DATA SHEET

NEC

MOS INTEGRATED CIRCUIT

μPD70136A

V33A™
16-BIT MICROPROCESSOR

DESCRIPTION

The μPD70136A (V33A) is a 16-bit, high-speed microprocessor whose instruction execution time is approximately four times faster than that for the μPD70116 (V30™). The μPD70136A has the same instruction set as that for the μPD70116 native mode. In addition, the μPD70136A is provided with the address expansion function and bus sizing function.

In terms of exception processing, etc., the μPD70136A is more compatible with the μPD70116 (V30™) when compared with the μPD70136 (V33™). Furthermore, a floating-point coprocessor is offered.

Details are given in the following manuals. Be sure to read when carrynig out design work.

- **V33A User's Manual : U10032E**
- **16-bit V series™ User's Manual -Instruction : IEU-804 (O. D. No.)**

FEATURES

- 125-ns minimum instruction execution time at 16MHz
- Address space
  - Normal address mode: 1M bytes
  - Expanded address mode: 16M bytes
- Bus sizing function (16-bit/8-bit selectable): Effective for both memory and I/O
- μPD70116 (native mode) software compatible
- Undefined instruction code exception trapping function
- High-speed multiplication/division instructions
  - 16-bit multiplication: 12 clocks (0.75 μs at 16MHz)
  - 16-bit division: 19 clocks (1.19 μs at 16MHz)
- Bit field manipulation instruction
- Packed BCD operation instruction
- CMOS

Document No. U10136EJ4V0DS00 (4th edition)
Date Published September 1995 P
Printed in Japan

The mark ★ shows the the major revised points.

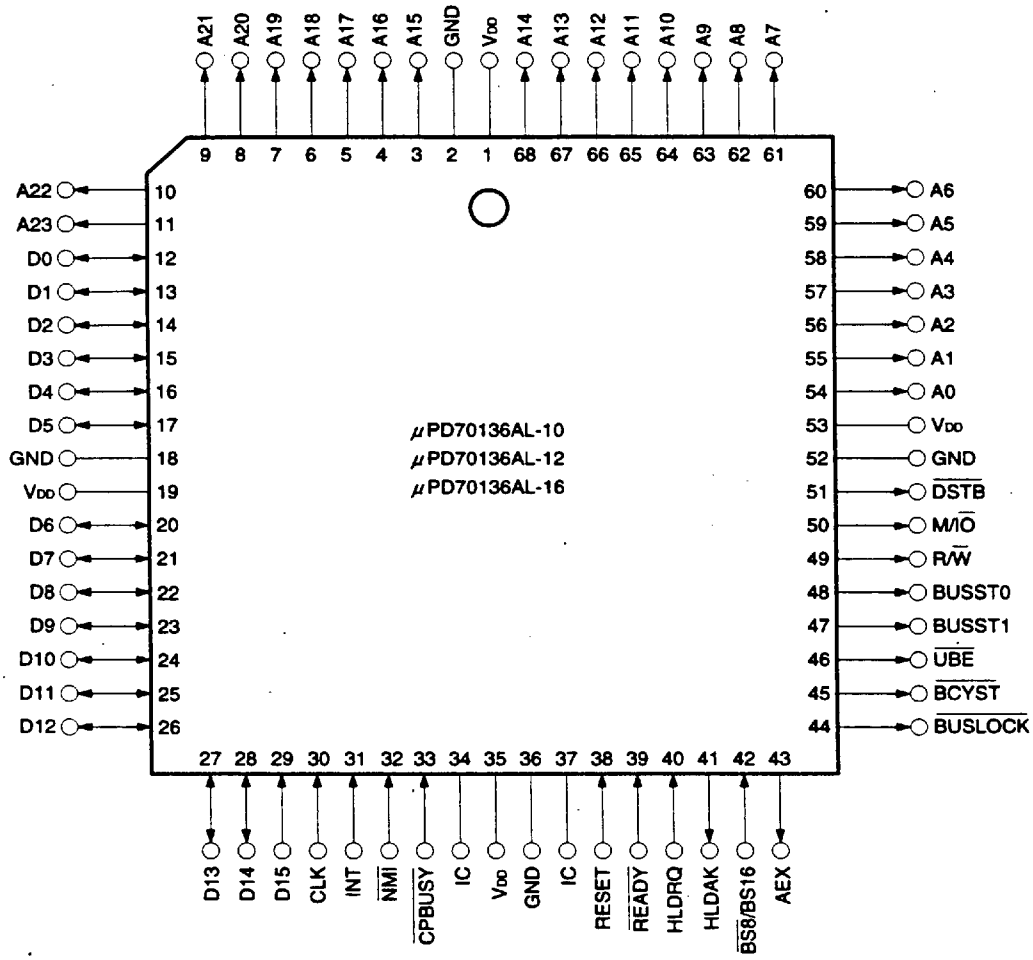
6427525 0086933 661

ORDERING INFORMATION

Part number	Package	Clock frequency (MHz)
μPD70136AL-10	68-pin plastic QFJ (□950 mil)	10.0
μPD70136AL-12	68-pin plastic QFJ (□950 mil)	12.5
μPD70136AL-16	68-pin plastic QFJ (□950 mil)	16.0
μPD70136AR-12	68-pin ceramic PGA (seam welding)	12.5
μPD70136AR-16	68-pin ceramic PGA (seam welding)	16.0

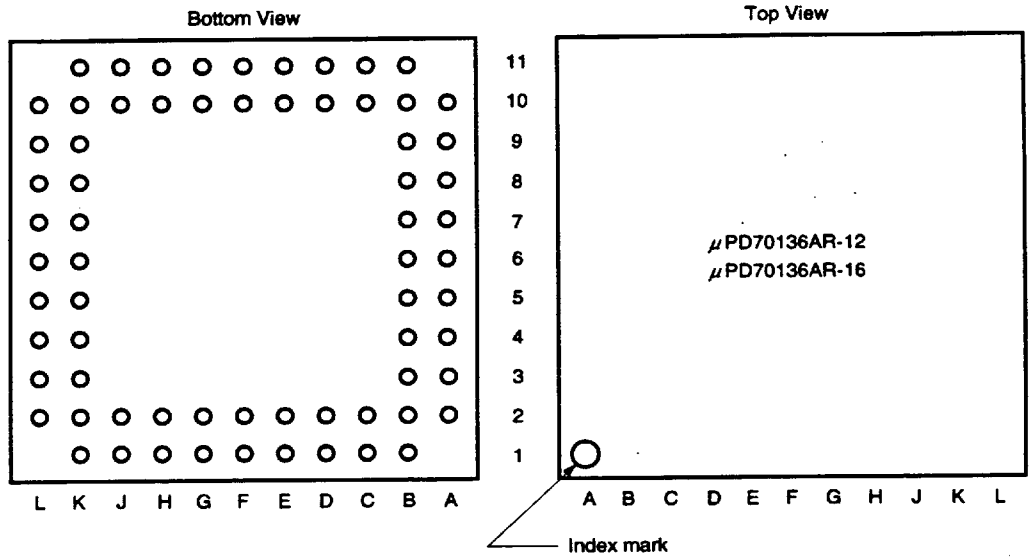
PIN CONFIGURATIONS

68-Pin Plastic QFJ (top view)



Caution IC: Connect to GND.

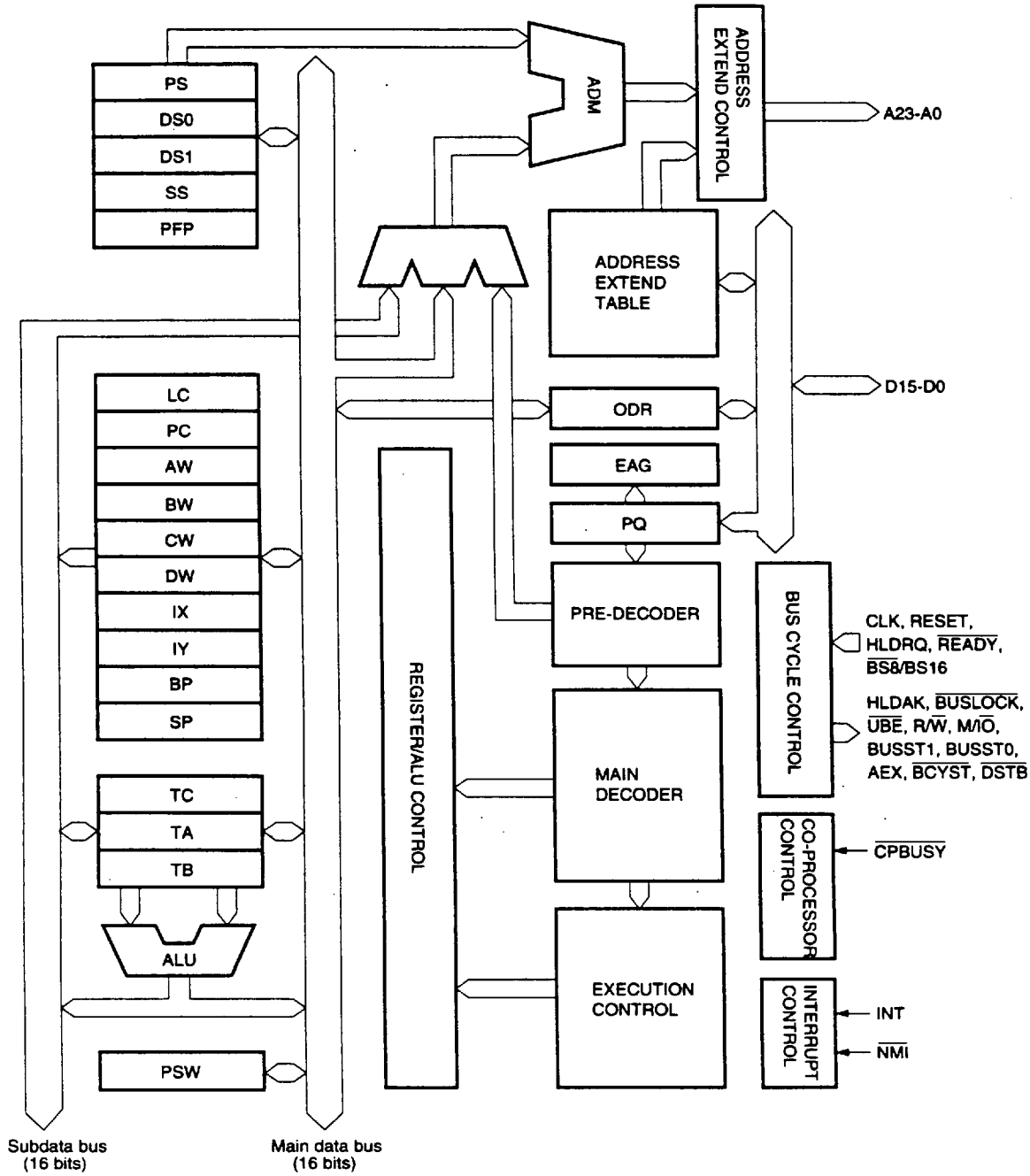
68-Pin Ceramic PGA (top view)



No.	Name	No.	Name	No.	Name	No.	Name
A2	AEX	B9	CLK	F10	V_{DD}	K4	A12
A3	HLDAK	B10	D14	F11	GND	K5	A14
A4	$\overline{READY}$	B11	D12	G1	A0	K6	GND
A5	IC	C1	$\overline{UBE}$	G2	A1	K7	A16
A6	V_{DD}	C2	BUSST1	G10	D5	K8	A18
A7	$\overline{CPBUSY}$	C10	D11	G11	D4	K9	A20
A8	INT	C11	D10	H1	A2	K10	A23
A9	D15	D1	BUSST0	H2	A3	K11	A22
A10	D13	D2	R/$\overline{W}$	H10	D3	L2	A7
B1	$\overline{BUSLOCK}$	D10	D9	H11	D2	L3	A9
B2	$\overline{BCYST}$	D11	D8	J1	A4	L4	A11
B3	$\overline{BS8}$/BS16	E1	M/$\overline{IO}$	J2	A5	L5	A13
B4	HLDRQ	E2	$\overline{DSTB}$	J10	D1	L6	V_{DD}
B5	RESET	E10	D7	J11	D0	L7	A15
B6	GND	E11	D6	K1	A6	L8	A17
B7	IC	F1	GND	K2	A8	L9	A19
B8	$\overline{NMI}$	F2	V_{DD}	K3	A10	L10	A21

Caution IC: Connect to GND.

INTERNAL BLOCK DIAGRAM



CONTENTS

1. PIN FUNCTIONS

★ 1.1 PIN IDENTIFICATION

Symbol	Input/Output	Function	Processing of Unused Pins
CLK	Input	System clock	–
A23-A0	3-state output	Address bus	Open
D15-D0	3-state input/output	Data bus	Open
$\overline{UBE}$	3-state output	Data bus upper byte enable	Open
R/$\overline{W}$	3-state output	Read/write	Open
M/$\overline{IO}$	3-state output	Memory/(I/O)	Open
BUSST1, BUSST0	3-state output	Bus status	Open
$\overline{BCYST}$	3-state output	Bus cycle start	Open
$\overline{DSTB}$	3-state output	Data strobe	Open
$\overline{BUSLOCK}$	Output	Bus lock	Open
$\overline{READY}$	Input	Ready	Connected to GND via resistor
$\overline{BS8}$/BS16	Input	Dynamic bus sizing control	Connected to V_{DD}/GND via resistor
AEX	Output	Address expansion flag	Open
HLDRQ	Input	Bus hold request	Connected to GND via resistor
HLDAK	Output	Bus hold enable	Open
INT	Input	Maskable interrupt request	Connected to GND via resistor
$\overline{NMI}$	Input	Non-maskable interrupt request	Connected to V_{DD} via resistor
$\overline{CPBUSY}$	Input	Coprocessor busy	Connected to GND via resistor
RESET	Input	System reset	–
V_{DD}	–	Positive supply	–
GND	–	Ground potential	–
IC	–	Internal connection pin	Connected to GND

1.2 PIN STATUS IN SPECIFIC STATUS

Table 1-1 shows the pin statuses in specific statuses such as on bus hold, in standby mode, and on reset.

Table 1-1. Pin Status in Specific Status

Pin Name	I/O	Bus Latch Note 1	Status		
			Bus Hold	Standby	Reset
CLK	Input	×	–	–	–
A0-A23	3-state output	O	Hi-Z	L	Hi-Z
D0-D15	3-state I/O	O	Hi-Z	Note 2	Hi-Z
$\overline{UBE}$	3-state output	O	Hi-Z	H	Hi-Z
R/$\overline{W}$	3-state output	O	Hi-Z	L	Hi-Z
M/$\overline{IO}$	3-state output	O	Hi-Z	L	Hi-Z
BUSST0, BUSST1	3-state output	O	Hi-Z	H	Hi-Z
$\overline{BCYST}$	3-state output	O	Hi-Z	Note 3	Hi-Z
$\overline{DSTB}$	3-state output	O	Hi-Z	H	Hi-Z
$\overline{BUSLOCK}$	Output	×	Note 4	Note 4	Note 5 ★
$\overline{READY}$	Input	×	–	–	–
$\overline{BS8}$/BS16	Input	×	–	–	–
AEX	Output	×	Note 6	Note 6	Note 7 ★
HLDRQ	Input	×	–	–	–
HLDAK	Output	×	H	L	L
INT	Input	×	–	–	–
$\overline{NMI}$	Input	–	–	–	–
$\overline{CPBUSY}$	Input	×	–	–	Note 8
RESET	Input	×	–	–	–

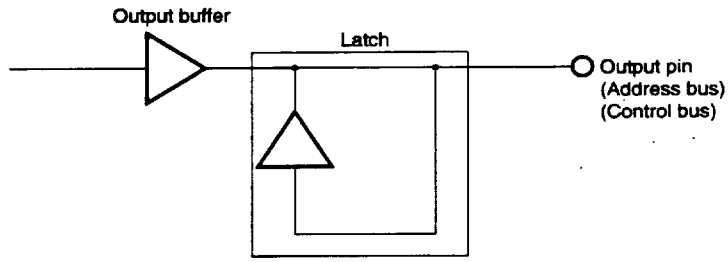
Notes 1. O : Latch is provided.
× : Latch is not provided.
The pins provided with a latch retains the status before they go into a high-impedance state until driven by an external source. Therefore, it is not necessary to pull up or down these pins. However, be sure to pull up $\overline{DSTB}$ (1.9-10kΩ). To invert the level of a pin in the high-impedance state from an external source, a drive capability of higher than the latch inverting current (I_{ILH}, I_{IHL}) is necessary.

2. Undefined for the duration of the first 2 clocks in the halt acknowledge cycle, and then goes into a high-impedance state.
3. Low for the duration of the first clock in the halt acknowledge cycle and then goes high.
4. L in either of the following cases, and H in other cases:
   - When an instruction with bus lock prefix is executed on hold
   - When HALT instruction with bus lock prefix is executed
5. Undefined during reset signal input. H following reset.
6. H in address expansion mode, and L in non-expansion mode
7. Undefined during reset signal input. L following reset.
8. Whether the processor is connected or not is judged by sampling the status of this pin on reset.

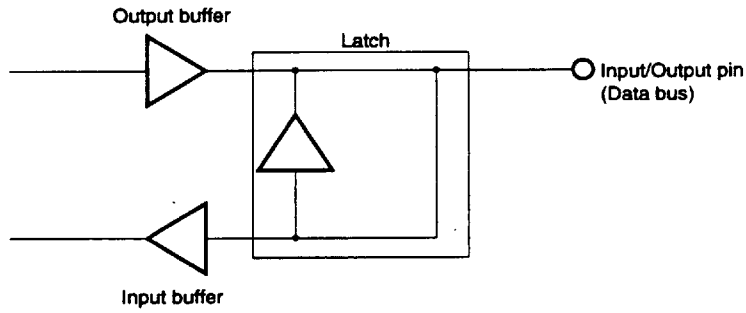
Remark H: high level L: low level Hi-Z: high impedance

Figure 1-1. Latch Configuration

Output Pin



Input/Output Pin



2. REGISTER CONFIGURATION

2.1 PFP (PREFETCH POINTER)

The prefetch pointer is a 16-bit binary counter, which retains the program memory address offset information.

The PFP is incremented each time an instruction is fetched from the program memory. When branch, call, return, or break instruction is executed, a new location is loaded into the PFP. In this case, the PFP contents become the same as for the PC (Program Counter).

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

2.2 PQ (PREFETCH QUEUE)

The μPD70136A contains an 8-byte instruction queue (FIFO) which can store up to 8 bytes of prefetched instruction code.

The queue contents are cleared when a branch, call, return, or break instruction is executed, or when processing external interrupt, and an instruction at a new location will be prefetched.

Normally, the μPD70136A prefetches instruction, when there is 1 word (2 bytes) or more space in the queue.

When successively executing a number of instructions, if the average instruction execution time is faster than the number of clock periods necessary to prefetch each instruction code, when an instruction execution is completed, the next instruction code to be executed by the EXU is already provided in the queue. Therefore, the time necessary to fetch an instruction from the external memory can be excluded from the instruction execution time. As a result, the processing speed can be improved, compared to that for a CPU, which fetches and executes an instruction one at a time.

The queue effectiveness decreases as the number of instructions which clear the queue increases, or when instructions, having shorter instruction execution time, have been continued.

2.3 ODR (OPERAND DATA REGISTER)

ODR can be accessed in byte units. Therefore, the upper byte and the lower byte are independently read/written. Write operation is completed, when data is written in the ODR. Read operation is terminated, upon confirming that data is transferred form the external data bus to the ODR.

2.4 SEGMENT REGISTERS (PS, SS, DS0, DS1)

In the μPD70136A, memory addresses are divided into logical segments, which are 64K bytes each. The start address for each segment register is specified by the corresponding segment register. The offset from the start address is specified by a different register or by the effective address.

The μPD70136A contains the following four kinds 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

The PS and the PFP (Prefetch Pointer), and the DS1 and the IY registers are always paired.

The SS is normally paired with the SP. However, when the BP register is selected as the base register, effective address is used as the offset.

The DS0 is used together with the IX register for block transfer processing. However, for other general processing, the effective address is used as the offset.

In addressing using the BP register as the base register and the SS register as the segment register, any one of three other registers can be selected as the segment selection, using the the segment override prefix instruction (PS:, DS0:, DS1:). However, eight or more prefix instructions cannot be attached to other than prefix instructions.

2.5 ADM (ADDRESS MODIFIER)

ADM (Address Modifier) is used for generating a physical address (addition of segment register and PFP or DP), and increments the PFP (Prefetch Pointer).

2.6 GENERAL-PURPOSE REGISTERS (AW, BW, CW, DW)

The μPD70136A contains four 16-bit registers. Each of these 16-bit registers can be used as a 16-bit register. In addition, each of these 16-bit registers can be divided into upper and lower 8 bits, so that each can be accessed as an 8-bit register (AH, AL, BH, BL, CH, CL, DH, DL).

Therefore, these registers can be used as 8-bit or 16-bit registers for various instructions, such as transfer, arithmetic operation, logical operation instruction, etc.

These registers are used as default registers for certain instruction processings as follows:

- AW : Word multiplication/division, word input/output, translation, BCD rotation, data conversion
- AL : Byte multiplication/division, byte input/output, BCD rotation, data conversion
- 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 input/ output

2.7 POINTERS (SP, BP) AND INDEX REGISTER (IX, IY)

SP and BP, and IX and IY are used as the base pointers or index registers, when accessing the memory in the based addressing mode, indexed addressing mode, based indexed addressing mode, etc.

In the same way as for general-purpose registers, these pointers and index registers are used for transfer, arithmetic, or logical operation instructions. However, in this case, these pointers and index registers cannot be used as 8-bit registers.

These registers are used as default registers for certain instruction processing, as follows:

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

2.8 TA/TB (TEMPORARY REGISTER/SHIFTER A/B)

TA/TB is a 16-bit temporary register/shifter used for multiplication/division, and shift/rotate (including BCD rotate) instructions.

When executing a multiplication/division instruction, TA and TB are paired to form a 32-bit temporary register/shifter. When executing a shift/rotate instruction, only the TB serves as the 16-bit temporary register/shifter.

The upper and lower bytes individually for TA and TB can be independently read/written through the internal bus.

TA/TB becomes the ALU input.

2.9 TC (TEMPORARY REGISTER C)

TC is a 16-bit temporary register, used for multiplication, division, and other internal processings.

The TC becomes the ALU input.

2.10 ALU (ARITHMETIC LOGIC UNIT)

ALU (Arithmetic Logic Unit) consists of the full adder and the logic operation circuit, and performs arithmetic operations (addition, subtraction, increment, decrement, and complement operations) and logical operations (test, AND, OR, XOR, and test, set, clear, and invert in bit units).

2.11 PSW (PROGRAM STATUS WORD)

The program status word consists of six status flags and three control flags.

Status flags

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

Control flags

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

These flags are manipulated in the stack in the following word image:

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

Status flags are automatically set or reset, according to instruction execution results (data value).

The CY flag can be directly set, reset, or inverted by an instruction.

The control flags are set or reset by an instruction, in order to control the CPU operation.

2.12 LC (LOOP COUNTER)

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

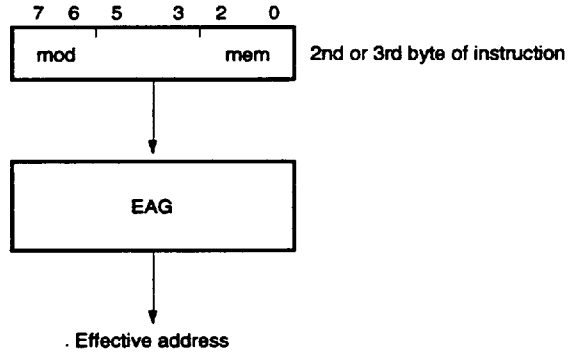
2.13 PC (PROGRAM COUNTER)

PC (Program Counter) is a 16-bit binary counter, which retains the offset information for the program memory address in the instruction to be executed next.

The PC is incremented each time the decoder fetches an instruction byte from the instruction queue. When a branch, call, return, or break instruction is executed, a new address location is loaded into the PC, in this case, the PC contents become the same as those for the PFP (Prefetch Pointer).

2.14 EAG (EFFECTIVE ADDRESS GENERATOR)

EAG (Effective Address Generator) logic computes an effective address necessary for accessing the memory at high speed. An address computation is completed in one clock period in any addressing mode.



The byte (2nd or 3rd byte) in which the instruction operand is specified is clocked in. If memory accessing is necessary, the EAG generates a control signal necessary for manipulating the respective register, and computes the effective address.

In addition, the EAG initiates a bus cycle (memory read, etc.) as necessary.

3. BUS CYCLE

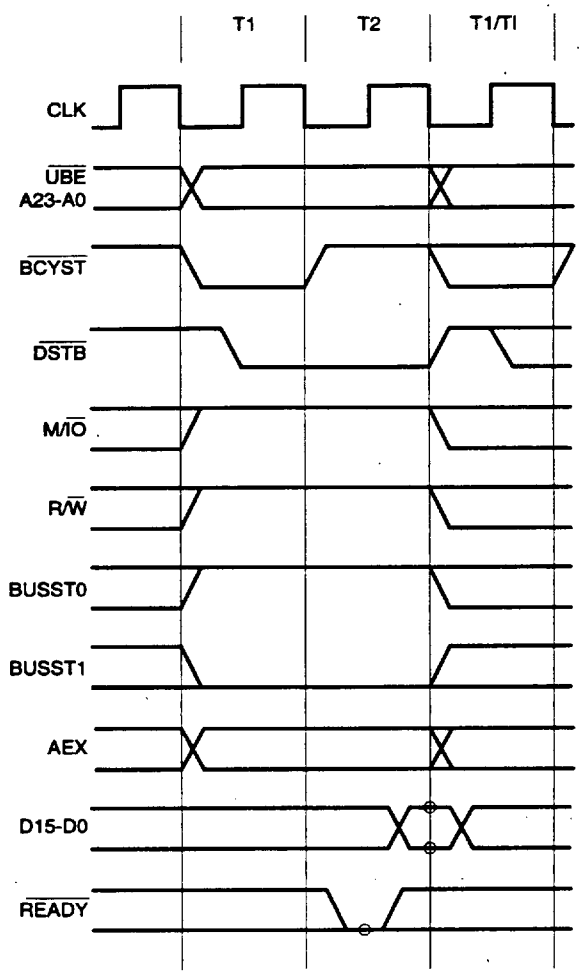
3.1 TYPE OF BUS CYCLES

The μPD70136A starts 11 bus cycles of the external bus as shown in the following table, by using the combinations of the signals shown in the table.

M/$\overline{IO}$	BUSST1	BUSST0	R/$\overline{W}$	Bus Cycle Type
0	0	0	1	Interrupt acknowledge
		1	1	I/O read
		1	0	I/O write
	1	0	1	Coprocessor read
		0	0	Coprocessor write
		1	0	Hold acknowledge
1	0	0	1	Instruction fetch
		1	1	Memory read
		1	0	Memory write
	1	0	1	Coprocessor memory read
		0	0	Coprocessor memory write

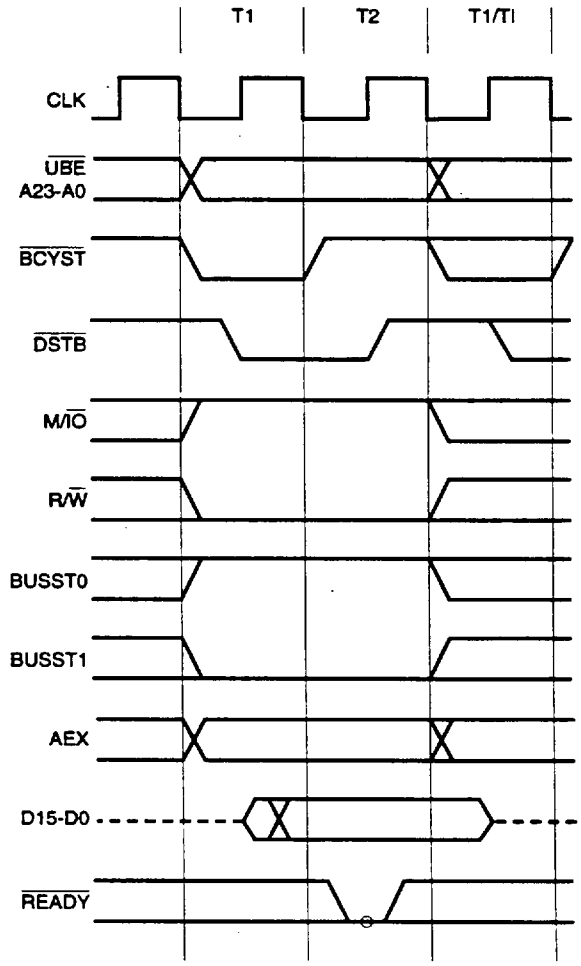
3.2 BUS TIMING

(1) Memory read (0 wait)



Remark ○ indicates sampling timing.

