μPD70136 (V33) 16-Bit Microprocessor: High-Speed, CMOS

Description

The μPD70136 (V33™) is a 16-bit, high-speed CMOS microprocessor that is object and source code compatible with the μPD70116 (V30®). Performance is four times that of the 10-MHz V30 due to a number of architectural features, such as hard-wired data path control and dedicated high-speed logic. The address space is expanded to 16M bytes using an internal address translation table.

The powerful instruction set includes bit processing, bit-field insertion and extraction, and BCD string arithmetic. Using a modified Booth's algorithm, the 16-MHz device can execute a 16-bit multiply in 750 ns.

The μPD70136 has separate 16-bit data and 24-bit address buses. Bus control is synchronous. The nominal bus cycle is two clock periods. Dynamic bus sizing is supported for devices that require an 8-bit data path. This allows the μPD70136 to be used in either 16- or 8-bit systems.

An undefined instruction trap allows instructions that are not part of the V-Series instruction set (such as commands for proprietary MMUs) to be emulated. The μPD72291, a high-speed CMOS floating-point coprocessor capable of 530K floating-point operations per second at 16 MHz, is offered.

Features

- 125-ns minimum instruction execution time at 16 MHz
- Expanded address space
  - 24-bit addressing to 16M bytes
  - LIM 4.0 compatible
- No microcode; better performance with hard-wired data path control
- Dynamic bus sizing for both memory and I/O
- Fully μPD70116 software compatible
- Undefined instruction trap
- High-speed multiplication: 16-bit multiply in 12 clocks (0.75 μs at 16 MHz)
- High-speed division: 16-bit divide in 19 clocks (1.19 μs at 16 MHz)
- μPD72291 floating-point coprocessor executes 530K floating-point operations per second
- BCD string arithmetic instructions
- CMOS with low-power standby mode
- 12.5-MHz or 16-MHz clock
- Single power supply

3e

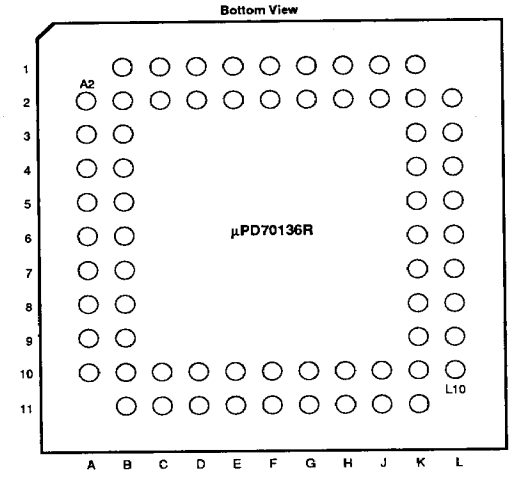
Ordering Information

Part Number	Clock (MHz)	Package
μPD70136R-12	12.5	68-pin ceramic PGA
R-16	16	
L-12	12.5	68-pin PLCC
L-16	16	
GJ-12	12.5	74-pin plastic QFP
GJ-16	16	

V30 is a registered trademark of NEC Corporation
V20, V33, V40, and V50 are trademarks of NEC Corporation

