments by the 64K bytes, the start address of each segment is specified by a segment register, the offset after the start address is specified either by another register or an effective address.

There are four types of segment registers:

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

A pair of PS and PFP (Prefetch Pointer) and that of DS1 and IY are fixed.

SS is paired with SP in normal stack operation, but it offsets effective address when BP register is selected as a base register.

DS0 is used together with IX in a block transfer processing, but it offsets effective address in the other processing.

In the addressing which defines SS as a segment register when using BP register as a base register, it is possible to use the other 3 types of segment registers for a segment selection by using segment overlaid prefix instruction (PS:, DS0:, DS1:).

2.6 ADM (Address Modifier)

ADM (Address Modifier) performs the generation of physical address (addition of segment register to PFP or DP) and the increments of PFP (Prefetch Pointer).

2.7 General Registers (AW, BW, CW, DW)

There are four different types of 16-bit general registers. It is possible to access as an 8-bit register (AH, AL, BH, BL, CH, CL, DH, and DL) by dividing each register into the upper and lower 8 bits.

Therefore, these registers can be used as an 8-bit or 16-bit register for a variety of instructions, such as transfer instructions, arithmetic operation instructions, and logical operation instructions.

Also, the following list shows that the each register can be used as a default register for a specific instruction processing.

- AW : Word multiplication/division, word I/O, data conversion
- AL : Byte multiplication/division, byte I/O, BCD rotate, data conversion, translation
- AH : Byte multiplication/division
- BW : Translation
- CW : Loop control branch, repeat prefix
- CL : Shift instruction, rotate instruction, BCD operation
- DW : Word multiplication/division, indirect addressing I/O

2.8 Pointer (SP, BP) and Index Register (IX, IY)

These are used as a base pointer or an index register during the memory access executed by based addressing, indexed addressing, and based/indexed addressing.

Like a general register, they can be used for instructions, such as transfers, arithmetic operations, and logical operations, but they cannot be used as an 8-bit register for the same instructions.

The following list shows that each register can be used as a default register for the purpose of a specific processing.

SP : Stack manipulation
IX : Block transfer (on the source side), BCD string operation
IY : Block transfer (on the destination side), BCD string operation

2.9 TA/TB (Temporary Register/Shifter A/B)

TA/TB are 16-bit temporary registers/shifters which are used for multiplication/division and shift/rotate (including BCD rotate) instructions.

TA and TB work as a 32-bit temporary register/shifter when executing the multiplication/division instructions, while only TB works as a 16-bit temporary register/shifter when executing the shift/rotate instructions.

Both TA and TB can read/write the upper and lower byte independently between the internal buses.

TA/TB are inputs of ALU.

2.10 TC (Temporary Register C)

TC is a 16-bit temporary register which is used for the internal processing, such as multiplication/division, etc.

TC is an input of ALU.

2.11 ALU (Arithmetic & Logic Unit)

ALU (Arithmetic & Logic Unit) consists of a full adder and a logic unit, and it performs the arithmetic operations (addition/subtraction/multiplication/division, increment, decrement, and complement operations) and the logical operations (test, AND, OR, and XOR, and the bit-wise test, set, clear, and inversion).

2.12 PSW (Program Status Word)

Program status word consists of the 6 types of status flags and the 4 types of control flags.

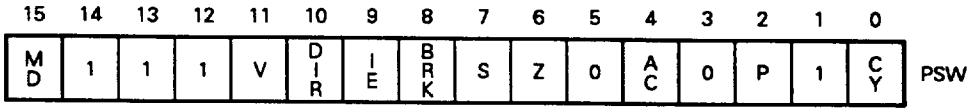
Status flags

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

Control flags

- MD (Mode)
- DIR (Direction)
- IE (Interrupt Enable)
- BRK (Break)

These flags are stack processed by manipulating the following word images.



Status flags are automatically set and reset according to the execution result (data value) of each instruction.

CY flag can be set, reset, or inverted directly by instructions.

Control flags are set or reset by instructions, then control the CPU operations.

MD flag is reloadable only between the execution of BRKEM instruction and that of RETEM instruction, it may not be restored in other places even if RETI, or POP PSW instruction is executed.

2.13 LC (Loop Counter)

LC (Loop Counter) is a 16-bit register which counts the number of loops of the primitive block transfer/I/O instructions (MOVBK, OUTM, etc.) controlled by repeat prefix instructions (REP, REPC, etc.), and the number of shifts of multi-bit shift/rotate instructions.

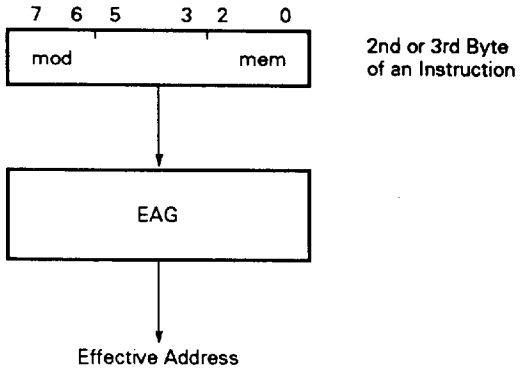
2.14 PC (Program Counter)

The program counter is a 16-bit binary counter which holds the offset information of the program memory address which the EXU is currently to execute.

The PC is incremented each time a microprogram fetches an instruction byte out of an instruction queue. Also, a new location is loaded when branch, call, return, or break instruction is executed. The contents of the PC at this point are same as the PFP (Prefetch Pointer).

2.15 EAG (Effective Address Generator)

EAG (Effective Address Generator) is a circuit which performs high-speed effective address calculations needed during the memory access. It terminates the calculation by two clocks in all addressing modes.



If it reads the byte (the 2nd or 3rd byte) specified by the instruction's operand and requires memory access, it will generate a control signal related to the ALU and the associated register operation, and will calculate an effective address to transfer the signal to the DP (Data Pointer).

If necessary, it requests the BCU to activate a bus cycle (memory read).

2.16 Instruction Decoder

Instruction decoder classifies the 1st byte of an instruction code into the groups with a specific function, and holds it during the execution of microinstruction.

2.17 Microaddress Register

Microaddress register specifies the address of microinstruction ROM which should be executed consecutively.

When starting the execution of microinstructions, the 1st byte of instructions stored in a queue as a start address is read into this register, and the register specifies the start address of the specific microinstruction sequence.

2.18 Microinstruction ROM

Microinstruction ROM holds 29-bit width of microinstructions for 1024 words.

2.19 Microinstruction Sequence Circuit

This circuit manages the control of a microaddress register, the output control of a microinstruction ROM, and synchronization of the EXU and BCU.

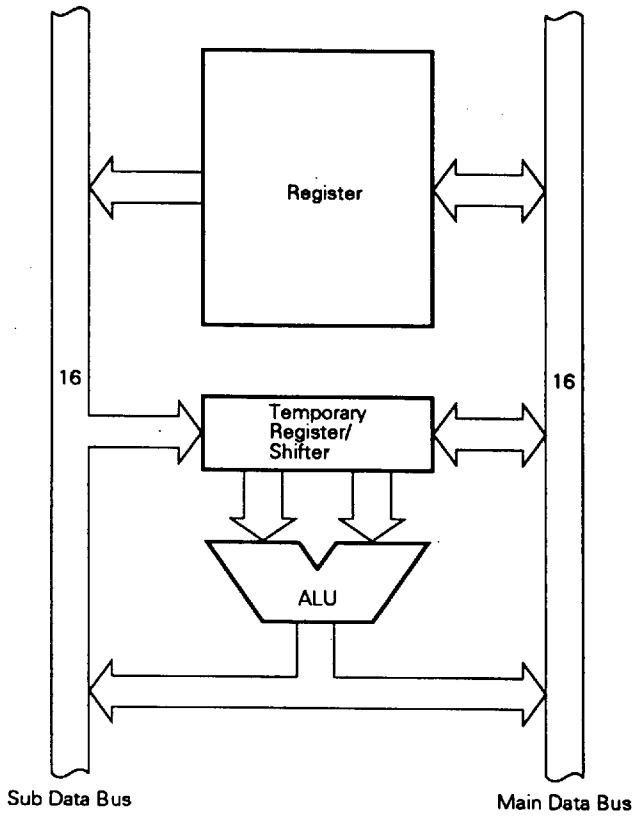
3. HIGH SPEED EXECUTION OF INSTRUCTIONS

In order to reduce the instruction execution time, the μPD70108 is equipped with the following hardware features.

- EXU internal dual data bus
- Effective address generator
- 16-/32-bit temporary register/shifter (TA, TB)
- 16-bit loop counter (LC)
- PC (Program Counter) and PFP (Prefetch Pointer)

3.1 Dual Data Bus Method

In order to reduce the number of processing steps required for executing instructions, the dual data bus method of main data bus (16-bit) and sub data bus (16-bit) is adopted. This method realizes roughly 30% reduction of processing time (compared with a single bus method) in addition/subtraction, logical operations, and compare instructions.



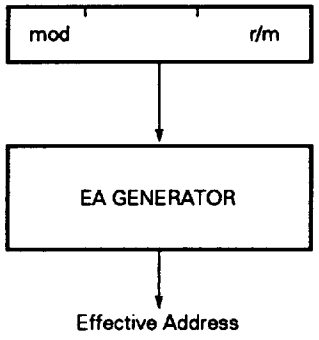
Example ADD AW, BW; AW ← AW + BW

	Single bus	Dual bus
Step 1	ALU ← AW	ALU ← AW, BW
2	ALU ← BW	AW ← ALU
3	AW ← ALU	

3.2 Effective Address Generator

This is a circuit which may perform high-speed processing of effective address calculation required during the memory access.

This dedicated hardware has realized the high-speed processing which is several times faster than the microprogram method. It requires just 2 clocks for effective address calculations in all addressing modes, while the microprogram method requires 5 to 12 clocks for the calculation.



3.3 16-/32-Bit Temporary Register/Shifter (TA, TB)

Temporary register/shifter (TA, TB) is offered for multiplication/division and shift/rotate instructions.

The adoption of this circuit has increased the speed of multiplication/division instructions particularly. This speed is 4 times as fast as that of the microprogram method.

TA + TB : 32-bit temporary register/shifter for multiplication/division instructions
TB : 16-bit temporary register/shifter for shift/rotate instructions

3.4 Loop Counter (LC)

This counts the number of loops of primitive block transfer/I/O instruction which is controlled by repeat prefix instruction, and the number of shifts of multi-bit shift/rotate instruction.

For example, the multi-bit rotate of register is executed as follows. It has increased the speed up to two times that of microprogram method.

RORC AW, CL; CL = 5

Microprogram method	LC method
8 + 4 × 5 = 28 clocks	7 + 5 = 12 clocks

3.5 PC and PFP

The hardware contains both a prefetch pointer (PFP) which addresses program memory prefetching, and a program counter (PC) which addresses program memory which is going to be executed. Because of this, the instruction execution time for branch, call return, and break instructions has been reduced for another few clocks, compared with a microprocessor with one PFP.

4. DESCRIPTION OF CHARACTERISTIC INSTRUCTIONS

4.1 Variable Length Bit Field Operation Instructions

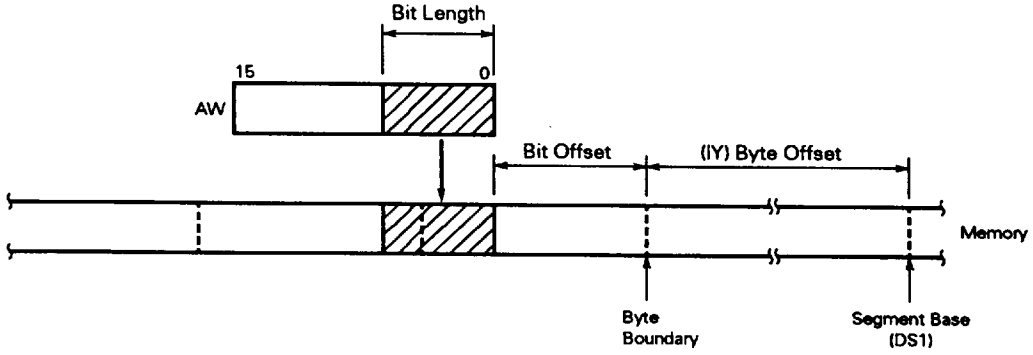
There are two types of instructions, INS (Insert Bit Field) and EXT (Extract Bit Field). These instructions are very effective for a computer plotting and a high-level language. For example, they may be applied to a packed array of Pascal and a record-type data structure.

(1) INS reg8, reg8'/INS reg8, imm4

This instruction transfers the lower bit data (out of the 16-bit data of AW register) which has a length specified by the 2nd operand, to memory area which is decided by a byte offset which is addressed by a segment register DS1 and a indexed register IY, and a bit offset which is specified by the values (0 to 15) of the 1st operand.

After the completion of transfer, the register which is specified by both IY register and the 1st operand is automatically updated to show the next bit field.

The effective values of the 2nd operand are 0 to 15 (1-bit length at 0, 16-bit length at 15) only.



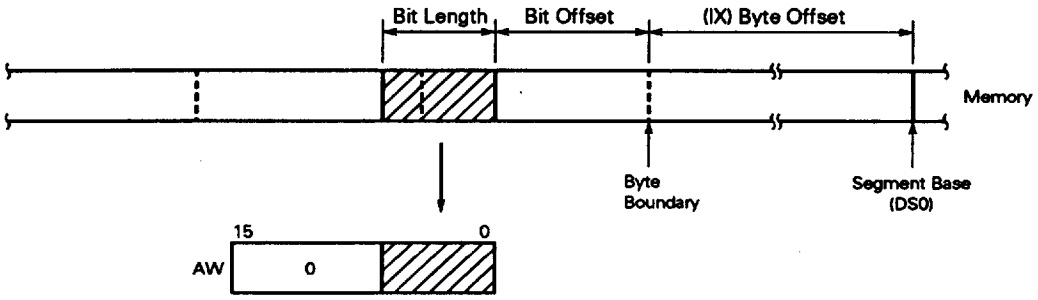
Bit field data can extend over the byte boundaries of memory.

(2) EXT reg8, reg8'/EXT reg8, imm4

This loads a bit field data with a bit length defined by the 2nd operand, from the memory area decided by the bit offset specified by a byte offset which is addressed by a segment register DS0 and an index register IX and a bit offset which is specified by the values (0 to 15) of the 1st operand, to the AW register.

After the completion of transfer, the register which is specified by both IX register and the 1st operand is automatically updated to show the next bit field.

The effective values of the 2nd operand are 0 to 15 (1-bit length at 0, 16-bit length at 15) only.



Bit field data can extend over the byte boundaries of memory.

4.2 Packed BCD Operation Instructions

The instructions consist of ADD4S, SUB4S, and CMP4S which may process packed BCD in a string form, and ROR4 and ROL4 which process it as a byte/word operand.

(1) ADD4S

This instruction sums a packed BCD string addressed by an index register IX and that by a index register IY, and stores the result to a string which is addressed by IY. The string length (number of BCD digits) is decided by a CL register. The operation result affects both a zero flag (Z) and a carry flag (CY).

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

(2) SUB4S

This instruction subtracts a packed BCD string addressed by an index register IX from that by an index register IY, and stores the result to a string addressed by IY. The string length (number of BCD digits) is decided by a CL register. The operation result affects both a zero flag (Z) and a carry flag (CY).

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

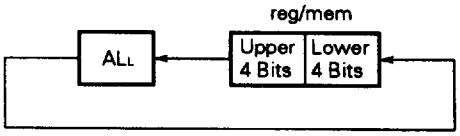
(3) CMP4S

This instruction performs the same subtraction as SUB4S does, but it does not store the result, and only affects a zero flag (Z) and a carry flag (CY).

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

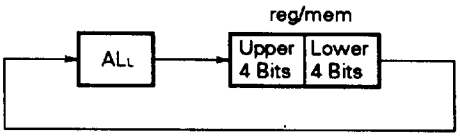
(4) ROL4

This instruction handles either a register which is directly addressed by an instruction byte or byte data of memory as BCD data, then rotates its one digit to the left through the lower 4 bits (AL$_L$) of an AL register.



(5) ROR4

This instruction handles either a register which is directly addressed by an instruction byte or byte data of memory as BCD data, then rotates its one digit to the right through the lower 4 bits (AL$_L$) of an AL register.



4.3 Stack Operation Instructions

(1) PREPARE imm16, imm8

This instruction is used to create a "Stack Frame" which is necessary for a block-structured high-level language (e.g. Pascal, Ada, etc.). A stack frame contains both a pointer group pointing a frame of variables which may be referred from the procedures and the area of local variables.
Description is continued below using an example program made by a Pascal type language.