(2) Memory write (0 wait)



Remarks 1. Dashed line indicates high impedance.
2. O indicates sampling timing.

4. ADDRESS SPACE EXPANSION FUNCTION

4.1 OUTLINE

20-bit address space can be expanded to 24-bit address space by EA (Effective Address) generation.

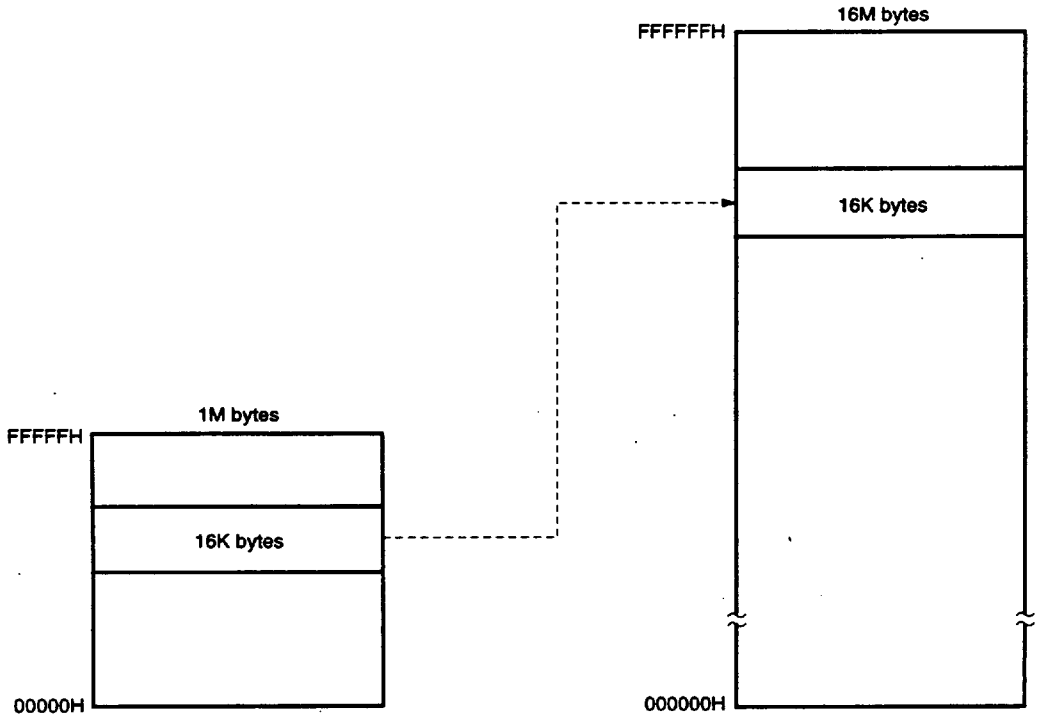
An address relocation method is used as the expansion method.

Relocation is made 16K bytes for each page, and expansion is possible up to 16M bytes.

Expansion is specified by software.

When the expansion address mode is specified, the upper 6 bits for the 20-bit address are expanded to 10 bits, by referencing the translation table to generate a 24-bit expansion address.

When the expansion address mode is not specified, 20-bit physical address is output as is, and the upper 4 bits (A23-A20) output low.



4.2 EXPANSION ADDRESS MODE SETTING/RELEASE

The following instructions are used for setting/releasing the expansion address mode (XA mode).

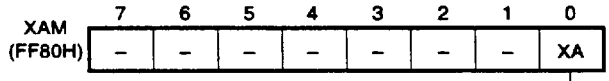
BRKXA instruction
RETXA instruction

The table below indicates the XA flag operation.

Instruction	Operation	XA Mode
BRKXA	Reads vector n, and branches. Sets XA flag to 1.	Set
RETXA	Reads vector n, and branches. Resets XA flag to 0.	Reset

Remark The expansive address mode/normal address mode becomes valid starting from the fetch cycle of the branch destination address when the BRKXA or RETXA instruction is executed.

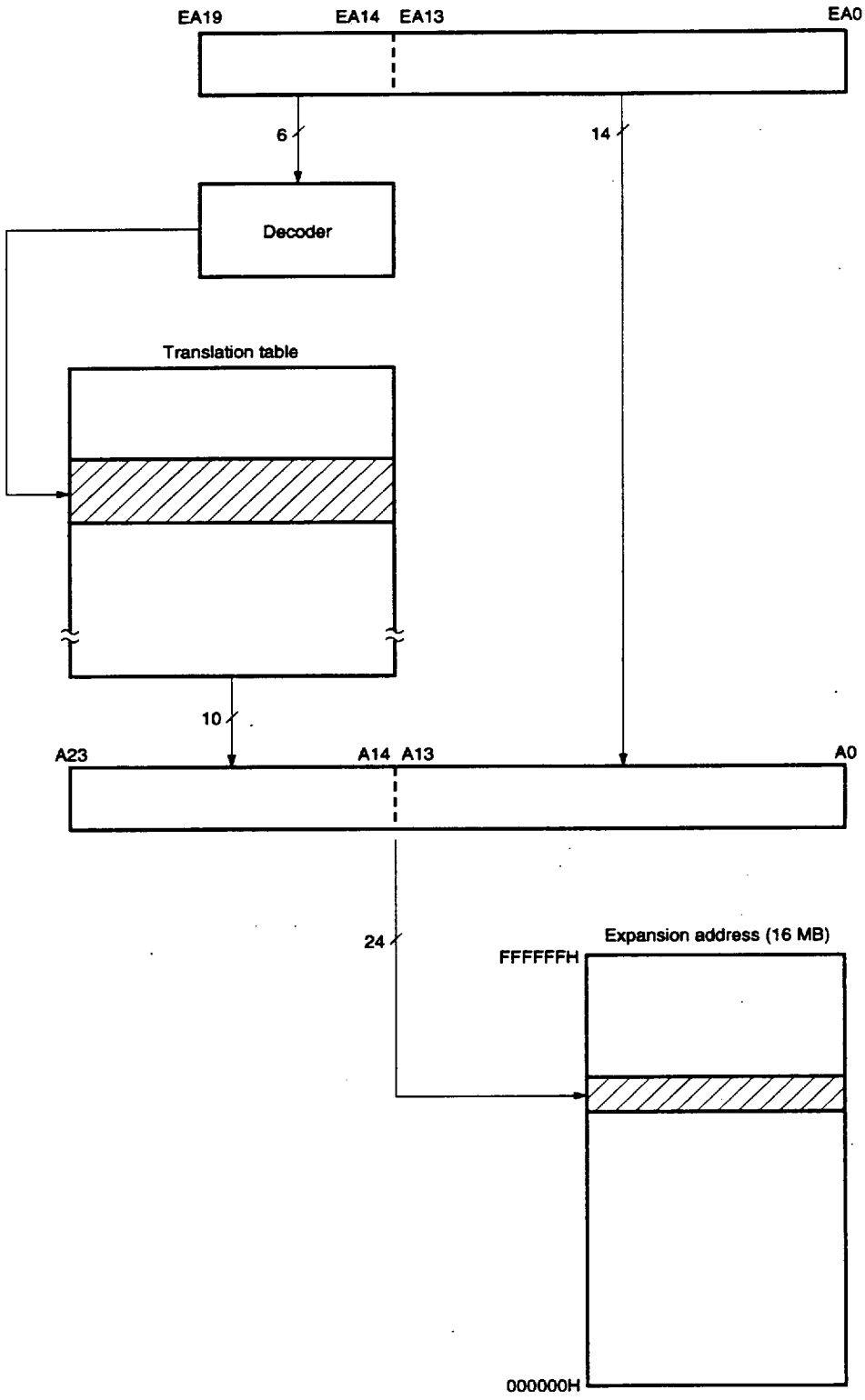
Whether or not the expansion address mode is set can be determined by bit 0 (XA flag) for the XAM register (FF80H).



XA	Expansion Address
0	Normal address mode (NA mode)
1	Expansion address mode (XA mode)

The XA flag is a read-only flag, which can be read by the byte IN instruction. When a reset is input, this flag is cleared to 0. In the expansion address mode, no accessing should be made to the page register.

4.3 ADDRESS TRANSLATION METHOD



4.4 ADDRESS TRANSLATION TABLE

The address translation table inputs the upper 6 bits for the 20-bit address generated by the effective address generator. It selects one of 64 page registers (PGR) to generate the upper 10 bits in the expansion address.

The page registers (PGRs) are allocated to addresses FF00H-FF7EH in the I/O space, and can be read/written using the word IN/OUT instructions.

The page registers must not be accessed in the expansion address mode (XA = 1).

(1) Address translation

A19	A18	A17	A16	A15	A14	Page Register	A19	A18	A17	A16	A15	A14	Page Register
0	0	0	0	0	0	PGR 1	1	0	0	0	0	0	PGR 33
0	0	0	0	0	1	PGR 2	1	0	0	0	0	1	PGR 34
0	0	0	0	1	0	PGR 3	1	0	0	0	1	0	PGR 35
0	0	0	0	1	1	PGR 4	1	0	0	0	1	1	PGR 36
0	0	0	1	0	0	PGR 5	1	0	0	1	0	0	PGR 37
0	0	0	1	0	1	PGR 6	1	0	0	1	0	1	PGR 38
0	0	0	1	1	0	PGR 7	1	0	0	1	1	0	PGR 39
0	0	0	1	1	1	PGR 8	1	0	0	1	1	1	PGR 40
0	0	1	0	0	0	PGR 9	1	0	1	0	0	0	PGR 41
0	0	1	0	0	1	PGR 10	1	0	1	0	0	1	PGR 42
0	0	1	0	1	0	PGR 11	1	0	1	0	1	0	PGR 43
0	0	1	0	1	1	PGR 12	1	0	1	0	1	1	PGR 44
0	0	1	1	0	0	PGR 13	1	0	1	1	0	0	PGR 45
0	0	1	1	0	1	PGR 14	1	0	1	1	0	1	PGR 46
0	0	1	1	1	0	PGR 15	1	0	1	1	1	0	PGR 47
0	0	1	1	1	1	PGR 16	1	0	1	1	1	1	PGR 48
0	1	0	0	0	0	PGR 17	1	1	0	0	0	0	PGR 49
0	1	0	0	0	1	PGR 18	1	1	0	0	0	1	PGR 50
0	1	0	0	1	0	PGR 19	1	1	0	0	1	0	PGR 51
0	1	0	0	1	1	PGR 20	1	1	0	0	1	1	PGR 52
0	1	0	1	0	0	PGR 21	1	1	0	1	0	0	PGR 53
0	1	0	1	0	1	PGR 22	1	1	0	1	0	1	PGR 54
0	1	0	1	1	0	PGR 23	1	1	0	1	1	0	PGR 55
0	1	0	1	1	1	PGR 24	1	1	0	1	1	1	PGR 56
0	1	1	0	0	0	PGR 25	1	1	1	0	0	0	PGR 57
0	1	1	0	0	1	PGR 26	1	1	1	0	0	1	PGR 58
0	1	1	0	1	0	PGR 27	1	1	1	0	1	0	PGR 59
0	1	1	0	1	1	PGR 28	1	1	1	0	1	1	PGR 60
0	1	1	1	0	0	PGR 29	1	1	1	1	0	0	PGR 61
0	1	1	1	0	1	PGR 30	1	1	1	1	0	1	PGR 62
0	1	1	1	1	0	PGR 31	1	1	1	1	1	0	PGR 63
0	1	1	1	1	1	PGR 32	1	1	1	1	1	1	PGR 64

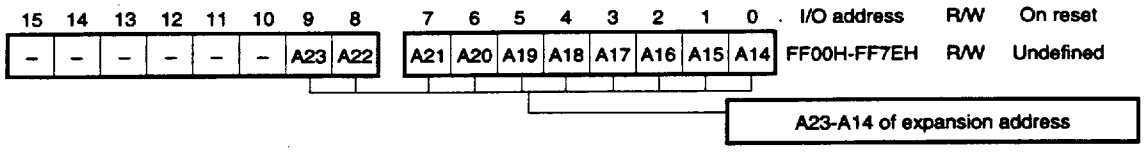
(2) Page registers (PGR1-PGR64)

The page registers are accessed by word IN/word OUT instruction.

The lower 10 bits (D9-D0) of a page register are valid. When the other higher bits (D15-D10) are read, 0 is read. When data is written to these bits, the data is ignored. The page registers are not affected by reset.

Do not access the page registers in the expansive address mode (XA = 1)

Page registers (PGR1-PGR64)



I/O Address	Page Register	I/O Address	Page Register	I/O Address	Page Register
FF00	PGR1	FF2C	PGR23	FF58	PGR45
FF02	PGR2	FF2E	PGR24	FF5A	PGR46
FF04	PGR3	FF30	PGR25	FF5C	PGR47
FF06	PGR4	FF32	PGR26	FF5E	PGR48
FF08	PGR5	FF34	PGR27	FF60	PGR49
FF0A	PGR6	FF36	PGR28	FF62	PGR50
FF0C	PGR7	FF38	PGR29	FF64	PGR51
FF0E	PGR8	FF3A	PGR30	FF66	PGR52
FF10	PGR9	FF3C	PGR31	FF68	PGR53
FF12	PGR10	FF3E	PGR32	FF6A	PGR54
FF14	PGR11	FF40	PGR33	FF6C	PGR55
FF16	PGR12	FF42	PGR34	FF6E	PGR56
FF18	PGR13	FF44	PGR35	FF70	PGR57
FF1A	PGR14	FF46	PGR36	FF72	PGR58
FF1C	PGR15	FF48	PGR37	FF74	PGR59
FF1E	PGR16	FF4A	PGR38	FF76	PGR60
FF20	PGR17	FF4C	PGR39	FF78	PGR61
FF22	PGR18	FF4E	PGR40	FF7A	PGR62
FF24	PGR19	FF50	PGR41	FF7C	PGR63
FF26	PGR20	FF52	PGR42	FF7E	PGR64
FF28	PGR21	FF54	PGR43		
FF2A	PGR22	FF56	PGR44		

Caution Always access page registers (PGR1-PGR64) with word accesses at even addresses.

4.5 INTERNAL I/O AREA

The following two kinds of registers are provided as internal I/O registers for address expansion (refer to **4.1 OUTLINE** through **4.4 ADDRESS TRANSLATION TABLE**).

- XAM register
- PGR1 to PGR2

Register	I/O address	Read/write	Access
XAM	FF80H	Read only possible	Word IN instruction
PGR1-PGR64	FF00H-FF7EH	Read/write possible	Word IN/OUT instruction

The method to access internal I/O, at FF00H to FF80H, differs from normal external I/O access.

Signal	External I/O access (000H-FEFFH)	Internal I/O access (FF00H-FF80H)
A23-A0	Output	Output
D15-D0	Output (Hi-Z when read)	Output (Hi-Z when read)
$\overline{BCYST}$	Output	Output
$\overline{DSTB}$	Output	No output
M/$\overline{IO}$	Output	Output
R/$\overline{W}$	Output	Output
$\overline{READY}$	Accepted	Not accepted
$\overline{BS8}$/BS16	Accepted	Not accepted

Caution **The page registers (PGR1-PGR64) must not be accessed in the expansion address mode (XA =1). When accessing a page register (PGR1-PGR64), the word IN/OUT instruction, which accesses the page register in page units, must be used.**

5. DYNAMIC BUS SIZING FUNCTION

The μPD70136A has a 16-bit data bus. However, in order to facilitate connection to a system having an 8-bit data bus, the μPD70136A is provided with the dynamic bus sizing function.

The bus sizing function is effective for both memory accessing and I/O accessing (external I/O only). When the $\overline{BS8}$/BS16 pin is set to low, only the lower 8 bits of the data bus become effective.

When executing a word access to an even address, the second bus cycle is started after the normal first bus cycle, and the upper 8 bits of data are read/written through D7-D0.

Table 5-1. Access of Data

$\overline{UBE}$	A0	Operation
0	0	16-bit access
0	1	Upper 8-bit access
1	0	Lower 8-bit access
1	1	Second cycle in bus sizing

Table 5-2. Write Operation

Byte/word	Address	A0	$\overline{UBE}$	Cycle	No Sizing ($\overline{BS8}$/BS16 = 1)		Sizing ($\overline{BS8}$/BS16 = 0)	
					D15-D8	D7-D0	D15-D8	D7-D0
Byte	Even	0	1	First	Undefined	Lower byte	Undefined	Lower byte
	Odd	1	0	First	Lower byte	Lower byte	Lower byte	Lower byte
Word	Even	0	0	First	Upper byte	Lower byte	Upper byte	Lower byte
		1	1	Second	No cycle	No cycle	Upper byte	Upper byte
	Odd	1	0	First	Lower byte	Lower byte	Lower byte	Lower byte
		0	1	Second	Lower byte	Upper byte	Lower byte	Upper byte

Table 5-3. Read Operation

Byte/word	Address	A0	$\overline{UBE}$	Cycle	No Sizing ($\overline{BS8}$/BS16 = 1)		Sizing ($\overline{BS8}$/BS16 = 0)	
					D15-D8	D7-D0	D15-D8	D7-D0
Byte	Even	0	1	First	—	Lower byte	—	Lower byte
	Odd	1	0	First	Lower byte	—	—	Lower byte
Word	Even	0	0	First	Upper byte	Lower byte	—	Lower byte
		1	1	Second	No cycle	No cycle	—	Upper byte
	Odd	1	0	First	Lower byte	—	—	Lower byte
		0	1	Second	—	Upper byte	—	Upper byte

Caution **For memory operand of floating-point operation chip instructions, the dynamic bus sizing must always be set to 16-bit bus.**

Table 5-4. Bus Sizing and Sampling for Bus Cycles

Bus Cycle	D15-D0 Pin Valid Bus Size	$\overline{BS8}$/BS16 Pin Sampling
Interrupt acknowledge cycle	8 bits	–
External I/O read cycle	8/16 bits	○
Internal I/O read cycle	8/16 bits	–
External I/O read cycle	8/16 bits	○
External I/O write cycle	8/16 bits	–
Coprocessor read cycle	16 bits	–
Coprocessor write cycle	16 bits	–
Halt acknowledge cycle	Hi-Z (no meaning)	–
Instruction fetch cycle	8/16 bits	○
CPU memory read cycle	8/16 bits	○
CPU memory write cycle	8/16 bits	○
Coprocessor memory read cycle	16 bits	○ Note
Coprocessor memory write cycle	16 bits	○ Note

Note When bus size is set to 8 bits, operation is not performed normally.

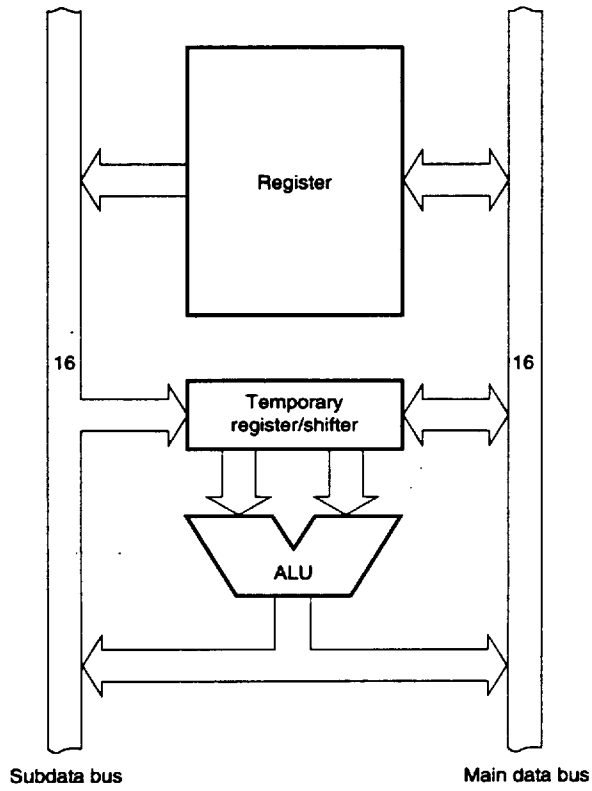
6. INCREASING INSTRUCTION EXECUTION SPEED

The μPD70136A is provided with the following hardware functions in order to reduce the instruction execution time:

- 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)

6.1 DUAL DATA BUS METHOD

In order to reduce the number of processing steps necessary for instruction execution, a dual data bus concept, with the main data bus (16 bits) and the sub data bus (16 bits), is employed. With this concept, the processing time is reduced approximately 30%, compared to the processing time for a single bus system in implementing addition, subtraction, logic operation, and compare instructions.



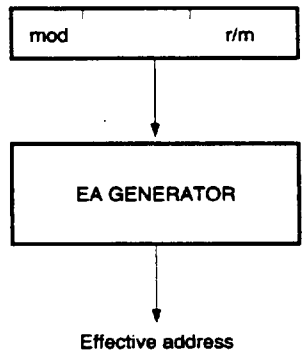
Example ADD AW, BW; AW ← AW+BW

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

6.2 EFFECTIVE ADDRESS GENERATOR

The effective address generator computes at a high speed an effective address necessary for accessing the memory.

In the microprogram method, it took 5 to 12 clock periods to compute an effective address. However, with this sole use hardware, an effective address can be computed in one clock period for any addressing mode.



6.3 16/32-BIT TEMPORARY REGISTER/SHIFTER (TA, TB)

The temporary register/shifter (TA, TB) are provided for multiplication, division, shift, and rotate instructions. With this circuit, especially, multiplication and division instruction execution speed is increased to approximately 4 time faster than a method using the microprogram.

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

6.4 LOOP COUNTER (LC)

The loop counter (LC) counts the number of loops for primitive block transfer, and input/output instructions controlled by the repeat prefix instruction, or counts the number of shifts for the multiple-bit shift/rotate instructions.

For example, register multiple-bit rotation will be performed as shown below, and the processing speed is increased approximately 2 time faster than that of microprogram method.

RORC AW, CL; CL = 5

Microprogram method	LC method
8 + 4 × 5 = 28 clocks	2 + 5 = 7 clocks

6.5 PC AND PFP

With the prefetch pointer (PFP), which addresses the program memory when prefetching an instruction, and the program counter (PC), which addresses the program memory for the current instruction execution, provided by hardware, the instruction execution time is reduced by several clock periods for branch, call, return, and break instructions, compared to that for the PFP only.

7. UNIQUE μPD70136A INSTRUCTIONS

7.1 VARIABLE BIT FIELD MANIPULATION INSTRUCTIONS

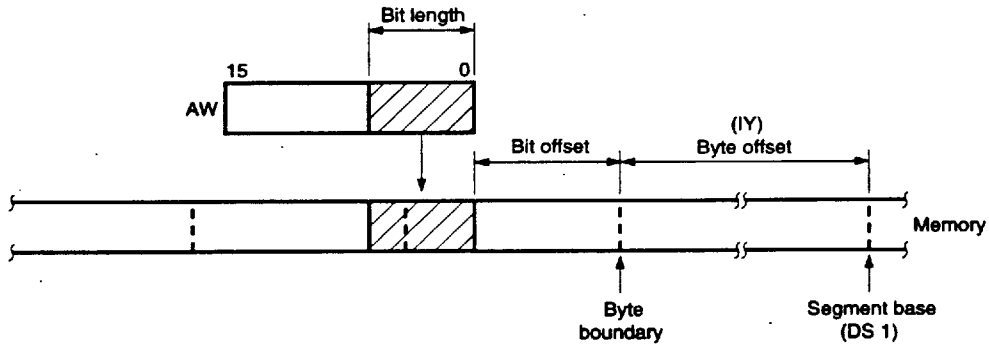
The INS (Insert Bit Field) and EXT (Extract Bit Field) instructions are provided as variable bit field manipulation instructions. These instructions are very effective for computer graphics and high level language. For example, these instructions are effective for Pascal packed array and record type data structure.

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

Of the 16 bits of data contained in the AW register, the data for the lower bits, specified by the second operand, is transferred to the memory area determined by the byte offset addressed by the DS1 segment register and IY indexed register plus the bit offset specified by the value (0-15) for the first operand.

After the transfer, the IY register and the register, specified by the first operand, are automatically updated to indicate the next bit field.

Only 0-15 (0 specifies 1-bit length, 15 specifies 16-bit length) are effective as the value for the second operand.



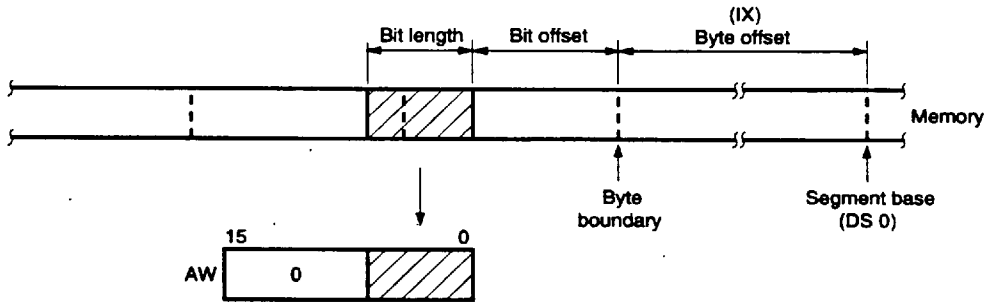
Bit field data can cross memory byte boundaries.

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

Data for the bit field, whose length is specified by the second operand, is loaded from the memory area determined by the byte offset addressed by the DS0 segment register and IX indexed register, plus the bit offset specified by the value (0-15) for the first operand to the AW register.

After the transfer, the IX register and the register specified by the first operand are automatically updated to indicate the next bit field.

Only 0–15 (0 specifies 1 bit length, 15 specifies 16 bit length) are effective as the value for the second operand.



Bit-field data can cross memory byte boundaries.

7.2 PACKED BCD OPERATION INSTRUCTIONS

The ADD4S, SUB4S, and CMP4S instructions process packed BCD as strings. The ROR4 and ROL4 instructions process packed BCD as byte or word format operands.

Assembler macro processing must be used for string processing of rotate instructions.

(1) ADD4S

This instruction adds the packed BCD string, addressed by the IX index register, to the packed BCD string, addressed by the IY index register, and stores the result in the string, addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register, and the operation result will affect the zero (Z) and carry (CY) flags.

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

(2) SUB4S

This instruction subtracts the packed BCD string, addressed by the IX index register, from the packed BCD string, addressed by the IY index register, and stores the result in the string, addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register, and the operation result will affect the zero (Z) and carry (CY) flags.

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

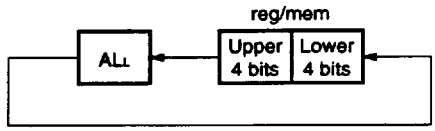
(3) CMP4S

This instruction performs the same operation as SUB4S, except that the result is not stored and only the zero flag (Z) and carry flag (CY) are affected.

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

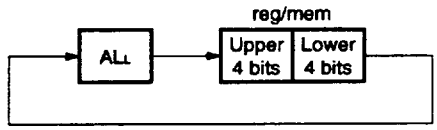
(4) ROL4

This instruction treats the byte data for the register or memory operand specified by the instruction as BCD data and uses the lower 4 bits of the AL register (ALL) to rotate that data one BCD digit to the left.



(5) ROR4

This instruction treats the byte data for the register or memory operand specified by the instruction as BCD data and uses the lower 4 bits of the AL register (ALL) to rotate that data one BCD digit to the right.