Pin Configurations

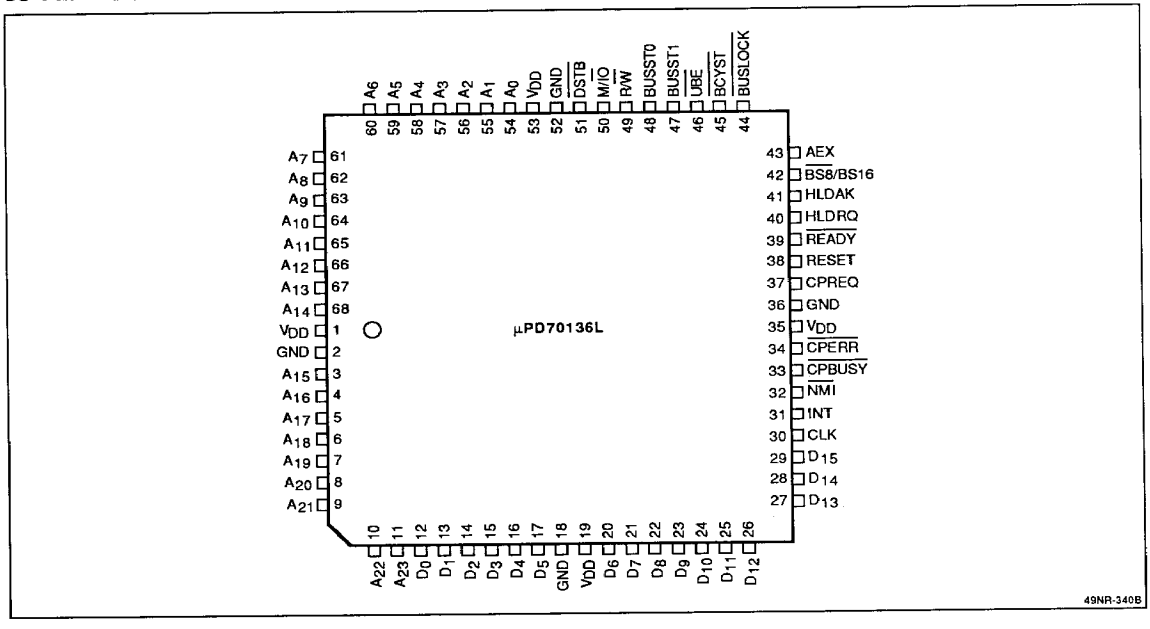
68-Pin Ceramic PGA



Pin	Symbol	Pin	Symbol	Pin	Symbol	Pin	Symbol
A2	AEX	B9	CLK	F10	V_{DD}	K4	A_{12}
A3	HLDAK	B10	D_{14}	F11	GND	K5	A_{14}
A4	$\overline{READY}$	B11	D_{12}	G1	A_0	K6	GND
A5	CPREQ	C1	$\overline{UBE}$	G2	A_1	K7	A_{16}
A6	V_{DD}	C2	BUSST1	G10	D_5	K8	D_{18}
A7	$\overline{CPBUSY}$	C10	D_{11}	G11	D_4	K9	A_{20}
A8	INT	C11	D_{10}	H1	A_2	K10	A_{23}
A9	D_{15}	D1	BUSST0	H2	A_3	K11	A_{22}
A10	D_{13}	D2	$R/\overline{W}$	H10	D_3	L2	A_7
B1	$\overline{BUSLOCK}$	D10	D_9	H11	D_2	L3	A_9
B2	$\overline{BCYST}$	D11	D_8	J1	A_4	L4	A_{11}
B3	$\overline{BS8}$/BS16	E1	$M/\overline{IO}$	J2	A_5	L5	A_{13}
B4	HLDRQ	E2	$\overline{DSTB}$	J10	D_1	L6	V_{DD}
B5	RESET	E10	D_7	J11	D_0	L7	A_{15}
B6	GND	E11	D_6	K1	A_6	L8	A_{17}
B7	$\overline{CPERR}$	F1	GND	K2	A_8	L9	A_{19}
B8	$\overline{NMI}$	F2	V_{DD}	K3	A_{10}	L10	A_{21}

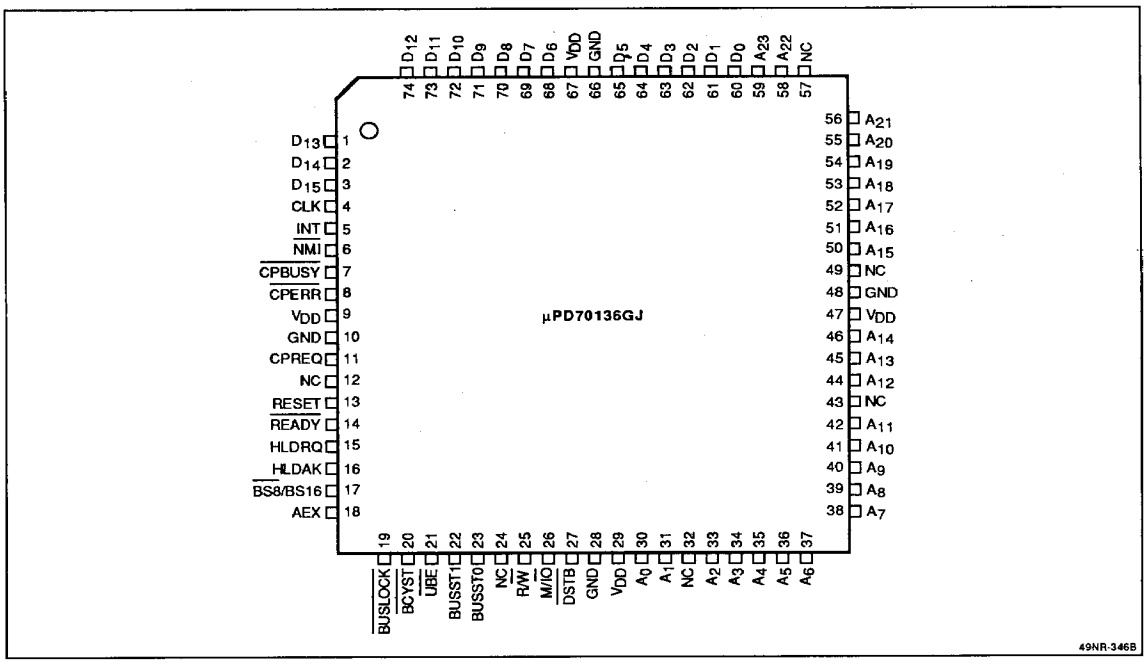
49NR-339B

68-Pin PLCC



49NR-340B

74-Pin Plastic QFP



3e

Pin Identification

Symbol	I/O	Function
A_0-A_{23}	3-state	Address bus
D_0-D_{15}	3-state	Data bus
$\overline{UBE}$	3-state	Upper byte enable
R/$\overline{W}$	3-state	Read/write
M/$\overline{IO}$	3-state	Memory I/O
BUSST0, BUSST1	3-state	Bus status
$\overline{BCYST}$	3-state	Bus cycle start strobe
$\overline{DSTB}$	3-state	Data strobe
$\overline{BUSLOCK}$	Out	Bus lock
$\overline{READY}$	In	Ready
$\overline{BS8}$/BS16	In	Dynamic bus sizing control
AEX	Out	Address expansion flag
HLDRQ	In	Bus hold request
HLDAK	Out	Bus hold acknowledge
INT	In	Maskable interrupt
$\overline{NMI}$	In	Nonmaskable interrupt
$\overline{CPBUSY}$	In	Coprocessor busy
$\overline{CPERR}$	In	Coprocessor error
CPREQ	In	Coprocessor request
RESET	In	Reset
CLK	In	Clock
V_{DD}	—	+5-volt power supply
GND	—	Ground
IC	—	Internal connection; connect to ground
NC	—	No connection