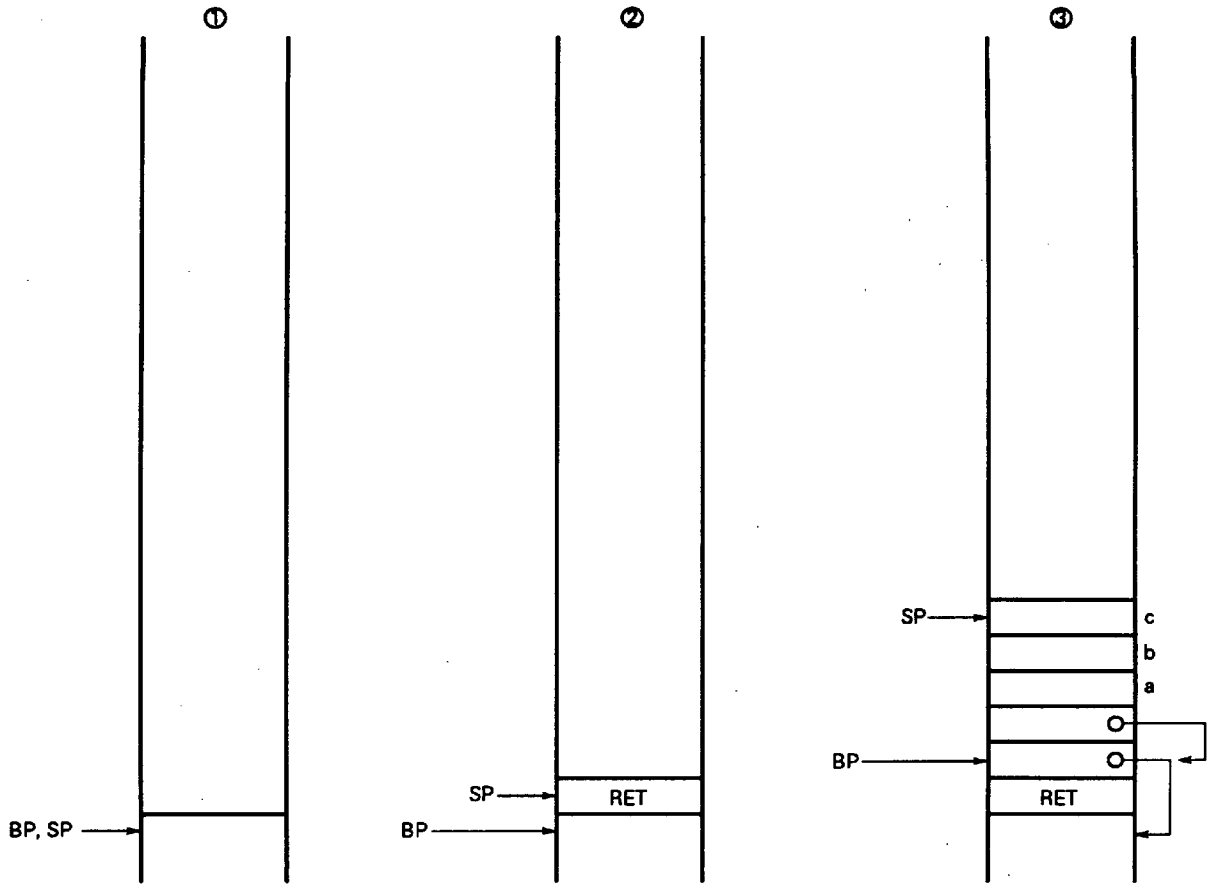
```
program EXAMPLE;
  procedure P;
    var  a,b,c,;
    procedure Q;
      var  d,e;
      procedure R;
        var  f,g;
        begin
          d:=a+f+g;
        end;
      begin
        R;
        b:=d+e;
      end;
    begin
      a:=b+c;
      Q;
    end;
(*main program*)
  begin
      P;
  end.
```

Remark A word is used for all variables.

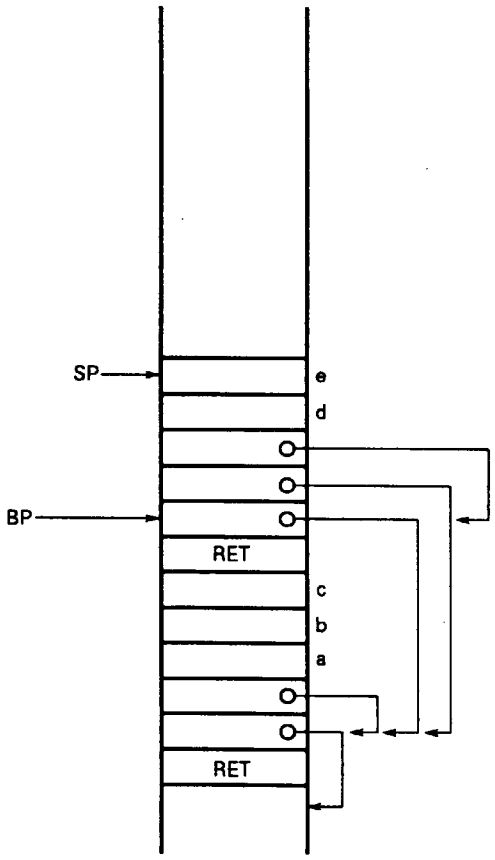
This is a program example in which 3-layered procedure blocks are nesting. Procedure P defines variables a, b, and c, procedure Q defines d and e, and procedure R defines f and g. Therefore, global variables a, b, and c, are referred from procedure Q, and variables a, b, c, d, and e from procedure R.
The PREPARE instruction copies a frame pointer to reserve the area of local variables and to enable the reference to global variables. The 1st operand specifies an area size (byte unit) to be reserved for local variables, and the 2nd operand shows the depth of the procedure block (the depth is called "lexical level").
The frame's base address which is created by the PREPARE instruction is set to a base pointer BP.
After having compiled the EXAMPLE program, this program converts itself to a program listed in the next page (The DISPOSE instruction which is used in an assembler program returns the state of both a stack pointer SP and a base pointer BP to the state immediately before the PREPARE instruction is executed. Please refer to (2)).

```
; ASSEMBLER PROGRAM

START:  MOV       SP, SPTOP
        MOV       BP, SP          ; ①
        CALL      P               ; ②
        BR        SYSTEM
P:      PREPARE   6, 1            ; ③
        MOV       AW, [BP][B+BLEVEL*2]
        ADD       AW, [BP][C+CLEVEL*2]
        MOV       [BP][A+ALEVEL*2],AW
        CALL      Q
        DISPOSE
        RET
Q:      PREPARE   4, 2            ; ④
        CALL      R
        MOV       AW, [BP][D+DLEVEL*2]
        ADD       AW, [BP][E+ELEVEL*2]
        MOV       IY, [BP][BLEVEL*2]
        MOV       SS:[IY][B+BLEVEL*2], AW
        DISPOSE
        RET
R:      PREPARE   4, 3            ; ⑤
        MOV       AW, [BP][F+FLEVEL*2]
        ADD       AW, [BP][G+GLEVEL*2]
        MOV       IY, [BP][ALEVEL*2]
        ADD       AW, SS:[IY][A+ALEVEL*2]
        MOV       IY, [BP][DLEVEL*2]
        MOV       SS:[IY][D+DLEVEL*2], AW
        DISPOSE
        RET

  ; A = -2        ALEVEL = -1
  ; B = -4        BLEVEL = -1
  ; C = -6        CLEVEL = -1
  ; D = -2        DLEVEL = -2
  ; E = -4        ELEVEL = -2
  ; F = -2        FLEVEL = -3
  ; G = -4        GLEVEL = -3
```

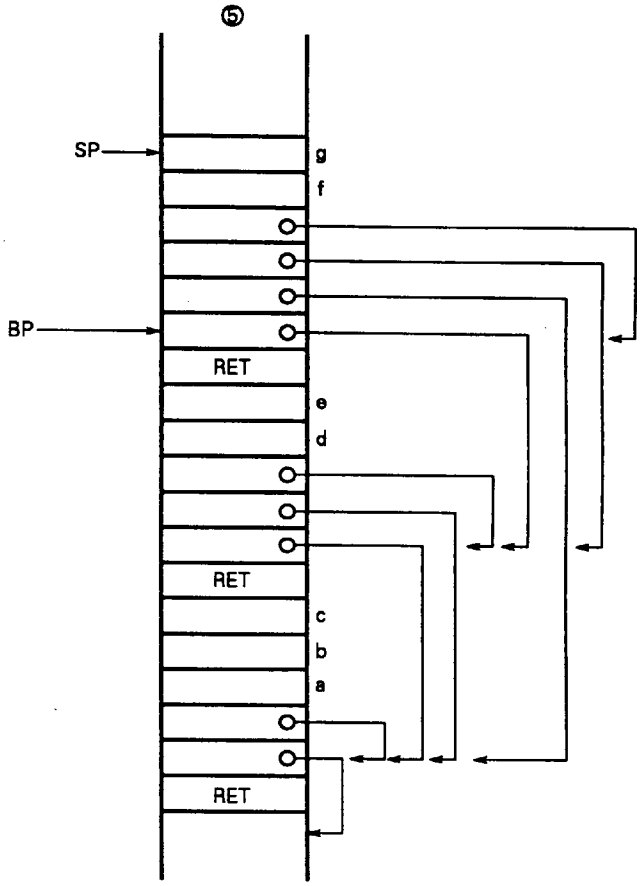
The process in which a stack frame is created as the program runs is illustrated in the following pages. Numbers correspond to those placed in the program's comment list.



④



⑤



The PREPARE instruction saves BP to a stack first in order to restore the BP of a procedure at the called side when the procedure finishes. Then, it pushes a frame pointer (a saved BP) onto the stack within the range accessible from the called procedure. The accessible range equals to the value which is subtracted by one from the lexical level of the procedure.
If the lexical level is one or more, the instruction pushes its own frame point onto the stack. This is done to copy a frame pointer of the called procedure, when the instruction copies a frame pointer in the other procedure which was called from this procedure.
Then, the instruction sets the value of new frame pointer to BP, and reserve the area of the local variables to be used in the procedure, onto the stack. I.e. it subtracts the value worthy of local variables from SP.

```
   display  =  2nd  operand
   dynamics  =  1st  operand

SP  =  SP-2;
(SP)  =  BP;
temp  =  SP;
if  display  >  0  then  begin
   repeat  display  -  1  times
      begin
         SP  =  SP-2;
         BP  =  BP-2;
         (SP)  =  (BP);
      end;
   SP  =  SP-2;
   (SP)  =  temp;
   end;
BP  =  temp;
SP  =  SP-dynamics
```

Data access

(a) Access of local variables

Local variables are placed in the frame of the procedure itself. Therefore, the effective address EA.L of a local variable is calculated by the following formula.

EA.L = SS: (BP + offset)

This "offset" is the sum of the offset values which are located from a frame size stacked onto the frame (the base value of an accessible frame) and the base value of local variable area, to that variable.

(b) Access of global variables

Global variables are located in the address added by the offset value which accesses the target base pointer out of the old base pointers loaded onto the stack frame and attempts to access the value.

Therefore, the effective address EA.G of global variables is calculated by the following expression:

EA.G = SS: ((SS: (BP + offset1)) + offset2)

This offset1 is the offset value from the base value (BP value) of the current frame to the address in which the base address of a frame (including the global variable to be referred) is stored. Also, the offset2 is the offset value from the base value of a frame which holds the variable to be referred to that variable.

(2) DISPOSE

This instruction releases one of the stack frames which is created by PREPARE instruction. For BP it loads a point value which points the previous frame, while for SP it loads a point value which points the least significant address of a frame.

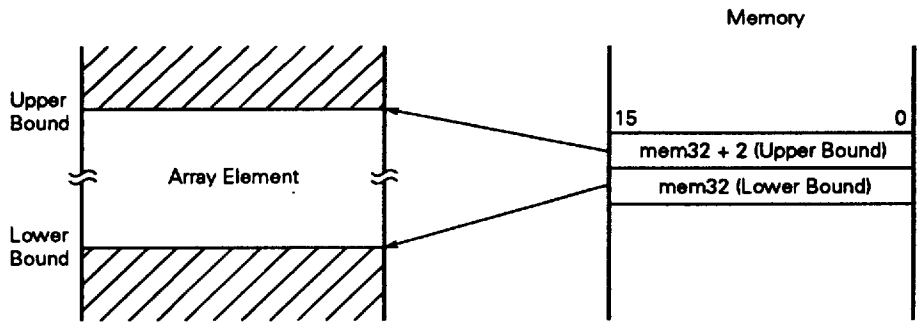
SP = BP;
BP = (SP);
SP = SP + 2

4.4 Array Index Check Instructions

This is an instruction to check whether the index value to specify if an element exists in the defined area or not, in array-type data structure. If the index value exceeds the area, it activates BRK5.

The defined area value should be set to the 2 words (setting the lower bound value at the 1st word, and the upper bound value at the 2nd word) in the memory, before CHKIND instruction is executed. The index value is for the register (any 16-bit register) which an array manipulation program is using.

```
CHKIND    reg 16, mem 32

  If (mem 32) > reg16 or (mem 32 + 2) < reg16
  TA←(015H, 014H)
  TC←(017H, 016H)
  SP←SP−2, (SP+1, SP)←PSW
  IE←0, BRK←0                          } = BRK5
  SP←SP−2, (SP+1, SP)←PS
  PS←TC
  SP←SP−2, (SP+1, SP)←PC
  PC←TA
```



4.5 Mode Operation Instructions

The operating modes of the μPD70108 consist of native mode (normal operation) and emulation mode (emulation operation of the μPD8080AF instruction set). As a flag to switch these modes, a mode flag (MD) is provided in the bit 15 of PSW. The mode is changed to native mode when MD is 1, and to emulation mode when MD is 0. MD is set/reset directly or indirectly by the mode operation instruction.

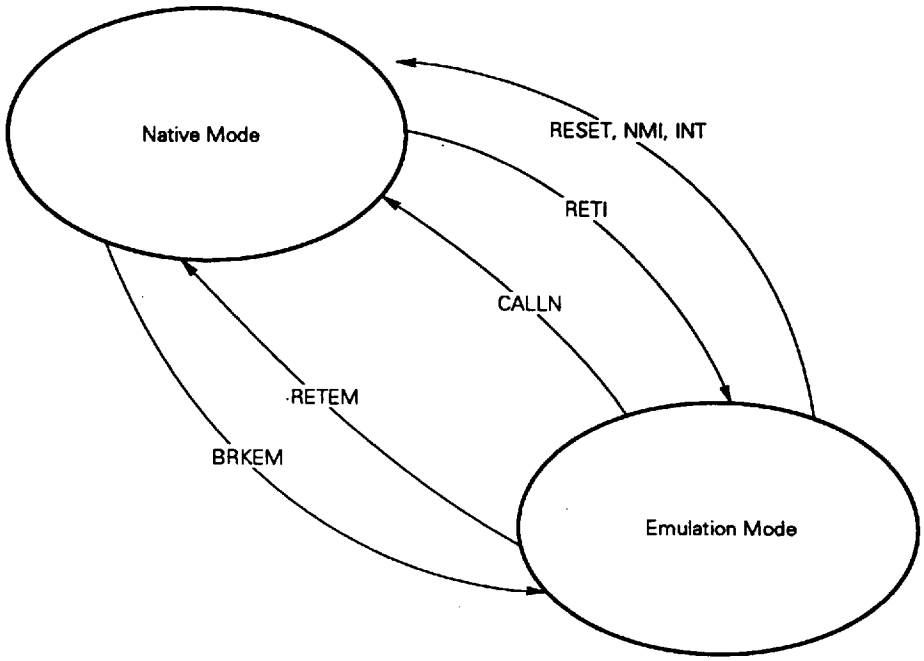
The instructions to change modes from native to emulation are:

BRKEM (Break for Emulation)
RETI (Return from Interrupt)

The instructions to change modes from emulation to native are:

RETEM (Return from Emulation)
CALLN (Call Native routine)

Also, either RESET input or external interrupt input (NMI, INT) turns emulation mode back to native mode.



(1) BRKEM imm 8

This is a basic instruction to activate emulation mode. This instruction saves PSW, PS, and PC, resets MD (0), and loads an interrupt vector specified by operand to PS and PC. This instruction neither affects interrupt enable flag (IE) nor breaks flag (BRK).

Fetching the instruction code of interrupt service routine (for emulation) which has jumped, the CPU interprets the code as an instruction of the μPD8080AF and executes it.

The CPU interprets emulation mode as interrupt servicing.

In emulation mode, the register and flag actions of the μPD8080AF are alternatively done by the register and flag of the μPD70108 shown below.

μPD8080AF	μPD70108
A	AL
B	CH
C	CL
D	DH
E	DL
H	BH
L	BL
SP	BP
PC	PC

μPD8080AF	μPD70108
C	CY
Z	Z
S	S
P	P
AC	AC

Regarding stack operations, either SP in native mode or BP in emulation mode works as a stack pointer. Adoption of this independent stack pointer allows both modes to reserve independent stack area, and prevents them from destroying any stack in other mode by erroneous stack operation.

SP, IX, IY, and AH in native mode and the four segment register (PS, SS, DS0, and DS1) are not affected by emulation mode.

In emulation mode, the segment base of instruction is decided by the PS register (automatically decided by interrupt vector), and that of data by the DS0 register (decided by a programmer just before entering emulation mode).

(2) RETEM (without operand)

When RETEM instruction is executed in emulation mode (this instruction is interpreted as an instruction of the μPD8080AF), the CPU restores PS, PC, and PSW and returns to native mode, as if it returns from interrupt servicing. At this point, the contents (i.e. "1") in native mode which was saved in the stack by BRKEM instruction are restored, which sets the CPU to native mode.

(3) CALLN imm 8

When this instruction is executed in emulation mode (this instruction is interpreted as an instruction of the μPD8080AF), the CPU save PS, PC, and PSW to the stack (MD = 0 is saved), sets (1) a mode flag (MD), then loads the interrupt vector specified by operand to PS and PC. This instruction neither affects interrupt enable flag (IE) nor break flag (BRK).

Thus, interrupt routine in native mode can be called from emulation mode.

To return from this interrupt routine to emulation mode, RETI instruction should be executed.

(4) RETI (without operand)

This is a general instruction to return from an interrupt routine activated by BRK instruction or an external interrupt in native mode. If this instruction is executed at the end of an interrupt service routine activated by CALLN instruction in emulation mode, PS, PC, and PSW restoration is exactly the same as normal. Because the value (= 0) of mode flag (MD) in emulation mode is restored to MD if PSW is restored, the CPU is set to emulation mode, then further instructions are interpreted as an instruction of the μPD8080AF and executed. RETI instruction is executed to return from an interrupt routine of native mode which was activated by NMI or INT interrupt request generated in emulation mode in the same way.

4.6 Floating-Point Operation Coprocessor Instruction

FPO1 fp-op/FPO1 fp-op, mem
FPO2 fp-op/FPO2 fp-op, mem

These are coprocessor's instructions for external floating-point operation. They leave operations to a coprocessor when the CPU fetches these instructions, then they only execute auxiliary processing (calculation of effective address, generation of physical address, and activation of memory read cycle) for a coprocessor if necessary.

When a coprocessor monitors these instructions, it interprets them as an instruction to itself and executes them. At this point, the coprocessor uses only the address information of memory read cycle only activated by the CPU, or both the address and read data, depending on a type of instruction.

FPO1 and FPO2 instructions have the same function, but different type of codes.

Also in the description of an actual assembler language, it is more common to use mnemonic to each instruction in a coprocessor, rather than to use the mnemonic, FPO1 or FPO2.

When the CPU fetches FPO1 or FPO2 and either of them requests memory access, it activates a memory read cycle. However, the data read by this should be used by a floating-point operation coprocessor, so it will never be handled by the CPU.

Also, the CPU activates a memory read cycle even if a floating-point operation coprocessor needs a memory write cycle, the data resulted from this activation is ignored as a dummy data, only memory address information is latched by a floating-point operation coprocessor. Then a floating-point operation coprocessor uses the address information to execute a memory write cycle.