7.3 STACK MANIPULATION INSTRUCTIONS

(1) PREPARE imm16,imm8

This instruction is used to generate a "stack frame" necessary for a block structure high level language (such as Pascal and Ada). The stack frame contains the pointers to point frames of variables that can be referenced from the procedure, and local variable area.

The following explanation uses the following program example written in a Pascal style 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 All variables are word variables.

In this program, procedure blocks are nested in three levels. Procedure P defines variables a, b, and c, procedure Q defines variables d and e, and procedure R defines variables f and g. Therefore, a, b, and c are referenced from procedure Q, and d and e in addition to a, b, and c, are referenced from procedure R as global variables.

The PREPARE instruction copies the frame pointer, in order to assure local variable area and enable referencing to global variables. The first operand specifies the size (bytes) of the area assured for the local variables. The second operand indicates the depth of the procedure block (this depth is referred to as lexical level).

The base address for the frame generated by the PREPARE instruction is set into the base pointer BP.

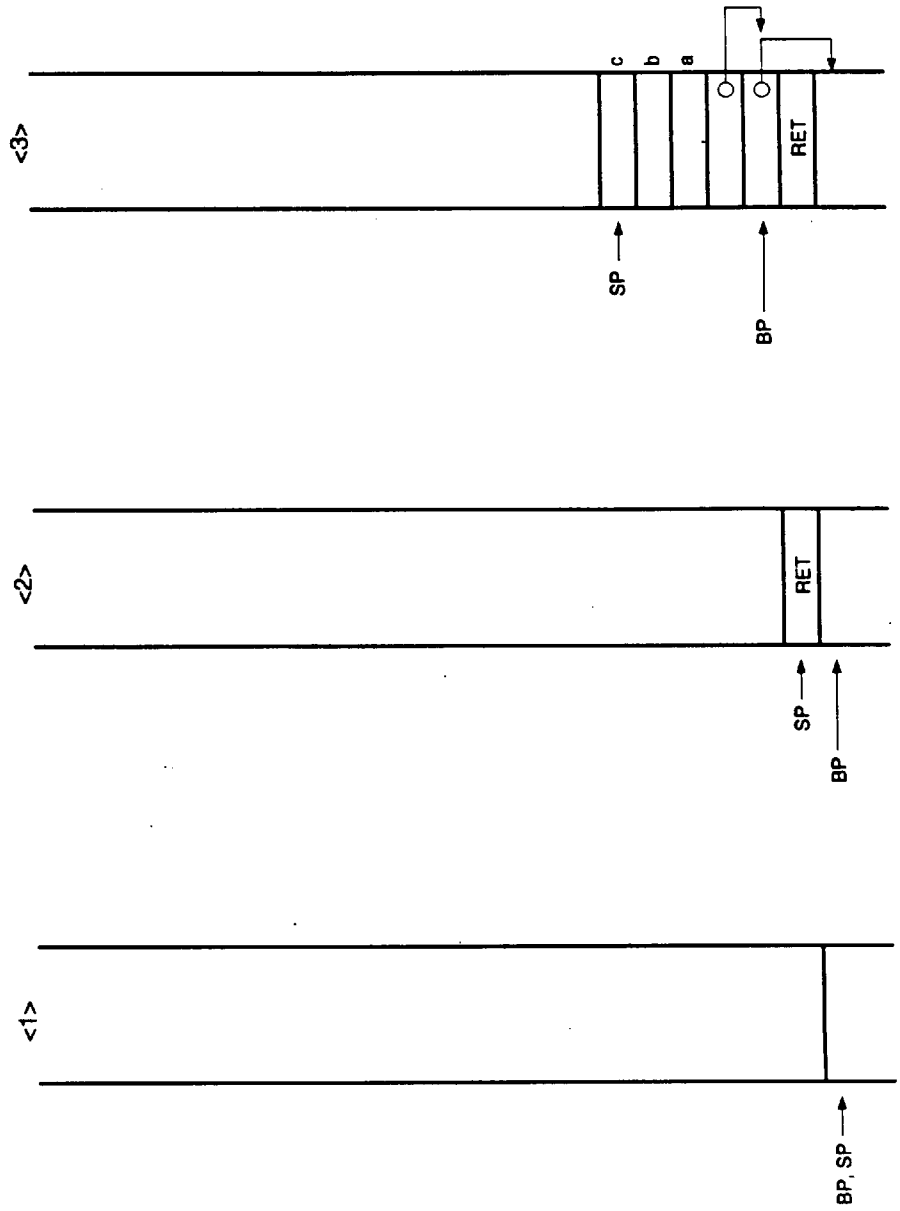
When the above EXAMPLE program is compiled, the assembler program, shown on the next page will be created (the DISPOSE instruction used in the assembler program returns the stack pointer SP and the base pointer BP to the condition which existed before the PREPARE instruction was executed. Refer to (2)).

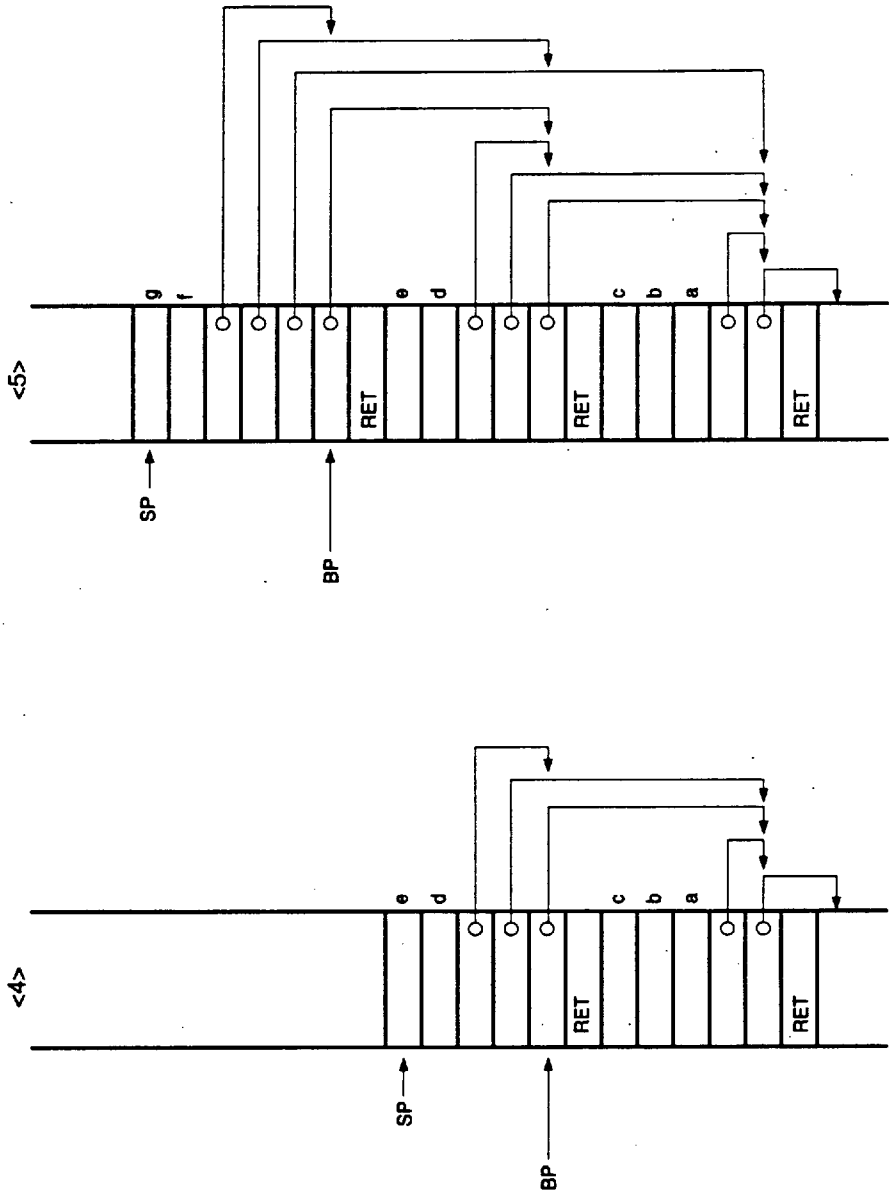
:ASSEMBLER PROGRAM

```
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], AW
         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    DELVEL = -2
;  E = -4    ELEVEL = -2
;  F = -2    FLEVEL = -3
;  G = -4    GLEVEL = -3
```

The following shows the process in which stack frames are generated according to the program execution progress. The numbers correspond to the numbers written in the comment fields.





The PREPARE instruction first stores the BP into the stack. This is to restore the BP for the procedure which made the call, when the procedure is completed. The frame pointers (stored BP) in the range that can be referenced from the called procedure are then loaded into the stack. The range that can be referenced means a value which is the lexical level of the procedure minus 1.

If the lexical level is 1 or greater, the own frame pointer is also loaded into the stack. This is to copy the frame pointer for the procedure, which made the call, when copying the frame pointer in the procedure called from this procedure.