Table 1. Output Pin States

Symbol	States		
	Hold	Standby	Reset
A_0-A_{23} (Note 1)	Hi-z	L	Hi-z
D_{15}-D_0 (Note 1)	Hi-z	(Note 2)	Hi-z
$\overline{UBE}$ (Note 1)	Hi-z	H	Hi-z
R/$\overline{W}$ (Note 1)	Hi-z	L	Hi-z
M/$\overline{IO}$ (Note 1)	Hi-z	L	Hi-z
BUSST0, BUSST1 (Note 1)	Hi-z	H	Hi-z
$\overline{BCYST}$ (Note 1)	Hi-z	(Note 3)	Hi-z
$\overline{DSTB}$ (Note 1)	Hi-z	H	Hi-z
$\overline{BUSLOCK}$	(Note 4)	(Note 4)	H
AEX	(Note 5)	(Note 5)	L
HLDAK	H	L	L

Notes:

(1) Latched internally.

(2) Undefined during first two clock periods of the halt acknowledge cycle; three-state thereafter.

(3) Low during first clock period of the halt acknowledge cycle; high thereafter.

(4) Low if $\overline{BUSLOCK}$ prefix is used for halt instruction; high otherwise.

(5) Low if in extended addressing mode; high otherwise.

PIN FUNCTIONS

CLK (Clock)

CLK is the main clock. All timing is relative to this input. Each bus state is one CLK period wide. Instruction clock counts refer to this CLK input.

A_0-A_{23} (Address Bus)

A_0-A_{23} form the 24-bit physical address bus. It is used to access both the 16M-byte expanded and 1M-byte normal memory spaces and the 64K-byte I/O space. These three-state outputs become valid during T1 of all bus cycles and remain valid until after the bus cycle is completed. During HLDAK and when RESET is active, these outputs are not driven.

D_0-D_{15} (Data Bus)

D_0-D_{15} form the 16-bit data bus, which is used to transfer 16- and 8-bit data between the μPD70136 and the external system. To accommodate 8-bit devices, dynamic bus sizing can be selected so that only the lower 8 bits, D_0-D_7, are used. During CPU read cycles, the value on the data bus is latched by the CPU on the trailing edge of T2 or the last TW state. During CPU write cycles, D_0-D_{15} become valid after the rising edge of T1 and remain valid until after the rising edge of the clock cycle following T2 or the last TW state. During HLDAK and when RESET is asserted, D_0-D_{15} are not driven.

$\overline{UBE}$ (Upper Byte Enable)

$\overline{UBE}$ indicates that the upper 8 bits of the data bus will be used in the current CPU bus cycle. This signal is used in conjunction with A_0 as shown in table 2.

Table 2. Bus Operation vs $\overline{UBE}$ and A_0

Bus Operation	$\overline{UBE}$	A_0	Number of Bus Cycles
Word at even address, $\overline{BS8}$/BS16 = 0	0	0 (Note 1)	2
	1	1 (Note 3)	
Word at odd address	0	1 (Note 1)	2
	1	0 (Note 2)	
Byte at even address	1	0	1
Byte at odd address	0	1	1

Notes:

(1) First bus cycle

(2) Second bus cycle

(3) Second cycle for bus sizing

$\overline{UBE}$ has the same timing as A_0-A_{23} and is not driven during HLDAK or while RESET is asserted.

R/$\overline{W}$ (Read/Write)

R/$\overline{W}$ indicates whether the current bus cycle will be a read or a write. If R/$\overline{W}$ is high, then the cycle will be a read; if low, a write cycle. R/$\overline{W}$ has the same timing as A_0-A_{23} and is not driven during HLDAK or while RESET is asserted.

M/$\overline{IO}$ (Memory/$\overline{IO}$)

M/$\overline{IO}$ indicates whether the current bus cycle will be an access to the memory or I/O space. If M/$\overline{IO}$ is high, access will be to memory; if low, to the I/O space. M/$\overline{IO}$ is used with BUSST0 and BUSST1 to identify the cycle type. M/$\overline{IO}$ has the same timing as A_0-A_{23} and is not driven during HLDAK or while RESET is asserted.

BUSST0-BUSST1 (Bus Cycle Status)

BUSST0 and BUSST1, in conjunction with M/$\overline{IO}$ and R/$\overline{W}$, identify the current cycle type as shown in table 3.

Table 3. Bus Cycle Types

Status				
M/$\overline{IO}$	R/$\overline{W}$	BUSST1	BUSST0	Type of Bus Cycle
0	1	0	0	Interrupt acknowledge
0	1	0	1	I/O read
0	0	0	1	I/O write
0	1	1	0	Coprocessor read
0	0	1	0	Coprocessor write
0	0	1	1	HALT acknowledge
1	1	0	0	Instruction fetch
1	1	0	1	Memory read
1	0	0	1	Memory write
1	1	1	0	CP data read
1	0	1	0	CP data write

Note: All bus status signals change after the start of the T1 state.

The 11 cycle types are described in detail in the bus cycles section. The remaining five combinations of these inputs are reserved for future use.

BUSST0-BUSST1 have the same timing as the address bus, A_0-A_{23}, and are not driven during HLDAK or while RESET is asserted.

$\overline{BCYST}$ (Bus Cycle Start)

$\overline{BCYST}$ indicates the start of a bus cycle. It is asserted low during T1 of every bus cycle, and only for the first clock period of each bus cycle. $\overline{BCYST}$ is not driven during HLDAK or while RESET is asserted.