5. INTERRUPT OPERATIONS

The μPD70108 has mainly two types of interrupts; one by an external interrupt request, and the other by software processing.

They are classified further as follows:

(1) External interrupt

(a) NMI input (non-maskable)

(b) INT input (maskable)

(2) Software instruction

(a) Processing results of instruction

- Divide error by DIV or DIVU instruction
- Memory boundary over detection by CHKIND instruction

(b) Conditional break instruction

- When V = 1 in BRKV instruction

(c) Unconditional break instruction

- 1-byte break instruction, BRK 3
- 2-byte break instruction, BRK imm8

(d) Flag processing (single step)

- Sets BRK flag using stack operation

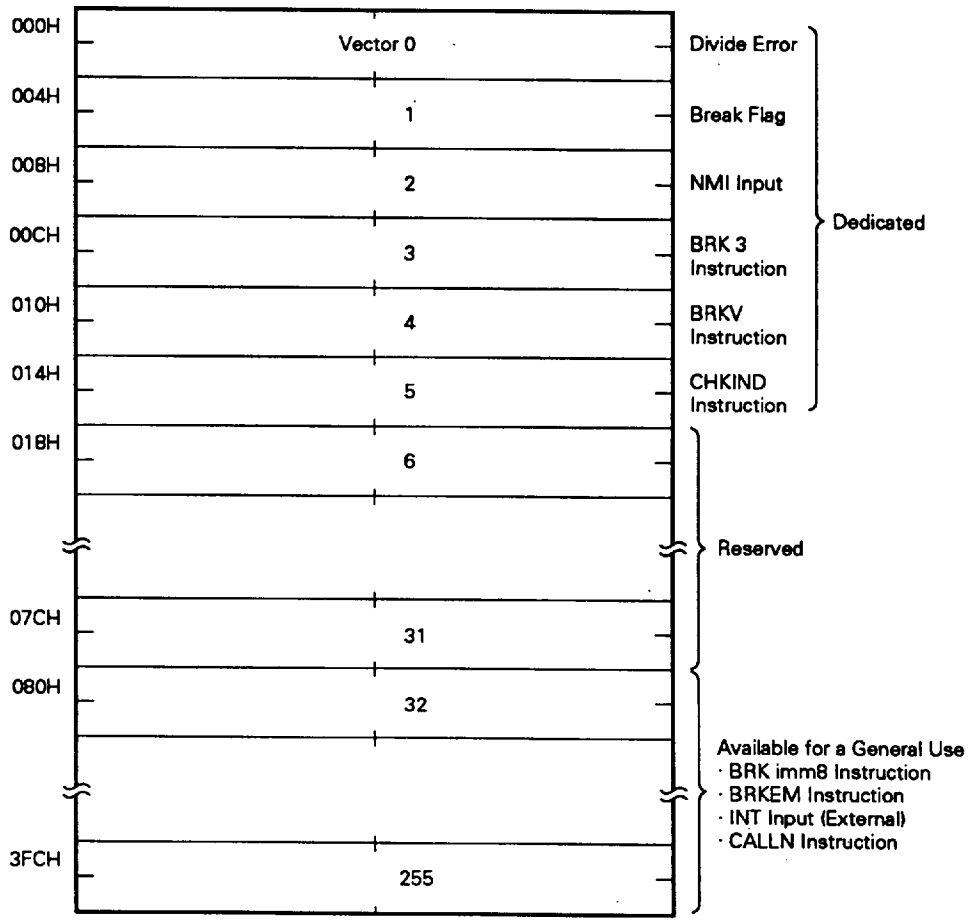
(e) Emulation-related instruction

- BRKEM imm8
- CALLN imm8

Any of the above interrupts should be selected by either automatically or sequentially specifying one point in the interrupt vector table which has been arranged beforehand, then decide interrupt routine start address.

Interrupt vector table is shown in the Figure 5-1. This table is assigned to the 1K-byte of the memory 000H to 3FFH, and it may hold 256 vectors (using 4 bytes per vector).

Figure 5-1. Interrupt Vector Table

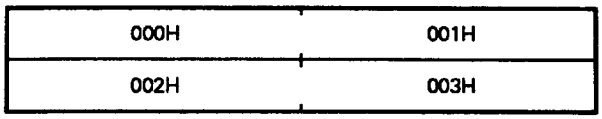


The vectors 0 to 5 are specified by a use factor, and the vectors 6 to 31 are reserved. They are not available for a general use.

In the vectors 32 to 255, the 2-byte break instruction, BRKEM instruction, INT input, and CALLN instruction (during the emulation) are available for a general use.

One interrupt vector consists of 4 bytes. 2 bytes in the lower address is loaded to PC as an offset, while the other 2 bytes in the upper address is loaded to PS as a base.

Example Vector 0



PS ← (003H, 002H)
PC ← (001H, 000H)

Based on this format, a programmer should initialize the contents of each vector to use at the beginning of a program.

The basic steps to jump in an interrupt service routine are listed as follows:

TA ← vector lower (offset)
TC ← vector upper (segment base)
SP ← SP – 2, (SP + 1, SP) ← PSW
IE ← 0, BRK ← 0, MD ← 0
SP ← SP – 2, (SP + 1, SP) ← PS
PS ← TC
SP ← SP – 2, (SP + 1, SP) ← PC
PC ← TA

6. STANDBY FUNCTIONS

μPD70108 incorporates standby mode to decrease the power consumption while it is waiting for program's processing.

Standby mode is set by HALT instruction in native mode or HLT instruction in emulation mode.

In standby mode, internal clock is provided only for the circuit related to the function necessary for releasing the standby mode and the circuit related to bus hold control function, then no internal clock is provided for the other circuits. This may reduce the power consumption to a fraction of that for normal operation (native/emulation mode).

Standby mode is released by either RESET input or external interrupt inputs (NMI, INT).

Bus hold function is effective during the standby mode, however, it returns to standby mode when bus hold request is cleared.

7. I/O ADDRESS RESERVE

The upper 256 bytes (FF00H to FFFFH) of I/O address might be used in the future. Do not use it at this time.

8. INSTRUCTION SET

Table 8-1. Legend of Operand Type

Identifier	Description
reg	8-/16-bit general register (destination-side register in the instruction which uses two 8-/16-bit general registers)
reg'	Source-side register in the instruction which uses two 8-/16-bit general registers
reg8	8-bit general register (destination-side register in the instruction which uses two 8-bit general registers)
reg8'	Source-side register in the instruction which uses two 8-bit general registers
reg16	16-bit general register (destination-side register in the instruction which uses two 8-bit general registers)
reg16'	Source-side register in the instruction which uses two 16-bit general registers
dmem	8-/16-bit memory location
mem	8-/16-bit memory location
mem8	8-bit memory location
mem16	16-bit memory location
mem32	32-bit memory location
imm	Constant in the range of 0 to FFFFH
imm3	Constant in the range of 0 to 7
imm4	Constant in the range of 0 to FH
imm8	Constant in the range of 0 to FFH
imm16	Constant in the range of 0 to FFFFH
acc	Register AW or AL
sreg	Segment register
src-table	Name of 256-byte conversion table
src-block	Name of a block which is addressed by register IX
dst-block	Name of a block which is addressed by register IY
near-proc	Procedure in the current program segment
far-proc	Procedure in the other program segment
near-label	Label in the current program segment
short-label	Label in the range, from the end of instruction to the –128 to +127-byte
far-label	Label in other program segment
memptr16	Word which includes the location's offset in the current program segment to which control attempts to move
memptr32	Double word which includes the location's offset and segment base address in other program segment to which control attempts to move
regptr16	16-bit general register which includes the location's offset in other program segment to which control attempts to move
pop-value	Number of bytes which are dumped from the stack (0 to 64K, usually even number)
fp-op	Immediate value to identify the instruction code of an external floating-point operation coprocessor
R	Register set

Table 8-2. Legend of Operation Code

Identifier	Description
W	Byte/word specification bit (0: byte, 1: word). However, when s is 1, sign extended byte data is specified 16-bit operand even if W = 1.
reg	Register field (000 to 111)
reg'	Register field (000 to 111) (source-side register in the instruction which uses two registers)
mem	Memory field (000 to 111)
mod	Mode field (00 to 10)
s	Sign extension specification bit (0: sign is not extended, 1: sign is extended)
X,XXX,YYY,ZZZ	Data to identify the instruction code of an external floating-point operation coprocessor.

Table 8-3. Legend of Operation Description

Identifier	Description
AW	Accumulator (16-bit)
AH	Accumulator (upper byte)
AL	Accumulator (lower byte)
BW	Register BW (16-bit)
CW	Register CW (16-bit)
CL	Register CW (lower byte)
DW	Register DW (16-bit)
BP	Base pointer (16-bit)
SP	Stack pointer (16-bit)
PC	Program counter (16-bit)
PSW	Program status word (16-bit)
IX	Index register (source) (16-bit)
IY	Index register (destination) (16-bit)
PS	Program segment register (16-bit)
SS	Stack segment register (16-bit)
DS0	Data segment 0 register (16-bit)
DS1	Data segment 1 register (16-bit)
AC	Auxiliary carry flag
CY	Carry flag
P	Parity flag
S	Sign flag
Z	Zero flag
DIR	Direction flag
IE	Interrupt enable flag
V	Overflow flag
BRK	Break flag
MD	Mode flag
(...)	Contents of memory shown in the parentheses
disp	Displacement (8-/16-bit)
ext-disp 8	16-bit displacement which is sign-extended from 8-bit displacement
temp	Temporary register (8-/16-/32-bit)
TA	Temporary register A (16-bit)
TB	Temporary register B (16-bit)
TC	Temporary register C (16-bit)
tmpcy	Temporary carry flag (1-bit)
seg	Immediate segment register (16-bit)
offset	Immediate offset register (16-bit)
←	Transfer direction
+	Addition
−	Subtraction
×	Multiplication
÷	Division
%	Modulo
∧	AND
∨	OR
∀	Exclusive-OR
××H	Numeric value of 2-digit hexadecimal number
××××H	Numeric value of 4-digit hexadecimal number

Table 8-4. Legend of Flag Operation

Identifier	Description
(Blank)	No change
0	Cleared to 0
1	Set to 1
×	Set or cleared according to the result
U	Undefined
R	Pre-saved value is restored

Table 8-5. Memory Addressing

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

Table 8-6. Selection of 8-/16-Bit General Registers

reg, 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 8-7. Selection of Segment Registers

sreg	
00	DS1
01	PS
10	SS
11	DS0

★ The instruction set is described in table form on the following pages.

The clock cycles indicated in the tables represent the time needed for the execution unit to execute instructions, and are based on the following conditions.

- The prefetch time and waiting time to use the bus are not included.
- Zero wait time is assumed for memory access. In other words, one bus cycle's clock cycle equals 4 clock cycles.
- Zero wait time is assumed for I/O access.
- Primitive block transfer instructions and primitive I/O instructions include the repeat prefix.

If the instruction performs both byte and word processing (holding W-bit), the value of the clock cycle is shown as follows:

The left side of / shows the clock cycle for the byte processing (W=0);
The right side of / shows the clock cycle for the word processing (W=1).

For the clock cycles of block transfer-related instructions, refer to **Table 8-8.**

Table 8-8. Clock Cycles of Block Transfer-related Instructions

Instruction	Clock Cycles	
	Byte processing (W=0)	Word processing (W=1)
MOVBK	11+8/rep (11)	11+16/rep (19)
CMPBK	7+14/rep (13)	7+22/rep (21)
CMPM	7+10/rep (7)	7+14/rep (11)
LDM	7+9/rep (7)	7+13/rep (11)
STM	7+4/rep (7)	7+8/rep (11)
INM	9+8/rep (10)	9+16/rep (18)
OUTM	9+8/rep (10)	9+16/rep (18)

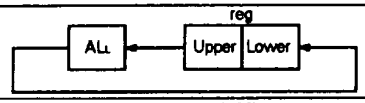
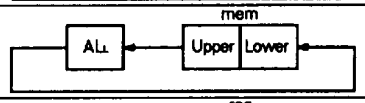
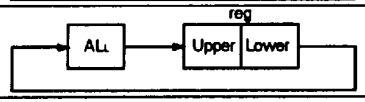
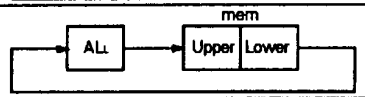
Remark Numeric values in parentheses are for single processing.

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Data transfer instructions	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	If W = 0, AL←(dmem) If W = 1, AH←(dmem+1), AL←(dmem)						
		dmem,acc	1 0 1 0 0 0 1 W		3	9/13	If W = 0, (dmem)←AL If 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 sreg: SS,DS0,DS1						
		sreg,mem16	1 0 0 0 1 1 1 0	mod 0 sreg mem	2 – 4	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	14	(mem16)←sreg						
		DS0,reg16, mem32	1 1 0 0 0 1 0 1	mod reg mem	2 – 4	26	reg16←(mem32) DS0←(mem32+2)						
		DS1,reg16, mem32	1 1 0 0 0 1 0 0	mod reg mem	2 – 4	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	3	AW↔reg16						

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Repeat prefix	REPC		0 1 1 0 0 1 0 1		1	2	Executes the primitive block transfer instruction of the consecutive byte during the CW ≠ 0, then decrements (–1) CW. Processes hold interrupt, if any. Exits from a loop if CY ≠ 1.						
	REPNC		0 1 1 0 0 1 0 0		1	2	Same as above Exits from a loop if CY ≠ 0.						
	REP REPE REPZ		1 1 1 1 0 0 1 1		1	2	Executes the primitive block transfer instruction of the consecutive byte during the CW ≠ 0, then decrements (–1) CW. Processes hold interrupt, if any. Exits from a loop if the primitive block transfer instruction is CMPBK or CMPM, and at the same time Z ≠ 1.						
	REPNE REPNZ		1 1 1 1 0 0 1 0		1	2	Same as above Exists from a loop if Z ≠ 0.						
Primitive block transfer instructions	MOVBK	dst-block, src-block	1 0 1 0 0 1 0 W		1	See Table 8-8	If W = 0, (IY)←(IX) DIR = 0 : IX←IX+1, IY←IY+1 DIR = 1 : IX←IX–1, IY←IY–1 If 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	See Table 8-8	If W = 0, (IX)–(IY) DIR = 0 : IX←IX+1, IY←IY+1 DIR = 1 : IX←IX–1, IY←IY–1 If 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 8-8	If W = 0, AL–(IY) DIR = 0 : IY←IY+1; DIR=1; IY←IY–1 If W = 1, AW–(IY+1, IY) DIR = 0 : IY←IY+2; DIR=1; IY←IY–2	×	×	×	×	×	×
	LDM	src-block	1 0 1 0 1 1 0 W		1	See Table 8-8	If W = 0, AL←(IX) DIR = 0 : IX←IX+1; DIR=1; IX←IX–1 If 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	See Table 8-8	If W = 0, (IY)←AL DIR = 0 : IY←IY+1; DIR=1; IY←IY–1 If W = 1, (IY+1,IY)←AW DIR = 0 : IY←IY+2; DIR=1; IY←IY–2						

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Bit field operation instructions	INS	reg8,reg8'	0 0 0 0 1 1 1 1	0 0 1 1 0 0 0 1	3	35 – 133	16-bit field←AW						
			1 1 reg' reg										
		reg8,imm4	0 0 0 0 1 1 1 1	0 0 1 1 1 0 0 1	4	35 – 133	16-bit field←AW						
			1 1 0 0 0 reg										
	EXT	reg8,reg8'	0 0 0 0 1 1 1 1	0 0 1 1 0 0 1 1	3	34 – 59	AW←16-bit field						
			1 1 reg' reg										
		reg8,imm4	0 0 0 0 1 1 1 1	0 0 1 1 1 0 1 1	4	34 – 59	AW←16-bit field						
			1 1 0 0 0 reg										
I/O instructions	IN	acc,imm8	1 1 1 0 0 1 0 W		2	9/13	If W = 0, AL←(imm8) If W = 1, AH←(imm8+1),AL←(imm8)						
		acc,DW	1 1 1 0 1 1 0 W		1	8/12	If W = 0, AL←(DW) If W = 1, AH←(DW+1),AL←(DW)						
	OUT	imm8,acc	1 1 1 0 0 1 1 W		2	8/12	If W = 0, (imm8)←AL If W = 1, (imm8+1)←AH, (imm8)←AL						
		DW,acc	1 1 1 0 1 1 1 W		1	8/12	If W = 0, (DW)←AL If W = 1, (DW+1)←AH, (DW)←AL						
Primitive I/O instructions	INM	dst-block,DW	0 1 1 0 1 1 0 W		1	See Table 8-8	If W = 0, (IY)←(DW) DIR = 0 : IY←IY+1; DIR=1; IY←IY–1						
							If W = 1, (IY+1, IY)←(DW+1,DW) DIR = 0 : IY←IY+2; DIR=1; IY←IY–2						
	OUTM	DW,src-block	0 1 1 0 1 1 1 W		1	See Table 8-8	If W = 0, (DW)←(IX) DIR = 0 : IX←IX+1; DIR=1; IX←IX–1						
							If W = 1 (DW+1, DW)←(IX+1,IX) DIR = 0 : IX←IX+2; DIR=1; IX←IX–2						