The value for the new frame pointer is then set into the BP, and the local variable area to be used by the procedure is assured in the stack. That is, the SP is decremented by the number of local variables.

```
   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 Method

(a) Accessing local variable

A local variable is allocated in the procedure's own frame. Therefore, the effective address for the local variable EA.L can be computed as follows:

EA.L = SS:(BP+offset)

Where, offset value is the result of addition of the frame size (base value for the frame that can be referenced) loaded in the flame, with the offset value from the base value of the local variable area to the variable.

(b) Accessing global variable

A global variable accesses the base pointer to be referenced from the old base pointers loaded in the stack frame, then add the offset value to the variable to be referenced to that value. This value is the address at which the global variable is located. Therefore, the the effective address for the global variable EA.G can be computed as follows:

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

Where, offset1 is the offset value from the base value (BP value) for the current frame to the address, in which the base address for frame containing the global variable is stored.

Offset2 is the offset value from the base value for the frame containing the variable to be referenced to that variable.

(2) DISPOSE

The DISPOSE instruction releases one frame from stack frames generated by the PREPARE instruction. The point value, indicating the previous frame, is loaded into the BP, and the point value, indicating the bottom position of the frame, is loaded into the SP.

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

7.4 CHECK ARRAY BOUNDARY INSTRUCTION

This instruction is used to verify that index values, pointing to the elements of an array data structure, are within the defined range. If the index value is not between these defined ranges when CHKIND is executed, a BRK5 will occur.

When using the CHKIND instruction, the defined value must be set into 2 words (the 1st word defines the lower limit, the second word specifies the upper limit) in the memory in advance. The index value should be a register (an arbitrary 16-bit register) used by the array manipulation program.

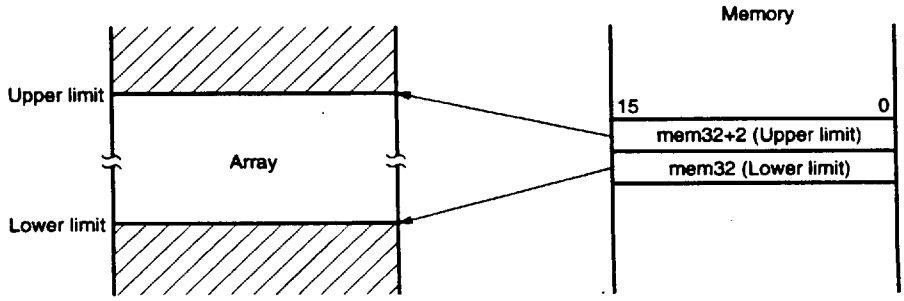
CHKIND, reg 16, mem 32

When (mem32) > reg16 or (mem 32+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[Note]
PC ← TA

} = BRK 5

Note Start address for CHKIND instruction



7.5 ADDRESS EXPANSION MODE CONTROL INSTRUCTIONS

The μPD70136A has the following two instructions in order to turn on/turn off the expanded addressing mode (XA mode):

- BRKXA instruction
- RETXA instruction

With these two instructions, bit 0 (XA flag) for the XAM register (internal I/O port: FF80H) can be set/reset to turn on/turn off the expanded addressing mode.

(1) BRKXA instruction

This instruction is used to turn on the expanded addressing mode.

The control is transferred to the address stored in the interrupt vector table entry, specified by the instruction, and bit 0 (XA flag) for the XAM register (internal I/O port: FF80H) is set to 1.

Interrupt acknowledge bus cycle will not be executed.

PC, PS, and PSW will not be stored into the stack.

(2) RETXA instruction

This instruction is used to turn off the expanded addressing mode.

The control is transferred to the address stored in the interrupt vector table entry, specified by the instruction, and bit 0 (XA flag) for the XAM register (internal I/O port: FF80H) is reset to 0.

Interrupt acknowledge bus cycle will not be executed.

PC, PS, and PSW will not be stored into the stack.

7.6 FLOATING-POINT OPERATION COPROCESSOR CONTROL INSTRUCTIONS

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

These instructions are used to control the external coprocessor μPD72291.
These instructions perform different operations, depending on whether or not the coprocessor is connected.

(1) When coprocessor is connected

Upon fetching the FPO1 or FPO2 instruction, the CPU outputs instructions for coprocessor to perform operations and the CPU only performs supportive processing for operation chips (effective address computation, physical address generation, and memory read cycle initiation) necessary for the coprocessor to perform operations.

In terms of function, FPO1 and FPO2 are identical, except that the type of code is different.

In general, when writing in an assembler language, a mnemonic, corresponding to each instruction for the coprocessor, is used rather than FPO1 and FPO2 mnemonics.

When the FPO1 or FPO2 instruction is fetched, the CPU initiates memory read cycle, if the instruction is requesting memory accessing. However, the data read out by this operation is to be used by the coprocessor, so that the CPU will not use clocks in this data.

When the coprocessor needs memory write cycle, the CPU initiates memory write cycle for the coprocessor, and the coprocessor accesses the data bus.

When the CPU fetches the POLL instruction, the CPU samples the $\overline{\text{CPBUSY}}$ pin every two clock periods, in order to synchronize with the coprocessor, and waits for execution until the $\overline{\text{CPBUSY}}$ pin becomes high.

(2) When coprocessor is not connected

When the CPU fetches the FPO1, FPO2, or POLL instruction, vector No.7 interrupt is generated, regardless of the interrupt enable flag status.

FPO1, FPO2, POLL instruction
↓
Reads vector 7 table
↓
Fetches interrupt destination
↓
Stores PC[Note], PS, PSW

Go to coprocessor, not present routine (user created routine)

Note Start address for the coprocessor control instruction

Whether or not the coprocessor is connected is determined by the $\overline{\text{CPBUSY}}$ pin status.

$\overline{\text{CPBUSY}}$ pin status when reset
$\overline{\text{CPBUSY}}$ (input) = "L" .... Coprocessor not connected
$\overline{\text{CPBUSY}}$ = "H" .... Coprocessor connected

7.7 UNDEFINED INSTRUCTION CODES

Vector number 6 interrupt is generated for any undefined instruction code listed in Table 7-1, regardless interrupt enable flag states.

Undefined instruction
↓
Reads vector 6 table
↓
Fetches interrupt destination
↓
Stores PC[Note], PS, PSW
↓
Go to undefined instruction processing routine (user created routine)

Note Trapped undefined code start address

Table 7-1. Undefined Codes List

No.	1st	2nd
1-16	00001111	00000000-00001111
17	00001111	00100001
18-20	00001111	00100011-00100101
21	00001111	00100111
22	00001111	00101001
23-27	00001111	00101011-00101111
28	00001111	00110000
29	00001111	00110010
30-34	00001111	00110100-00111000
35	00001111	00111010
36-39	00001111	00111100-00111111
40-185	00001111	01000000-11001110
186-200	00001111	11010000-11011111
201-215	00001111	11100001-11101111
216-230	00001111	11110001-11111111
231	01100011	
232-295	10001101	11000000-11111111
296-359	11000100	11000000-11111111
360-423	11000101	11000000-11111111
424-551	110x000x	xx110xxx
552-615	1101001x	xx110xxx
616	11010110	
617-680	1111011x	xx001xxx
681-744	1111111x	xx111xxx

8. INTERRUPT OPERATION

The interrupts supported by the μPD70136A can be divided into two types; interrupts generated by external interrupt requests and traps generated by software processing. They are:

(1) External interrupts

(i) $\overline{\text{NMI}}$ input (nonmaskable)

(ii) INT input (maskable)

(2) Software traps

(a) By instruction execution result
- Divide error during DIV or DIVU instruction
- Array bound error during CHKIND
- Undefined instruction
- Coprocessor error
- Coprocessor not connected

(b) Conditional break instruction
- When V = 1 when BRKV instruction is executed

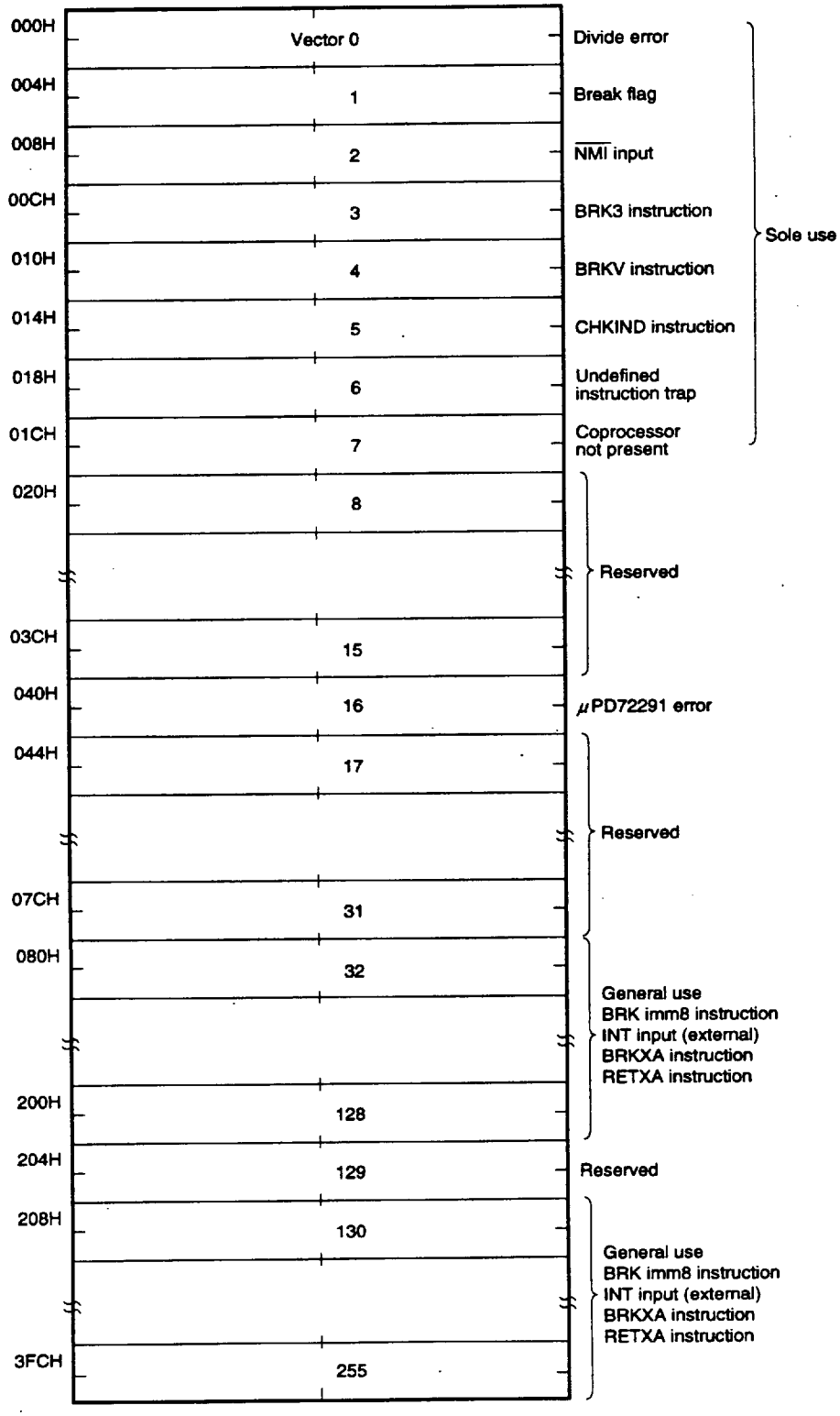
(c) Unconditional break instruction
- 1-byte break instruction BRK 3
- 2-byte break instruction BRK imm8

(d) Flag processing (single step)
- Sets the BRK flag by stack manipulation

For any interrupt, one location of the provided interrupt vector table is automatically selected, or is selected each time by specification, to determine the interrupt routine start address.

Figure 8-1 shows the interrupt vector table. This table is allocated to the 1K-byte area for memory addresses 000H to 3FFH, and can contain interrupt routine start addresses for 256 vectors (4 bytes for each vector).

Figure 8-1. Interrupt Vector Table



Vectors 0-7, and 16 are indicated for specific uses, and vectors 8-15, 17-31, and 129 are reserved and cannot be used for general purpose.

Vectors 32-128, 130-255 can be used for general purposes, and the following 3 can be used.

- Two-byte break instruction ····· BRK imm8
- Expansion address mode instruction ····· BRKXA, RETXA
- INT input

Each interrupt vector consists of 4 bytes. The lower 2 bytes are loaded into the PC as offset, and the upper 2 bytes are loaded into the PS as base.

Example Vector 0

000H	001H
002H	003H

PS ← (003H, 002H)
PC ← (001H, 000H)

The programmer must initialize the contents of each vector in the beginning of a program, according to this format. The following indicates basic steps to jump to an interrupt service routine.

TA ← Lower bytes of vector (offset)
TC ← Upper bytes of vector (base)
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

Cautions 1. For interrupts generated by the following causes, the PC and PS values, which indicate the start address for the instruction in which an interrupt is generated, are stored into the stack.

- **Undefined instruction code trap**
- **Coprocessor not present interrupt**
- **μPD72291 error interrupt**

If an interrupt is generated by other than the above mentioned causes, the PC and PS values, which indicate the start address for the next instruction among the instructions in which the interrupt is generated, are stored into the stack.

2. The μPD70136A will not accept other NMI interrupt in the NMI interrupt service routine, until the RETI instruction is executed. However, when the standby mode is initiated, by executing the HALT instruction in the NMI service routine, the NMI interrupt is accepted. In this case, the standby mode is released and an NMI interrupt (vector 2) is generated.

9. INTERFACING TO THE μPD72291 FLOATING-POINT COPROCESSOR

The following describes interfacing the μPD72291 floating-point coprocessor to the μPD70136A.

(1) System configuration

Figure 9-1 shows a system configuration example, in which the μPD72291 is connected to the μPD70136A.

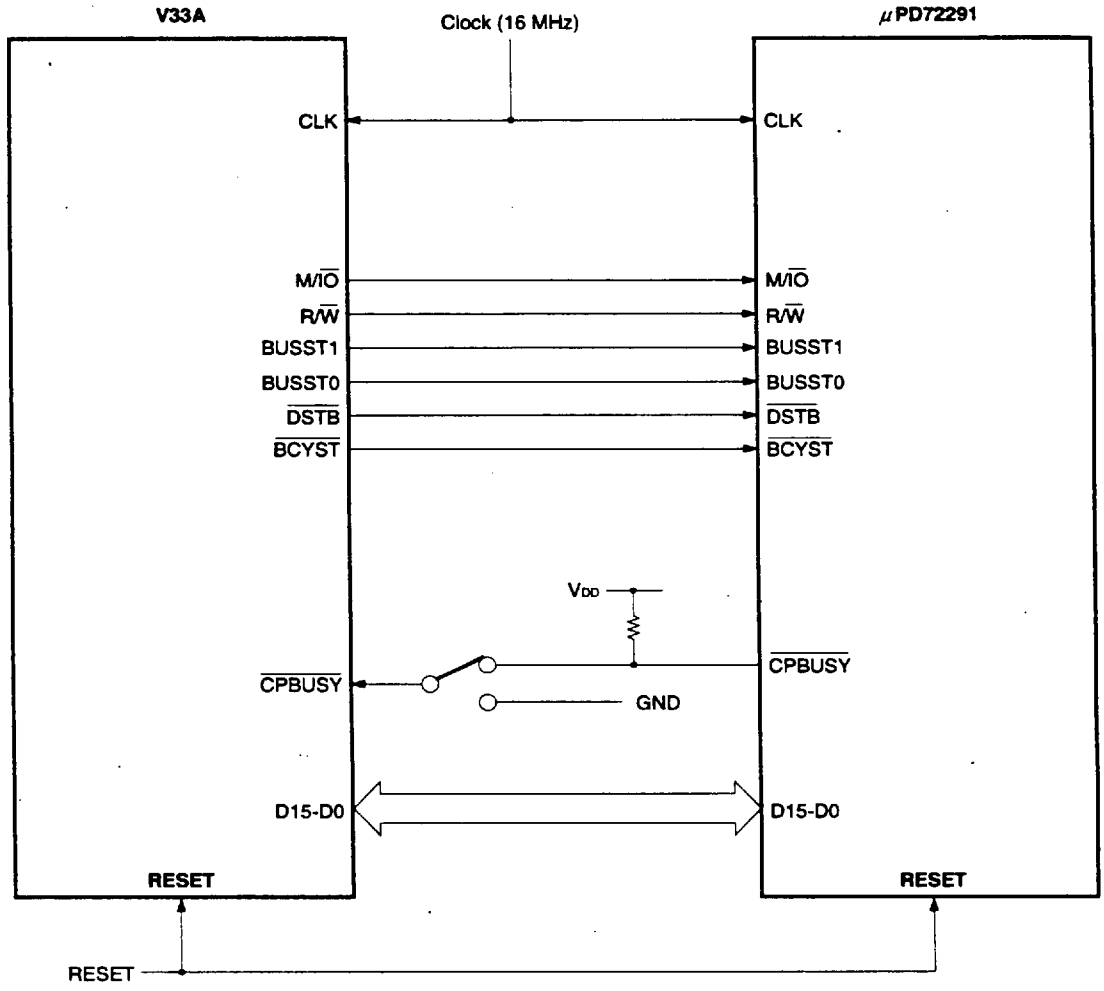
The μPD72291 can be connected to the μPD70136A without an additional external circuit.

Directly connect the input/output signals (CLK, $\overline{\text{RESET}}$, BUSST1, BUSST0, M/$\overline{\text{IO}}$, R/$\overline{\text{W}}$, $\overline{\text{DSTB}}$, D15-D0) for μPD72291 to the μPD70136A, as shown in Figure 9-1 The same clock must be supplied to the clock for the μPD72291 and that for the μPD70136A.

The μPD72291 $\overline{\text{CPBUSY}}$ signal must be connected to the μPD70136A $\overline{\text{CPBUSY}}$ signal, and must be pulled-up with a resistor (approximately 2kΩ) to V_{DD}.

When only the socket for the μPD72291 is provided, but the μPD72291 is not connected, provide a switch to the $\overline{\text{CPBUSY}}$ signal for the μPD70136A, so that the $\overline{\text{CPBUSY}}$ pin can be connected to GND by the switch. The μPD70136A samples the $\overline{\text{CPBUSY}}$ signal when reset. If the $\overline{\text{CPBUSY}}$ pin is low, the coprocessor, not present exception, will be generated, if an instruction for the μPD72291 is attempted to be executed.

Figure 9-1. Typical System Configuration



Cautions 1. The same clock must be supplied to the CLK pin for the μPD72291 and that for the μPD70136A. When connecting the μPD72291 to the μPD70136A, the μPD72291 $\overline{\text{CPBUSY}}$ signal must be connected to the μPD70136A $\overline{\text{CPBUSY}}$ pin, and pulled up with a resistor. When only the socket for the μPD72291 is provided, but the μPD72291 is not connected, provide a switch to the μPD70136A $\overline{\text{CPBUSY}}$ signal, so that the $\overline{\text{CPBUSY}}$ pin can be connected to GND by the switch.

2. Data bus lines for the μPD70136A and those for the μPD72291 must be directly connected, without anything including buffers.

(2) Bus cycle

The following describes coprocessor related bus cycles generated by the μPD70136A.

The μPD70136A generates bus cycles for accessing the memory and I/O. In addition to these bus cycles, the μPD70136A also generates bus cycles for the coprocessor. The data transfer source and destination are indicated by the bus status signals (M/$\overline{IO}$, R/$\overline{W}$, BUSST1, BUSST0). Table 9-1 indicates the relationship between the bus status signals and the bus cycles.

Table 9-1. μPD70136A Bus Status Signals and Bus Cycles

M/IO	R/W	BUSST1	BUSST0	Bus Cycle	Transfer Direction
0	1	1	0	Coprocessor read	[COP → CPU]
0	0	1	0	Coprocessor write	[CPU → COP]
1	1	1	0	Memory read for coprocessor	[Memory → COP]
1	0	1	0	Memory write for coprocessor	[COP → memory]

Caution To execute the instruction for the μPD72291, be sure to place the memory operand in an even address. Specify the 16-bit length by using the dynamic bus sizing function. This is because the μPD72291 accesses 16-bit data in one bus cycle. Therefore, if a memory sized to be 8 bits is specified as an operand, the normal operation cannot be performed.

Remark COP is an abbreviation for coprocessor.

(a) Coprocessor read

This bus cycle is used to read the status from the status word port (STWP) for the μPD72291. The address (A23-A0) output from the μPD70136A has no meaning (the μPD70136A outputs 000008H). The data bus is driven by the μPD72291.

The bus cycle has two clock periods.

(b) Coprocessor write

This bus cycle is used to write an instruction to the command word port (CMWP) for the the μPD72291. The address (A23-A0) output from the μPD70136A has no meaning (the μPD70136A outputs 000000H). The data bus is driven by the μPD70136A.

The bus cycle has two clock periods.

(c) Coprocessor memory read

This bus cycle is used to transfer memory data to the source operand word port (SOPWP) for the μPD72291. The address (A23-A0) output from the μPD70136A is the memory address. The data bus is driven by the memory.

The bus cycle has three clock periods (the bus cycle is automatically extended by one clock by the μPD70136A). When inserting a wait cycle, set the $\overline{READY}$ input for the μPD70136A to high.

(d) Coprocessor memory write

This bus cycle is used to transfer an operation result from the destination operand word port (DOPWP) for the μPD72291 to the memory. The address (A23-A0) output from the μPD70136A is the memory address. The data bus is not driven by the μPD70136A, but by the μPD72291.

The bus cycle has three clock periods (the bus cycle is automatically extended by one clock by the μPD70136A). When inserting a wait cycle, set the READY input for the μPD70136A to high.

Remark STWP, CMWP, and DOPWP are built-in ports of μPD72291.

10. RESET FUNCTION

When a low level has been input to the RESET pin for the duration of six clocks and then the RESET pin is made high, the μPD70136A is reset.

When reset, the CPU is initialized as shown in Table 10-1, and instruction prefetch is started from address FFFF0H.

Table 10-1. Resetting CPU

Target	Initial Value	Remarks
PFP	0000H	Start address: FFFF0
PC	0000H	
PS	FFFFH	
SS	0000H	
DS0	0000H	
DS1	0000H	
PSW	V DIR IE BRK Upper: 1 1 1 1 0 0 0 0 S Z AC P CY Lower: 0 0 0 0 0 0 0 0	
Queue	Clear	
XAM	7 6 5 4 3 2 1 0 – – – – – – × ×	XA flag: normal address
PGR1-PGR64	15 14 13 12 11 10 9 8 – – – – – – × × 7 6 5 4 3 2 1 0 × × × × × × × ×	Page register: undefined

Remark ×: Retains status immediately before reset.

11. STANDBY FUNCTION

The μPD70136A offers standby mode to reduce power consumption. The standby mode is entered after executing a HALT instruction.

In the standby mode, the internal clock is supplied only to those circuits related to functions required to exit this mode and bus hold control functions. As a result, power consumption is reduced by several times the normal operation.