$\overline{DSTB}$ (Data Strobe)

$\overline{DSTB}$ indicates the status of the data on D_0-D_{15}. When asserted low during a write cycle, the μPD70136 drives the write data on D_0-D_{15}. When the CPU asserts this output during a read cycle, external logic should drive the read data onto D_0-D_{15}.

$\overline{DSTB}$ is asserted following the rising edge (middle) of T1, and stays asserted through T2 and any TW (wait) state that may be inserted. During write cycles, $\overline{DSTB}$ will be deasserted after the rising edge of either T2 or the last wait state. During read cycles, $\overline{DSTB}$ is deasserted after the trailing edge of T2 or the last wait state. $\overline{DSTB}$ is not driven during HLDAK, HALT acknowledge cycles, or while RESET is asserted.

$\overline{BUSLOCK}$ (Bus Lock)

$\overline{BUSLOCK}$ should be used by external logic to exclude any other bus master (e.g., a DMA controller) from using a shared resource that the μPD70136 currently is using. When $\overline{BUSLOCK}$ is asserted high, HLDRQ will be ignored.

$\overline{BUSLOCK}$ is asserted when the $\overline{BUSLOCK}$ prefix is executed or when the μPD70136 is performing a bus operation that must not be interfered with, such as an interrupt acknowledge cycle. $\overline{BUSLOCK}$ has the same timing as the address bus A_0-A_{23} and is driven high during HLDAK and RESET.

$\overline{READY}$ (System Ready)

$\overline{READY}$ is asserted low when the external system is ready for the current bus cycle to terminate. While $\overline{READY}$ is not asserted, the μPD70136 will add TW (wait) states to the current bus cycle. The bus state in which $\overline{READY}$ is sampled low will be the last state of the cycle.

$\overline{READY}$ is used during CPU read cycles to give slow devices time to drive the D_0-D_7 inputs, and during write cycles to give slow devices enough time to finish the write operation.

$\overline{READY}$ is sampled on the rising (middle) edge of T2 and all TW states. $\overline{READY}$ is ignored during the HLDAK state. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

$\overline{BS8}$/BS16 (8-Bit Bus Size/16-Bit Bus Size)

$\overline{BS8}$/BS16 is driven low by external logic when the μPD70136 addresses a device with an 8-bit data path. If the μPD70136 operand is 16 bits wide and $\overline{BS8}$/BS16 is low, then the μPD70136 will perform two 8-bit bus cycles. The current bus cycle will handle the low byte on D_0-D_7, and the next bus cycle will handle the upper byte also on D_0-D_7. This input is ignored during HLDAK, interrupt acknowledge, and coprocessor cycles.

$\overline{BS8}$/BS16 is sampled on the rising (middle) edge of T2 or the last TW state, coincident with $\overline{READY}$. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

AEX (Address Expansion)

AEX is asserted when the expanded addressing mode is enabled. When AEX is high, the memory address space is 16M bytes (24-bit address), and when low, 1M bytes (20-bit address).

HLDRQ (Hold Request)

HLDRQ is asserted high by external logic when an external bus master (e.g., a DMA controller) wants to take over the μPD70136 bus. When HLDRQ is detected high, the μPD70136 will release the bus after the current bus operation is completed. Note that this is not necessarily the current bus cycle. The μPD70136 releases its bus by floating the address, data, and control buses. See table 1.

HLDRQ is sampled on the rising edge of each clock. It will be ignored while $\overline{\text{BUSLOCK}}$ is asserted. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

HLDAK (Hold Acknowledge)

HLDAK is asserted when the μPD70136 enters the hold acknowledge state in response to HLDRQ. Data, address, and control buses are not driven. See table 1.

INT (Interrupt Request)

INT is asserted high by external logic to notify the CPU that an external event has occurred that requires the CPU's attention. After INT has been sampled high, and if the IE (enable interrupts) bit in the PSW is high, interrupt processing will begin after the current instruction is completed.

INT is sampled on the rising edge of each clock. After being asserted high, INT must be kept high until the first INTAK cycle begins. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

$\overline{\text{NMI}}$ (Nonmaskable Interrupt Request)

$\overline{\text{NMI}}$ is asserted by external logic to notify the CPU that an external event has occurred which requires the CPU's immediate attention. When $\overline{\text{NMI}}$ is sampled low, interrupt processing will begin immediately after the current instruction is completed. A trap will be taken through vector 2. The state of the IE bit in the PSW has no effect on $\overline{\text{NMI}}$ acceptance.

$\overline{\text{NMI}}$ is sampled on the rising edge of each clock. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

Once $\overline{\text{NMI}}$ is samped low, an internal flag is set, so that a one-clock pulse meeting setup and hold times will be recognized. The flag is cleared when the $\overline{\text{NMI}}$ is accepted and can be set again immediately.

$\overline{\text{CPBUSY}}$ (Coprocessor Busy)

$\overline{\text{CPBUSY}}$ is asserted low by a coprocessor (such as μPD72291) when it is busy with an internal operation. The μPD70136 uses this pin to check the status of the coprocessor.

$\overline{\text{CPBUSY}}$ is sampled on the falling edge of each clock. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

$\overline{\text{CPERR}}$ (Coprocessor Error)

$\overline{\text{CPERR}}$ is asserted low by a coprocessor to notify the μPD70136 of an error.

$\overline{\text{CPERR}}$ is sampled on the falling edge of each clock. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

CPREQ (Coprocessor Request)

CPREQ is asserted high by a coprocessor to request the μPD70136 to run a memory operation for the coprocessor.