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Addition/subtraction instructions	ADD	reg,reg'	0 0 0 0 0 0 1 W	1 1 reg reg'	2	2	reg←reg+reg'	×	×	×	×	×	×
		mem,reg	0 0 0 0 0 0 0 W	mod reg mem	2 – 4	16/24	(mem)←(mem)+reg	×	×	×	×	×	×
		reg,mem	0 0 0 0 0 0 1 W	mod reg mem	2 – 4	11/15	reg←reg+(mem)	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 s W	1 1 0 0 0 reg	3 – 4	4	reg←reg+imm	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 s W	mod 0 0 0 mem	3 – 6	18/26	(mem)←(mem)+imm	×	×	×	×	×	×
		acc,imm	0 0 0 0 0 1 0 W		2 – 3	4	If W = 0, AL←AL+imm If W = 1, AW←AW+imm	×	×	×	×	×	×
	ADDC	reg,reg'	0 0 0 1 0 0 1 W	1 1 reg reg'	2	2	reg←reg+reg'+CY	×	×	×	×	×	×
		mem,reg	0 0 0 1 0 0 0 W	mod reg mem	2 – 4	16/24	(mem)←(mem)+reg+CY	×	×	×	×	×	×
		reg,mem	0 0 0 1 0 0 1 W	mod reg mem	2 – 4	11/15	reg←reg+(mem)+CY	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 s W	1 1 0 1 0 reg	3 – 4	4	reg←reg+imm+CY	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 s W	mod 0 1 0 mem	3 – 6	18/26	(mem)←(mem)+imm+CY	×	×	×	×	×	×
		acc,imm	0 0 0 1 0 1 0 W		2 – 3	4	If W = 0, AL←AL+imm+CY If W = 1, AW←AW+imm+CY	×	×	×	×	×	×
	SUB	reg,reg'	0 0 1 0 1 0 1 W	1 1 reg reg'	2	2	reg←reg–reg'	×	×	×	×	×	×
		mem,reg	0 0 1 0 1 0 0 W	mod reg mem	2 – 4	16/24	(mem)←(mem)–reg	×	×	×	×	×	×
		reg,mem	0 0 1 0 1 0 1 W	mod reg mem	2 – 4	11/15	reg←reg–(mem)	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 s W	1 1 1 0 1 reg	3 – 4	4	reg←reg–imm	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 s W	mod 1 0 1 mem	3 – 6	18/26	(mem)←(mem)–imm	×	×	×	×	×	×
		acc,imm	0 0 1 0 1 1 0 W		2 – 3	4	If W = 0, AL←AL–imm If W = 1, AW←AW–imm	×	×	×	×	×	×
	SUBC	reg,reg'	0 0 0 1 1 0 1 W	1 1 reg reg'	2	2	reg←reg–reg'–CY	×	×	×	×	×	×
		mem,reg	0 0 0 1 1 0 0 W	mod reg mem	2 – 4	16/24	(mem)←(mem)–reg–CY	×	×	×	×	×	×
		reg,mem	0 0 0 1 1 0 1 W	mod reg mem	2 – 4	11/15	reg←reg–(mem)–CY	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 s W	1 1 0 1 1 reg	3 – 4	4	reg←reg–imm–CY	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 s W	mod 0 1 1 mem	3 – 6	18/26	(mem)←(mem)–imm–CY	×	×	×	×	×	×
		acc,imm	0 0 0 1 1 1 0 W		2 – 3	4	If W = 0, AL←AL–imm–CY If W = 1, AW←AW–imm–CY	×	×	×	×	×	×

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
BCD operation instructions	ADD4S		0 0 0 0 1 1 1 1	0 0 1 0 0 0 0 0	2	19×n+7	dst BCD string←dst BCD string+src BCD string *	U	×	U	U	U	×
	SUB4S		0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 0	2	19×n+7	dst BCD string←dst BCD string−src BCD string *	U	×	U	U	U	×
	CMP4S		0 0 0 0 1 1 1 1	0 0 1 0 0 1 1 0	2	19×n+7	dst BCD string−src BCD string *	U	×	U	U	U	×
	ROL4	reg8	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 0 0	3	13							
		mem8	0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 0 0	3 – 5	28							
	ROR4	reg8	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 1 0	3	17							
		mem8	0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 1 0	3 – 5	32							
Increment-decrement instructions	INC	reg8	1 1 1 1 1 1 1 0	1 1 0 0 0 reg	2	2	reg8←reg8+1	×		×	×	×	×
		mem	1 1 1 1 1 1 1 W	mod 0 0 0 mem	2 – 4	16/24	(mem)←(mem)+1	×		×	×	×	×
		reg16	0 1 0 0 0 reg		1	2	reg16←reg16+1	×		×	×	×	×
	DEC	reg8	1 1 1 1 1 1 1 0	1 1 0 0 1 reg	2	2	reg8←reg8−1	×		×	×	×	×
		mem	1 1 1 1 1 1 1 W	mod 0 0 1 mem	2 – 4	16/24	(mem)←(mem)−1	×		×	×	×	×
		reg16	0 1 0 0 1 reg		1	2	reg16←reg16−1	×		×	×	×	×

n: 1/2 of the number of BCD digits

*: The number of BCD digits is given at CL register. It is possible to set the values 1 to 254.

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Multiplication instructions	MULU	reg8	1 1 1 1 0 1 1 0	1 1 1 0 0 reg	2	21 – 22	AW←AL×reg8 AH = 0: CY←0, V←0 AH ≠ 0: CY←1, V←1	U	×	×	U	U	U
		mem8	1 1 1 1 0 1 1 0	mod 1 0 0 mem	2 – 4	27 – 28	AW←AL×(mem8) AH = 0: CY←0, V←0 AH ≠ 0: CY←1, V←1	U	×	×	U	U	U
		reg16	1 1 1 1 0 1 1 1	1 1 1 0 0 reg	2	29 – 30	DW,AW←AW×reg16 DW = 0: CY←0, V←0 DW = 1: CY←1, V←1	U	×	×	U	U	U
		mem16	1 1 1 1 0 1 1 1	mod 1 0 0 mem	2–4	39 – 40	DW,AW←AW×(mem16) DW = 0: CY←0, V←0 DW = 1: CY←1, V←1	U	×	×	U	U	U
	MUL	reg8	1 1 1 1 0 1 1 0	1 1 1 0 1 reg	2	33 – 39	AW←AL×reg8 AH = sign extension of AL: CY←0, V←0 AH ≠ sign extension of AL: CY←1, V←1	U	×	×	U	U	U
		mem8	1 1 1 1 0 1 1 0	mod 1 0 1 mem	2 – 4	39 – 45	AW←AL×(mem8) AH = sign extension of AL: CY←0, V←0 AH ≠ sign extension of AL: CY←1, V←1	U	×	×	U	U	U
		reg16	1 1 1 1 0 1 1 1	1 1 1 0 1 reg	2	41 – 47	DW,AW←AW×reg16 DW = sign extension of AW: CY←0, V←0 DW ≠ sign extension of AW: CY←1, V←1	U	×	×	U	U	U
		mem16	1 1 1 1 0 1 1 1	mod 1 0 1 mem	2 – 4	51 – 57	DW,AW←AW×(mem16) DW = sign extension of AW: CY←0, V←0 DW ≠ sign extension of AW: CY←1, V←1	U	×	×	U	U	U
		reg16, (reg16',)* imm8	0 1 1 0 1 0 1 1	1 1 reg reg'	3	28 – 34	reg16←reg16'×imm8 Product ≤ 16-bit: CY←0, V←0 Product > 16-bit: CY←1, V←1	U	×	×	U	U	U
		reg16, mem16, imm8	0 1 1 0 1 0 1 1	mod reg mem	3 – 5	38 – 44	reg16←(mem16)×imm8 Product ≤ 16-bit: CY←0, V←0 Product > 16-bit: CY←1, V←1	U	×	×	U	U	U
		reg16, (reg16',)* imm16	0 1 1 0 1 0 0 1	1 1 reg reg'	4	36 – 42	reg16←reg16'×imm16 Product ≤ 16-bit: CY←0, V←0 Product > 16-bit: CY←1, V←1	U	×	×	U	U	U
		reg16, mem16, imm16	0 1 1 0 1 0 0 1	mod reg mem	4 – 6	46 – 52	reg16←(mem16)×imm16 Product ≤ 16-bit: CY←0, V←0 Product > 16-bit: CY←1, V←1	U	×	×	U	U	U

*: The 2nd operand can be omitted, in which case the same register as for the 1st operand is taken as specified.

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Unsigned division instructions	DIVU	reg8	1 1 1 1 0 1 1 0	1 1 1 1 0 reg	2	19	temp←AW If temp ÷ reg8 ≤ FFH, AH←temp%reg8, AL←temp ÷ reg8 If temp ÷ reg8 > FFH, TA←(001H, 000H), TC←(003H, 002H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA	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 If temp ÷ (mem8) ≤ FFH, AH←temp%(mem8), AL←temp ÷ (mem8) If temp ÷ (mem8) > FFH, TA←(001H, 000H), TC←(003H, 002H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA	U	U	U	U	U	U
		reg16	1 1 1 1 0 1 1 1	1 1 1 1 0 reg	2	25	temp←DW, AW If temp ÷ reg16 ≤ FFFFH, DW←temp%reg16, AW←temp ÷ reg16 If temp ÷ reg16 > FFFFH, TA←(001H, 000H), TC←(003H, 002H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA	U	U	U	U	U	U
		mem16	1 1 1 1 0 1 1 1	mod 1 1 0 mem	2 – 4	34	temp←DW, AW If temp ÷ (mem16) ≤ FFFFH, DW←temp%(mem16), AW←temp ÷ (mem16) If temp ÷ (mem16) > FFFFH, TA←(001H, 000H), TC←(003H, 002H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA	U	U	U	U	U	U

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Signed division instructions	DIV	reg8	1 1 1 1 0 1 1 0	1 1 1 1 1 reg	2	29 – 34	temp←AW If temp ÷ reg8 > 0 and temp ÷ reg8 ≤ 7FH, or temp ÷ reg8 < 0 and temp ÷ reg8 > 0 – 7FH – 1, AH←temp%reg8, AL←temp ÷ reg8 If temp ÷ reg8 > 0 and temp ÷ reg8 > 7FH, or temp ÷ reg8 < 0 and temp ÷ reg8 ≤ 0 – 7FH – 1, TA←(001H, 000H), TC←(003H, 002H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA	U	U	U	U	U	U
		mem8	1 1 1 1 0 1 1 0	mod 1 1 1 mem	2 – 4	34 – 39	temp←AW If temp ÷ (mem8) > 0 and temp ÷ (mem8) ≤ 7FH, or temp ÷ (mem8) < 0 and temp ÷ (mem8) > 0 – 7FH – 1, AH←temp%(mem8), AL←temp ÷ (mem8) If temp ÷ (mem8) > 0 and temp ÷ (mem8) > 7FH, or temp ÷ (mem8) < 0 and temp ÷ (mem8) ≤ 0 – 7FH – 1, TA←(001H, 000H), TC←(003H, 002H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA	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 If temp ÷ reg16 > 0 and temp ÷ reg16 ≤ 7FFFH, or temp ÷ reg16 < 0 and temp ÷ reg16 > 0 – 7FFFH – 1, DW←temp%reg16, AW←temp ÷ reg16 If temp ÷ reg16 > 0 and temp ÷ reg16 > 7FFFH, or temp ÷ reg16 < 0 and temp ÷ reg16 ≤ 0 – 7FFFH – 1, TA←(001H, 000H), TC←(003H, 002H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA	U	U	U	U	U	U
		mem16	1 1 1 1 0 1 1 1	mod 1 1 1 mem	2 – 4	47 – 52	temp←DW, AW If temp ÷ (mem16) > 0 and temp ÷ (mem16) ≤ 7FFFH, or temp ÷ (mem16) < 0 and temp ÷ (mem16) > 0 – 7FFFH – 1, DW←temp%(mem16), AW←temp ÷ (mem16) If temp ÷ (mem16) > 0 and temp ÷ (mem16) > 7FFFH, or temp ÷ (mem16) < 0 and temp ÷ (mem16) ≤ 0 – 7FFFH – 1, TA←(001H, 000H), TC←(003H, 002H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA	U	U	U	U	U	U

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
BCD auxiliary instructions	ADJBA		0 0 1 1 0 1 1 1		1	7	If AL ∧ 0FH > 9 or AC = 1, AL←AL+6 AH←AH+1, AC←1, CY←AC, AL←AL ∧ 0FH	×	×	U	U	U	U
	ADJ4A		0 0 1 0 0 1 1 1		1	3	If AL ∧ 0FH > 9 or AC = 1, AL←AL+6, AC←1 If AL > 9FH or CY = 1, AL←AL+60H, CY←1	×	×	U	×	×	×
	ADJBS		0 0 1 1 1 1 1 1		1	7	If AL ∧ 0FH > 9 or AC = 1, AL←AL−6, AH←AH−1, AC←1 CY←AC, AL←AL ∧ 0FH	×	×	U	U	U	U
	ADJ4S		0 0 1 0 1 1 1 1		1	3	If AL ∧ 0FH > 9 or AC =1, AL←AL−6, AC←1 If AL > 9FH or CY = 1, AL←AL−60H, CY←1	×	×	U	×	×	×
Data 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	×	×	×
	CVTDB		1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	2	7	AL←AH×0AH+AL, AH←0	U	U	U	×	×	×
	CVTBW		1 0 0 1 1 0 0 0		1	2	If AL < 80H, AH←0. Otherwise, AH←FFH.						
	CVTWL		1 0 0 1 1 0 0 1		1	4 – 5	If AW < 8000H, DW←0. Otherwise, DW←FFFFH.						
Compare instructions	CMP	reg,reg'	0 0 1 1 1 0 1 W	1 1 reg reg'	2	2	reg−reg'	×	×	×	×	×	×
		mem,reg	0 0 1 1 1 0 0 W	mod reg mem	2 – 4	11/15	(mem)−reg	×	×	×	×	×	×
		reg,mem	0 0 1 1 1 0 1 W	mod reg mem	2 – 4	11/15	reg−(mem)	×	×	×	×	×	×
		reg,imm	1 0 0 0 0 0 s W	1 1 1 1 1 reg	3 – 4	4	reg−imm	×	×	×	×	×	×
		mem,imm	1 0 0 0 0 0 s W	mod 1 1 1 mem	3 – 6	13/17	(mem)−imm	×	×	×	×	×	×
		acc,imm	0 0 1 1 1 1 0 W		2 – 3	4	If W = 0, AL − imm If W = 1, AW − imm	×	×	×	×	×	×
Complement operation instructions	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	×	×	×	×	×	×
		mem	1 1 1 1 0 1 1 W	mod 0 1 1 mem	2 – 4	16/24	(mem)←$\overline{\text{(mem)}}$+1	×	×	×	×	×	×

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Logical operation instructions	TEST	reg,reg'	1 0 0 0 0 1 0 W	1 1 reg' reg	2	2	reg∧reg'	U	0	0	×	×	×
		mem,reg reg,mem	1 0 0 0 0 1 0 W	mod reg mem	2 – 4	10/14	(mem)∧reg	U	0	0	×	×	×
		reg,imm	1 1 1 1 0 1 1 W	1 1 0 0 0 reg	3 – 4	4	reg∧imm	U	0	0	×	×	×
		mem,imm	1 1 1 1 0 1 1 W	mod 0 0 0 mem	3 – 6	11/15	(mem)∧imm	U	0	0	×	×	×
		acc,imm	1 0 1 0 1 0 0 W		2 – 3	4	If W = 0, AL∧imm8 If W = 1, AW∧imm16	U	0	0	×	×	×
	AND	reg,reg'	0 0 1 0 0 0 1 W	1 1 reg reg'	2	2	reg←reg∧reg'	U	0	0	×	×	×
		mem,reg	0 0 1 0 0 0 0 W	mod reg mem	2 – 4	16/24	(mem)←(mem)∧reg	U	0	0	×	×	×
		reg,mem	0 0 1 0 0 0 1 W	mod reg mem	2 – 4	11/15	reg←reg∧(mem)	U	0	0	×	×	×
		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	×	×	×
		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	×	×	×
		acc,imm	0 0 1 0 0 1 0 W		2 – 3	4	If W = 0, AL←AL∧imm8 If W = 1, AW←AW∧imm16	U	0	0	×	×	×
	OR	reg,reg'	0 0 0 0 1 0 0 W	1 1 reg reg'	2	2	reg←reg ∨ reg'	U	0	0	×	×	×
		mem,reg	0 0 0 0 1 0 0 W	mod reg mem	2 – 4	16/24	(mem)←(mem) ∨ reg	U	0	0	×	×	×
		reg,mem	0 0 0 0 1 0 1 W	mod reg mem	2 – 4	11/15	reg←reg ∨ (mem)	U	0	0	×	×	×
		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	×	×	×
		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	×	×	×
		acc,imm	0 0 0 0 1 1 0 W		2 – 3	4	If W = 0, AL←AL ∨ imm8 If W = 1, AW←AW ∨ imm16	U	0	0	×	×	×
	XOR	reg,reg'	0 0 1 1 0 0 1 W	1 1 reg reg'	2	2	reg←reg ∀ reg'	U	0	0	×	×	×
		mem,reg	0 0 1 1 0 0 0 W	mod reg mem	2 – 4	16/24	(mem)←(mem) ∀ reg	U	0	0	×	×	×
		reg,mem	0 0 1 1 0 0 1 W	mod reg mem	2 – 4	11/15	reg←reg ∀ (mem)	U	0	0	×	×	×
		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	×	×	×
		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	×	×	×
		acc,imm	0 0 1 1 0 1 0 W		2 – 3	4	If W = 0, AL←AL ∀ imm8 If W = 1, AW←AW ∀ imm16	U	0	0	×	×	×

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Bit manipulation 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 Bit NO.CL of reg8 = 1 : Z←0	U	0	0	U	U	×
		mem8,CL		0 0 0 0 mod 0 0 0 mem	3 – 5	8	Bit NO.CL of (mem8) = 0 : Z←1 Bit NO.CL of (mem8) = 1 : Z←0	U	0	0	U	U	×
		reg16,CL		0 0 0 1 1 1 0 0 0 reg	3	3	Bit NO.CL of reg16 = 0 : Z←1 Bit NO.CL of reg16 = 1 : Z←0	U	0	0	U	U	×
		mem16,CL		0 0 0 1 mod 0 0 0 mem	3 – 5	12	Bit NO.CL of (mem16) = 0 : Z←1 Bit NO.CL of (mem16) = 1 : Z←0	U	0	0	U	U	×
		reg8,imm3		1 0 0 0 1 1 0 0 0 reg	4	4	Bit NO.imm3 of reg8 = 0 : Z←1 Bit NO.imm3 of reg8 = 1 : Z←0	U	0	0	U	U	×
		mem8,imm3		1 0 0 0 mod 0 0 0 mem	4 – 6	9	Bit NO.imm3 of (mem8) = 0 : Z←1 Bit NO.imm3 of (mem8) = 1 : Z←0	U	0	0	U	U	×
		reg16,imm4		1 0 0 1 1 1 0 0 0 reg	4	4	Bit NO.imm4 of reg16 = 0 : Z←1 Bit NO.imm4 of reg16 = 1 : Z←0	U	0	0	U	U	×
		mem16,imm4		1 0 0 1 mod 0 0 0 mem	4 – 6	13	Bit NO.imm4 of (mem16) = 0 : Z←1 Bit NO.imm4 of (mem16) = 1 : Z←0	U	0	0	U	U	×
	NOT1	reg8,CL		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 1 1 0 mod 0 0 0 mem	3 – 5	13	Bit NO.CL of (mem8)←$\overline{\text{Bit NO.CL of (mem8)}}$						
		reg16,CL		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 1 1 1 mod 0 0 0 mem	3 – 5	21	Bit NO.CL of (mem16)←$\overline{\text{Bit NO.CL of (mem16)}}$						
		reg8,imm3		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		1 1 1 0 mod 0 0 0 mem	4 – 6	14	Bit NO.imm3 of (mem8)←$\overline{\text{Bit NO.imm3 of (mem8)}}$						
		reg16,imm4		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		1 1 1 1 mod 0 0 0 mem	4 – 6	22	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←$\overline{\text{CY}}$		×				