The standby mode is exited, when RESET, or when an external interrupt ($\overline{NMI}$, INT) is received.

The bus hold function still operates during the standby mode. However, the CPU returns to the standby mode, when the bus hold request is removed.

12. INSTRUCTION SET

Table 12-1. Operand Types

Symbol	Meaning
reg,reg'	8/16-bit general-purpose register
reg 8, reg 8'	8-bit general-purpose register
reg 16, reg 16'	16-bit general-purpose register
dmem	8/16-bit memory location
mem	8/16-bit memory location
mem 8	8-bit memory location
mem 16	16-bit memory location
mem 32	32-bit memory location
imm	Constant (0 to FFFFH)
imm 3	Constant (0 to 7)
imm 4	Constant (0 to FH)
imm 8	Constant (0 to FFH)
imm 16	Constant (0 to FFFFH)
acc	Register AW or AL
sreg	Segment register
src-table	256-byte conversion table
src-block	Block name addressed by register IX
dst-block	Block name addressed by register IY
near-proc	Procedure within the current segment
far-proc	Procedure within a different program segment
near-label	Label within the current segment
short-label	Label between −128 and +127 bytes from the end of the current instruction
far-label	Label within a different program segment
memptr 16	Word containing the destination offset address in the current program segment
memptr 32	Double word containing the destination offset address and segment base address in another program segment
regptr 16	16-bit general-purpose register containing the destination offset address in another segment
pop-value	Number of bytes to discard from the stack (0-64K, normally an even number)
fp-op	Immediate value to identify the instruction code of the external floating-point arithmetic coprocessor
R	Register set

Table 12-2. Operation Codes

Symbol	Meaning	
W	Byte/word specification bit (0: byte, 1: word). However, when s = 1, even if w = 1, sign expansion byte data is used as the word operand.	★
reg,reg'	Register field (000-111)	
mem	Memory field (000-111)	
mod	Mode field (00-10)	
s	Sign expansion specification bit (0: no sign expansion, 1: sign expansion)	★
X,XXX,YYY,ZZZ	Data to identify the instruction code for the external floating-point arithmetic coprocessor	

Table 12-3. Operand Types

Symbol	Meaning
AW	Accumulator (16 bits)
AH	Accumulator (upper byte)
AL	Accumulator (lower byte)
BW	Register BW (16 bits)
CW	Register CW (16 bits)
CL	Register CW (low byte)
DW	Register DW (16 bits)
BP	Base pointer (16 bits)
SP	Stack pointer (16 bits)
PC	Program counter (16 bits)
PSW	Program status word (16 bits)
IX	Index register (source) (16 bits)
IY	Index register (destination) (16 bits)
PS	Program segment register (16 bits)
SS	Stack segment register (16 bits)
DS0	Data segment 0 register (16 bits)
DS1	Data segment 1 register (16 bits)
AC	Auxiliary carry flag
CY	Carry flag
P	Parity flag
S	Sign flag
Z	Zero flag
DIR	Direction flag
IE	Interrupt enable flag
V	Overflow flag
BRK	Break flag
XA	Expansion address flag
(...)	Memory contents indicated by parentheses
disp	Displacement (8/16 bits)
ext-disp8	8-bit displacement sign expanded to 16-bit
temp	Temporary register (8/16/32 bits)
temp1, 2	Temporary register (16 bits)
TA	Temporary register A (16 bits)
TB	Temporary register B (16 bits)
TC	Temporary register C (16 bits)
tmpcy	Temporary carry flag (1 bit)
seg	Immediate segment data (16 bits)
offset	Immediate offset data (16 bits)
←	Transfer direction
+	Addition
−	Subtraction
×	Multiplication
÷	Division
%	Modulo
∧	AND
∨	OR
∀	XOR
xxH	Two-digit hexadecimal value
xxxxH	Four-digit hexadecimal value

Table 12-4. Flag Operation

Symbol	Meaning
(blank)	No change
0	Cleared to 0
1	Set to 1
×	Set or cleared, according to result.
U	Undefined
R	Value saved earlier is restored

Table 12-5. Memory Addressing Mode

mem \ mod	00	01	10
000	BW+IX	BW+IX+disp 8	BW+IX+disp 16
001	BW+IY	BW+IY+disp 8	BW+IY+disp 16
010	BP+IX	BP+IX+disp 8	BP+IX+disp 16
011	BP+IY	BP+IY+disp 8	BP+IY+disp 16
100	IX	IX+disp 8	IX+disp 16
101	IY	IY+disp 8	IY+disp 16
110	DIRECT ADDRESS	BP+disp 8	BP+disp 16
111	BW	BW+disp 8	BW+disp 16

Table 12-6. 8- and 16-Bit General Register Selection

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 12-7. Segment Register Selection

sreg	
00	DS1
01	PS
10	SS
11	DS0

Instruction sets are explained in the following pages.

The number of clocks shown in the table is the time required for the execution unit to execute an instruction and is under the following conditions:

(1) Excluding prefetch time, pre-decode time, and wait time to use the bus

(2) It is assumed that know wait state is inserted in memory access, i.e., the number of clocks for one bus cycle is two

(3) It is assumed that know wait state is inserted in I/O access

(4) Primitive block transfer instruction and primitive I/O instruction include the repeat prefix

(5) In the case of the instruction with byte processing and word processing (with W bit), the figure on the left of the / indicates the value of byte or word processing to and even address, and the figure on the right indicates the value of word processing to an odd address

(6) The number of clocks when 16 bits are specified for the bus sizing function. To specify 8 bits double the bus cycle for the word data to an even address

(7) Indicates the normal address mode

For the inter-block transfer instruction, refer to **Table 12-8.**

★ **Table 12-8. Number of Clocks for Block Transfer Related Instructions**

Instruction Processing	Number of Clocks			
	Byte (W = 0)	Word Processing (W = 1)		
		Odd, Odd Address	Odd, Even Address	Even, Even Address
MOVBK	6/rep (6)	10/rep (10)	8/rep (8)	6/rep (6)
CMPBK	12/rep – 1 (11)	16/rep – 1 (15)	14/rep – 1 (13)	12/rep – 1 (11)
CMPM	10/rep – 1 (9)	12/rep – 1 (11)	———	10/rep – 1 (9)
LDM	3/rep + 2 (5)	5/rep + 2 (7)	———	3/rep + 2 (5)
STM	3/rep (3)	5/rep (5)	———	3/rep (3)
INM	8/rep + 4 (12)	14/rep + 8 (22)	When I/O address is odd: 12/rep + 8 (20) When memory address is odd: 10/rep + 4 (14)	8/rep + 4 (12)
OUTM	12/rep – 6 (6)	22/rep – 6 (16)	When I/O address is odd: 20/rep – 6 (14) When memory address is odd: 14/rep – 6 (8)	12/rep – 6 (6)

Remark 1. Value indicated in parentheses applies only for one processing.

2. "/rep" gives the repeat count. For execution one time, the repeat count is 1.

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags				
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Data transfer instruction	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	3/5	(mem)←reg						
		reg, mem	1	0	0	0	1	0	1	W	mod		reg		mem				2-4	5/7	reg←(mem)						
		mem, imm	1	1	0	0	0	1	1	W	mod	0	0	0	mem				3-6	3/5	(mem)←imm						
		reg, imm	1	0	1	1	W		reg										2-3	2	reg←imm						
		acc, dmem	1	0	1	0	0	0	0	W									3	5/7	When W = 0, AL←(dmem) When W = 1, AH←(dmem+1), AL←(dmem)						
		dmem,acc	1	0	1	0	0	0	1	W									3	3/5	When W = 0, (dmem)←AL When W = 1, (dmem+1)←AH, (dmem)←AL						
		sreg, reg16	1	0	0	0	1	1	1	0	1	1	0	sreg		reg			2	2	sreg←reg16 sreg: SS, DS0, DS1						
		sreg,mem16	1	0	0	0	1	1	1	0	mod	0	sreg		mem				2-4	5/7	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	3/5	(mem16)←sreg						
		DS0, reg16, mem32	1	1	0	0	0	1	0	1	mod		reg		mem				2-4	10/14	reg16←(mem32), DS0←(mem32+2)						
		DS1, reg16, mem32	1	1	0	0	0	1	0	0	mod		reg		mem				2-4	10/14	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	2	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	2	reg16←mem16						

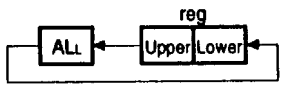
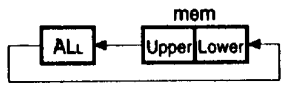
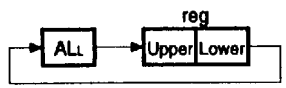
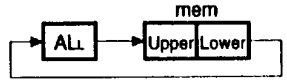
Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Data transfer instruction	TRANS	src-table	1	1	0	1	0	1	1	1									1	5	AL←(BW+AL)						
	XCH	reg, reg'	1	0	0	0	0	1	1	W	1	1	reg			reg'			2	3	reg↔reg'						
		mem, reg reg, mem	1	0	0	0	0	1	1	W	mod		reg			mem			2-4	8/12	(mem)↔reg						
		AW, reg16 reg16, AW	1	0	0	1	0	reg											1	3	AW↔reg16						
Repeat prefix	REPC		0	1	1	0	0	1	0	1									1	2	While CW ≠ 0, executes the primitive block transfer of the successive bytes, and decrements (−1) CW. If any interrupt has been on hold, the interrupt is processed. Exits from loop, when CY ≠ 1.						
	REPNC		0	1	1	0	0	1	0	0									1	2	Same as above. Exits from loop, when CY ≠ 0.						
	REP REPE REPZ		1	1	1	1	0	0	1	1									1	2	While CW ≠ 0, executes the primitive block transfer of the successive bytes, and decrements (−1) CW. If any interrupt has been on hold, the interrupt is processed. Exits from loop, when the primitive block transfer instruction is CMPBK or CMPM and Z ≠ 1.						
	REPNE REPNZ		1	1	1	1	0	0	1	0									1	2	Same as above. Exits from loop, when Z ≠ 0.						

Caution **Eight or more repeat prefix instructions should not be attached to non-prefix instruction.**

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Primitive block transfer instruction	MOVBK	dst-block, src-block	1	0	1	0	0	1	0	W									1	See Table 12-8	When W = 0, (IY)←(IX) DIR = 0: IX←IX+1, IY←IY+1 DIR = 1: IX←IX−1, IY←IY−1 When W = 1, (IY+1,IY)←(IX+1,IX) DIR = 0: IX←IX+2, IY←IY+2 DIR = 1: IX←IX−2, IY←IY−2						
	CMPBK	src-block, dst-block	1	0	1	0	0	1	1	W									1	See Table 12-8	When W = 0, (IX) − (IY) DIR = 0: IX←IX+1, IY←IY+1 DIR = 1: IX←IX−1, IY←IY−1 When W = 1, (IX+1,IX) − (IY+1,IY) DIR = 0: IX←IX+2, IY←IY+2 DIR = 1: IX←IX−2, IY←IY−2	x	x	x	x	x	x
	CMPM	dst-block	1	0	1	0	1	1	1	W									1	See Table 12-8	When W = 0, AL−(IY) DIR = 0: IY←IY+1; DIR = 1: IY←IY−1 When W = 1, AW − (IY+1, IY) DIR = 0: IY←IY+2; DIR = 1: IY←IY−2	x	x	x	x	x	x
	LDM	src-block	1	0	1	0	1	1	0	W									1	See Table 12-8	When W = 0, AL←(IX) DIR = 0: IX←IX+1; DIR = 1: IX←IX−1 When W = 1, AW←(IX+1, IX) DIR = 0: IX+2; DIR = 1: IX←IX−2						
	STM	dst-block	1	0	1	0	1	0	1	W									1	See Table 12-8	When W = 0, (IY)←AL DIR = 0: IY←IY+1; DIR = 1: IY←IY−1 When W = 1, (IY+1, IY)←AW DIR = 0: IY←IY+2; DIR = 1: IY←IY−2						

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Bit field operation instruction	INS	reg8,reg8'	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	1	3	37-69 /39-77	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	37-69 /39-77	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	29-61 /33-63	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	29-61 /33-63	AW←16 bit field						
			1	1	0	0	0	reg																			
I/O instruction	IN	acc,imm8	1	1	1	0	0	1	0	W									2	5/7	When W = 0, AL←(imm8) When W = 1, AH←(imm8+1), AL←(imm8)						
		acc,DW	1	1	1	0	1	1	0	W									1	5/7	When W = 0, AL←(DW) When W = 1, AH←(DW+1), AL←(DW)						
	OUT	imm8,acc	1	1	1	0	0	1	1	W									2	3/5	When W = 0, (imm8)←AL When W = 1, (imm8+1)←AH,(imm8)←AL						
		DW,acc	1	1	1	0	1	1	1	W									1	3/5	When W = 0, (DW)←AL When W = 1, (DW+1)←AH,(DW)←AL						
Primitive I/O instruction	INM	dst–block, DW	0	1	1	0	1	1	0	W									1	See Table 12-8	When W = 0, (IY)←(DW) DIR = 0: IY←IY+1; DIR = 1: IY←IY−1						
																					When W = 1, (IY+1, IY)←(DW+1, DW) DIR = 0: IY←IY+2; DIR = 1: IY←IY−2						
	OUTM	DW, src–block	0	1	1	0	1	1	1	W									1	See Table 12-8	When W = 0, (DW)←(IX) DIR = 0: IX←IX+1; DIR = 1: IX←IX−1						
																					When W = 1, (DW+1,DW)←(IX+1,IX) DIR = 0: IX←IX+2; DIR = 1: IX←IX−2						

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Addition/Subtraction instruction	ADD	reg,reg'	0	0	0	0	0	0	1	W	1	1	reg				reg'		2	2	reg←reg+reg'	x	x	x	x	x	x
		mem,reg	0	0	0	0	0	0	0	W	mod		reg			mem			2-4	7/11	(mem)←(mem)+reg	x	x	x	x	x	x
		reg,mem	0	0	0	0	0	0	1	W	mod		reg			mem			2-4	6/8	reg←reg+(mem)	x	x	x	x	x	x
		reg,imm	1	0	0	0	0	0	s	W	1	1	0	0	0	reg			3-4	2	reg←reg+imm	x	x	x	x	x	x
		mem,imm	1	0	0	0	0	0	s	W	mod		0	0	0	mem			3-6	7/11	(mem)←(mem)+imm	x	x	x	x	x	x
		acc,imm	0	0	0	0	0	1	0	W									2-3	2	When W = 0, AL←AL+imm When W = 1, AW←AW+imm	x	x	x	x	x	x
	ADDC	reg,reg'	0	0	0	1	0	0	1	W	1	1	reg				reg'		2	2	reg←reg+reg'+CY	x	x	x	x	x	x
		mem,reg	0	0	0	1	0	0	0	W	mod		reg			mem			2-4	7/11	(mem)←(mem)+reg+CY	x	x	x	x	x	x
		reg,mem	0	0	0	1	0	0	1	W	mod		reg			mem			2-4	6/8	reg←reg+(mem)+CY	x	x	x	x	x	x
		reg,imm	1	0	0	0	0	0	s	W	1	1	0	1	0	reg			3-4	2	reg←reg+imm+CY	x	x	x	x	x	x
		mem,imm	1	0	0	0	0	0	s	W	mod		0	1	0	mem			3-6	7/11	(mem)←(mem)+imm+CY	x	x	x	x	x	x
		acc,imm	0	0	0	1	0	1	0	W									2-3	2	When W = 0, AL←AL+imm+CY When W = 1, AW←AW+imm+CY	x	x	x	x	x	x
	SUB	reg,reg'	0	0	1	0	1	0	1	W	1	1	reg				reg'		2	2	reg←reg−reg'	x	x	x	x	x	x
		mem,reg	0	0	1	0	1	0	0	W	mod		reg			mem			2-4	7/11	(mem)←(mem)−reg	x	x	x	x	x	x
		reg,mem	0	0	1	0	1	0	1	W	mod		reg			mem			2-4	6/8	reg←reg−(mem)	x	x	x	x	x	x
		reg,imm	1	0	0	0	0	0	s	W	1	1	1	0	1	reg			3-4	2	reg←reg−imm	x	x	x	x	x	x
		mem,imm	1	0	0	0	0	0	s	W	mod		1	0	1	mem			3-6	7/11	(mem)←(mem)−imm	x	x	x	x	x	x
		acc,imm	0	0	1	0	1	1	0	W									2-3	2	When W = 0, AL←AL−imm When W = 1, AW←AW−imm	x	x	x	x	x	x
	SUBC	reg,reg'	0	0	0	1	1	0	1	W	1	1	reg				reg'		2	2	reg←reg−reg'−CY	x	x	x	x	x	x
		mem,reg	0	0	0	1	1	0	0	W	mod		reg			mem			2-4	7/11	(mem)←(mem)−reg−CY	x	x	x	x	x	x
		reg,mem	0	0	0	1	1	0	1	W	mod		reg			mem			2-4	6/8	reg←reg−(mem)−CY	x	x	x	x	x	x
		reg,imm	1	0	0	0	0	0	s	W	1	1	0	1	1	reg			3-4	2	reg←reg−imm−CY	x	x	x	x	x	x
		mem,imm	1	0	0	0	0	0	s	W	mod		0	1	1	mem			3-6	7/11	(mem)←(mem)−imm−CY	x	x	x	x	x	x
		acc,imm	0	0	0	1	1	1	0	W									2-3	2	When W = 0, AL←AL−imm−CY When W = 1, AW←AW−imm−CY	x	x	x	x	x	x

Instruction Group	Mnemonic	Operand	Operation Code	Number of Bytes	Number of Clocks	Operation	Flags
			7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0				AC CY V P S Z
BCD operation instruction	ADD4S		0 0 0 0 1 1 1 1 / 0 0 1 0 0 0 0 0	2	18×n+2	dst BCD string←dst BCD string + src BCD string*	U x U U U x
	SUB4S		0 0 0 0 1 1 1 1 / 0 0 1 0 0 0 1 0	2	18×n+2	dst BCD string←dst BCD string – src BCD string*	U x U U U x
	CMP4S		0 0 0 0 1 1 1 1 / 0 0 1 0 0 1 1 0	2	14×n+7	dst BCD string – src BCD string*	U x U U U x
	ROL4	reg8	0 0 0 0 1 1 1 1 / 0 0 1 0 1 0 0 0 1 1 0 0 0 reg	3	9		
		mem8	0 0 0 0 1 1 1 1 / 0 0 1 0 1 0 0 0 mod 0 0 0 mem	3-5	15		
	ROR4	reg8	0 0 0 0 1 1 1 1 / 0 0 1 0 1 0 1 0 1 1 0 0 0 reg	3	13		
		mem8	0 0 0 0 1 1 1 1 / 0 0 1 0 1 0 1 0 mod 0 0 0 mem	3-5	19		
Increment/decrement instruction	INC	reg8	1 1 1 1 1 1 1 0 / 1 1 0 0 0 reg	2	2	reg8←reg8+1	x x x x x
		mem	1 1 1 1 1 1 1 W / mod 0 0 0 mem	2-4	7/11	(mem)←(mem)+1	x x x x x
		reg16	0 1 0 0 0 reg	1	2	reg16←reg16+1	x x x x x
	DEC	reg8	1 1 1 1 1 1 1 0 / 1 1 0 0 1 reg	2	2	reg8←reg8–1	x x x x x
		mem	1 1 1 1 1 1 1 W / mod 0 0 1 mem	2-4	7/11	(mem)←(mem)–1	x x x x x
		reg16	0 1 0 0 1 reg	1	2	reg16←reg16–1	x x x x x

n : Half of the number of digits of BCD

* : The number of digits in BCD is specified using the CL register. Its value can range from 1 to 254.

Instruction Group	Mnemonic	Operand	Operation Code	Number of Bytes	Number of Clocks	Operation	Flags
			7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0				AC CY V P S Z
Multiplication instruction	MULU	reg8	1 1 1 1 0 1 1 0 / 1 1 1 0 0 reg	2	8	AW←ALxreg8 AH = 0: CY←0,V←0 AH ≠ 0: CY←1,V←1	U x x U U U
		mem8	1 1 1 1 0 1 1 0 / mod 1 0 0 mem	2-4	12	AW←ALx(mem8) AH = 0: CY←0,V←0 AH ≠ 0: CY←1,V←1	U x x U U U
		reg16	1 1 1 1 0 1 1 1 / 1 1 1 0 0 reg	2	12	DW, AW←AWxreg16 DW = 0: CY←0,V←0 DW ≠ 0: CY←1,V←1	U x x U U U
		mem16	1 1 1 1 0 1 1 1 / mod 1 0 0 mem	2-4	16/18	DW,AW←AWx(mem16) DW = 0: CY←0,V←0 DW ≠ 0: CY←1,V←1	U x x U U U
	MUL	reg8	1 1 1 1 0 1 1 0 / 1 1 1 0 1 reg	2	8	AW←ALxreg8 AH = Sign extension for AL: CY←0,V←0 AH ≠ Sign extension for AL: CY←1,V←1	U x x U U U
		mem8	1 1 1 1 0 1 1 0 / mod 1 0 1 mem	2-4	12	AW←ALx(mem8) AH = Sign extension for AL: CY←0,V←0 AH ≠ Sign extension for AL: CY←1,V←1	U x x U U U
		reg16	1 1 1 1 0 1 1 1 / 1 1 1 0 1 reg	2	12	DW,AW←AWxreg16 DW = Sign extension for AW: CY←0,V←0 DW ≠ Sign extension for AW: CY←1,V←1	U x x U U U
		mem16	1 1 1 1 0 1 1 1 / mod 1 0 1 mem	2-4	16/18	DW,AW←AWx(mem16) DW = Sign extension for AW: CY←0,V←0 DW ≠ Sign extension for AW: CY←1,V←1	U x x U U U

Instruction Group	Mnemonic	Operand	Operation Code 7 6 5 4 3 2 1 0														Number of Bytes	Number of Clocks	Operation	Flags AC	CY	V	P	S	Z
Multiplication instruction	MUL (cont'd)	reg16, (reg16',)* imm8	0	1	1	0	1	0	1	1	1	1	reg		reg'		3	12	reg16←reg16'ximm8 Product ≤ 16 bits: CY←0,V←0 Product > 16 bits: CY←1,V←1	U	×	×	U	U	U