CPREQ is sampled on the falling edge of each clock. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

RESET (Reset)

RESET is asserted high when external logic needs to initialize the μPD70136; for instance, after power-up. When RESET is asserted for at least 6 clock periods, the μPD70136 will abort any current bus cycles and initialize the registers as shown in table 4.

Table 4. Register Initialization by Reset

Register	Offset Value
PFP	0000H
PC	0000H
PS	FFFFH
SS	0000H
DS0	0000H
DS1	0000H

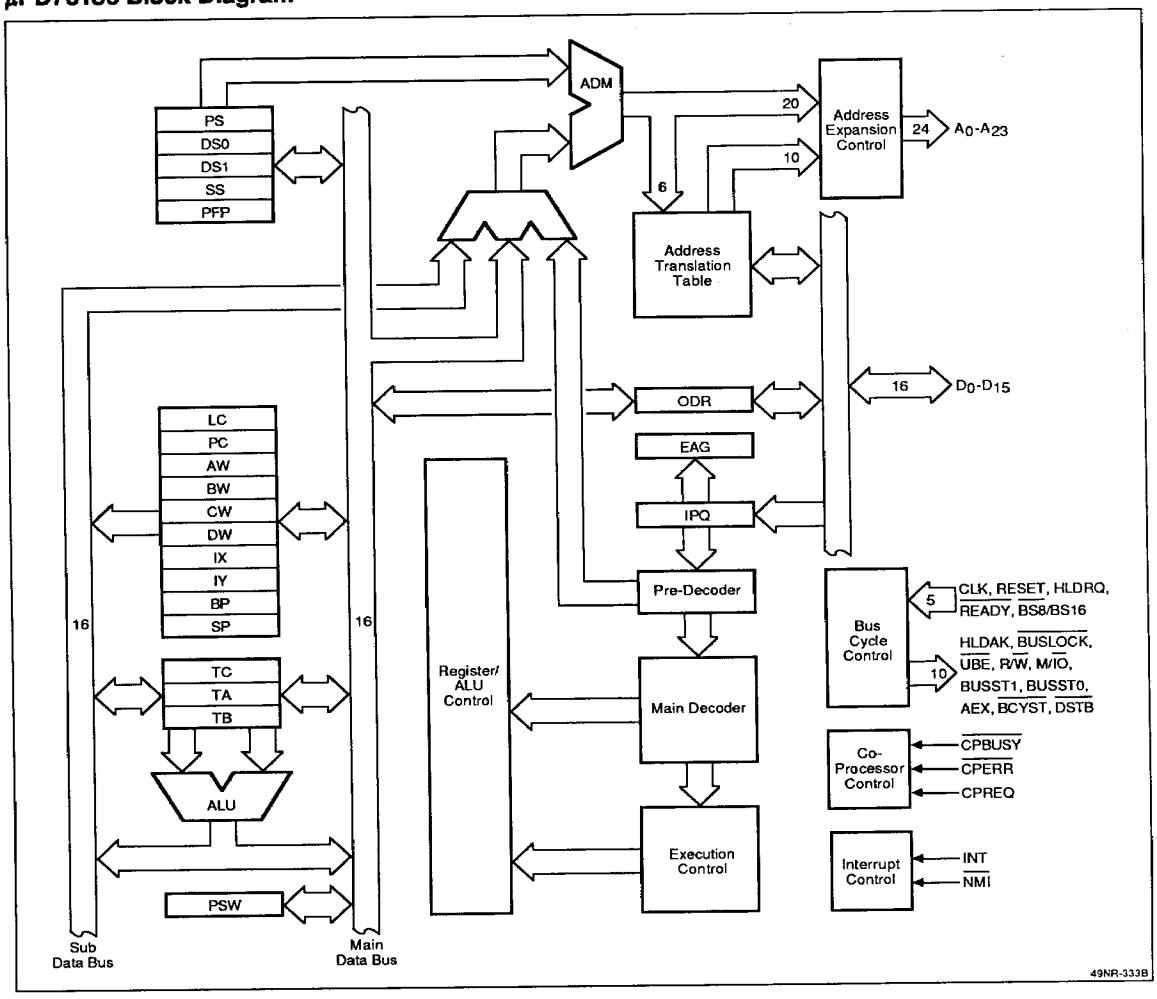
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSW	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0

Prefetch Queue	Cleared
Address Mode	Normal Address Mode
Other Registers	Undefined (if power has just been turned on) Unchanged (if power on, but RESET is asserted)

Refer to table 1 for the state of the μPD70136 outputs during reset. When RESET is deasserted low, the μPD70136 will begin fetching from address 0FFFF0H. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

3e

μPD70136 Block Diagram



ELECTRICAL SPECIFICATIONS

Absolute Maximum Ratings

T_A = +25°C

Power supply voltage, V_{DD}	−0.5 to +7.0 V
Input voltage, V_I	−0.5 V to V_{DD} + 0.3 V
CLK input voltage, V_K	−0.5 V to V_{DD} + 1.0 V
Output voltage, V_O	−0.5 V to V_{DD} + 0.3 V
Operating temperature, T_{OPT}	−10 to +70°C
Storage temperature, T_{STG}	−65 to +150°C

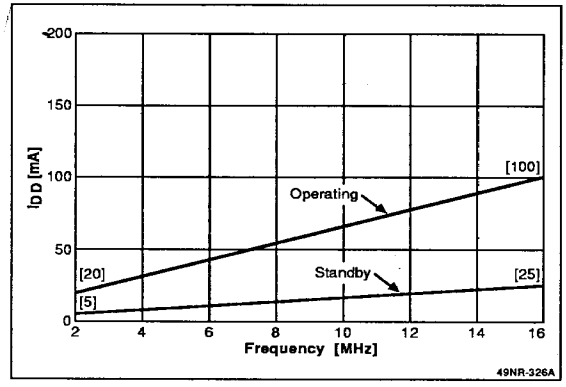
Exposure to Absolute Maximum Ratings for extended periods may affect device reliability; exceeding the ratings could cause permanent damage.