Instruction Group	Mnemonic	Operand	Operation Code														Bytes	Clocks	Operation	Flag					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Bit manipulation instructions	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	1	0	mod	0	0	0	mem				3 - 5	14	Bit NO.CL of (mem8)←0						
		reg16,CL					0	0	1	1	1	1	0	0	0	reg			3	5	Bit NO.CL of reg16←0						
		mem16,CL					0	0	1	1	mod	0	0	0	mem				3 - 5	22	Bit NO.CL of (mem16)←0						
		reg8,imm3					1	0	1	0	1	1	0	0	0	reg			4	6	Bit NO.imm3 of reg8←0						
		mem8,imm3					1	0	1	0	mod	0	0	0	mem				4 - 6	15	Bit NO.imm3 of (mem8)←0						
		reg16,imm4					1	0	1	1	1	1	0	0	0	reg			4	6	Bit NO.imm4 of reg16←0						
		mem16,imm4					1	0	1	1	mod	0	0	0	mem				4 - 6	23	Bit NO.imm4 of (mem16)←0						
	SET1	reg8,CL					0	1	0	0	1	1	0	0	0	reg			3	4	Bit NO.CL of reg8←1						
		mem8,CL					0	1	0	0	mod	0	0	0	mem				3 - 5	13	Bit NO.CL of (mem8)←1						
		reg16,CL					0	1	0	1	1	1	0	0	0	reg			3	4	Bit NO.CL of reg16←1						
		mem16,CL					0	1	0	1	mod	0	0	0	mem				3 - 5	21	Bit NO.CL of (mem16)←1						
		reg8,imm3					1	1	0	0	1	1	0	0	0	reg			4	5	Bit NO.imm3 of reg8←1						
		mem8,imm3					1	1	0	0	mod	0	0	0	mem				4 - 6	14	Bit NO.imm3 of (mem8)←1						
		reg16,imm4					1	1	0	1	1	1	0	0	0	reg			4	5	Bit NO.imm4 of reg16←1						
		mem16,imm4					1	1	0	1	mod	0	0	0	mem				4 - 6	22	Bit NO.imm4 of (mem16)←1						
							2nd Byte*				3rd Byte*																

*: 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						

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Shift instructions	SHL	reg,1	1 1 0 1 0 0 0 W	1 1 1 0 0 reg	2	6	CY←MSB of reg, reg←reg×2 If MSB of reg ≠ CY, V←1 If MSB of reg = CY, V←0	U	×	×	×	×	×
		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)×2 If MSB of (mem) ≠ CY, V←1 If MSB of (mem) = CY, V←0	U	×	×	×	×	×
		reg,CL	1 1 0 1 0 0 1 W	1 1 1 0 0 reg	2	7 + n	While temp←CL, temp ≠ 0, repeats the consecutive operation CY←MSB of reg, reg←reg×2 temp←temp–1	U	×	U	×	×	×
		mem,CL	1 1 0 1 0 0 1 W	mod 1 0 0 mem	2 – 4	19/27 + n	While temp←CL, temp ≠ 0, repeats the consecutive operation CY←MSB of (mem), (mem)←(mem)×2 temp←temp–1	U	×	U	×	×	×
		reg,imm8	1 1 0 0 0 0 0 W	1 1 1 0 0 reg	3	7 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←MSB of reg, reg←reg×2 temp←temp–1	U	×	U	×	×	×
		mem,imm8	1 1 0 0 0 0 0 W	mod 1 0 0 mem	3 – 5	19/27 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←MSB of (mem), (mem)←(mem)×2 temp←temp–1	U	×	U	×	×	×

n: Number of shifts

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Shift instructions	SHR	reg,1	1 1 0 1 0 0 0 W	1 1 1 0 1 reg	2	6	CY←LSB of reg, reg←reg+2 MSB of reg ≠ the consecutive bit of MSB of reg: V←1 MSB of reg = the consecutive bit of MSB of reg: V←0	U	×	×	×	×	×
		mem,1	1 1 0 1 0 0 0 W	mod 1 0 1 mem	2 – 4	16/24	CY←LSB of (mem), (mem)←(mem)+2 MSB of (mem) ≠ the consecutive bit of MSB of (mem): V←1 MSB of (mem) = the consecutive bit of MSB of (mem): V←0	U	×	×	×	×	×
		reg,CL	1 1 0 1 0 0 1 W	1 1 1 0 1 reg	2	7 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation CY←LSB of reg, reg←reg+2 temp←temp–1	U	×	U	×	×	×
		mem,CL	1 1 0 1 0 0 1 W	mod 1 0 1 mem	2 – 4	19/27 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation CY←LSB of (mem), (mem)←(mem)+2 temp←temp–1	U	×	U	×	×	×
		reg,imm8	1 1 0 0 0 0 0 W	1 1 1 0 1 reg	3	7 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←LSB of reg, reg←reg+2 temp←temp–1	U	×	U	×	×	×
		mem,imm8	1 1 0 0 0 0 0 W	mod 1 0 1 mem	3 – 5	19/27 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←LSB of (mem), (mem)←(mem)+2 temp←temp–1	U	×	U	×	×	×
	SHRA	reg,1	1 1 0 1 0 0 0 W	1 1 1 1 1 reg	2	6	CY←LSB of reg, reg←reg+2, V←0 MSB of operand does not change.	U	×	0	×	×	×
		mem,1	1 1 0 1 0 0 0 W	mod 1 1 1 mem	2 – 4	16/24	CY←LSB of (mem), (mem)←(mem)+2, V←0 MSB of operand does not change.	U	×	0	×	×	×
		reg,CL	1 1 0 1 0 0 1 W	1 1 1 1 1 reg	2	7 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation CY←LSB of reg, reg←reg+2 temp←temp–1, MSB of operand does not change.	U	×	U	×	×	×
		mem,CL	1 1 0 1 0 0 1 W	mod 1 1 1 mem	2 – 4	19/27 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation CY←LSB of (mem), (mem)←(mem)+2 temp←temp–1, MSB of operand does not change.	U	×	U	×	×	×
		reg,imm8	1 1 0 0 0 0 0 W	1 1 1 1 1 reg	3	7 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←LSB of reg, reg←reg+2 temp←temp–1, MSB of operand does not change.	U	×	U	×	×	×
		mem,imm8	1 1 0 0 0 0 0 W	mod 1 1 1 mem	3 – 5	19/27 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←LSB of (mem), (mem)←(mem)+2 temp←temp–1, MSB of operand does not change.	U	×	U	×	×	×

n: Number of shifts

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag						
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z	
Rotate instructions	ROL	reg,1	1 1 0 1 0 0 0 W	1 1 0 0 0 reg	2	6	CY←MSB of reg, reg←reg×2+CY MSB of reg ≠ CY: V←1 MSB of reg = CY: V←0		×	×				
		mem,1	1 1 0 1 0 0 0 W	mod 0 0 0 mem	2 – 4	16/24	CY←MSB of (mem), (mem)←(mem)×2+CY MSB of (mem) ≠ CY: V←1 MSB of (mem) = CY: V←0		×	×				
		reg,CL	1 1 0 1 0 0 1 W	1 1 0 0 0 reg	2	7 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation CY←MSB of reg, reg←reg×2+CY temp←temp–1		×	U				
		mem,CL	1 1 0 1 0 0 1 W	mod 0 0 0 mem	2 – 4	19/27 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation CY←MSB of (mem), (mem)←(mem)×2+CY temp←temp–1		×	U				
		reg,imm8	1 1 0 0 0 0 0 W	1 1 0 0 0 reg	3	7 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←MSB of reg, reg←reg×2+CY temp←temp–1		×	U				
		mem,imm8	1 1 0 0 0 0 0 W	mod 0 0 0 mem	3 – 5	19/27 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←MSB of (mem), (mem)←(mem)×2+CY temp←temp–1		×	U				
	ROR	reg,1	1 1 0 1 0 0 0 W	1 1 0 0 1 reg	2	6	CY←LSB of reg, reg←reg÷2 MSB of reg←CY MSB of reg ≠ the consecutive bit of MSB of reg: V←1 MSB of reg = the consecutive bit of MSB of reg: V←0		×	×				
		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) ≠ the consecutive bit of MSB of (mem): V←1 MSB of (mem) = the consecutive bit of MSB of (mem): V←0		×	×				
		reg,CL	1 1 0 1 0 0 1 W	1 1 0 0 1 reg	2	7 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation CY←LSB of reg, reg←reg÷2 MSB of reg←CY temp←temp–1		×	U					★
		mem,CL	1 1 0 1 0 0 1 W	mod 0 0 1 mem	2 – 4	19/27 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation CY←LSB of (mem), (mem)←(mem)÷2 MSB of (mem)←CY temp←temp–1		×	U					★
		reg,imm8	1 1 0 0 0 0 0 W	1 1 0 0 1 reg	3	7 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←LSB of reg, reg←reg÷2 MSB of reg←CY temp←temp–1		×	U					★
		mem,imm8	1 1 0 0 0 0 0 W	mod 0 0 1 mem	3 – 5	19/27 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation CY←LSB of (mem), (mem)←(mem)÷2 MSB of (mem)←CY temp←temp–1		×	U					★

n: Number of shifts

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag						
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z	
Rotate instructions	ROLC	reg,1	1 1 0 1 0 0 0 W	1 1 0 1 0 reg	2	8	tmpcy←CY, CY←MSB of reg reg←regx2+tmpcy MSB of reg ≠ CY: V←1 MSB of reg = CY: V←0		×	×				
		mem,1	1 1 0 1 0 0 0 W	mod 0 1 0 mem	2 – 4	16/24	tmpcy←CY, CY←MSB of (mem) (mem)←(mem)x2+tmpcy MSB of (mem) ≠ CY: V←1 MSB of (mem) = CY: V←0		×	×				
		reg,CL	1 1 0 1 0 0 1 W	1 1 0 1 0 reg	2	7 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation tmpcy←CY, CY←MSB of reg reg←regx2+tmpcy temp←temp–1		×	U				★
		mem,CL	1 1 0 1 0 0 1 W	mod 0 1 0 mem	2 – 4	19/27 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation tmpcy←CY, CY←MSB of (mem) (mem)←(mem)x2+tmpcy temp←temp–1		×	U				★
		reg,imm8	1 1 0 0 0 0 0 W	1 1 0 1 0 reg	3	7 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation tmpcy←CY, CY←MSB of reg reg←regx2+tmpcy temp←temp–1		×	U				★
		mem,imm8	1 1 0 0 0 0 0 W	mod 0 1 0 mem	3 – 5	19/27 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation tmpcy←CY, CY←MSB of (mem) (mem)←(mem)x2+tmpcy temp←temp–1		×	U				★

n: Number of shifts

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag						
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z	
Rotate instructions	RORC	reg,1	1 1 0 1 0 0 0 W	1 1 0 1 1 reg	2	8	tmpcy←CY, CY←LSB of reg reg←reg+2 MSB of reg←tmpcy MSB of reg ≠ the consecutive bit of MSB of reg: V←1 MSB of reg = the consecutive bit of MSB of reg: V←0		×	×				
		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) ≠ the consecutive bit of MSB of (mem): V←1 MSB of (mem) = the consecutive bit of MSB of (mem): V←0		×	×				
		reg,CL	1 1 0 1 0 0 1 W	1 1 0 1 1 reg	2	7 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation tmpcy←CY, CY←LSB of reg reg←reg+2 MSB of reg←tmpcy temp←temp–1		×	U				★
		mem,CL	1 1 0 1 0 0 1 W	mod 0 1 1 mem	2 – 4	19/27 + n	While temp←CL and temp ≠ 0, repeats the consecutive operation tmpcy←CY, CY←LSB of (mem) (mem)←(mem)+2 MSB of (mem)←tmpcy temp←temp–1		×	U				★
		reg,imm8	1 1 0 0 0 0 0 W	1 1 0 1 1 reg	3	7 + n	While temp←imm8 and temp ≠ 0, repeats the consecutive operation tmpcy←CY, CY←LSB of reg reg←reg+2 MSB of reg←tmpcy temp←temp–1		×	U				★
		mem,imm8	1 1 0 0 0 0 0 W	mod 0 1 1 mem	3 – 5	19/27 + n	While temp←imm8, temp ≠ 0, repeats the consecutive operation tmpcy←CY, CY←LSB of (mem) (mem)←(mem)+2 MSB of (mem)←tmpcy temp←temp–1		×	U				★

n: Number of shifts

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Subroutine control instructions	CALL	near-proc	1 1 1 0 1 0 0 0		3	20	SP←SP−2, (SP+1, SP)←PC PC←PC+disp						
		regptr16	1 1 1 1 1 1 1 1	1 1 0 1 0 reg	2	18	SP←SP−2, (SP+1, SP)←PC PC←regptr16						
		memptr16	1 1 1 1 1 1 1 1	mod 0 1 0 mem	2 – 4	31	TA←(memptr16) SP←SP−2, (SP+1, SP)←PC, PC←TA						
		far-proc	1 0 0 1 1 0 1 0		5	29	SP←SP−2, (SP+1, SP)←PS, PS←seg SP←SP−2, (SP+1, SP)←PC, PC←offset						
		memptr32	1 1 1 1 1 1 1 1	mod 0 1 1 mem	2 - 4	47	TA←(memptr32), TB←(memptr32+2) SP←SP−2, (SP+1, SP)←PS, PS←TB SP←SP−2, (SP+1, SP)←PC, PC←TA						
	RET		1 1 0 0 0 0 1 1		1	19	PC←(SP+1, SP) SP←SP+2						
		pop-value	1 1 0 0 0 0 1 0		3	24	PC←(SP+1, SP) SP←SP+2, SP←SP+pop-value						
			1 1 0 0 1 0 1 1		1	29	PC←(SP+1, SP) PS←(SP+3, SP+2) SP←SP+4						
		pop-value	1 1 0 0 1 0 1 0		3	32	PC←(SP+1, SP) PS←(SP+3, SP+2) SP←SP+4, SP←SP+pop-value						

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag						
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z	
Stack manipulation instructions	PUSH	mem16	1 1 1 1 1 1 1 1	mod 1 1 0 mem	2 – 4	26	SP←SP–2 (SP+1, SP)←(mem16)							
		reg16	0 1 0 1 0 reg		1	12	SP←SP–2 (SP+1, SP)←reg16							
		sreg	0 0 0 sreg 1 1 0		1	12	SP←SP–2 (SP+1, SP)←sreg							
		PSW	1 0 0 1 1 1 0 0		1	12	SP←SP–2 (SP+1, SP)←PSW							
		R	0 1 1 0 0 0 0 0		1	67	Push registers on the stack							
		imm8	0 1 1 0 1 0 1 0		2	11	(SP–1, SP–2)←Sign extension of imm8 SP←SP–2							★
		imm16	0 1 1 0 1 0 0 0		3	12	(SP–1, SP–2)←imm16 SP←SP–2							★
	POP	mem16	1 0 0 0 1 1 1 1	mod 0 0 0 mem	2 – 4	25	SP←SP+2 (mem16)←(SP–1, SP–2)							★
		reg16	0 1 0 1 1 reg		1	12	SP←SP+2 reg16←(SP–1, SP–2)							★
		sreg	0 0 0 sreg 1 1 1		1	12	SP←SP+2 sreg: SS, DS0, DS1 sreg←(SP–1, SP–2)							★
		PSW	1 0 0 1 1 1 0 1		1	12	SP←SP+2 PSW←(SP–1, SP–2)	R	R	R	R	R	R	★
		R	0 1 1 0 0 0 0 1		1	75	Pop registers from the stack							
	PREPARE	imm16, imm8	1 1 0 0 1 0 0 0		4	*	Prepare New Stack Frame							
	DISPOSE		1 1 0 0 1 0 0 1		1	10	Dispose of Stack Frame							
Branch instructions	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	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	35	PS←(memptr32+2) PC←(memptr32)							

*: If imm8 = 0, 16
If imm8 ≥ 1, 23+16(imm8–1)

Instruction Group	Mnemonic	Operand	Operation Code	Bytes	Clocks[Note]	Operation	Flag
			7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0				AC CY V P S Z
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	short-label	0 0 0 1	2	14/4	if V = 0 PC←PC+ext-disp8	
	BC BL	short-label	0 0 1 0	2	14/4	if CY = 1 PC←PC+ext-disp8	
	BNC BNL	short-label	0 0 1 1	2	14/4	if CY = 0 PC←PC+ext-disp8	
	BE BZ	short-label	0 1 0 0	2	14/4	if Z = 1 PC←PC+ext-disp8	
	BNE BNZ	short-label	0 1 0 1	2	14/4	if Z = 0 PC←PC+ext-disp8	
	BNH	short-label	0 1 1 0	2	14/4	if CY ∨ Z = 1 PC←PC+ext-disp8	
	BH	short-label	0 1 1 1	2	14/4	if CY ∨ Z = 0 PC←PC+ext-disp8	
	BN	short-label	1 0 0 0	2	14/4	if S = 1 PC←PC+ext-disp8	
	BP	short-label	1 0 0 1	2	14/4	if S = 0 PC←PC+ext-disp8	
	BPE	short-label	1 0 1 0	2	14/4	if P = 1 PC←PC+ext-disp8	
	BPO	short-label	1 0 1 1	2	14/4	if P = 0 PC←PC+ext-disp8	
	BLT	short-label	1 1 0 0	2	14/4	if S ∀ V = 1 PC←PC+ext-disp8	
	BGE	short-label	1 1 0 1	2	14/4	if S ∀ V = 0 PC←PC+ext-disp8	
	BLE	short-label	1 1 1 0	2	14/4	if (S ∀ V) ∨ Z = 1 PC←PC+ext-disp8	
	BGT	short-label	1 1 1 1	2	14/4	if (S ∀ V) ∨ Z = 0 PC←PC+ext-disp8	
	DBNZNE	short-label	1 1 1 0 0 0 0 0	2	14/5	CW = CW−1 if Z = 0 and CW ≠ 0 PC←PC+ext-disp8	
	DBNZE	short-label	0 0 0 1	2	14/5	CW = CW−1 if Z = 1 and CW ≠ 0 PC←PC+ext-disp8	
	DBNZ	short-label	0 0 1 0	2	13/5	CW = CW−1 if CW≠0 PC←PC+ext-disp8	
	BCWZ	short-label	0 0 1 1	2	13/5	if CW = 0 PC←PC+ext-disp8	