		reg16, mem16, imm8	0	1	1	0	1	0	1	1	mod		reg		mem		3-5	16/18	reg16←(mem16)ximm8 Product ≤ 16 bits: CY←0,V←0 Product > 16 bits: CY←1,V←1	U	×	×	U	U	U
		reg16, (reg16',)* imm16	0	1	1	0	1	0	0	1	1	1	reg		reg'		4	12	reg16←reg16'ximm16 Product ≤ 16 bits: CY←0,V←0 Product > 16 bits: CY←1,V←1	U	×	×	U	U	U
		reg16, mem16, imm16	0	1	1	0	1	0	0	1	mod		reg		mem		4-6	16/18	reg16←(mem16)ximm16 Product ≤ 16 bits: CY←0,V←0 Product > 16 bits: CY←1,V←1	U	×	×	U	U	U

* : The second operand can be omitted. When omitted, the same register, specified for the first operand, is assumed to be specified.

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Unsigned division instruction	DIVU	reg8	1	1	1	1	0	1	1	0	1	1	1	1	0	reg			2	11	temp←AW When temp+reg8 ≤ FFH, AH←temp%reg8, AL←temp+reg8 When 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[Note],PC←TA	U	U	U	U	U	U
		mem8	1	1	1	1	0	1	1	0	mod	1	1	0	mem				2-4	15	temp←AW When temp+(mem8) ≤ FFH, AH←temp%(mem8), AL←temp+(mem8) When 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[Note],PC←TA	U	U	U	U	U	U
		reg16	1	1	1	1	0	1	1	1	1	1	1	1	0	reg			2	19	temp←DW,AW When temp+reg16 ≤ FFFFH, DW←temp%reg16, AW←temp+reg16 When 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[Note],PC←TA	U	U	U	U	U	U
		mem16	1	1	1	1	0	1	1	1	mod	1	1	0	mem				2-4	23/25	temp←DW, AW When temp+(mem16) ≤ FFFFH, DW←temp%(mem16), AW←temp+(mem16) When 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[Note],PC←TA	U	U	U	U	U	U

Note Start address of DIVU instruction.

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags						
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z	
Signed division instruction	DIV	reg8	1	1	1	1	0	1	1	0	1	1	1	1	1	reg			2	17	temp←AW When temp+reg8 > 0 and temp+reg8 ≤ 7FH or when temp+reg8 < 0 and temp+reg8 > 0–7FH–1, AH←temp%reg8, AL←temp+reg8 When temp+reg8 > 0 and temp+reg8 > 7FH or when 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[Note],PC←TA	U	U	U	U	U	U	★
		mem8	1	1	1	1	0	1	1	0	mod	1	1	1	mem				2-4	20	temp←AW When temp+(mem8) > 0 and temp+(mem8) ≤ 7FH or when temp+(mem8) < 0 and temp+(mem8) > 0–7FH–1, AH←temp%(mem8), AL←temp+(mem8) When temp+(mem8) > 0 and temp+(mem8) > 7FH or when temp+(mem8) < 0 and temp+(mem8) ≤ 0–7FH–1, 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[Note],PC←TA	U	U	U	U	U	U	
		reg16	1	1	1	1	0	1	1	1	1	1	1	1	1	reg			2	24	temp←DW,AW When temp+reg16 > 0 and temp+reg16 ≤ 7FFFH or when temp+reg16 < 0 and temp+reg16 > 0–7FFFH–1, DW←temp%reg16,AW←temp+reg16 When temp+reg16 > 0 and temp+reg16 > 7FFFH or when temp+reg16 < 0 and temp+reg16 ≤ 0–7FFFH–1, 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[Note],PC←TA	U	U	U	U	U	U	
		mem16	1	1	1	1	0	1	1	1	mod	1	1	1	mem				2-4	28/30	temp←DW,AW When temp+(mem16) > 0 and temp+(mem16) ≤ 7FFFH or when temp+(mem16) < 0 and temp+(mem16) > 0–7FFFH–1, DW←temp%(mem16),AW←temp+(mem16) When temp+(mem16) > 0 and temp+(mem16) > 7FFFH or when temp+(mem16) < 0 and temp+(mem16) ≤ 0–7FFFH–1, 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[Note],PC←TA	U	U	U	U	U	U	

Note The address following DIV instruction.

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
BCD coprocessor instruction	ADJBA		0	0	1	1	0	1	1	1									1	4	When AL ∧0FH > 9 or AC = 1, AL←AL+6 AH←AH+1, AC←1, CY←AC, AL←AL ∧0FH	x	x	U	U	U	U
	ADJ4A		0	0	1	0	0	1	1	1									1	2	When AL ∧0FH > 9 or AC = 1, AL←AL+6, AC←1 When AL > 9FH or CY = 1, AL←AL+60H, CY←1	x	x	U	x	x	x
	ADJBS		0	0	1	1	1	1	1	1									1	4	When AL ∧0FH > 9 or AC = 1, AL←AL−6, AH←AH−1, AC←1 CY←AC, AL←AL ∧0FH	x	x	U	U	U	U
	ADJ4S		0	0	1	0	1	1	1	1									1	2	When AL ∧0FH > 9 or AC = 1, AL←AL−6, AC←1 When AL > 9FH or CY = 1, AL←AL−60H, CY←1	x	x	U	x	x	x
Data conversion	CVTBD		1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	0	2	12	AH←AL+0AH, AL←AL%0AH	U	U	U	x	x	x
	CVTDB		1	1	0	1	0	1	0	1	0	0	0	0	1	0	1	0	2	8	AH←0,AL←AHx0AH+AL	U	U	U	x	x	x
	CVTBW		1	0	0	1	1	0	0	0									1	2	When AL < 80H, AH←0. Otherwise, AH←FFH						
	CVTWL		1	0	0	1	1	0	0	1									1	2	When AW < 8000H, DW←0. Otherwise, DW←FFFFH						
Compare instruction	CMP	reg,reg'	0	0	1	1	1	0	1	W	1	1	reg				reg'		2	2	reg−reg'	x	x	x	x	x	x
		mem,reg	0	0	1	1	1	0	0	W	mod		reg				mem		2-4	6/8	(mem)−reg	x	x	x	x	x	x
		reg,mem	0	0	1	1	1	0	1	W	mod		reg				mem		2-4	6/8	reg−(mem)	x	x	x	x	x	x
		reg,imm	1	0	0	0	0	0	s	W	1	1	1	1	1		reg		3-4	2	reg−imm	x	x	x	x	x	x
		mem,imm	1	0	0	0	0	0	s	W	mod		1	1	1		mem		3-6	6/8	(mem)−imm	x	x	x	x	x	x
		acc,imm	0	0	1	1	1	1	0	W									2-3	2	When W = 0, AL−imm When W = 1, AW−imm	x	x	x	x	x	x

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Complements	NOT	reg	1	1	1	1	0	1	1	W	1	1	0	1	0	reg			2	2	reg←$\overline{reg}$						
		mem	1	1	1	1	0	1	1	W	mod		0	1	0	mem			2-4	7/11	(mem)←$\overline{(mem)}$						
	NEG	reg	1	1	1	1	0	1	1	W	1	1	0	1	1	reg			2	2	reg←$\overline{reg}$+1	x	x	x	x	x	x
		mem	1	1	1	1	0	1	1	W	mod		0	1	1	mem			2-4	7/11	(mem)←$\overline{(mem)}$+1	x	x	x	x	x	x
Logical operation instruction	TEST	reg,reg'	1	0	0	0	0	1	0	W	1	1	reg'			reg			2	2	reg ∧ reg'	U	0	0	x	x	x
		mem,reg reg,mem	1	0	0	0	0	1	0	W	mod		reg			mem			2-4	6/8	(mem) ∧ reg	U	0	0	x	x	x
		reg,imm	1	1	1	1	0	1	1	W	1	1	0	0	0	reg			3-4	2	reg ∧imm	U	0	0	x	x	x
		mem,imm	1	1	1	1	0	1	1	W	mod		0	0	0	mem			3-6	6/8	(mem) ∧imm	U	0	0	x	x	x
		acc,imm	1	0	1	0	1	0	0	W									2-3	2	When W = 0, AL ∧ imm8 When W = 1, AW ∧imm16	U	0	0	x	x	x
	AND	reg,reg'	0	0	1	0	0	0	1	W	1	1	reg			reg'			2	2	reg←reg ∧reg'	U	0	0	x	x	x
		mem,reg	0	0	1	0	0	0	0	W	mod		reg			mem			2-4	7/11	(mem)←(mem) ∧reg	U	0	0	x	x	x
		reg,mem	0	0	1	0	0	0	1	W	mod		reg			mem			2-4	6/8	reg←reg ∧(mem)	U	0	0	x	x	x
		reg,imm	1	0	0	0	0	0	0	W	1	1	1	0	0	reg			3-4	2	reg←reg ∧imm	U	0	0	x	x	x
		mem,imm	1	0	0	0	0	0	0	W	mod		1	0	0	mem			3-6	7/11	(mem)←(mem) ∧imm	U	0	0	x	x	x
		acc,imm	0	0	1	0	0	1	0	W									2-3	2	When W = 0, AL←AL ∧ imm8 When W = 1, AW←AW ∧imm16	U	0	0	x	x	x
	OR	reg,reg'	0	0	0	0	1	0	1	W	1	1	reg			reg'			2	2	reg←reg ∨reg'	U	0	0	x	x	x
		mem,reg	0	0	0	0	1	0	0	W	mod		reg			mem			2-4	7/11	(mem)←(mem) ∨reg	U	0	0	x	x	x
		reg,mem	0	0	0	0	1	0	1	W	mod		reg			mem			2-4	6/8	reg←reg ∨(mem)	U	0	0	x	x	x
		reg,imm	1	0	0	0	0	0	0	W	1	1	0	0	1	reg			3-4	2	reg←reg ∨imm	U	0	0	x	x	x
		mem,imm	1	0	0	0	0	0	0	W	mod		0	0	1	mem			3-6	7/11	(mem)←(mem) ∨imm	U	0	0	x	x	x
		acc,imm	0	0	0	0	1	1	0	W									2-3	2	When W = 0, AL←AL ∨ imm8 When W = 1, AW←AW ∨imm16	U	0	0	x	x	x

Instruction Group	Mnemonic	Operand	Operation Code									Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0					AC	CY	V	P	S	Z
Logical operation instruction	XOR	reg,reg'	0	0	1	1	0	0	1	W	1 1 reg reg'	2	2	reg←reg ∀ reg'	U	0	0	x	x	x
		mem,reg	0	0	1	1	0	0	0	W	mod reg mem	2-4	7/11	(mem)←(mem) ∀ reg	U	0	0	x	x	x
		reg,mem	0	0	1	1	0	0	1	W	mod reg mem	2-4	6/8	reg←reg ∀ (mem)	U	0	0	x	x	x
		reg,imm	1	0	0	0	0	0	0	W	1 1 1 1 0 reg	3-4	2	reg←reg ∀ imm	U	0	0	x	x	x
		mem,imm	1	0	0	0	0	0	0	W	mod 1 1 0 mem	3-6	7/11	(mem)←(mem) ∀ imm	U	0	0	x	x	x
		acc,imm	0	0	1	1	0	1	0	W		2-3	2	When W = 0, AL←AL ∀ imm8 When W = 1, AW←AW ∀ imm16	U	0	0	x	x	x

Instruction Group	Mnemonic	Operand	Operation Code	Number of Bytes	Number of Clocks	Operation	Flags
			7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0				AC CY V P S Z
Bit operation instruction	TEST1	reg8,CL	0 0 0 1 0 0 0 0 / 1 1 0 0 0 reg	3	4	reg8 bit NO.CL = 0 : Z←1 reg8 bit NO.CL = 1 : Z←0	U 0 0 U U x
		mem8,CL	0 0 0 0 / mod 0 0 0 mem	3-5	8	(mem8) bit NO.CL = 0 : Z←1 (mem8) bit NO.CL = 1 : Z←0	U 0 0 U U x
		reg16,CL	0 0 0 1 / 1 1 0 0 0 reg	3	4	reg16 bit NO.CL = 0 : Z←1 reg16 bit NO.CL = 1 : Z←0	U 0 0 U U x
		mem16,CL	0 0 0 1 / mod 0 0 0 mem	3-5	8/10	(mem16) bit NO.CL = 0 : Z←1 (mem16) bit NO.CL = 1 : Z←0	U 0 0 U U x
		reg8,imm3	1 0 0 0 / 1 1 0 0 0 reg	4	4	reg8 bit NO.imm3 = 0 : Z←1 reg8 bit NO.imm3 = 1 : Z←0	U 0 0 U U x
		mem8,imm3	1 0 0 0 / mod 0 0 0 mem	4-6	8	(mem8) bit NO.imm3 = 0 : Z←1 (mem8) bit NO.imm3 = 1 : Z←0	U 0 0 U U x
		reg16,imm4	1 0 0 1 / 1 1 0 0 0 reg	4	4	reg16 bit NO.imm4 = 0 : Z←1 reg16 bit NO.imm4 = 1 : Z←0	U 0 0 U U x
		mem16,imm4	1 0 0 1 / mod 0 0 0 mem	4-6	8/10	(mem16) bit NO.imm4 = 0 : Z←1 (mem16) bit NO.imm4 = 1 : Z←0	U 0 0 U U x
	NOT1	reg8,CL	0 1 1 0 / 1 1 0 0 0 reg	3	4	reg8 bit NO.CL←$\overline{\text{reg8 bit NO.CL}}$	
		mem8,CL	0 1 1 0 / mod 0 0 0 mem	3-5	9	(mem8) bit NO.CL←$\overline{\text{(mem8) bit NO.CL}}$	
		reg16,CL	0 1 1 1 / 1 1 0 0 0 reg	3	4	reg16 bit NO.CL←$\overline{\text{reg16 bit NO.CL}}$	
		mem16,CL	0 1 1 1 / mod 0 0 0 mem	3-5	9/13	(mem16) bit NO.CL←$\overline{\text{(mem16) bit NO.CL}}$	
		reg8,imm3	1 1 1 0 / 1 1 0 0 0 reg	4	4	reg8 bit NO.imm3←$\overline{\text{reg8 bit NO.imm3}}$	
		mem8,imm3	1 1 1 0 / mod 0 0 0 mem	4-6	9	(mem8) bit NO.imm3←$\overline{\text{(mem8) bit NO.imm3}}$	
		reg16,imm4	1 1 1 1 / 1 1 0 0 0 reg	4	4	reg16 bit NO.imm4←$\overline{\text{reg16 bit NO.imm4}}$	
		mem16,imm4	1 1 1 1 / mod 0 0 0 mem	4-6	9/13	(mem16) bit NO.imm4←$\overline{\text{(mem16) bit NO.imm4}}$	
			2nd byte* / 3rd byte*			★ : 1st byte = 0FH	
	NOT1	CY	1 1 1 1 0 1 0 1	1	2	CY←$\overline{\text{CY}}$	x

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Bit operation instruction	CLR1	reg8,CL	0	0	0	1	0	0	1	0	1	1	0	0	0	reg			3	4	reg8 bit NO.CL←0						
		mem8,CL					0	0	1	0	mod		0	0	0	mem			3-5	9	(mem8) bit NO.CL←0						
		reg16,CL					0	0	1	1	1	1	0	0	0	reg			3	4	reg16 bit NO.CL←0						
		mem16,CL					0	0	1	1	mod		0	0	0	mem			3-5	9/13	(mem16) bit NO.CL←0						
		reg8,imm3					1	0	1	0	1	1	0	0	0	reg			4	4	reg8 bit NO.imm3←0						
		mem8,imm3					1	0	1	0	mod		0	0	0	mem			4-6	9	(mem8) bit NO.imm3←0						
		reg16,imm4					1	0	1	1	1	1	0	0	0	reg			4	4	reg16 bit NO.imm4←0						
		mem16,imm4					1	0	1	1	mod		0	0	0	mem			4-6	9/13	(mem16) bit NO.imm4←0						
	SET1	reg8,CL					0	1	0	0	1	1	0	0	0	reg			3	4	reg8 bit NO.CL←1						
		mem8,CL					0	1	0	0	mod		0	0	0	mem			3-5	9	(mem8) bit NO.CL←1						
		reg16,CL					0	1	0	1	1	1	0	0	0	reg			3	4	reg16 bit NO.CL←1						
		mem16,CL					0	1	0	1	mod		0	0	0	mem			3-5	9/13	(mem16) bit NO.CL←1						
		reg8,imm3					1	1	0	0	1	1	0	0	0	reg			4	4	reg8 bit NO.imm3←1						
		mem8,imm3					1	1	0	0	mod		0	0	0	mem			4-6	9	(mem8) bit NO.imm3←1						
		reg16,imm4					1	1	0	1	1	1	0	0	0	reg			4	4	reg16 bit NO.imm4←1						
		mem16,imm4					1	1	0	1	mod		0	0	0	mem			4-6	9/13	(mem16) bit NO.imm4←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																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Shift instruction	SHL	reg,1	1	1	0	1	0	0	0	W	1	1	1	0	0	reg			2	2	CY←MSB of reg, reg←regx2 When MSB of reg ≠ CY, V←1 When MSB of reg = CY, V←0	U	x	x	x	x	x
		mem,1	1	1	0	1	0	0	0	W	mod	1	0	0	mem				2-4	7/11	CY←MSB of (mem), (mem)←(mem)x2 When MSB of (mem) ≠ CY, V←1 When MSB of (mem) = CY, V←0	U	x	x	x	x	x
		reg,CL	1	1	0	1	0	0	1	W	1	1	1	0	0	reg			2	2+n	temp←CL, repeats following operation, while temp ≠ 0: CY←MSB of reg, reg←regx2 temp←temp−1	U	x	U	x	x	x
		mem,CL	1	1	0	1	0	0	1	W	mod	1	0	0	mem				2-4	6/10+n	temp←CL, repeats following operation, while temp ≠ 0: CY←MSB of (mem), (mem)←(mem)x2 temp←temp−1	U	x	U	x	x	x
		reg,imm8	1	1	0	0	0	0	0	W	1	1	1	0	0	reg			3	2+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←MSB of reg, reg←regx2 temp←temp−1	U	x	U	x	x	x
		mem,imm8	1	1	0	0	0	0	0	W	mod	1	0	0	mem				3-5	6/10+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←MSB of (mem), (mem)←(mem)x2 temp←temp−1	U	x	U	x	x	x
	SHR	reg,1	1	1	0	1	0	0	0	W	1	1	1	0	1	reg			2	2	CY←LSB of reg, reg←reg÷2 MSB of reg ≠ next bit of MSB of reg: V←1 MSB of reg = next bit of MSB of reg: V←0	U	x	x	x	x	x
		mem,1	1	1	0	1	0	0	0	W	mod	1	0	0	mem				2-4	7/11	CY←MSB of (mem), (mem)←(mem)÷2 MSB of reg ≠ next bit of MSB of reg: V←1 MSB of reg = next bit of MSB of reg: CY, V←0	U	x	x	x	x	x

n : Number of sifts

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Shift instruction	SHR (cont'd)	reg,CL	1	1	0	1	0	0	1	W	1	1	1	0	0	reg			2	2+n	temp←CL, repeats following operation, while temp ≠ 0: CY←LSB of reg, reg←reg+2 temp←temp−1	U	x	U	x	x	x
		mem,CL	1	1	0	1	0	0	1	W	mod		1	0	0	mem			2-4	6/10+n	temp←CL, repeats following operation, while temp ≠ 0: CY←LSB of (mem), (mem)←(mem)+2 temp←temp−1	U	x	U	x	x	x
		reg,imm8	1	1	0	0	0	0	0	W	1	1	1	0	0	reg			3	2+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←LSB of reg, reg←reg+2 temp←temp−1	U	x	U	x	x	x
		mem,imm8	1	1	0	0	0	0	0	W	mod		1	0	0	mem			3-5	6/10+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←LSB of (mem), (mem)←(mem)+2 temp←temp−1	U	x	U	x	x	x
	SHRA	reg,1	1	1	0	1	0	0	0	W	1	1	1	1	1	reg			2	2	CY←LSB of reg, reg←reg+2, V←0 MSB of operand is not affected.	U	x	U	x	x	x
		mem,1	1	1	0	1	0	0	0	W	mod		1	1	1	mem			2-4	7/11	CY←LSB of (mem), (mem)←(mem)+2, V←0 MSB of operand is not affected.	U	x	U	x	x	x
		reg,CL	1	1	0	1	0	0	1	W	1	1	1	1	1	reg			2	2+n	temp←CL, repeats following operation, while temp ≠ 0: CY←LSB of reg, reg←reg+2 temp←temp−1, MSB of operand is not affected.	U	x	U	x	x	x
		mem,CL	1	1	0	1	0	0	1	W	mod		1	1	1	mem			2-4	6/10+n	temp←CL, repeats following operation, while temp ≠ 0: CY←LSB of (mem), (mem)←(mem)+2 temp←temp−1, MSB of operand is not affected.	U	x	U	x	x	x
		reg,imm8	1	1	0	0	0	0	0	W	1	1	1	1	1	reg			3	2+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←LSB of reg, reg←reg+2 temp←temp−1, MSB of operand is not affected.	U	x	U	x	x	x
		mem,imm8	1	1	0	0	0	0	0	W	mod		1	1	1	mem			3-5	6/10+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←LSB of (mem), (mem)←(mem)+2 temp←temp−1, MSB of operand is not affected.	U	x	U	x	x	x

n : Number of sifts

Instruction Group	Mnemonic	Operand	Operation Code	Number of Bytes	Number of Clocks	Operation	Flags
			7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0				AC CY V P S Z
Rotate instruction	ROL	reg,1	1 1 0 1 0 0 0 W / 1 1 0 0 0 reg	2	2	CY←MSB of reg, reg←regx2+CY When MSB of reg ≠ CY: V←1 When MSB of reg = CY: V←0	x x
		mem,1	1 1 0 1 0 0 0 W / mod 0 0 0 mem	2-4	7/11	CY←MSB of (mem), (mem)←(mem)x2+CY When MSB of (mem) ≠ CY: V←1 When MSB of (mem) = CY: V←0	x x
		reg,CL	1 1 0 1 0 0 1 W / 1 1 0 0 0 reg	2	2+n	temp←CL, repeats following operation, while temp ≠ 0: CY←MSB of reg, reg←regx2+CY temp←temp−1	x U
		mem,CL	1 1 0 1 0 0 1 W / mod 0 0 0 mem	2-4	6/10+n	temp←CL, repeats following operation, while temp ≠ 0: CY←MSB of (mem), (mem)←(mem)x2+CY temp←temp−1	x U
		reg,imm8	1 1 0 0 0 0 0 W / 1 1 0 0 0 reg	3	2+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←MSB of reg, reg←regx2+CY temp←temp−1	x U
		mem,imm8	1 1 0 0 0 0 0 W / mod 0 0 0 mem	3-5	6/10+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←MSB of (mem), (mem)←(mem)x2+CY temp←temp−1	x U
	ROR	reg,1	1 1 0 1 0 0 0 W / 1 1 0 0 1 reg	2	2	CY←LSB of reg←reg+2 MSB of reg←CY MSB of reg ≠ next bit of MSB of reg: V←1 MSB of reg = next bit of MSB of reg: V←0	x x
		mem,1	1 1 0 1 0 0 0 W / mod 0 0 1 mem	2-4	7/11	CY←LSB of (mem)←(mem)+2 MSB of (mem)⁻CY MSB of (mem) π next bit of MSB of reg: V⁻1 MSB of (mem) = next bit of MSB of reg: V⁻0	x x

n : Number of sifts

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Rotate instruction	ROR (cont'd)	reg,CL	1	1	0	1	0	0	1	W	1	1	0	0	1	reg			2	2+n	temp←CL, repeats following operation, while temp ≠ 0: CY←LSB of reg, reg←reg+2 MSB of reg←CY temp←temp−1		x	U			
		mem,CL	1	1	0	1	0	0	1	W	mod	0	0	1	mem				2-4	6/10+n	temp←CL, repeats following operation, while temp ≠ 0: CY←LSB of (mem), (mem)←(mem)+2 MSB of (mem)←CY temp←temp−1		x	U			
		reg,imm8	1	1	0	0	0	0	0	W	1	1	0	0	1	reg			3	2+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←LSB of reg, reg←reg+2 MSB of reg←CY temp←temp−1		x	U			
		mem,imm8	1	1	0	0	0	0	0	W	mod	0	0	1	mem				3-5	6/10+n	temp←imm8, repeats following operation, while temp ≠ 0: CY←LSB of (mem), (mem)←(mem)+2 MSB of (mem)←CY temp←temp−1		x	U			
	ROLC	reg,1	1	1	0	1	0	0	0	W	1	1	0	1	0	reg			2	2	tmpcy←CY, CY←MSB of reg reg←regx2+tmpcy When MSB of reg ≠ CY: V←1 When MSB of reg = CY: V←0		x	x			
		mem,1	1	1	0	1	0	0	0	W	mod	0	1	0	mem				2-4	7/11	tmpcy←CY, CY←MSB of (mem) (mem)←(mem)x2+tmpcy When MSB of (mem) ≠ CY: V←1 When MSB of (mem) = CY: V←0		x	x			
		reg,CL	1	1	0	1	0	0	1	W	1	1	0	1	0	reg			2	2+n	temp←CL, repeats following operation, while temp ≠ 0: tmpcy←CY, CY←MBS of reg reg←regx2+tmpcy temp←temp−1		x	U			

n : Number of sifts

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Rotate instruction	ROLC (cont'd)	mem,CL	1	1	0	1	0	0	1	W	mod	0	1	0	mem				2-4	6/10+n	temp←CL, repeats following operation, while temp ≠ 0: tmpcy←CY, CY←MSB of (mem) (mem)←(mem)x2+tmpcy temp←temp−1		x	U			