Capacitance

T_A = +25°C, V_{DD} = 0 V

Parameter	Symbol	Max	Unit	Conditions
Input capacitance	C_I	15	pF	fc = 1 MHz; unmeasured pins returned to 0 V.
I/O capacitance	C_{IO}	15	pF	

Typical Supply Current vs Clock Frequency



3e

DC Characteristics

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

Parameter	Symbol	Min	Typ	Max	Unit	Conditions
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}	0.8 V_{DD}		V_{DD} + 0.5	V	
CLK input voltage low	V_{KL}	−0.5		0.6	V	
Output voltage high	V_{OH}	0.7 V_{DD}			V	I_{OH} −400 mA
Output voltage low	V_{OL}			0.45	V	I_{OL} = 2.5 mA
Input leakage current high	I_{LIH}			−10	μA	V_I = V_{DD}
Input leakage current low	I_{LIL}			10	μA	V_I = 0 V
Output leakage current high	I_{LOH}			10	μA	V_O = V_{DD}
Output leakage current low	I_{LOL}			−10	μA	V_O = 0 V
Supply current	I_{DD} (see graph)	16 MHz	100	150	mA	Normal operation
			25	35	mA	Standby mode
		12.5 MHz	75	110	mA	Normal operation
			20	30	mA	Standby mode
*Stop mode current is not a function of CPU clock frequency		*		200	μA	Stop mode

AC Characteristics

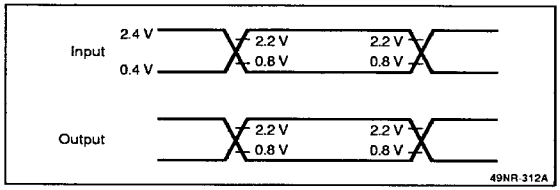
T_A = −10 to +70°C; V_{DD} = 5 V ±10%; C_L = 100 pF

Parameter	Symbol	12.5-MHz Limits		16 MHz-Limits		Unit
		Min	Max	Min	Max	
Clock period	t_{CYK}	80	500	62.5	500	ns
Clock high-level width	t_{KKH}	35		25		ns
Clock low-level width	t_{KKL}	35		25		ns
Clock rise time (1.7 V → 3.0 V)	t_{KR}		5		5	ns
Clock fall time (3.0 V → 1.7 V)	t_{KF}		5		5	ns
Reset delay time (V_{DD} valid)	t_{DVRST}	1		1		μs
Reset setup time (CLK ↓)	t_{SRSTK}	10		10		ns
Reset hold time (CLK ↓)	t_{HKRST}	15		15		ns
Reset high time	t_{WRSTH}	6		6		t_{CYK}
CLK ↓ → $\overline{BCYST}$ delay	t_{DKBC}	5	40	5	40	ns
$\overline{BCYST}$ low-level width	t_{BCBCL}	t_{CYK} − 10		t_{CYK} − 10		ns
$\overline{BCYST}$ high-level width	t_{BCBCH}	t_{CYK} (n + 1) −10		t_{CYK} (n + 1) −10		ns
CLK ↓ → address delay	t_{DKA}	5	40	5	40	ns
CLK ↓ → status delay	t_{DKST}	5	40	5	40	ns
$\overline{READY}$ setup time (CLK ↑)	t_{SRYK}	7		7		ns
$\overline{READY}$ hold time (CLK ↑)	t_{HKRY}	15		15		ns
CLK ↑ → data output delay	t_{DKD}	5	40	5	40	ns
Floating delay	t_{FK}	0	50	0	50	ns
CLK ↑ → $\overline{DSTB}$ delay	t_{DKDS}	5	40	5	40	ns
Address/status output → $\overline{DSTB}$ ↓ delay time	t_{DADSL}	$t_{KKL} + t_{KR} - 15$		$t_{KKL} + t_{KR} - 15$		ns
$\overline{DSTB}$ ↑ address/status hold time	t_{HDSHA}	$t_{KKL} + t_{KR} - 15$		$t_{KKL} + t_{KR} - 15$		ns
$\overline{DSTB}$ low-level width	t_{DSDSL}	t_{CYK} (n+1) − 10		t_{CYK} (n+1) − 10		ns
$\overline{DSTB}$ high-level width	t_{DSDSH}	$t_{KKL} + t_{KR} - 10$		$t_{KKL} + t_{KR} - 10$		ns
CLK ↓ → $\overline{DSTB}$ ↑ delay for read cycle	t_{DKDSRD}	5	40	5	40	ns
Address/status output → data delay time	t_{DAD}	$t_{KKL} + t_{KR} - 15$		$t_{KKL} + t_{KR} - 15$		ns
$\overline{DSTB}$ ↑ → data output delay time	t_{DDSHD}	$t_{KKL} + t_{KR} - 15$		$t_{KKL} + t_{KR} - 15$		ns
Data setup time (CLK ↓)	t_{SDK}	7		7		ns
Data hold time (CLK ↓)	t_{HKD}	10		10		ns
Data hold time ($\overline{DSTB}$ ↑)	t_{HDSD}	0		0		ns
Data hold time (R/W ↓)	t_{HRWD}	0		0		ns
$\overline{BS8}$/BS16 setup time	t_{SBSK}	7		7		ns
$\overline{BS8}$/BS16 hold time	t_{HKBS}	15		15		ns
HLDRQ setup time (CLK ↑)	t_{SHQK}	7		7		ns
HLDRQ hold time (CLK ↑)	t_{HKHQ}	15		15		ns
CLK ↑ → HLDAK delay time	t_{DKHA}	5	40	5	40	ns
Output float → HLDAK delay	t_{DFHA}	$t_{KKL} + t_{KR} - 15$		$t_{KKL} + t_{KR} - 15$		ns
$\overline{NMI}$, INT, $\overline{CPBUSY}$ setup time (CLK ↓)	t_{SIK}	10		10		ns
$\overline{NMI}$, INT, $\overline{CPBUSY}$ setup time (CLK ↓)	t_{HKI}	10		10		ns