Note Condition judgement: True/False

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
Interrupt instructions	BRK	3	1 1 0 0 1 1 0 0		1	50	TA←(00DH, 00CH), TC←(00FH, 00EH) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA						
		imm8 (≠ 3)	1 1 0 0 1 1 0 1		2	50	TA←(4n+1, 4n), TC←(4n+3, 4n+2) n = imm8 SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA						
	BRKV		1 1 0 0 1 1 1 0		1	Note 1	If V = 1, TA←(011H, 010H), TC←(013H, 012H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA						
	RETI		1 1 0 0 1 1 1 1		1	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	50	TA←(4n+1, 4n), TC←(4n+3, 4n+2) n = imm8 SP←SP–2, (SP+1, SP)←PSW, MD←0 Enables MD to be written SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA						
	CHKIND	reg16,mem32	0 1 1 0 0 0 1 0	mod reg mem	2 – 4	Note 2	If (mem32) > reg16 or (mem32 + 2) < reg16, TA←(015H, 014H), TC←(017H, 016H) SP←SP–2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP–2, (SP+1, SP)←PS, PS←TC SP←SP–2, (SP+1, SP)←PC, PC←TA						

Notes 1. If V=1, 52
If V=0, 3

2. If interrupt conditions are satisfied, 73-76
If interrupt conditions are not satisfied, 26

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
CPU control instructions	HALT		1 1 1 1 0 1 0 0		1	2	CPU Halt						
	POLL		1 0 0 1 1 0 1 1		1	2 + 5n	Poll and wait n: Number of $\overline{\text{POLL}}$ pin samplings						
	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	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	15	data bus←(mem)						
	NOP		1 0 0 1 0 0 0 0		1	3	No Operation						
	*		0 0 1 sreg 1 1 0		1	2	Segment overlaid prefix						

*: DS0:, DS1:, PS:, SS:

Instruction Group	Mnemonic	Operand	Operation Code		Bytes	Clocks	Operation	Flag					
			7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0				AC	CY	V	P	S	Z
8080	RETEM		1 1 1 0 1 1 0 1	1 1 1 1 1 1 0 1	2	39	PC←(SP+1, SP), PS←(SP+3, SP+2), PSW←(SP+5, SP+4), SP←SP+6, disable MD to be written	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	58	TA←(4n+1, 4n), TC←(4n+3, 4n+2) n = imm8 SP←SP−2, (SP+1, SP)←PSW, MD←1 SP←SP−2, (SP+1, SP)←PS, PS←TC SP←SP−2, (SP+1, SP)←PC, PC←TA						

9. ELECTRICAL SPECIFICATIONS

Absolute Maximum Rating (T_a = 25 °C)

Parameter	Symbol	Conditions	Rating	Unit
Power supply voltage	V_{DD}		−0.5 to +7.0	V
Input voltage	V_I	V_{DD} = 5 V ±10 %	−0.5 to V_{DD} +0.3	V
★ CLK input voltage	V_K		−0.5 to V_{DD} +1.0	V
Output voltage	V_O		−0.5 to V_{DD} +0.3	V
Operating ambient temperature	T_{opt}		−40 to +85	°C
Storage temperature	T_{stg}		−65 to +150	°C

Cautions 1. Do not connect output (and bidirectional) pins each other. Do not connect output (or bidirectional) pins directly to the V_{DD}, V_{CC}, or GND line. However, open drain pin and open collector pins can be directly connected to V_{DD}, V_{CC}, or GND line. If timing design is made so that so signal conflict occurs, three-state pins can also be connected directly to three-state pins of external circuit.

2. Exposure to Absolute Maximum Ratings for extended periods may affect device reliability; exceeding the ratings could cause permanent damage. The parameters apply independently. The device should be operated within the limits specified under DC and AC Characteristics.

DC Characteristics (μPD70108-5 T_a = −40 to +85 °C, V_{DD} = 5 V ±10 %)
(μPD70108-8 T_a = −10 to +70 °C, V_{DD} = 5 V ±5 %)
(μPD70108-10 T_a = −10 to +70 °C, V_{DD} = 5 V ±5 %)

Parameter	Symbol	Conditions		MIN.	TYP.	MAX.	Unit
Input voltage, high	V_{IH}			2.2		V_{DD}+0.3	V
Input voltage, low	V_{IL}			−0.5		0.8	V
CLK input voltage, high	V_{KH}			3.9		V_{DD}+1.0	V
CLK input voltage, low	V_{KL}			−0.5		0.6	V
Output voltage, high	V_{OH}	I_{OH} = −400 μA		0.7V_{DD}			V
Output voltage, low	V_{OL}	I_{OL} = 2.5 mA				0.4	V
Input leakage current, high	I_{LIH}	V_I = V_{DD}				10	μA
Input leakage current, low	I_{LIL}	V_I = 0 V				−10	μA
Output leakage current, high	I_{LOH}	V_O = V_{DD}				10	μA
Output leakage current, low	I_{LOL}	V_O = 0 V				−10	μA
HLDRQ input current, high	I_{HQH}	V_I = V_{DD}				10	μA
HLDRQ input current, low	I_{HQL}	V_I = 0 V				−0.5	mA
Power supply current	I_{DD}	Operating	70108-5		30	60	mA
			70108-8		45	80	mA
			70108-10		60	100	mA
		Standby	70108-5		5	10	mA
			70108-8		6	12	mA
			70108-10		7	14	mA

★ **Remark** TYP. value is reference at T_a = 25 °C and V_{DD} = 5.0 V.

Capacitance (T_a = 25 °C, V_{DD} = 0 V)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Input capacitance	C_I	f_c = 1 MHz			15	pF
I/O capacitance	C_{IO}	Unmeasured pins returned to 0 V			15	pF

AC Characteristics (μPD70108-5 T_a = −40 to +85 °C, V_{DD} = 5 V ±10 %)
(μPD70108-8 T_a = −10 to +70 °C, V_{DD} = 5 V ±5 %)
(μPD70108-10 T_a = −10 to +70 °C, V_{DD} = 5 V ±5 %)

Common to large/small scales

Parameter	Symbol	Conditions	70108-5 MIN.	70108-5 MAX.	70108-8 MIN.	70108-8 MAX.	70108-10 MIN.	70108-10 MAX.	Unit
Clock cycle	t_{CYK}		200	500	125	500	100	500	ns
Clock pulse high-level width	t_{KKH}	V_{KH} =3.0 V	69		44		41 39 Note		ns
Clock pulse low-level width	t_{KKL}	V_{KL} = 1.5 V	90		60		49		ns
Clock rise time	t_{KR}	1.5 to 3.0 V		10		10		5	ns
Clock fall time	t_{KF}	3.0 to 1.5 V		10		10		5	ns
RESET release delay time	t_{DVRST}	V_{DD} = 4.5 V	1		1		1		μs
RESET setup time (to CLK ↑)	t_{SRSTK}		15		15		15		ns
RESET hold time (from CLK ↑)	t_{HKRST}		15		15		15		ns
RESET high-level width	t_{WRSTH}		4 t_{CYK}		4 t_{CYK}		4 t_{CYK}		ns
READY inactive setup time (to CLK ↓)	t_{SRYLK}		−8		−8		−10		ns
READY inactive hold time (from CLK ↑)	t_{HKRYH}		30		20		20		ns
READY active setup time (to CLK ↑)	t_{SRYHK}		t_{KKL}−8		t_{KKL}−8		t_{KKL}−10		ns
READY active hold time (from CLK ↑)	t_{HKRYL}		30		20		20		ns
Data setup time (to CLK ↓)	t_{SDK}		30		20		10		ns
Data hold time (from CLK ↓)	t_{HKD}		10		10		10		ns
NMI, INT, $\overline{\text{POLL}}$ setup time (to CLK ↑)	t_{SIK}		30		15		15		ns
Input rise time (except CLK)	t_{IR}	0.8 to 2.2 V		20		20		20	ns
Input fall time (except CLK)	t_{IF}	2.2 to 0.8 V		12		12		12	ns
Output rise time	t_{OR}	0.8 to 2.2 V		20		20		20	ns
Output fall time	t_{OF}	2.2 to 0.8 V		12		12		12	ns

Note Applied only when using the μPD70108GC-10-3B6 and the μPD70108L-10.

AC Characteristics (cont'd)

Small scale

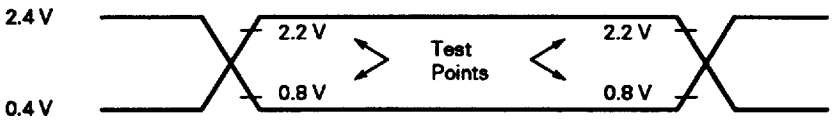
Parameter	Symbol	Conditions	70108-5 MIN.	70108-5 MAX.	70108-8 MIN.	70108-8 MAX.	70108-10 MIN.	70108-10 MAX.	Unit
Address delay time from CLK↓	t_{DKA}	C_L = 100 pF	10	90	10	60	10	48	ns
Address hold time from CLK↓	t_{HKA}		10		10		10		ns
PS delay time from CLK↓	t_{DKP}		10	90	10	60	10	50	ns
PS float delay time from CLK↑	t_{FKP}		10	80	10	60	10	50	ns
Address setup time (to ASTB↓)	t_{SAST}		t_{KKL}–60		t_{KKL}–30		t_{KKL}–30		ns
Address float delay time from CLK↓	t_{FKA}		t_{HKA}	80	t_{HKA}	60	t_{HKA}	50	ns
ASTB↑ delay time from CLK↓	t_{DKSTH}			80		50		40	ns
ASTB↓ delay time from CLK↑	t_{DKSTL}			85		55		45	ns
ASTB high-level width	t_{STST}		t_{KKL}–20		t_{KKL}–10		t_{KKL}–10		ns
Address hold time from ASTB↓	t_{HSTA}		t_{KKH}–10		t_{KKH}–10		t_{KKH}–10		ns
Control delay time from CLK	t_{DKCT}		10	110	10	65	10	55	ns
$\overline{RD}$↓ from address float	t_{AFRL}		0		0		0		ns
$\overline{RD}$↓ delay time from CLK↓	t_{DKRL}		10	165	10	80	10	70	ns
$\overline{RD}$↑ delay time from CLK↓	t_{DKRH}		10	150	10	80	10	60	ns
Address delay time from $\overline{RD}$↑	t_{DRHA}		t_{CYK}–45		t_{CYK}–40		t_{CYK}–35		ns
$\overline{RD}$ low-level width	t_{RR}		2t_{CYK}–75		2t_{CYK}–50		2t_{CYK}–40		ns
Data output delay time from CLK↓	t_{DKD}		10	90	10	60	10	50	ns
Data float delay time from CLK↓	t_{FKD}		10	80	10	60	10	50	ns
$\overline{WR}$ low-level width	t_{WW}		2t_{CYK}–60		2t_{CYK}–40		2t_{CYK}–35		ns
HLDRQ setup time (to CLK↑)	t_{SHOK}		35		20		20		ns
HLDAK delay time from CLK↓	t_{DKHA}		10	160	10	100	10	60	ns
$\overline{BUFEN}$↑ from $\overline{WR}$↑	t_{WCT}		t_{KKL}–20		t_{KKL}–20		t_{KKL}–20		ns

AC Characteristics (cont'd)

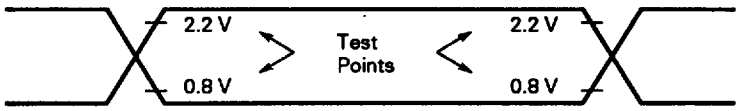
Large scale

Parameter	Symbol	Conditions	70108-5 MIN.	70108-5 MAX.	70108-8 MIN.	70108-8 MAX.	70108-10 MIN.	70108-10 MAX.	Unit
Address delay time from CLK↓	t_{DKA}	C_L = 100 pF	10	90	10	60	10	48	ns
Address hold time from CLK↓	t_{HKA}		10		10		10		ns
PS delay time from CLK↓	t_{DKP}		10	90	10	60	10	50	ns
PS float delay time from CLK↑	t_{FKP}		10	80	10	60	10	50	ns
Address float delay time from CLK↓	t_{FKA}		t_{HKA}	80	t_{HKA}	60	t_{HKA}	50	ns
Address delay time from $\overline{RD}$↑	t_{DRHA}		t_{CYK}–45		t_{CYK}–40		t_{CYK}–35		ns
ASTB↑ delay time from BS↓	t_{DBST}			15		15		15	ns
BS↓ delay time from CLK↑	t_{DKBL}		10	110	10	60	10	50	ns
BS↑ delay time from CLK↓	t_{DKBH}		10	130	10	65	10	50	ns
$\overline{RD}$↓ delay time from address float	t_{DAFRL}		0		0		0		ns
$\overline{RD}$↓ delay time from CLK↓	t_{DKRL}		10	165	10	80	10	70	ns
$\overline{RD}$↑ delay time from CLK↓	t_{DKRH}		10	150	10	80	10	60	ns
$\overline{RD}$ low-level width	t_{RR}		$2t_{CYK}$–75		$2t_{CYK}$–50		$2t_{CYK}$–40		ns
Data output delay time from CLK↓	t_{DKD}		10	90	10	60	10	50	ns
Data float delay time from CLK↓	t_{FKD}		10	80	10	60	10	50	ns
$\overline{AK}$ delay time from CLK↓	t_{DKAK}			70		50		40	ns
$\overline{RQ}$ setup time (to CLK↑)	t_{SRQK}		20		10		9		ns
$\overline{RQ}$ hold time (from CLK↓)	t_{HKRQ1}		0		0		0		ns
$\overline{RQ}$ hold time (from CLK↑)	t_{HKRQ2}		40		30		20		ns

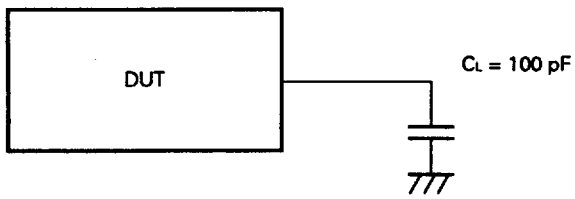
AC Test Input Waveform (Except CLK)



AC Test Output Test Points

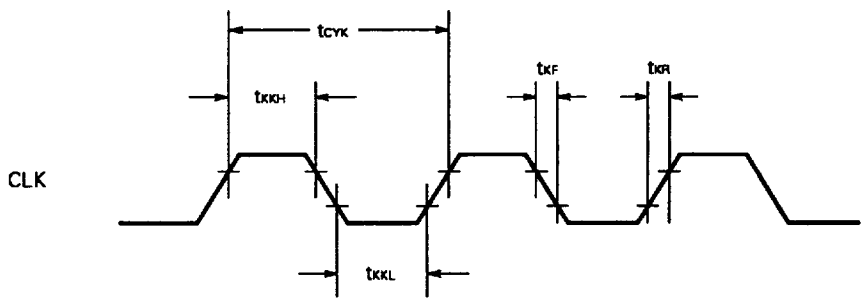


Load Condition

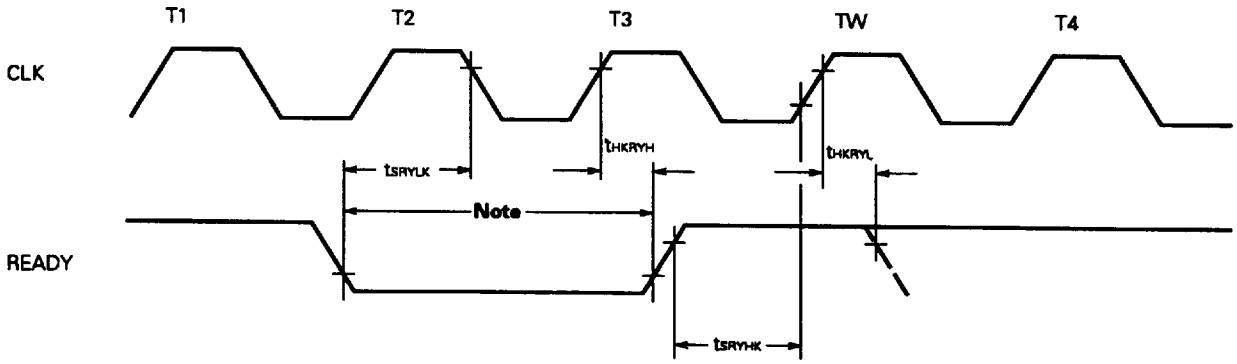


Caution If load capacitance exceeds 100 pF due to the configuration of circuits, lower the load capacitance to 100 pF or less by inserting a buffer, etc.

Clock Timing

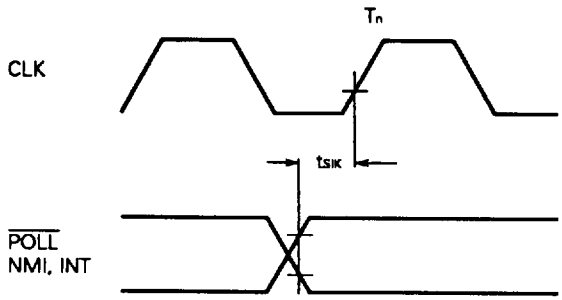


Wait (Ready) Timing

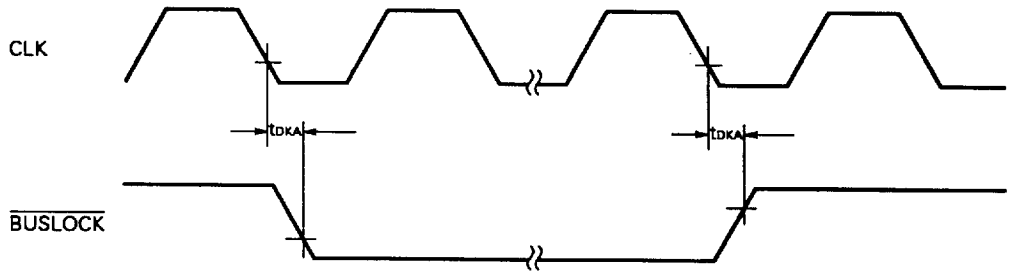


Note It is necessary to fix the READY signal to low (or to high) during this period.