		reg,imm8	1	1	0	0	0	0	0	W	1	1	0	1	0	reg			3	2+n	temp←imm8, repeats following operation, while temp ≠ 0: tmpcy←CY, CY←MSB of reg reg←regx2+tmpcy temp←temp−1		x	U			
		mem,imm8	1	1	0	0	0	0	0	W	mod	0	1	0	mem				3-5	6/10+n	temp←imm8, repeats following operation, while temp ≠ 0: tmpcy←CY, CY←MSB of (mem) (mem)←(mem)x2+tmpcy temp←temp−1		x	U			
	RORC	reg,1	1	1	0	1	0	0	0	W	1	1	0	1	1	reg			2	2	tmpcy←CY, CY← LSB of reg reg←reg+2 MSB of reg←tmpcy When MSB of reg ≠ next bit of MSB of: V←1 When MSB of reg = next bit of MSB of reg: V←0		x	x			
		mem,1	1	1	0	1	0	0	0	W	mod	0	1	1	mem				2-4	7/11	tmpcy←CY, CY←LSB of (mem) (mem)←(mem)+2 MSB of (mem)←tmpcy When MSB of (mem) ≠ next bit of MSB of (mem): V←1 When MSB of (mem) = next bit of MSB of (mem): V←0		x	x			
		reg,CL	1	1	0	1	0	0	1	W	1	1	0	1	1	reg			2	2+n	temp←CL, repeats following operation, while temp ≠ 0: tmpcy←CY, CY←LSB of reg reg←reg+2 MSB of reg←tmpcy temp←temp−1		x	U			

n : Number of sifts

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Rotate instruction	RORC (cont'd)	mem,CL	1	1	0	1	0	0	1	W	mod	0	1	1	mem				2-4	6/10+n	temp←CL, repeats following operation, while temp ≠ 0: tmpcy←CY, CY←LSB of (mem) (mem)←(mem)+2 MSB of (mem)←tmpcy temp←temp−1		x	U			
		reg,imm8	1	1	0	0	0	0	0	W	1	1	0	1	1	reg			3	2+n	temp←imm8, repeats following operation, while temp ≠ 0: tmpcy←CY, CY←LSB of reg reg←reg+2 MSB of reg←tmpcy temp←temp−1		x	U			
		mem,imm8	1	1	0	0	0	0	0	W	mod	0	1	1	mem				3-5	6/10+n	temp←imm8, repeats following operation, while temp ≠ 0: tmpcy←CY, CY←LSB of (mem) (mem)←(mem)+2 MSB of (mem)←tmpcy temp←temp−1		x	U			

n : Number of sifts

Instruction Group	Mnemonic	Operand	Operation Code								Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0									
														AC	CY	V	P	S	Z
Subroutine control instruction	CALL	near-proc	1	1	1	0	1	0	0	0	3	7/9	SP←SP−2, (SP+1,SP)←PC PC←PC+disp						
		regptr16	1	1	1	1	1	1	1	1	2	7/9	SP←SP−2, (SP+1,SP)←PC PC←regptr16						
		memptr16	1	1	1	1	1	1	1	1	2-4	11/15	TA←(memptr16) SP←SP−2, (SP+1,SP)←PC, PC←TA						
		far-proc	1	0	0	1	1	0	1	0	5	9/13	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	2-4	15/23	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	10/12	PC←(SP+1,SP) SP←SP+2						
		pop-value	1	1	0	0	0	0	1	0	3	10/12	PC←(SP+1,SP) SP←SP+2, SP←SP+pop-value						
			1	1	0	0	1	0	1	1	1	12/26	PC←(SP+1,SP) PS←(SP+3,SP+2) SP←SP+4						
		pop-value	1	1	0	0	1	0	1	0	3	12/26	PC←(SP+1,SP) PS←(SP+3,SP+2) SP←SP+4, SP←SP+pop-value						

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags						
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z	
Stack operation instruction	PUSH	mem16	1	1	1	1	1	1	1	1	mod	1	1	0	mem				2-4	5/9	SP←SP−2 (SP+1,SP)←(mem16)							
		reg16	0	1	0	1	0	reg											1	3/5	SP←SP−2 (SP+1,SP)←reg16							
		sreg	0	0	0	sreg		1	1	0									1	3/5	SP←SP−2 (SP+1,SP)←sreg							
		PSW	1	0	0	1	1	1	0	0									1	3/5	SP←SP−2 (SP+1,SP)←PSW							
		R	0	1	1	0	0	0	0	0									1	20/36	Push registers on the stack							
		imm8	0	1	1	0	1	0	1	0									2	3/5	when (SP−1,SP−2)←imm8 sign expansion SP←SP−2,S = 1, sign expansion							
		imm16	0	1	1	0	1	0	0	0									3	3/5	(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	5/9	SP←SP+2 (mem16)←(SP−1,SP−2)							★
		reg16	0	1	0	1	1	reg											1	5/7	SP←SP+2 reg16←(SP−1,SP−2)							★
		sreg	0	0	0	sreg		1	1	1									1	5/7	SP←SP+2 sreg: SS,DS0,DS1 sreg←(SP−1,SP−2)							★
		PSW	1	0	0	1	1	1	0	1									1	5/7	SP←SP+2 PSW←(SP−1,SP−2)	R	R	R	R	R	R	★
		R	0	1	1	0	0	0	0	1									1	22/38	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	6/8	Dispose of Stack Frame							

*: When imm8 = 0 : 15

When imm8 ≥ 1 : {15 + 8 (imm8 − 1)} / {17 + 12 (imm8 − 1)}

Instruction Group	Mnemonic	Operand	Operation Code	Number of Bytes	Number of Clocks[Note]	Operation	Flags
			7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0				AC CY V P S Z
Branch instruction	BR	near-label	1 1 1 0 1 0 0 1	3	7	PC←PC+disp	
		short-label	1 1 1 0 1 0 1 1	2	7	PC←PC+ext-disp8	
		regptr16	1 1 1 1 1 1 1 1 / 1 1 1 0 0 reg	2	7	PC←regptr16	
		memptr16	1 1 1 1 1 1 1 1 / mod 1 0 0 mem	2-4	11/13	PC←(memptr16)	
		far-label	1 1 1 0 1 0 1 0	5	7	PS←seg PC←offset	
		memptr32	1 1 1 1 1 1 1 1 / mod 1 0 1 mem	2-4	13/17	PS←(memptr32+2) PC←(memptr32)	
Condition branch instruction	BV	short-label	0 1 1 1 0 0 0 0	2	6/3	If V = 1 PC←PC+ext-disp8	
	BNV	short-label	0 0 0 1	2	6/3	If V = 0 PC←PC+ext-disp8	
	BC BL	short-label	0 0 1 0	2	6/3	If CY = 1 PC←PC+ext-disp8	
	BNC BNL	short-label	0 0 1 1	2	6/3	If CY = 0 PC←PC+ext-disp8	
	BE BZ	short-label	0 1 0 0	2	6/3	If Z = 1 PC←PC+ext-disp8	
	BNE BNZ	short-label	0 1 0 1	2	6/3	If Z = 0 PC←PC+ext-disp8	
	BNH	short-label	0 1 1 0	2	6/3	If CY ∨ Z = 1 PC←PC+ext-disp8	
	BH	short-label	0 1 1 1	2	6/3	If CY ∨ Z = 0 PC←PC+ext-disp8	
	BN	short-label	1 0 0 0	2	6/3	If S = 1 PC←PC+ext-disp8	
	BP	short-label	1 0 0 1	2	6/3	If S = 0 PC←PC+ext-disp8	
	BPE	short-label	1 0 1 0	2	6/3	If P = 1 PC←PC+ext-disp8	
	BPO	short-label	1 0 1 1	2	6/3	If P = 0 PC←PC+ext-disp8	
	BLT	short-label	1 1 0 0	2	6/3	If S ∀ V = 1 PC←PC+ext-disp8	
	BGE	short-label	1 1 0 1	2	6/3	If S ∀ V = 0 PC←PC+ext-disp8	
	BLE	short-label	1 1 1 0	2	6/3	If (S ∀ V) ∨ Z = 1 PC←PC+ext-disp8	
	BGT	short-label	1 1 1 1	2	6/3	If (S ∀ V) ∨ Z = 0 PC←PC+ext-disp8	

Note Condition determination: True/false

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Condition branch instruction	DBNZNE	short-label	1	1	1	0	0	0	0	0									2	6/3	CW = CW−1 PC←PC+ext-disp8 If Z = 0 and CW ≠ 0						
	DBNZE	short-label					0	0	0	1									2	6/3	CW = CW−1 PC←PC+ext-disp8 If Z = 1 and CW ≠ 0						
	DBNZ	short-label					0	0	1	0									2	6/3	CW = CW−1 PC←PC+ext-disp8 If CW ≠ 0						
	BCWZ	short-label					0	0	1	1									2	6/3	If CW = 0 PC←PC+ext-disp8						
Interrupt instruction	BRK	3	1	1	0	0	1	1	0	0									1	18/24	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[Note 1], PC←TA						
		imm8 (≠3)	1	1	0	0	1	1	0	1									2	18/24	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[Note 1], PC←TA						
	BRKV		1	1	0	0	1	1	1	0									1	**Note 2**	When 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[Note 1], PC←TA						
	RETI		1	1	0	0	1	1	1	1									1	13/19	PC←(SP+1,SP), PS←(SP+3,SP+2), PSW←(SP+5,SP+4), SP←SP+6	R	R	R	R	R	R

Notes 1. Start address of the next instruction of BRK, BRKV

2. V = 1 : 20/26
V = 0 : 3

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Interrupt instruction	CHKIND	reg16,mem32	0	1	1	0	0	0	1	0	mod		reg			mem			2-4	Note 3	When (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[Note 4], PC←TA						
CPU control instruction	HALT		1	1	1	1	0	1	0	0									1	2	CPU Halt						
	POLL[Note 1]		1	0	0	1	1	0	1	1									1	2+2n	Poll and wait n: Number of $\overline{\text{CPBUSY}}$ pin sampling operations						
	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[Note 1]	fp–op	1	1	0	1	1	X	X	X	1	1	Y	Y	Y	Z	Z	Z	2	Note 2	No Operation						
		fp–op,mem	1	1	0	1	1	X	X	X	mod		Y	Y	Y	mem			2-4	Note 2	data bus←(mem)						
	FPO2[Note 1]	fp–op	0	1	1	0	0	1	1	X	1	1	Y	Y	Y	Z	Z	Z	2	Note 2	No Operation						
		fp–op,mem	0	1	1	0	0	1	1	X	mod		Y	Y	Y	mem			2-4	Note 2	data bus←(mem)						
	NOP		1	0	0	1	0	0	0	0									1	3	No Operation						
Segment operation instruction	DS0:		0	0	1	1	1	1	1	0									1	2	Segment override prefix						
	DS1:		0	0	1	0	0	1	1	0																	
	PS:		0	0	1	0	1	1	1	0																	
	SS:		0	0	1	1	0	1	1	0																	

Caution **Eight or more repeat prefix instructions should not be attached to non-prefix instruction.**

Notes 1. If the POLL, FPO1, or FPO2 instruction is executed when the coprocessor is not connected, a coprocespor not present interrupt will be generated (interrupt vector 7).

2. The number of clocks differ depending on whether or not the coprocessor is connected.
3. When interrupt condition is established : (24–26)/(30–32)
   When interrupt condition is not established : 12/14
4. The address following CHKIND instruction.

Instruction Group	Mnemonic	Operand	Operation Code																Number of Bytes	Number of Clocks	Operation	Flags				
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				AC	CY	V	P	S	Z
Instructions dedicated to expansion address mode	BRKXA	imm8	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	3	12	temp1←(imm8 x 4 + 1, imm8 x 4) temp2←(imm8 x 4 + 3, imm8 x 4 + 2) XA←1 PC←temp1, PS←temp2 Sets expansion address mode						
	RETXA	imm8	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	3	12	temp1←(imm8 x 4 + 1, imm8 x 4) temp2←(imm8 x 4 + 1, imm8 x 4 + 2) XA←0 PC←temp1, PS←temp2 Release expansion address mode						

13. ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS (T_A = 25°C)

Item	Symbol	Conditions	Ratings	Unit
Power supply voltage	V_{DD}		−0.5 to +7.0 V	V
Input voltage	V_I		−0.5 to V_{DD}+0.3	V
Clock 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_A		−10 to +70	°C
Storage temperature	T_{stg}		−65 to +150	°C

★ **Cautions 1. Do not directly connect the output pins of two or more IC products and do not directly connect the output pins to V_{DD} or V_{CC} and GND. However, open-drain pins or open-collector pins may be connected directly. Moreover, an external circuit whose timing is designed to avoid output collision can be connected to pins that go into a high-impedance state.**

2. If even one of the above parameters exceeds the absolute maximum rating even momentarily, the quality of the program may be degraded. Absolute maximum ratings, therefore, are the values exceeding which the product may be physically damaged. Use the program keeping all the parameters within these rated values.

The standards and conditions shown in DC and AC Characteristics below specify the range within which the normal operation of the product is guaranteed.

DC CHARACTERISTICS (T_A = −10 to +70°C, V_{DD} = 5 V ± 10%)

Item	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}			0.8V_{DD}		V_{DD}+0.5	V
CLK input voltage, low	V_{KL}			−0.5		0.6	V
Output voltage, high	V_{OH}	I_{OH} = −400μA		0.7V_{DD}			V
Output voltage, low	V_{OL}	I_{OL} = 2.5mA				0.45	V
Input leakage current, high	I_{LIH}	V_I = V_{DD}				10	μA
Input leakage current, low	I_{LIL}	V_I = 0V				−10	μA
Output leakage current, high	I_{LOH}	V_O = V_{DD}				10	μA
Output leakage current, low	I_{LOL}	V_O = 0V				−10	μA
Latch leakage current, high	I_{LLH}	V_{IN} = 3.0V		−20		−200	μA
Latch leakage current, low	I_{LLL}	V_{IN} = 0.8V		20		200	μA
Latch inverse current, (L → H)	I_{ILH}					200	μA
Latch inverse current, (H → L)	I_{ILL}					−200	μA
Supply current	I_{DD}	Operation	16MHz		100	150	mA
			12.5MHz		75	110	mA
			10MHz		75	110	mA
		Standby	16MHz		25	35	mA
			12.5MHz		20	30	mA
			10MHz		20	30	mA

Remark The power supply current during operation is almost proportional to the operation clock frequency.

CAPACITANCE (T_A = 25°C, V_{DD} = 0V)

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

AC CHARACTERISTICS (T_A = −10°C to +70°C, V_{DD} = 5 V ± 10%)

(1) μPD70136A-10 (operating frequency = 10 MHz)

(1/2)

Item	Symbol	Conditions	MIN.	MAX.	Unit
Clock period	t_{CYK}		100	500	ns
Clock high-level width	t_{KKH}	V_{KH} = 3.5V	39		ns
Clock low-level width	t_{KKL}	V_{KL} = 1.7V	40		ns
Clock rise time (1.7V to 3.0V)	t_{KR}	1.7V-3.0V		5	ns
Clock fall time (3.0V to 1.7V)	t_{KF}	3.0V-1.7V		5	ns
Reset release delay time (V_{DD} VALID)	t_{DVRST}		1		us
Reset setup time (CLK ↑)	t_{SRSTK}		15		ns
Reset hold time (CLK ↑)	t_{HKRST}		15		ns
Reset high-level width	t_{WRSTH}		6		t_{CYK}
CLK ↓ → $\overline{\text{BCYST}}$ delay time	t_{DKBC}	C_L = 100pF	5	40	ns
CLK ↓ → address delay time	t_{DKA}		5	40	ns
CLK ↓ → status delay time	t_{DKST}		5	40	ns
CLK ↑ → data output delay time[Note 1]	t_{DKD}		5	45	ns
Floating delay time	t_{FK}		0	40	ns
CLK ↑ → $\overline{\text{DSTB}}$ output delay time	t_{DKDS}		5	40	ns
CLK ↓ → $\overline{\text{DSTB}}$ ↑ output delay	t_{DKDSH}		5	40	ns
CLK ↑ → HLDAK delay time	t_{DKHA}		5	50	ns
$\overline{\text{BCYST}}$ low-level width	t_{BCBCL}		t_{CYK}−10		ns
$\overline{\text{BCYST}}$ high-level width	t_{BCBCH}		$t_{CYK}(n+1)$[Note 2]−10		ns
$\overline{\text{DSTB}}$ low-level width	t_{DSDSL}		$t_{CYK}(n+1)$[Note 2]−10		ns
$\overline{\text{DSTB}}$ high-level width	t_{DSDSH}		$t_{KKL}+t_{KR}$−10		ns

★ **Notes 1.** This specification applies to the delay times <1>-<3> shown below following the falling edge of the CLK signal.

<1> Address delay time

<2> $\overline{\text{BUSLOCK}}$ delay time

<3> Each of the following signal delay times immediately following bus hold release:
A23-A0, D15-D0, M/$\overline{\text{IO}}$, BUSST1, BUSST0, $\overline{\text{UBE}}$, $\overline{\text{BCYST}}$, $\overline{\text{DSTB}}$

2. 'n' means number of wait clocks to be inserted into bus cycle. However, TC cycle (+1) is added to n in the case of the coprocessor cycle.

(2/2)

Item	Symbol	Conditions	MIN.	MAX.	Unit
Output float → HLDAK delay time	t_{DFHA}	C_L = 100pF	$t_{DXL}+t_{XR}-15$		ns
Address/status → output $\overline{DSTB}$ ↓ delay time	t_{DADSL}		$t_{DXL}+t_{XR}-15$		ns
$\overline{DSTB}$ ↑ output → address/status hold time (write operation)	t_{HDSHA}		$t_{DXL}+t_{XR}-15$		ns
$\overline{DSTB}$ ↑ output → data output delay time	t_{DDSHD}		$t_{DXL}+t_{XR}-15$		ns
$\overline{DSTB}$ ↓ output → data output floating time	t_{DHZ}			0	ns
$\overline{DSTB}$ ↑ output → data output set time	t_{DLZ}		$t_{DXL}+t_{XR}-15$		ns
Address/status output → data output delay time	t_{DAD}		$t_{DXL}+t_{XR}-15$		ns
Data setup time (CLK ↓)	t_{SDK}		10		ns
Data hold time (CLK ↓)[Note]	t_{HKD}		15		ns
Data hold time ($\overline{DSTB}$ ↑)[Note]	t_{HDSD}		0		ns
Data hold time (address/status change point)	t_{HASD}		0		ns
$\overline{READY}$ setup time (CLK ↑)	t_{SRYK}		10		ns
$\overline{READY}$ hold time (CLK ↑)	t_{HKRY}		20		ns
$\overline{BS8}$/BS16 setup time	t_{SBSK}		10		ns
$\overline{BS8}$/BS16 hold time	t_{HKBS}		20		ns
HLDRQ setup time (CLK ↑)	t_{SHQK}		10		ns
HLDRQ hold time (CLK ↑)	t_{HKHQ}		20		ns
$\overline{NMI}$, INT, $\overline{CPBUSY}$ setup time (CLK ↓)	t_{SIK}		15		ns
$\overline{NMI}$, INT, $\overline{CPBUSY}$ hold time (CLK ↓)	t_{HKI}		15		ns

Note For t_{HKD}, t_{HDSD}, t_{HASD} specifications, at least one of these must be observed.

Remark t_{HKD} means data hold request specification from rising edge of clock.
t_{HDSD} and t_{HASD} are data hold request specifications from $\overline{DSTB}$ and address/status change points, respectively. This means that the $\overline{DSTB}$ and the address/status signal do not change, until the μPD70136A completes data read operation. (Address/status means A0-A23, $\overline{UBE}$, R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, and AEX.)

(2) μPD70136A-12 (operating frequency = 12.5 MHz)

(1/2)

	Item	Symbol	Conditions	MIN.	MAX.	Unit
	Clock period	t_{CYK}		80	500	ns
	Clock high-level width	t_{KKH}	V_{KH} = 3.5V	35		ns
	Clock low-level width	t_{KKL}	V_{KL} = 1.7V	35		ns
	Clock rise time (1.7V to 3.0V)	t_{KR}	1.7V-3.0V		5	ns
	Clock fall time (3.0V to 1.7V)	t_{KF}	3.0V-1.7V		5	ns
	Reset release delay time (V_{DD} VALID)	t_{DVRST}		1		μs
	Reset setup time (CLK ↑)	t_{SRSTK}		10		ns
	Reset hold time (CLK ↑)	t_{HKRST}		15		ns
	Reset high-level width	t_{WRSTH}		6		t_{CYK}
	CLK ↓ → $\overline{\text{BCYST}}$ delay time	t_{DKBC}	C_L = 100pF	5	40	ns
	CLK ↓ → address delay time[Note 1]	t_{DKA}		5	40	ns
	CLK ↓ → status delay time	t_{DKST}		5	40	ns
	CLK ↑ → data output delay time	t_{DKD}		5	40	ns
	Floating delay time	t_{FK}		0	35	ns
	CLK ↑ → $\overline{\text{DSTB}}$ output delay time	t_{DKDS}		5	40	ns
	CLK ↓ → $\overline{\text{DSTB}}$ ↑ output delay time	t_{DKDSH}		5	40	ns
	CLK ↑ → HLDAK delay time	t_{DKHA}		5	40	ns
	$\overline{\text{BCYST}}$ low-level width	t_{BCBCL}		t_{CYK}–10		ns
★	$\overline{\text{BCYST}}$ high-level width	t_{BCBCH}		$t_{CYK}(n+1)$[Note 2]–10		ns
	$\overline{\text{DSTB}}$ low-level width	t_{DSDSL}		$t_{CYK}(n+1)$[Note 2]–10		ns
	$\overline{\text{DSTB}}$ high-level width	t_{DSDSH}		$t_{KKL}+t_{KR}$–10		ns

★ **Notes 1.** This specification applies to the delay times <1>-<3> shown below following the falling edge of the CLK signal.

<1> Address delay time

<2> $\overline{\text{BUSLOCK}}$ delay time

<3> Each of the following signal delay times immediately following bus hold release:
A23-A0, D15-D0, M/$\overline{\text{IO}}$, BUSST1, BUSST0, $\overline{\text{UBE}}$, $\overline{\text{BCYST}}$, $\overline{\text{DSTB}}$

2. 'n' means number of wait clocks to be inserted into bus cycle. However, TC cycle (+1) is added to n in the case of the coprocessor cycle.

(2/2)

Item	Symbol	Conditions	MIN.	MAX.	Unit
Output float → HLDAK delay time	t_{DFHA}	C_L = 100pF	$t_{DKL}+t_{KR}-15$		ns
Address/status → output $\overline{DSTB}$ ↓ delay time	t_{DADSL}		$t_{DKL}+t_{KR}-15$		ns
$\overline{DSTB}$ ↑ output → address/status hold time (write operation)	t_{HDSHA}		$t_{DKL}+t_{KR}-15$		ns
$\overline{DSTB}$ ↑ output → data output delay time	t_{DDSHD}		$t_{DKL}+t_{KR}-15$		ns
$\overline{DSTB}$ ↓ output → data output floating time	t_{DHZ}			0	ns
$\overline{DSTB}$ ↑ output → data output set time	t_{DLZ}		$t_{DKL}+t_{KR}-15$		ns
Address/status output → data output delay time	t_{DAD}		$t_{DKL}+t_{KR}-15$		ns
Data setup time (CLK ↓)	t_{SDK}		7		ns
Data hold time (CLK ↓)[Note]	t_{HKD}		10		ns
Data hold time ($\overline{DSTB}$ ↑)[Note]	t_{HDSD}		0		ns
Data hold time[Note] (address/status change point)	t_{HASD}		0		ns
$\overline{READY}$ setup time (CLK ↑)	t_{SRYK}		7		ns
$\overline{READY}$ hold time (CLK ↑)	t_{HKRY}		15		ns
$\overline{BS8}$/BS16 setup time	t_{SBSK}		7		ns
$\overline{BS8}$/BS16 hold time	t_{HKBS}		15		ns
HLDRQ setup time (CLK ↑)	t_{SHQK}		7		ns
HLDRQ hold time (CLK ↑)	t_{HKHQ}		15		ns
$\overline{NMI}$, INT, $\overline{CPBUSY}$ setup time (CLK ↓)	t_{SIK}		10		ns
$\overline{NMI}$, INT, $\overline{CPBUSY}$ hold time (CLK ↓)	t_{HKI}		10		ns

Note For t_{HKD}, t_{HDSD}, t_{HASD} specifications, at least one of these must be observed.

Remark t_{HKD} means data hold request specification from rising edge of clock.
t_{HDSD} and t_{HASD} are data hold request specifications from $\overline{DSTB}$ and address/status change points, respectively. This means that the $\overline{DSTB}$ and the address/status signal do not change, until the μPD70136A completes data read operation. (Address/status means A0-A23, $\overline{UBE}$, R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, and AEX.)

(3) μPD70136A-16 (operating frequency = 16 MHz)

(1/2)

	Item	Symbol	Conditions	MIN.	MAX.	Unit
	Clock period	t_{CYK}		62.5	500	ns
	Clock high-level width	t_{KKH}	V_{KH} = 3.5V	25		ns
	Clock low-level width	t_{KKL}	V_{KL} = 1.7V	25		ns
	Clock rise time (1.7V to 3.0V)	t_{KR}	1.7V-3.0V		5	ns
	Clock fall time (3.0V to 1.7V)	t_{KF}	3.0V-1.7V		5	ns
	Reset release delay time (V_{DD} VALID)	t_{DVRST}		1		μs
	Reset setup time (CLK ↑)	t_{SRSTK}		10		ns
	Reset hold time (CLK ↑)	t_{HKRST}		15		ns
	Reset high-level width	t_{WRSTH}		6		t_{CYK}
	CLK ↓ → $\overline{BCYST}$ delay time[Note 1]	t_{DKBC}	C_L = 100pF	5	40	ns
	CLK ↓ → address delay time	t_{DKA}		5	40	ns
	CLK ↓ → status delay time	t_{DKST}		5	40	ns
	CLK ↑ → data output delay time	t_{DKD}		5	40	ns
	Floating delay time	t_{FK}		0	30	ns
	CLK ↑ → $\overline{DSTB}$ output delay time	t_{DKDS}		5	40	ns
	CLK ↓ → $\overline{DSTB}$ ↑ output delay time	t_{DKDSH}		5	40	ns
	CLK ↑ → HLDAK delay time	t_{DKHA}		5	40	ns
	$\overline{BCYST}$ low-level width	t_{BCBCL}		t_{CYK}–10		ns
★	$\overline{BCYST}$ high-level width	t_{BCBCH}		$t_{CYK}(n+1)$[Note 2]–10		ns
	$\overline{DSTB}$ low-level width	t_{DSDSL}		$t_{CYK}(n+1)$[Note 2]–10		ns
	$\overline{DSTB}$ high-level width	t_{DSDSH}		$t_{KKL}+t_{KR}$–10		ns

★ **Notes 1.** This specification applies to the delay times <1>-<3> shown below following the falling edge of the CLK signal.

<1> Address delay time

<2> $\overline{BUSLOCK}$ delay time

<3> Each of the following signal delay times immediately following bus hold release:
A23-A0, D15-D0, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, $\overline{BCYST}$, $\overline{DSTB}$

2. 'n' means number of wait clocks to be inserted into bus cycle. However, TC cycle (+1) is added to n in the case of the coprocessor cycle.

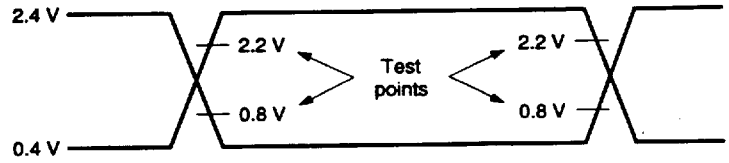
(2/2)

Item	Symbol	Conditions	MIN.	MAX.	Unit
Output float → HLDAK delay time	t_{DFHA}	C_L = 100pF	$t_{DKL}+t_{KR}-15$		ns
Address/status → output $\overline{DSTB}$ ↓ delay time	t_{DADSL}		$t_{DKL}+t_{KR}-15$		ns
$\overline{DSTB}$ ↑ output → address/status hold time (write operation)	t_{HDSHA}		$t_{DKL}+t_{KR}-15$		ns
$\overline{DSTB}$ ↑ output → data output delay time	t_{DDSHD}		$t_{DKL}+t_{KR}-15$		ns
$\overline{DSTB}$ ↓ output → data output floating time	t_{DHZ}			0	ns
$\overline{DSTB}$ ↑ output → data output set time	t_{DLZ}		$t_{DKL}+t_{KR}-15$		ns
Address/status output → data output delay time	t_{DAD}		$t_{DKL}+t_{KR}-15$		ns
Data setup time (CLK ↓)	t_{SDK}		7		ns
Data hold time (CLK ↓)[Note]	t_{HKD}		10		ns
Data hold time ($\overline{DSTB}$ ↑)[Note]	t_{HDSD}		0		ns
Data hold time[Note] (address/status change point)	t_{HASD}		0		ns
$\overline{READY}$ setup time (CLK ↑)	t_{SRYK}		7		ns
$\overline{READY}$ hold time (CLK ↑)	t_{HKRY}		15		ns
$\overline{BS8}$/BS16 setup time	t_{SBSK}		7		ns
$\overline{BS8}$/BS16 hold time	t_{HKBS}		15		ns
HLDRQ setup time (CLK ↑)	t_{SHQK}		7		ns
HLDRQ hold time (CLK ↑)	t_{HKHQ}		15		ns
$\overline{NMI}$, INT, $\overline{CPBUSY}$ setup time (CLK ↓)	t_{SIK}		10		ns
$\overline{NMI}$, INT, $\overline{CPBUSY}$ hold time (CLK ↓)	t_{HKI}		10		ns

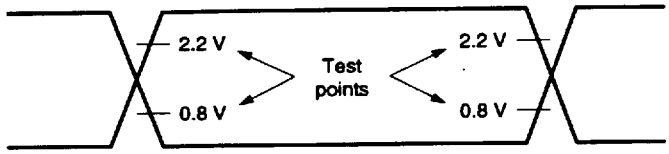
Note For t_{HKD}, t_{HDSD}, t_{HASD} specifications, at least one of these must be observed.

Remark t_{HKD} means data hold request specification from rising edge of clock.
t_{HDSD} and t_{HASD} are data hold request specifications from $\overline{DSTB}$ and address/status change points, respectively. This means that the $\overline{DSTB}$ and the address/status signal do not change, until the μPD70136A completes data read operation. (Address/status means A0-A23, $\overline{UBE}$, R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, and AEX.)

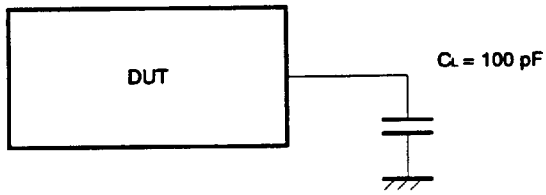
AC TEST INPUT WAVEFORMS (except CLK)



AC TEST OUTPUT MEASUREMENT POINTS

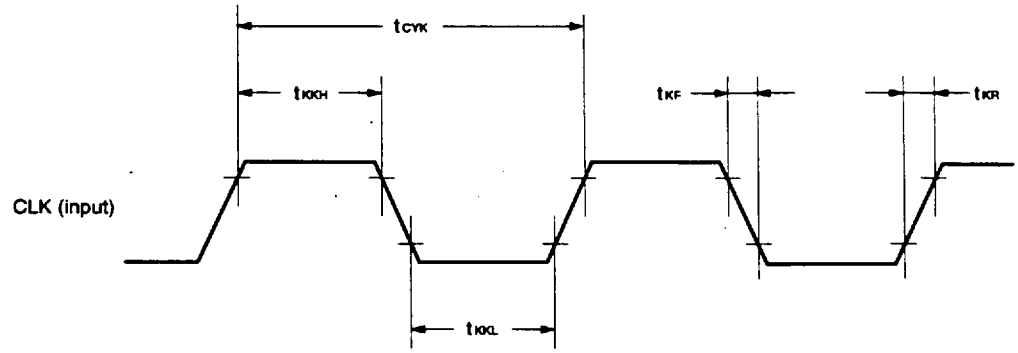


LOAD CONDITION

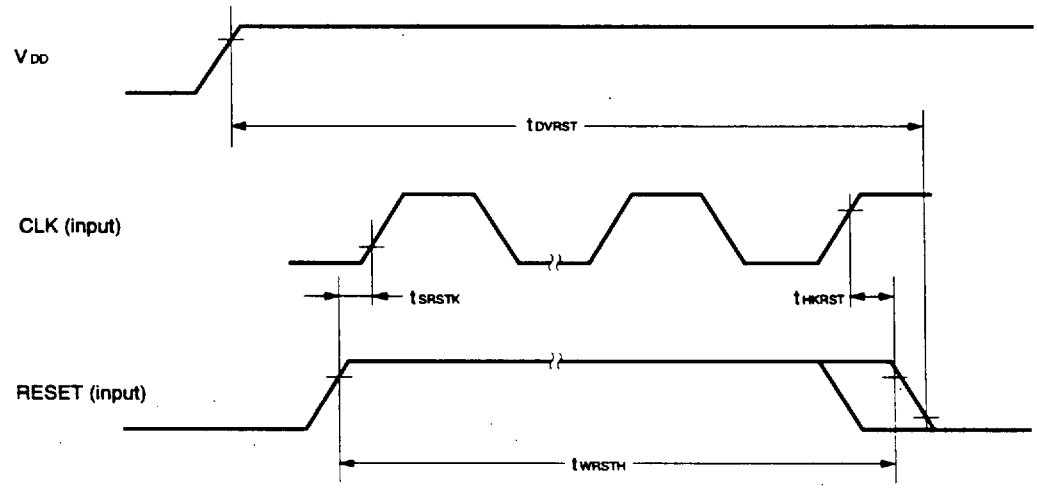


Caution **If load capacitance exceeds 100pF due to circuit configuration, reduce the load capacitance down to 100pF or less to this device by inserting a buffer, etc.**

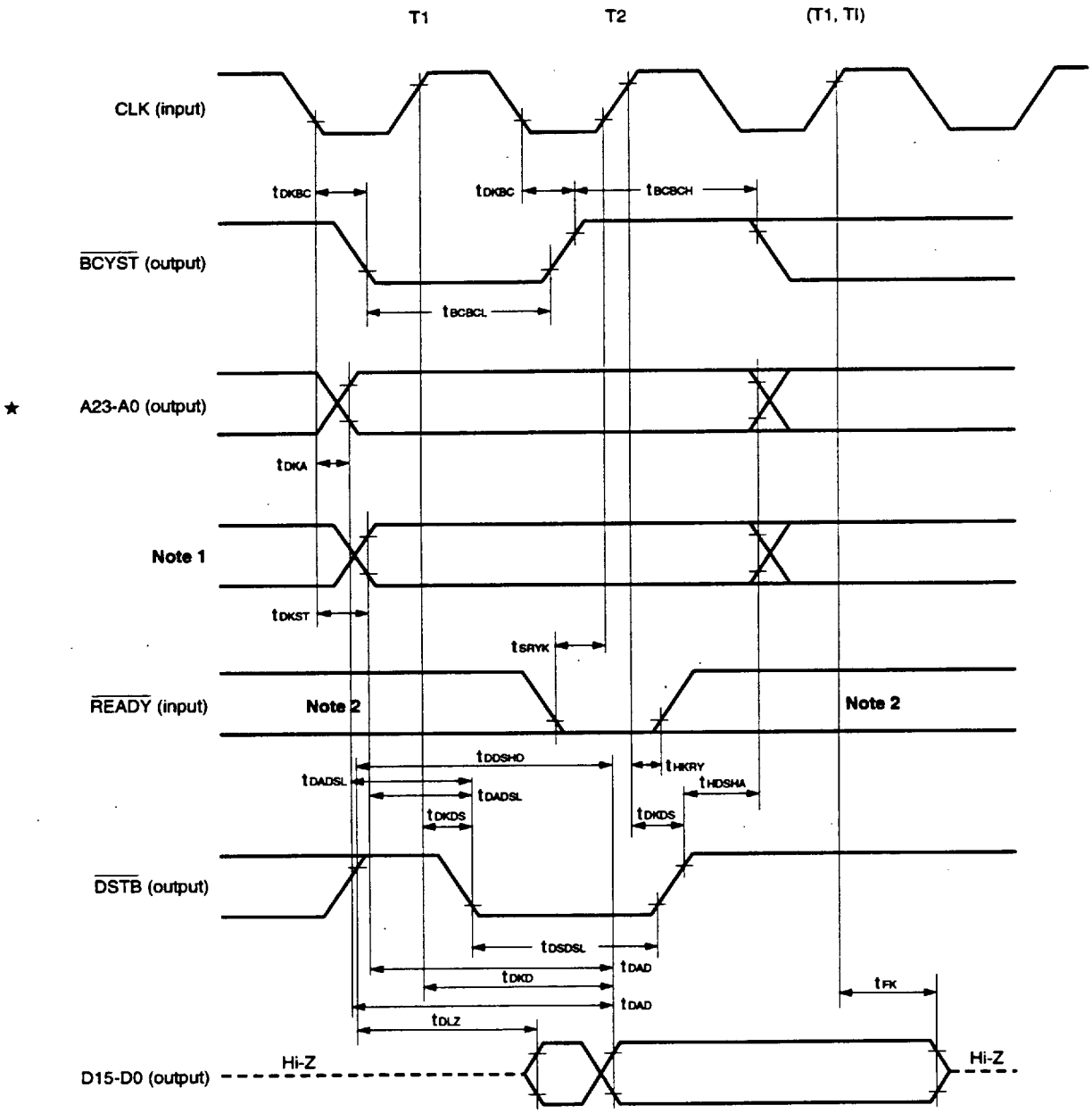
CLOCK TIMING



RESET TIMING ★



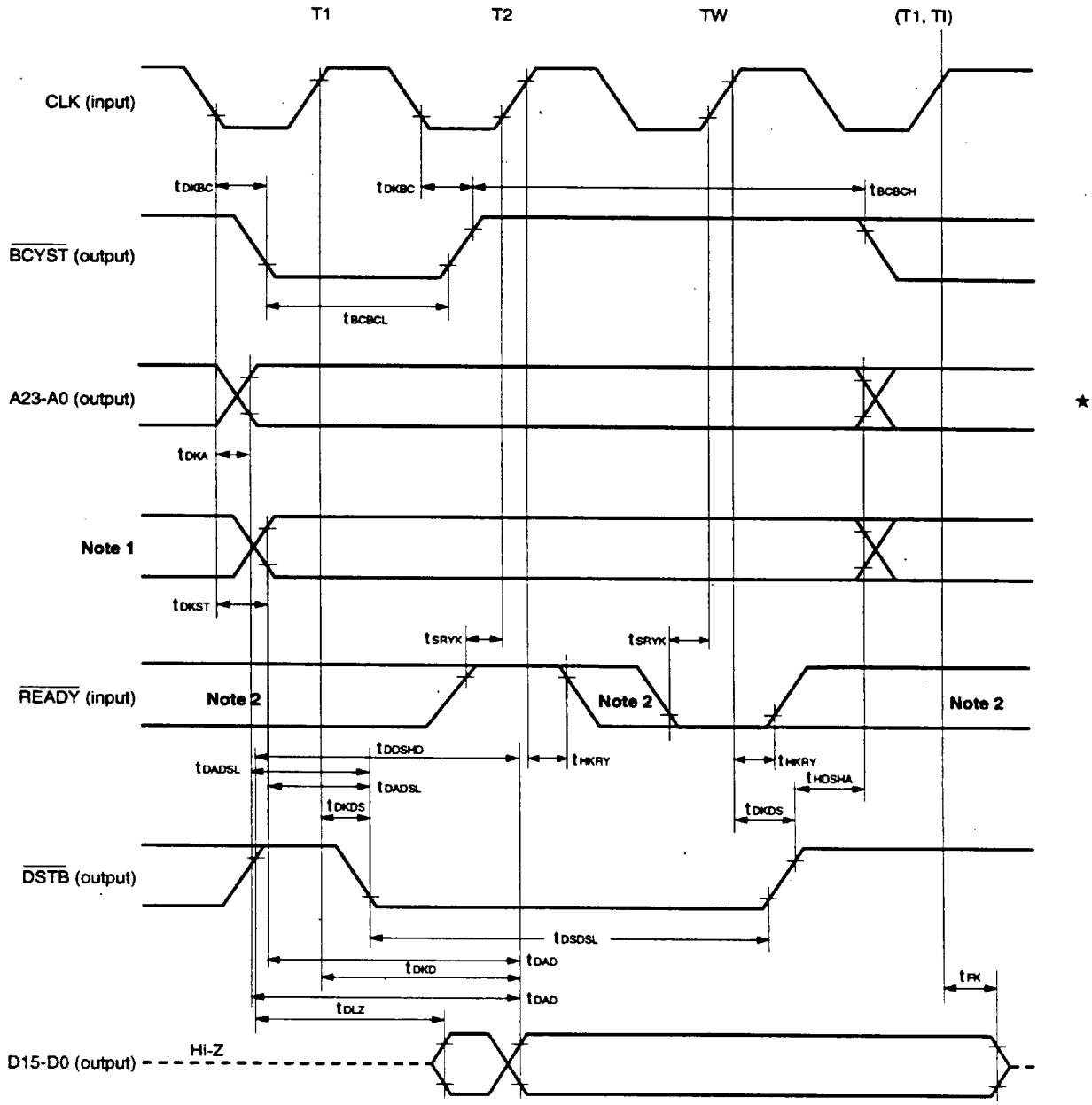
BASIC WRITE TIMING WAVEFORMS (0 wait)



Notes 1. R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, AEX (all outputs)

2. Change range

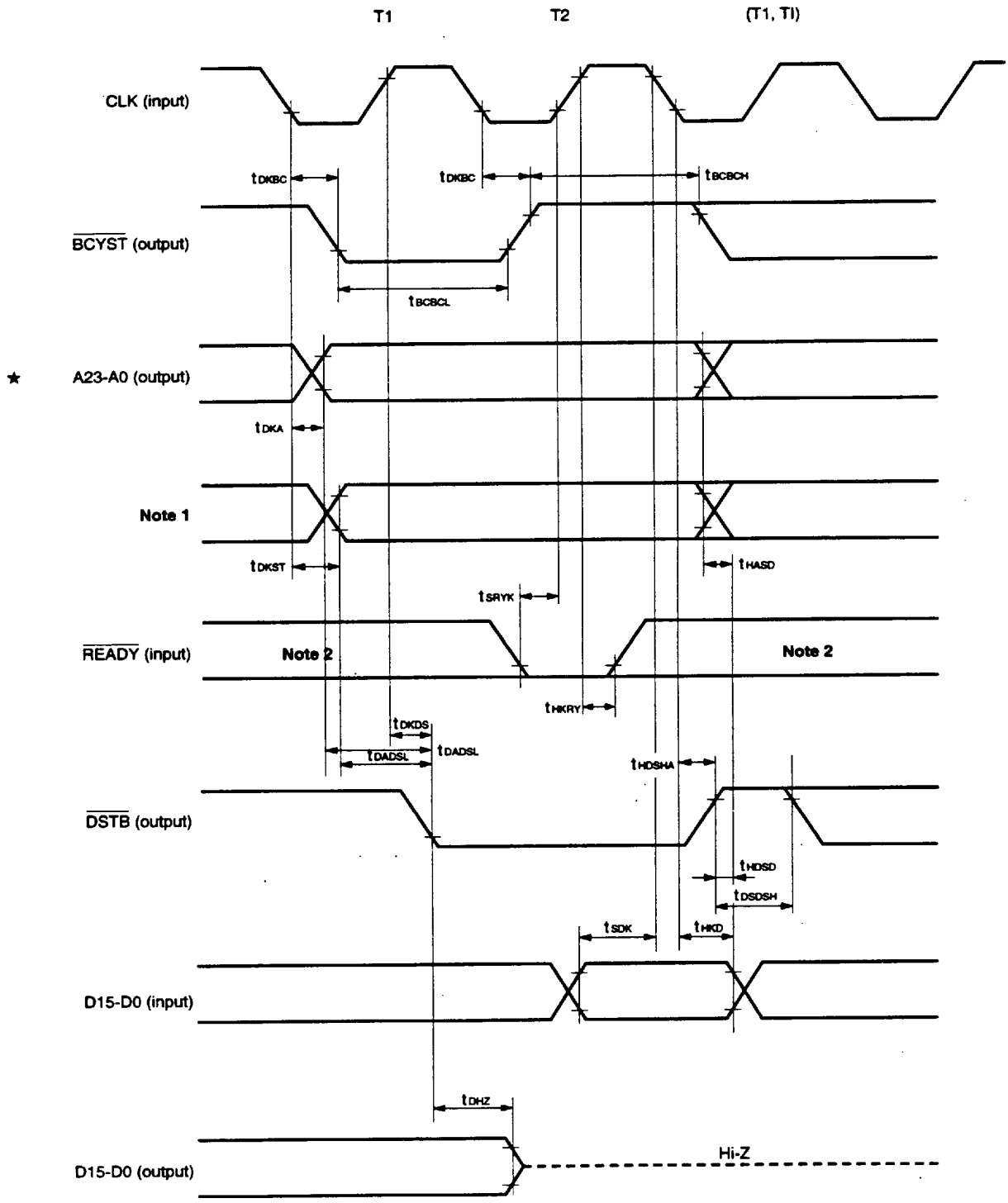
BASIC WRITE TIMING WAVEFORMS (1 wait)



Notes 1. R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, AEX (all outputs)

2. Change range

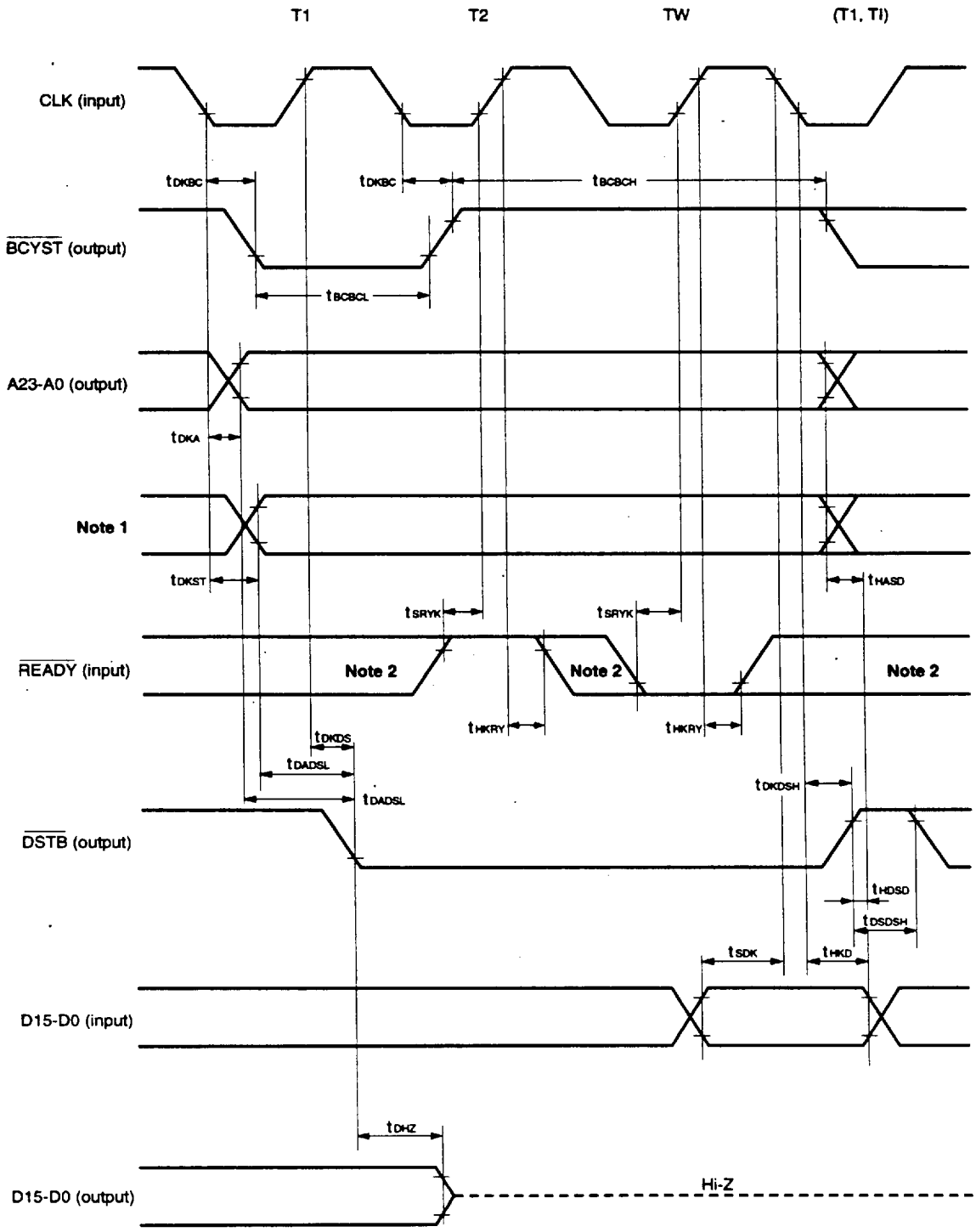
BASIC READ TIMING WAVEFORMS (0 wait)



Notes 1. R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, AEX (all outputs)

2. Change range

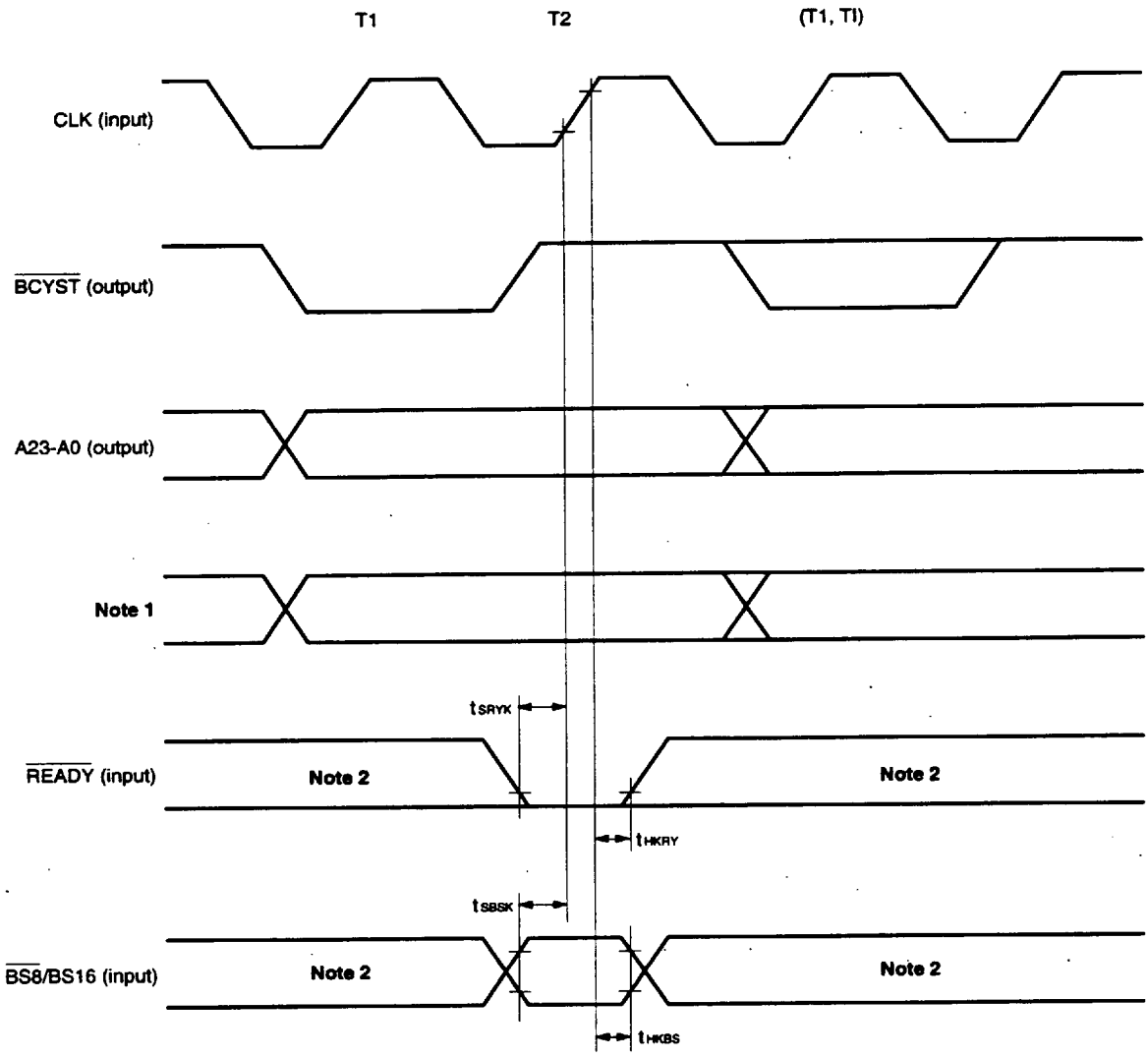
BASIC READ TIMING WAVEFORMS (1 wait)



Notes 1. R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, AEX (all outputs)

2. Change range

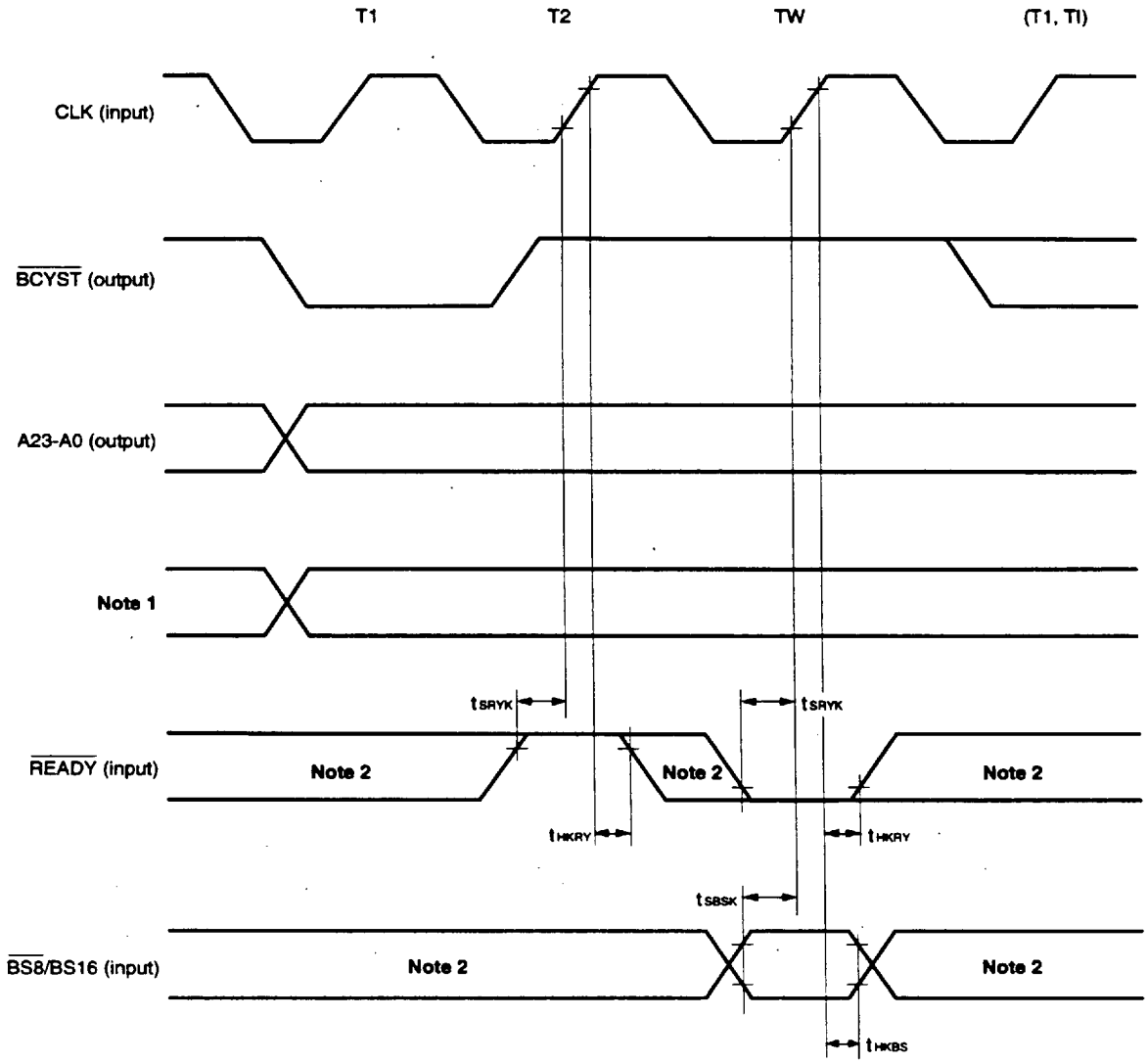
BUS SIZING TIMING WAVEFORMS (0 wait)



Notes 1. R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, AEX (all outputs)

2. Change range

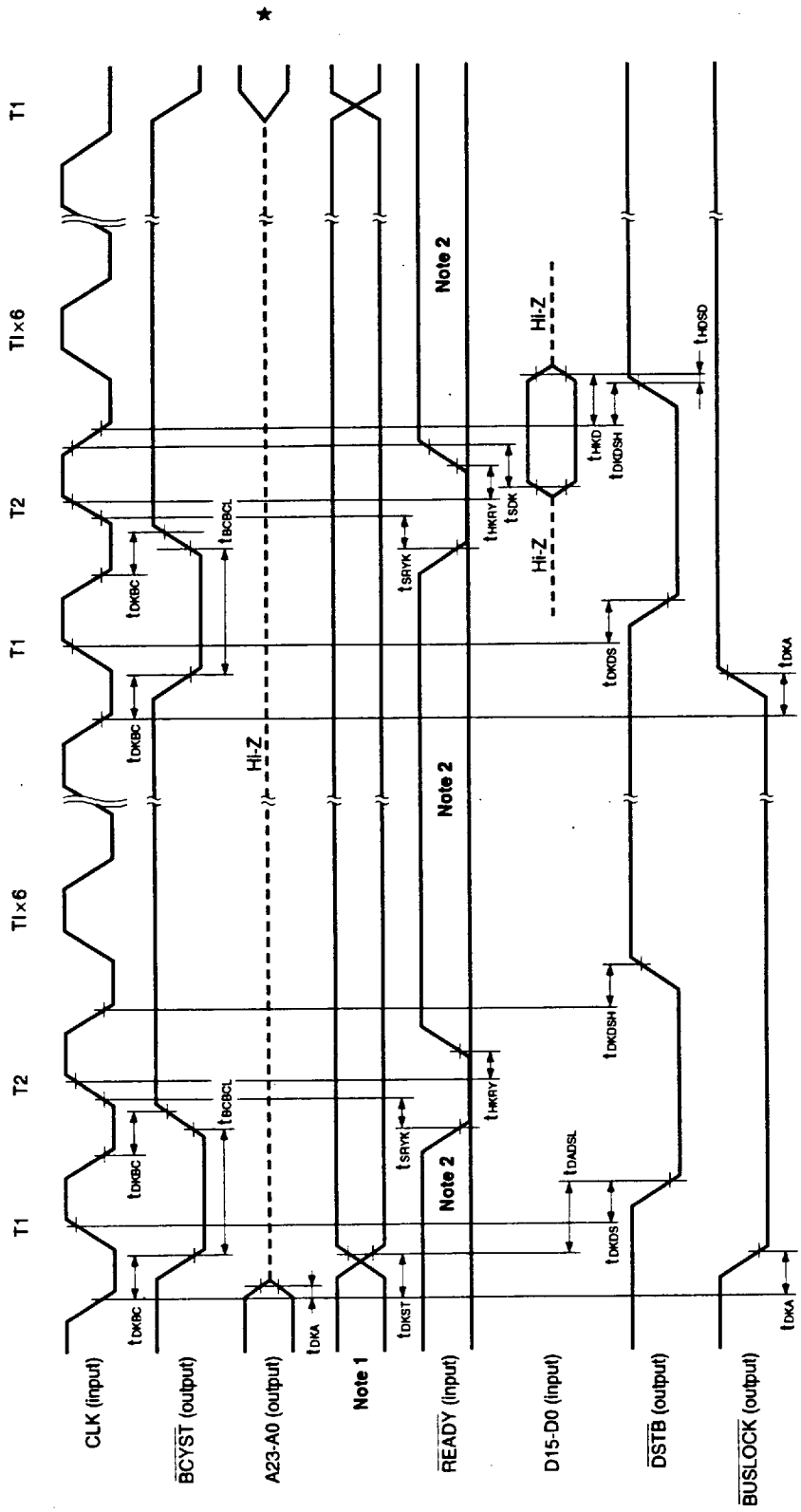
BUS SIZING TIMING WAVEFORMS (1 wait)



Notes 1. R/W, M/IO, BUSST1, BUSST0, UBE, AEX (all outputs)

2. Change range

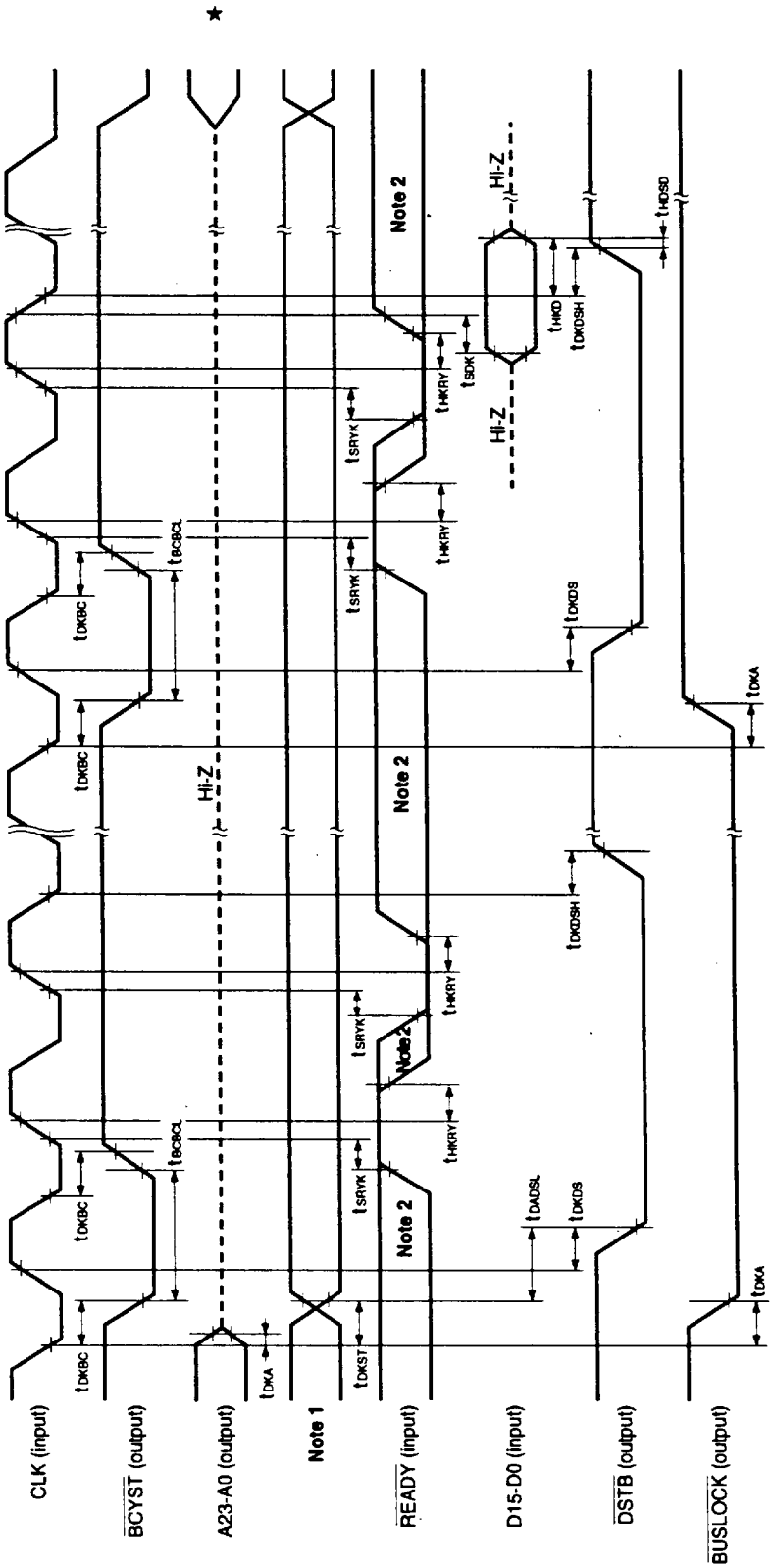
INTERRUPT ACKNOWLEDGE TIMING WAVEFORMS (0 wait)



Notes 1. R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, AEX (all outputs)

2. Change range

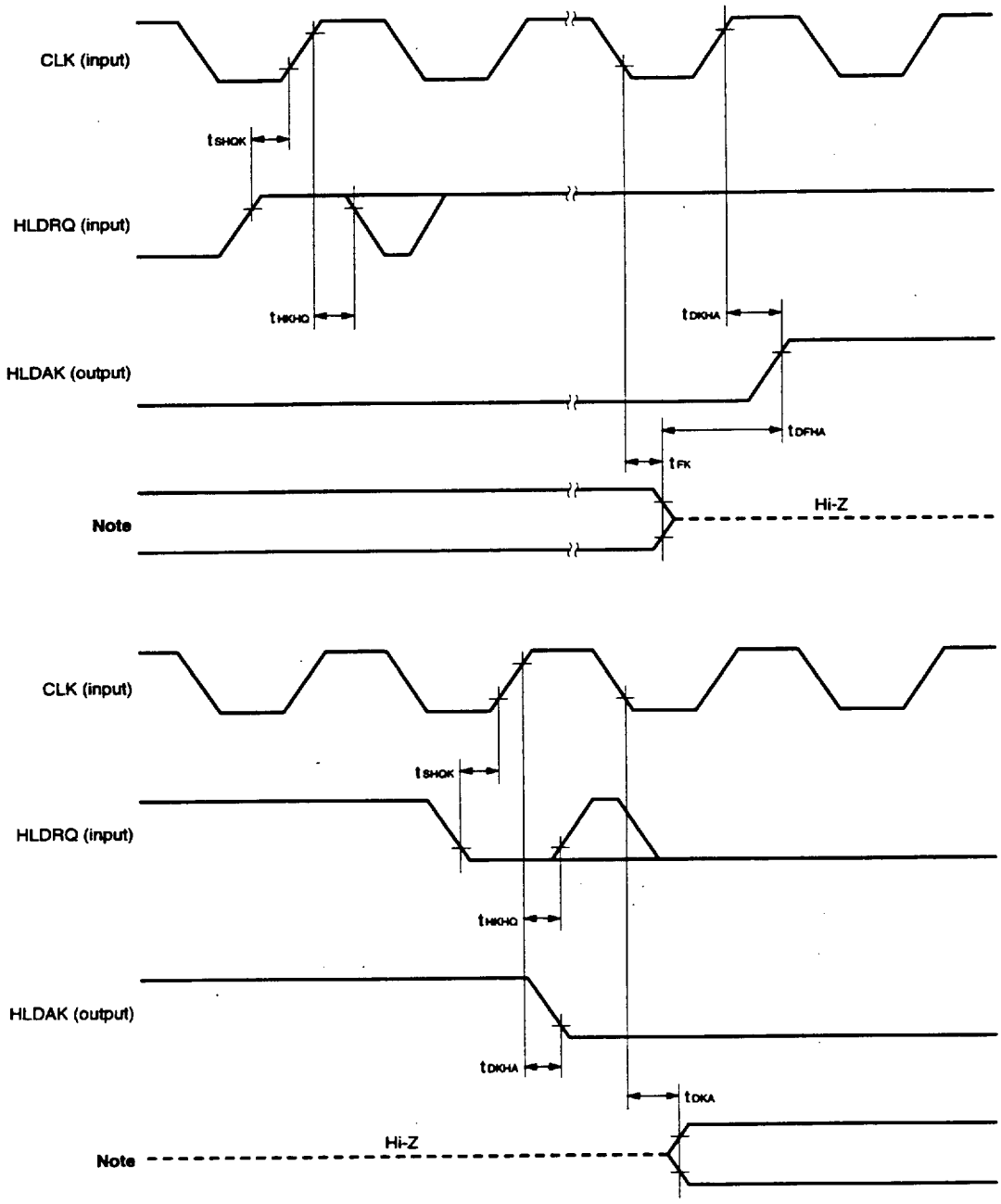
INTERRUPT ACKNOWLEDGE TIMING WAVEFORMS (1 wait)



Notes 1. R/$\overline{W}$, M/$\overline{IO}$, BUSST0, $\overline{UBE}$, AEX (all outputs)

2. Change range

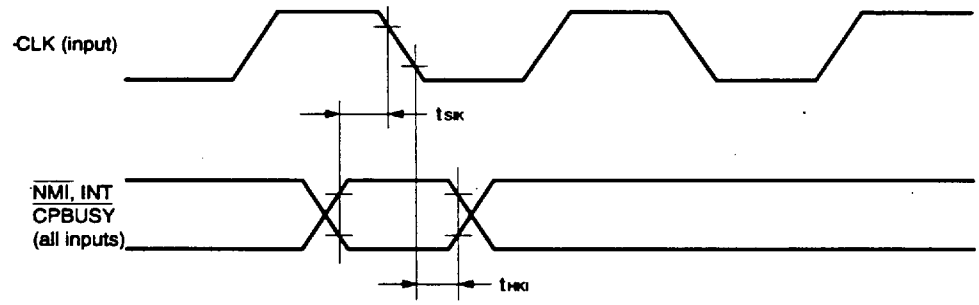
BUS HOLD TIMING WAVEFORMS



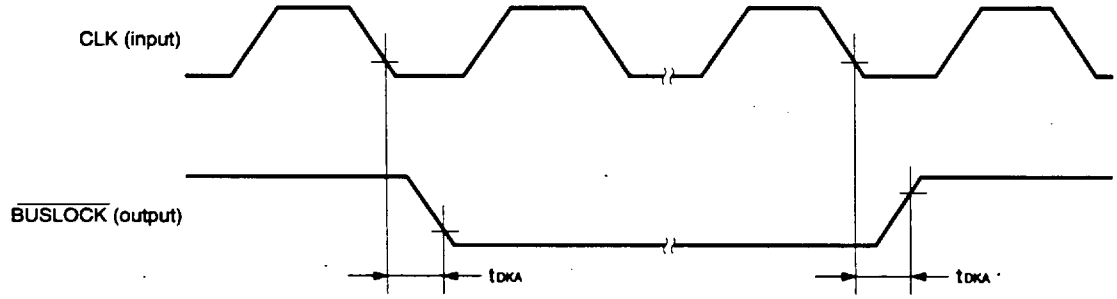
Note A23-A0, D15-D0, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, $\overline{BCYST}$, $\overline{DSTB}$

★ **Remarks**
1. If a hold cycle is executed immediately after an external I/O cycle, no idle state is inserted.
2. Pins that go to high impedance on bus hold or reset are internally latched; hence they will maintain their values if not externally driven.
3. If there is a hold request (high level input to the HLDRQ pin) during an I/O read cycle or memory read cycle, a hold cycle is executed with the $\overline{DSTB}$ signal at the active level (low level).

INPUT SET UP, INPUT HOLD TIMING WAVEFORMS

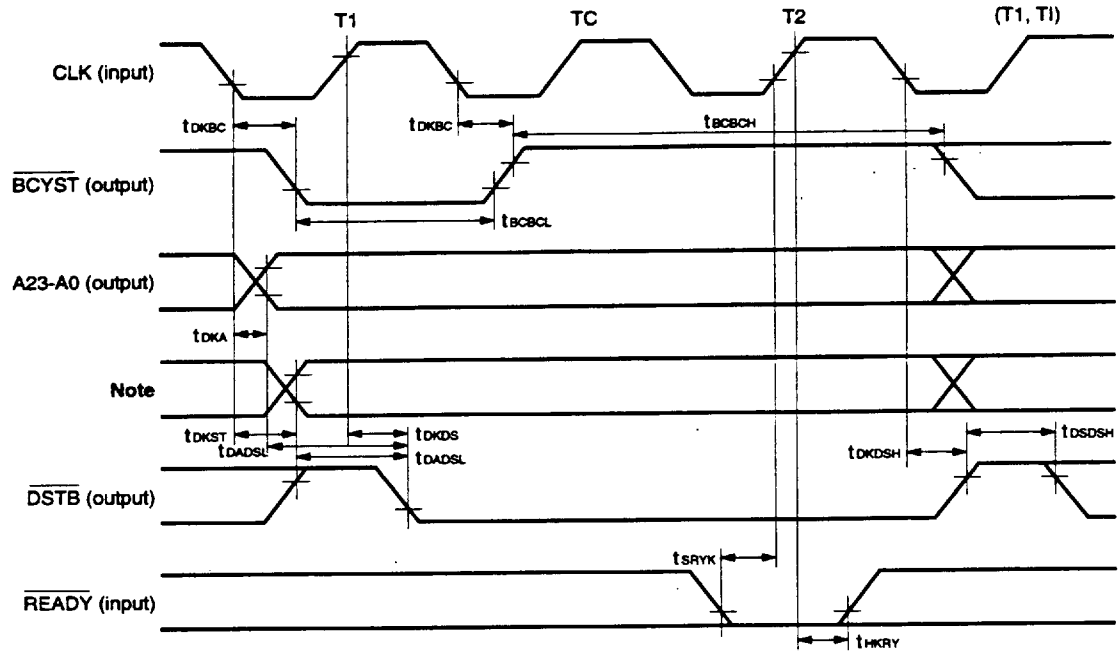


BUSLOCK TIMING WAVEFORMS



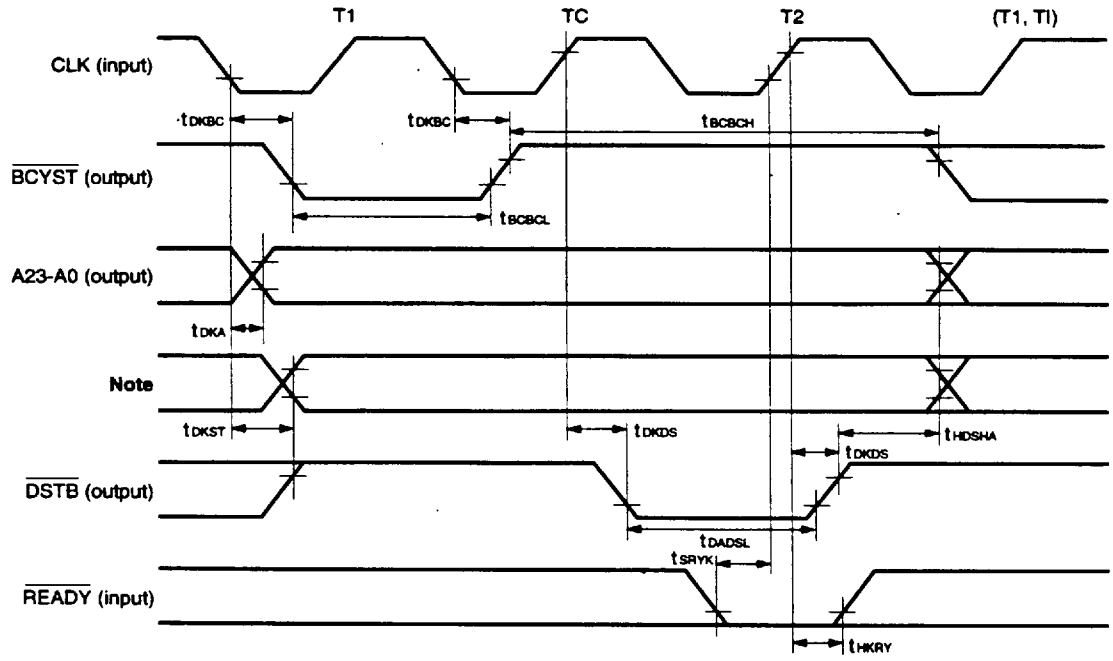
Remark When the BUSLOCK signal is output by an instruction with BUSLOCK prefix, the rising edge and falling edge do not depend on the bus cycle.
The rising edge and the falling edge are determined by the CPU's internal execution.
Refer to the timing waveform chart for the BUSLOCK signal in interrupt acknowledge cycle.

MEMORY READ CYCLE FOR COPROCESSOR (0 wait)



Note R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, AEX (all outputs)

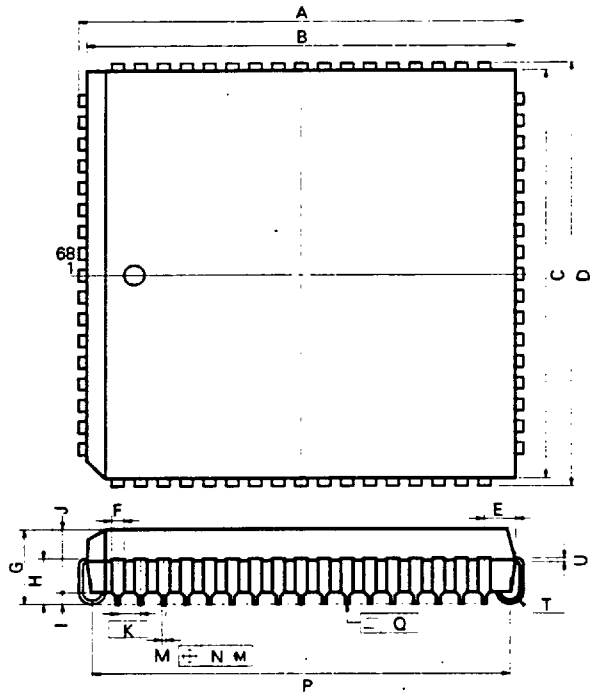
MEMORY WRITE CYCLE FOR COPROCESSOR (0 wait)



Note R/$\overline{W}$, M/$\overline{IO}$, BUSST1, BUSST0, $\overline{UBE}$, AEX (all outputs)

14. PACKAGE DRAWINGS

68 PIN PLASTIC QFJ (□950 mil)



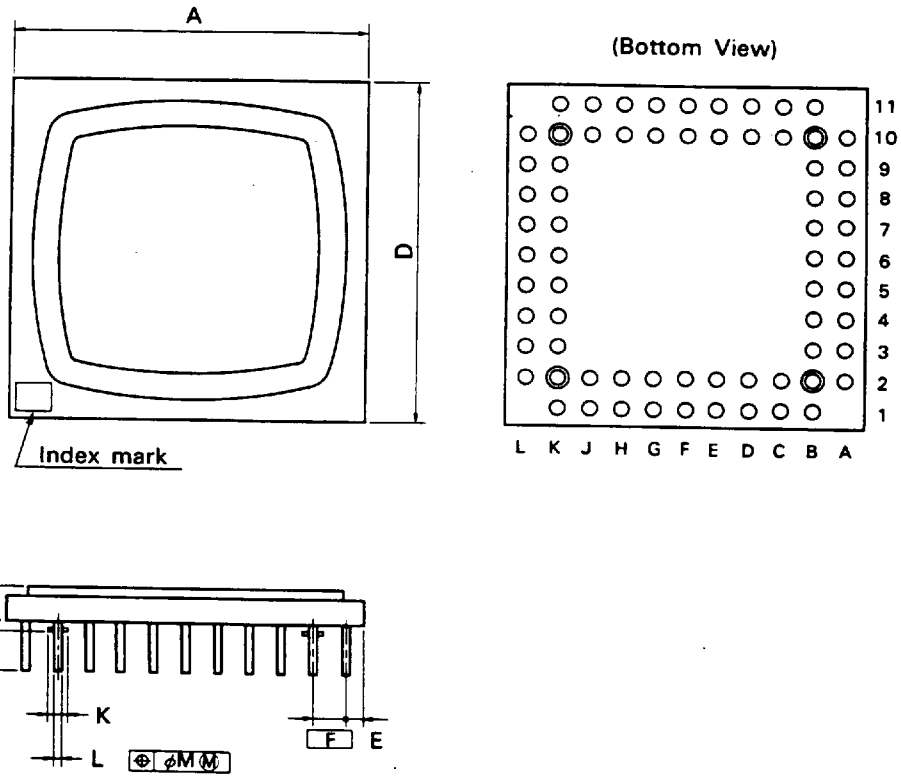
P68L-50A1-2

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	25.2±0.2	0.992±0.008
B	24.20	0.953
C	24.20	0.953
D	25.2±0.2	0.992±0.008
E	1.94±0.15	$0.076^{+0.007}_{-0.006}$
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±1.0	$0.016^{+0.004}_{-0.005}$
N	0.12	0.005
P	23.12±0.20	$0.910^{+0.009}_{-0.008}$
Q	0.15	0.006
T	R 0.8	R 0.031
U	$0.20^{+0.10}_{-0.05}$	$0.008^{+0.004}_{-0.002}$

68PIN CERAMIC PGA (SEAM WELD)



NOTE

Each lead centerline is located within φ0.5 mm (φ0.020 inch) of its true position (T.P.) at maximum material condition.

X68R-100A-1

ITEM	MILLIMETERS	INCHES
A	$27.94^{\pm 0.4}$	$1.100^{+0.018}_{-0.016}$
D	$27.94^{\pm 0.4}$	$1.100^{+0.018}_{-0.016}$
E	1.27	0.050
F	2.54 (T.P.)	0.100 (T.P.)
G	$2.8^{\pm 0.3}$	$0.110^{+0.012}_{-0.011}$
H	0.5 MIN.	0.019 MIN.
I	2.7	0.106
J	4.57 MAX.	0.180 MAX.
K	$\phi 1.2^{\pm 0.2}$	$\phi 0.047^{+0.008}_{-0.007}$
L	$\phi 0.46^{\pm 0.05}$	$\phi 0.018^{+0.002}_{-0.001}$
M	0.5	0.020

★ 15. RECOMMENDED SOLDERING CONDITIONS

For the μPD70136A, soldering must be performed under the following conditions.

For details of recommended conditions for surface mounting, refer to information document **"Semiconductor Device Mounting Technology Manual" (IEI-1207).**

For other soldering methods, please consult with NEC sales personnel.

Table 15-1. Surface Mounting Type Soldering Conditions

μPD70136AL: 68-pin plastic QFJ (□950 mil)

Soldering Method	Soldering Conditions	Recommended Conditions Reference Code
Infrared reflow	Package peak temperature: 235 °C max., time: up to 30 sec. (over 210 °C), count: twice or less, restriction days: 7[Note] (after that, 125 °C pre-bake for 36 hours is necessary) Precautions: (1) Reflow a second time should be started when the device temperature has returned to its normal state after the first reflow. (2) Avoid flux cleaning with water after the first reflow.	IR35-367-2
VPS	Package peak temperature: 215 °C, time: up to 40 sec. (over 200 °C), count: twice or less, restriction days: 7[Note] (after that, 125 °C pre-bake for 36 hours is necessary) Precautions: (1) Reflow a second time should be started when the device temperature has returned to its normal state after the first reflow. (2) Avoid flux cleaning with water after the first reflow.	VP15-367-2
Pin partial heating	Pin temperature: 300°C max., time: 3 seconds max. (per device)	–

Note This means the number of days after unpacking the dry pack. Storage conditions are 25°C and 65% RM max.

Caution Do not use one soldering method in combination with another. (howev er, pin partial heating can be performed with other soldering methods).

Table 15-2. Insertion Type Soldering Conditions

μPD70136AR: 68-pin ceramic PGA (seam welding)

Soldering Method	Soldering Conditions
Wave soldering (Only for pin)	Solder bath temperature: 260°C max., time: 10 seconds max.
Pin partial heating	Pin temperature: 300 °C max., time: 3 seconds max. (per pin)

Caution The wave soldering must be performed at the pin part only. Note that the solder must not be directly contacted to the package body.

Instruction	μPD70108, 70116	μPD70136	μPD70136A
Number of prefix instructions	<Block instruction> Up to 3 instructions can be stored. <Other instructions> No constraints	<Block instruction/exception> Up to 3 types can be stored. <Other instructions> Up to 3 types can be stored. Two or more prefixes of the same type must not be attached.	<Block instruction/exception> Up to 7 instructions can be stored. <Other instructions> Up to 7 instructions can be stored.
ADJ4A/ADJ4Sinstruction Note 1	When 9A ≤ AL ≤ 9F Upper 4 bits are adjusted only when AC = 0 [Instruction function] • When AL ^0FH > 9 or AC = 1 Lower 4 bits of AL register are adjusted • When AL > 9FH or CY = 1, Upper 4 bits of AL register are adjusted. • When 99H < AL < A0H and AC = 0, Upper 4 bits of AL register are adjusted	When 9A ≤ AL ≤ 9F Upper 4 bits are always adjusted [Instruction function] • When AL ^0FH > 9 or AC = 1, Lower 4 bits of AL register are adjusted • When AL > 99H or CY = 1, Upper 4 bits of AL register are adjusted.	When 9A ≤ AL ≤ 9F Upper 4 bits are adjusted, only when AC = 0 [Instruction function] • When AL ^0FH > 9 or AC = 1, Lower 4 bits of AL register are adjusted • When AL > 9FH or CY = 1, Upper 4 bits of AL register are adjusted. • When 99H < AL < A0H and AC = 0, Upper 4 bits of AL register are adjusted
PUSH SP POP SP	(SP–2) ← SP–2 Note 2 SP ← (SP)	(SP–2) ← SP SP ← (SP)	(SP–2) ← SP–2 Note 2 SP ← (SP)

Notes 1. These differences appear when adjustment is made on operation results, other than decimal data operation.
(For operation between decimal data, "99AH AL 9FH and AC = 1" will not occur.)

2. If PUSH and POP instructions are executed on the SP register, the result left in the SP register is two less than its initial value.

Remark The following three types of prefixes are available:

- Repeat ..... REPC, REPNC, REPZ, REPNZ
- Segment override ..... PS:, DS0:, DS1:, SS:
- Bus lock ..... BUSLOCK

Instruction		μPD70108, 70116	μPD70136	μPD70136A
Return address for exception oriented interrupt		Returns to the address for the next instruction of the instruction for which the interrupt is generated (DIV, CHKIND)	Returns to the address at which the exception is generated.	Returns to the address at which the exception is generated. (However, for DIV, CHKIND exceptions, returns to the next instruction address.)
DIV error		When quotient is 80H (byte), DIV error is generated.	When quotient is 81H (byte), or 8000H (word), the normal calculation is performed.	When quotient is 80H (byte), DIV error is generated.
Interrupt vector number	Undefined instruction	None	122	6
	Coprocessor not present		130	7
	μPD72291 error		128	16
POP R		Starts POP cycle 7 times [Instruction function] IY ← (SP+1, SP) IX ← (SP+3, SP+2) BP ← (SP+5, SP+4) BW ← (SP+9, SP+8) DW ← (SP+11, SP+10) CW ← (SP+13, SP+12) AW ← (SP+15, SP+14) SP ← (SP+16)	Starts POP cycle 8 times [Instruction function] IY ← (SP+1, SP) IX ← (SP+3, SP+2) BP ← (SP+5, SP+4) temp ← (SP+7, SP+6)Note BW ← (SP+9, SP+8) DW ← (SP+11, SP+10) CW ← (SP+13, SP+12) AW ← (SP+15, SP+14) SP ← (SP+16)	Starts POP cycle 8 times [Instruction function] IY ← (SP+1, SP) IX ← (SP+3, SP+2) BP ← (SP+5, SP+4) temp ← (SP+7, SP+6)Note BW ← (SP+9, SP+8) DW ← (SP+11, SP+10) CW ← (SP+13, SP+12) AW ← (SP+15, SP+14) SP ← (SP+16)
POLL		Waits until $\overline{\text{POLL}}$ pin becomes low.	• When coprocessor not present Generates coprocessor not present exception • When μPD72291 is connected Waits until $\overline{\text{CPBUSY}}$ pinbecomes high.	• When coprocessor not present Generates coprocessor not present exception • When μPD72291 is connected Waits until $\overline{\text{CPBUSY}}$ pin becomes high.

Note This operation has no meaning at all.