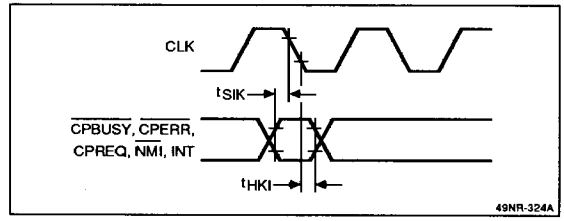
Note: 'n' means number of wait cycles to be inserted into bus cycle

Timing Waveforms

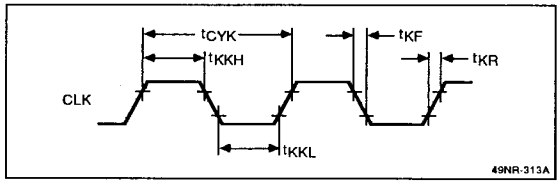
Input/Output Voltage Reference Levels



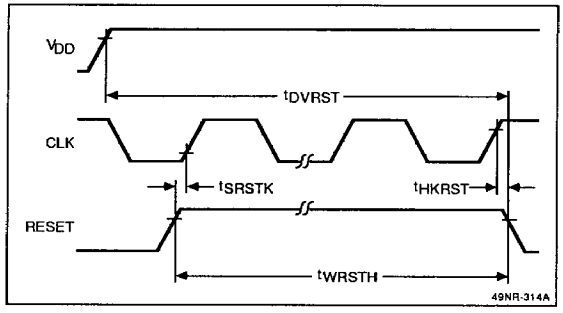
Input Setup/Hold Time



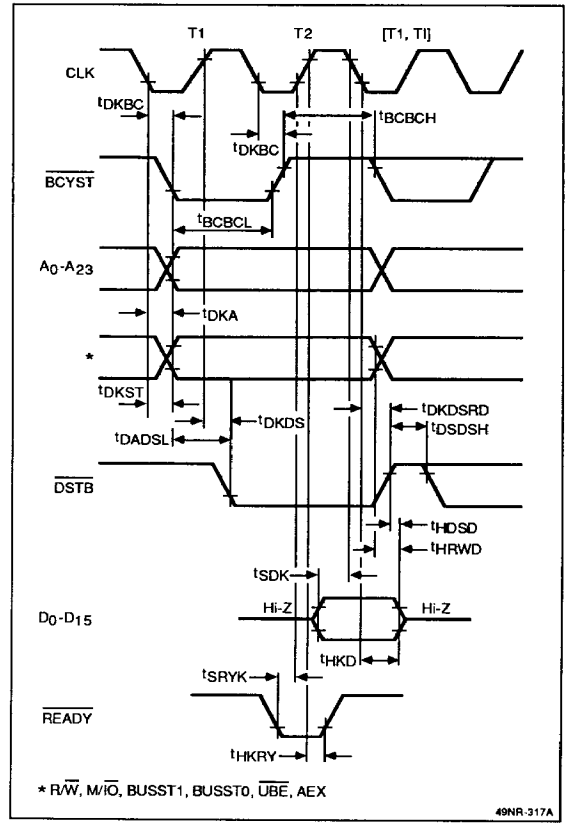
Clock Input



Reset

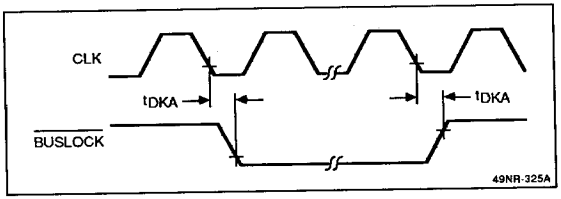


Basic Read Cycle (0 WAIT)