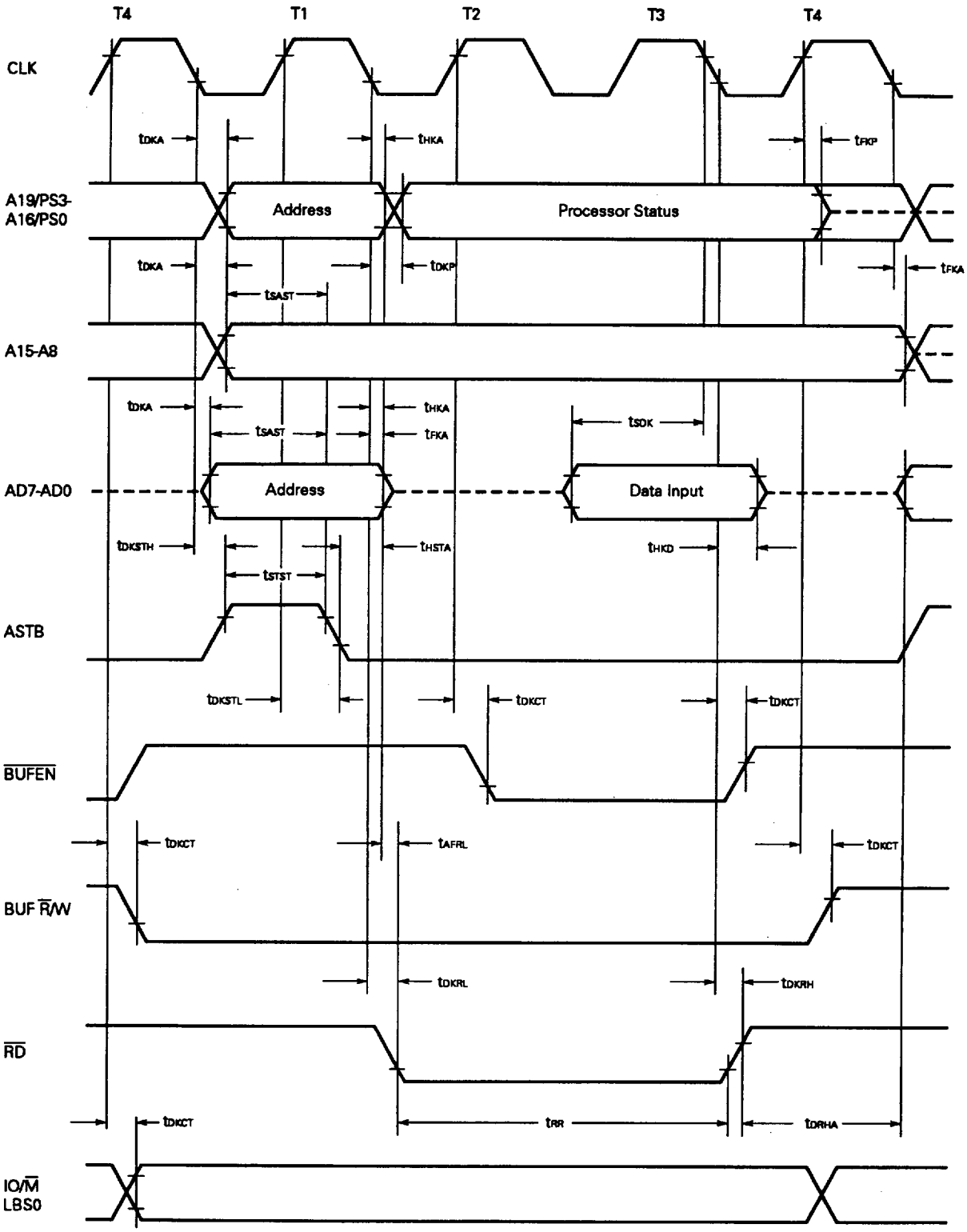
$\overline{\text{POLL}}$, NMI, INT Input Timing



$\overline{\text{BUSLOCK}}$ Output Timing

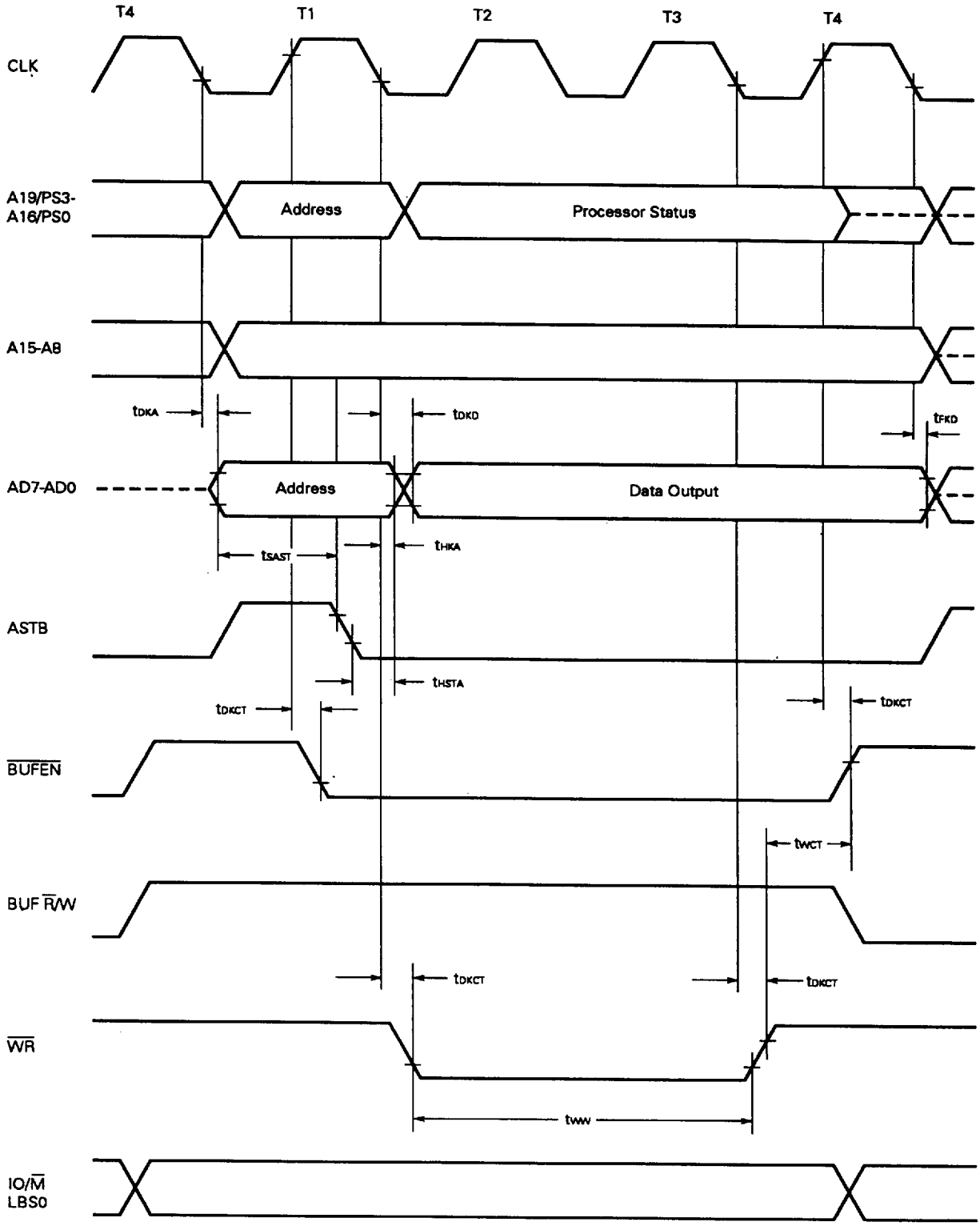


Read Timing (Small Scale)



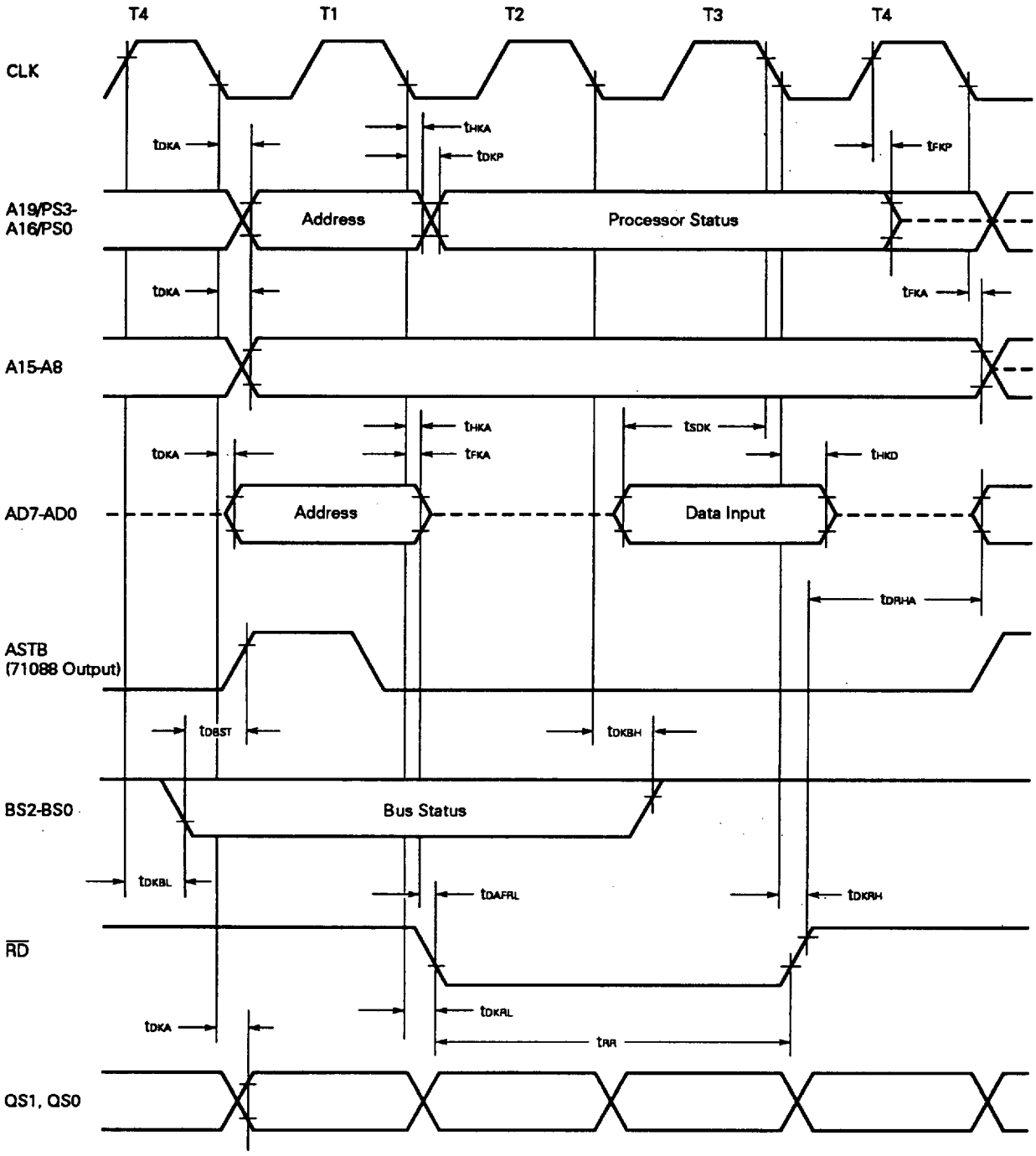
Remark A broken line shows high impedance.

Write Timing (Small Scale)



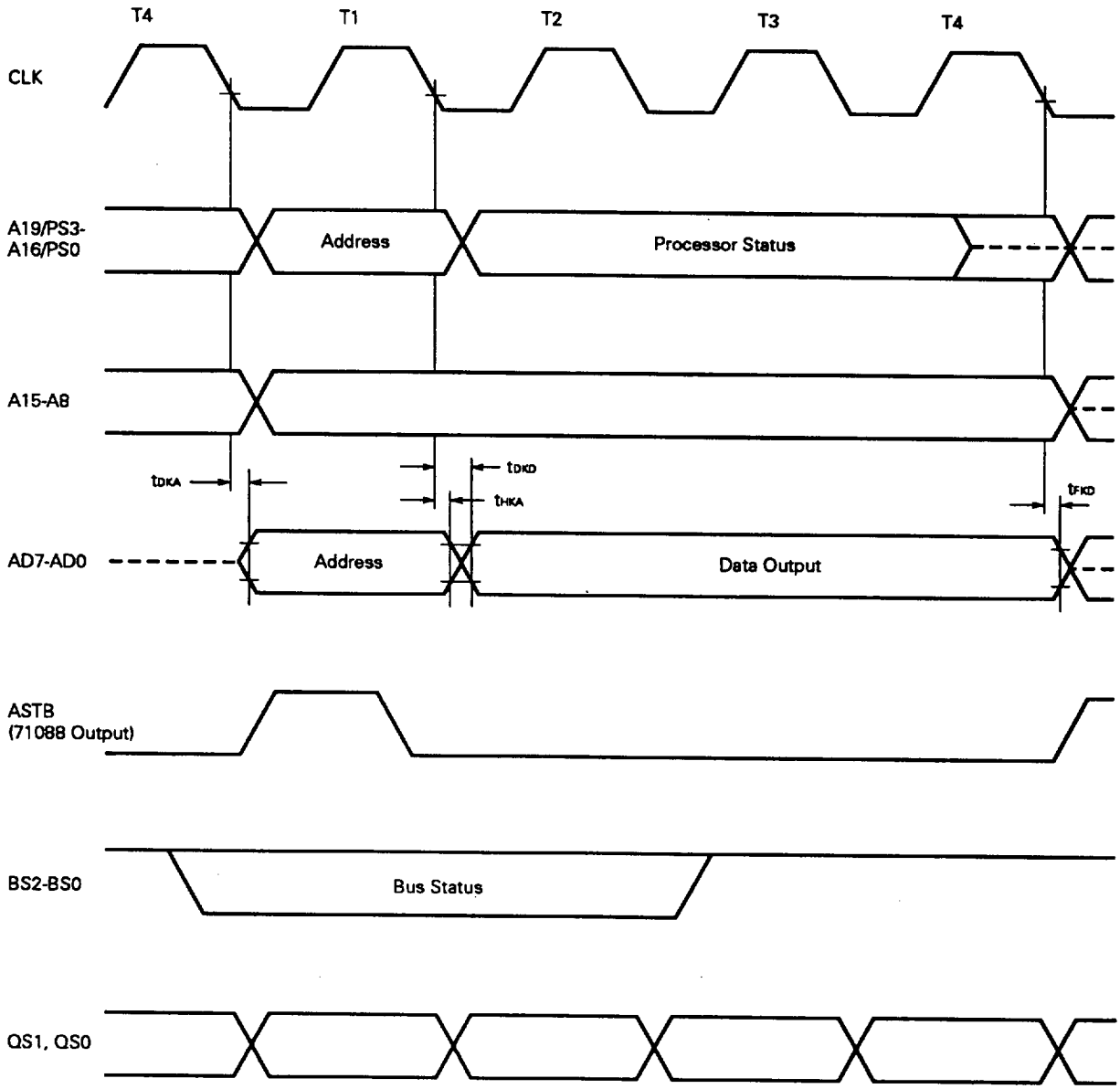
Remark A broken line shows high impedance.

Read Timing (Large Scale)



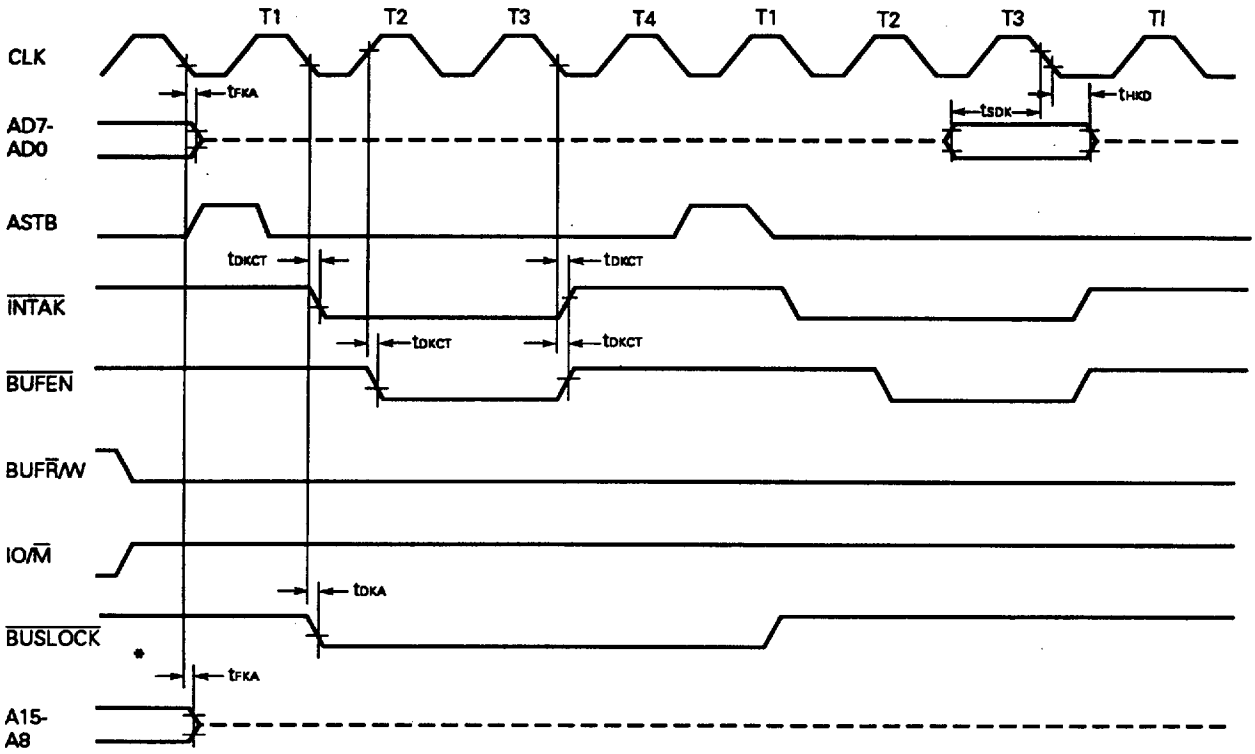
Remark A broken line shows high impedance.

Write Timing (Large Scale)



Remark A broken line shows high impedance.

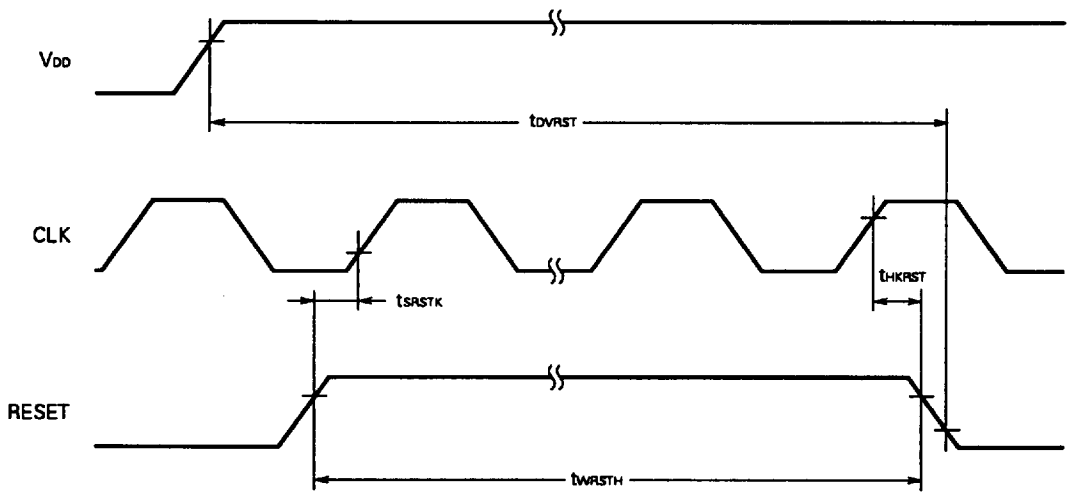
Interrupt Acknowledge Timing



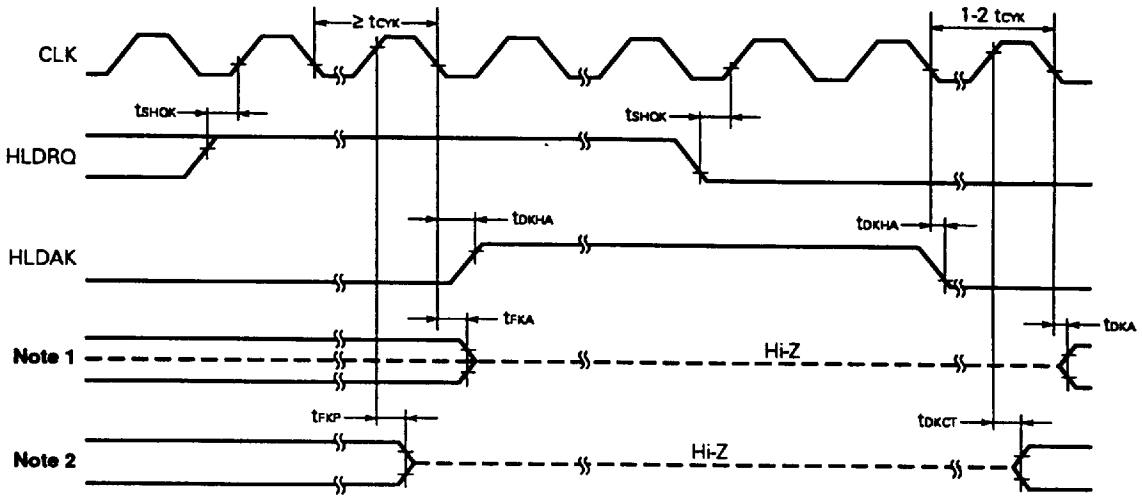
*: Only for large-scale mode

Remark A broken line shows high impedance.

Reset Timing

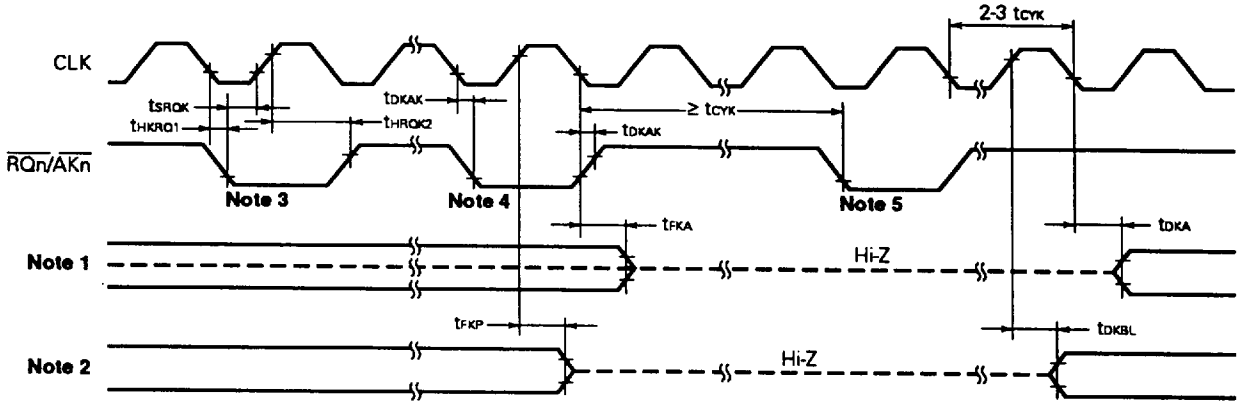


Hold Request/Acknowledge Timing (Small Scale) ★



Notes 1. AD0-AD7, A8-A15
2. A16/PS0-A19/PS3, $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$, BUF$\overline{R}$/W, $\overline{BUFEN}$, LBS0

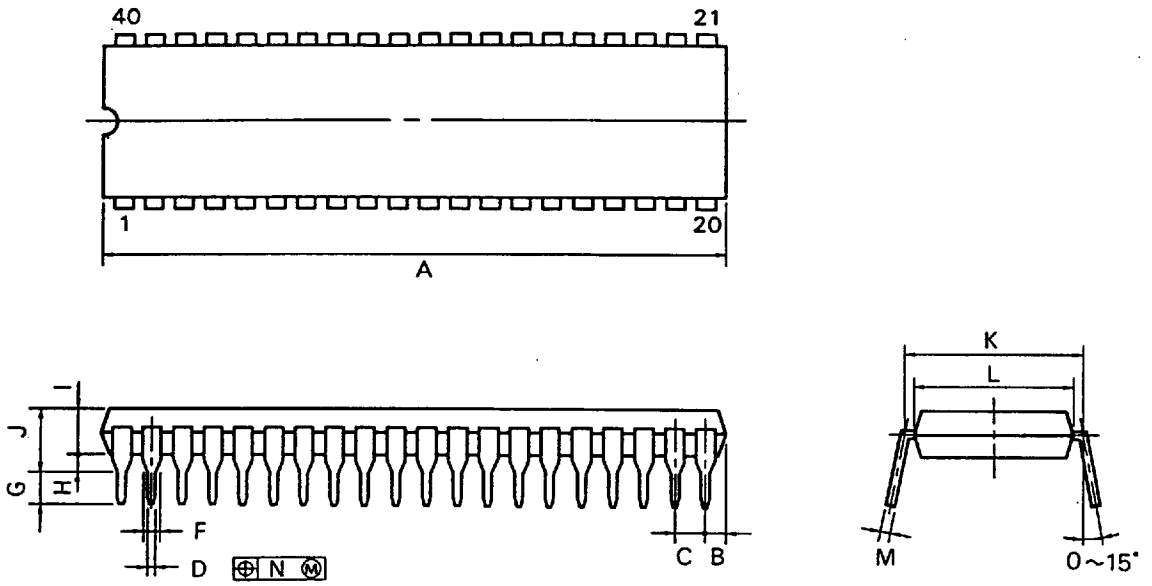
Bus Request/Acknowledge Timing (Large Scale) ★



Notes 1. AD0-AD7, A8-A15
2. A16/PS0-A19/PS3, $\overline{RD}$, BS0-BS2, $\overline{BUSLOCK}$
3. $\overline{RQn}$ (Input) : Request pulse
4. $\overline{AKn}$ (Output): Acknowledge pulse
5. $\overline{RQn}$ (Input) : Release pulse

10. PACKAGE DRAWINGS

40PIN PLASTIC DIP (600 mil)



P40C-100-600A

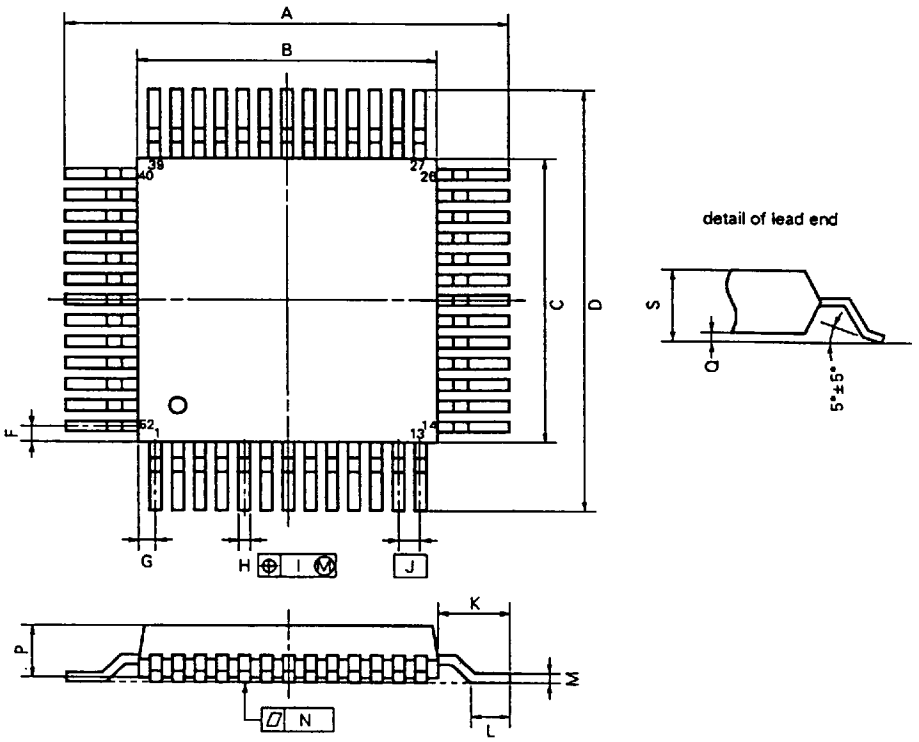
NOTES

1) Each lead centerline is located within 0.25 mm (0.01 inch) of its true position (T.P.) at maximum material condition.

2) Item "K" to center of leads when formed parallel.

ITEM	MILLIMETERS	INCHES
A	53.34 MAX.	2.100 MAX.
B	2.54 MAX.	0.100 MAX.
C	2.54 (T.P.)	0.100 (T.P.)
D	$0.50^{\pm 0.10}$	$0.020^{+0.004}_{-0.005}$
F	1.2 MIN.	0.047 MIN.
G	$3.6^{\pm 0.3}$	$0.142^{\pm 0.012}$
H	0.51 MIN.	0.020 MIN.
I	4.31 MAX.	0.170 MAX.
J	5.72 MAX.	0.226 MAX.
K	15.24 (T.P.)	0.600 (T.P.)
L	13.2	0.520
M	$0.25^{+0.10}_{-0.05}$	$0.010^{+0.004}_{-0.003}$
N	0.25	0.01

52 PIN PLASTIC QFP (□14) ★



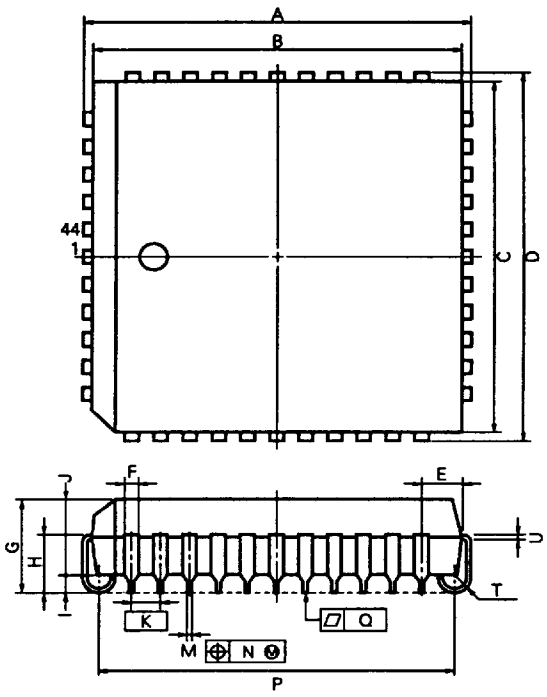
P52GC-100-3B6,3BH-2

NOTE

Each lead centerline is located within 0.20 mm (0.008 inch) of its true position (T.P.) at maximum material condition.

ITEM	MILLIMETERS	INCHES
A	17.6±0.4	0.693±0.016
B	14.0±0.2	$0.551^{+0.009}_{-0.008}$
C	14.0±0.2	$0.551^{+0.009}_{-0.008}$
D	17.6±0.4	0.693±0.016
F	1.0	0.039
G	1.0	0.039
H	0.40±0.10	$0.016^{+0.004}_{-0.005}$
I	0.20	0.008
J	1.0 (T.P.)	0.039 (T.P.)
K	1.8±0.2	$0.071^{+0.008}_{-0.009}$
L	0.8±0.2	$0.031^{+0.009}_{-0.008}$
M	$0.15^{+0.10}_{-0.05}$	$0.006^{+0.004}_{-0.003}$
N	0.10	0.004
P	2.7	0.106
Q	0.1±0.1	0.004±0.004
S	3.0 MAX.	0.119 MAX.

44 PIN PLASTIC QFJ (□650 mil)



P44L-50A1-2

NOTE

Each lead centerline is located within 0.12 mm (0.005 inch) of its true position (T.P.) at maximum material condition.

ITEM	MILLIMETERS	INCHES
A	17.5±0.2	0.689±0.008
B	16.58	0.653
C	16.58	0.653
D	17.5±0.2	0.689±0.008
E	1.94±0.15	$0.076^{+0.007}_{-0.008}$
F	0.6	0.024
G	4.4±0.2	$0.173^{+0.009}_{-0.008}$
H	2.8±0.2	$0.110^{+0.009}_{-0.008}$
I	0.9 MIN.	0.035 MIN.
J	3.4	0.134
K	1.27 (T.P.)	0.050 (T.P.)
M	0.40±0.10	$0.016^{+0.004}_{-0.005}$
N	0.12	0.005
P	15.50±0.20	$0.610^{+0.009}_{-0.008}$
Q	0.15	0.006
T	R 0.8	R 0.031
U	$0.20^{+0.10}_{-0.05}$	$0.008^{+0.004}_{-0.002}$

11. RECOMMENDED SOLDERING CONDITIONS ★

Solder this product under the soldering conditions indicated below.

For further information on the recommended soldering conditions, refer to information document "SEMICONDUCTOR DEVICE MOUNTING TECHNOLOGY MANUAL (IEI-1207)".

For soldering methods and conditions other than those of recommended, consult NEC.

Table 11-1. Soldering Conditions for Types of Surface Mounting Device

(1) μPD70108GC-xx-3B6: 52-pin plastic QFP (□ 14 mm)

Soldering method	Soldering condition	Symbol
Infrared ray reflow	Peak temperature of package surface: 230 °C, Time: 30 seconds max. (210 °C min.), Number of reflow process: 1 Exposure limit[Note]: 7 days (10 hours pre-baking is required at 125 °C afterwards)	IR30-107-1
VPS	Peak temperature of package surface: 215 °C, Time: 40 seconds max. (200 °C min.), Number of reflow process: 1 Exposure limit[Note]: 7 days (10 hours pre-baking is required at 125 °C afterwards)	VP15-107-1
Wave soldering	Solder temperature: 260 °C max., Time: 10 seconds max., Number of reflow process: 1 Exposure limit[Note]: 7 days (10 hours pre-baking is required at 125 °C afterwards) Preheating temperature: 120 °C max. (package surface temperature)	WS60-107-1
Partial heating	Pin temperature: 300 °C max., Time: 3 seconds max. (per one side of device)	—

Note Exposure limit before soldering after dry-pack package is opened.
Storage conditions: 25 °C and relative humidity at 65 % or less.

Caution Do not apply two or more soldering methods (except partial heating) in combination.

Information

Recommended soldering conditions for some parts of this product have been upgraded. (Improvements mode: Infrared ray reflow peak temperature expansion (235 °C), twice, restrictions on days, etc.)
For details, consult NEC.

(2) μPD70108L-xx: 44-pin plastic QFJ (□ 650 mil)

Soldering method	Soldering condition	Symbol
Infrared ray reflow	Peak temperature of package surface: 230 °C, Time: 30 seconds max. (210 °C min.), Number of reflow process: 1 Exposure limit[Note]: 7 days (10 hours pre-baking is required at 125 °C afterwards)	IR30-107-1
VPS	Peak temperature of package surface: 215 °C, Time: 40 seconds max. (200 °C min.), Number of reflow process: 1 Exposure limit[Note]: 7 days (10 hours pre-baking is required at 125 °C afterwards)	VP15-107-1
Partial heating	Pin temperature: 300 °C max., Time: 3 seconds max. (per one side of device)	—

Note Exposure limit before soldering after dry-pack package is opened.
Storage conditions: 25 °C and relative humidity at 65 % or less.

Caution Do not apply two or more soldering methods (except partial heating) in combination.

Table 11-2. Soldering Conditions for Types of Insert Mounting Device

μPD70108C-xx: 40-pin plastic DIP (600 mil)

Soldering method	Soldering condition
Wave soldering (Only leads)	Solder temperature: 260 °C max., Time: 10 seconds max.
Partial heating	Pin temperature: 260 °C max., Time: 10 seconds max.

Caution Solder only the leads by means of wave soldering, and exercise care that the jetted solder does not come in contact with the package.