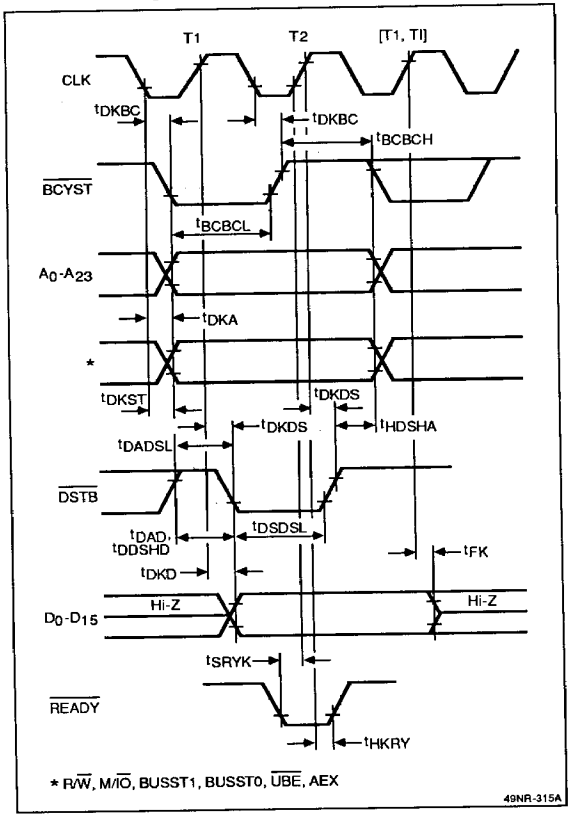


3e

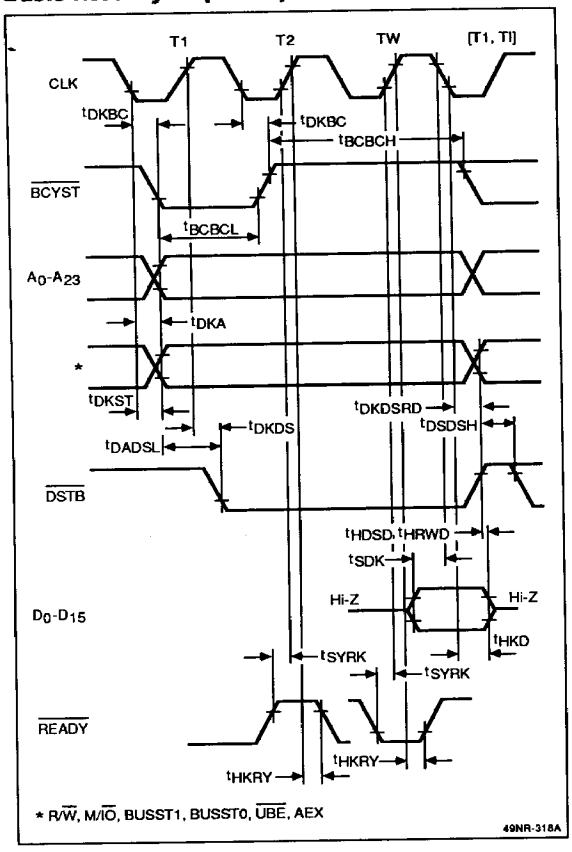
Bus Lock



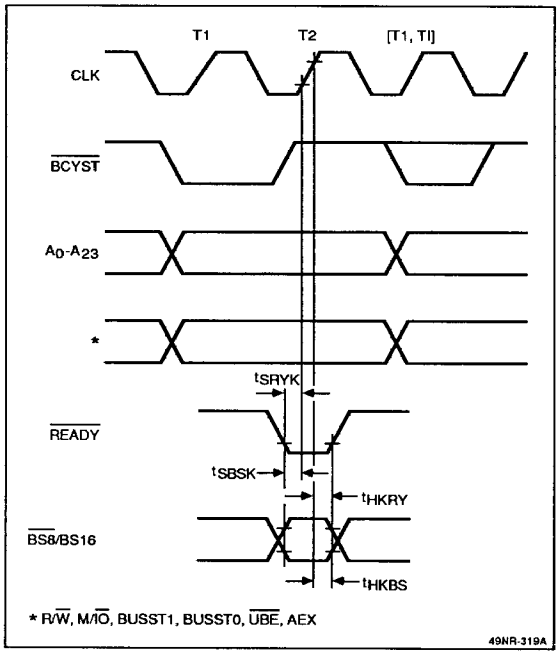
Basic Write Cycle (0 WAIT)



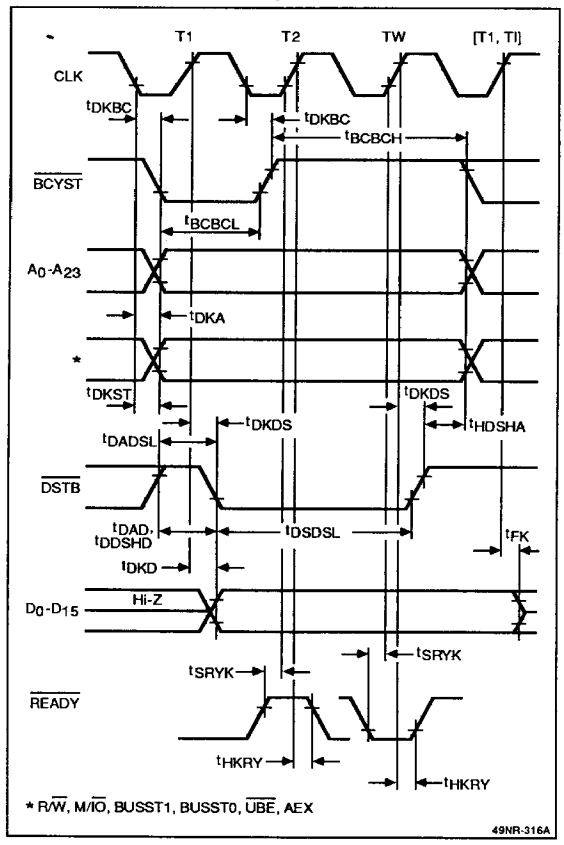
Basic Read Cycle (1 WAIT)



Basic Write Cycle (1 WAIT)

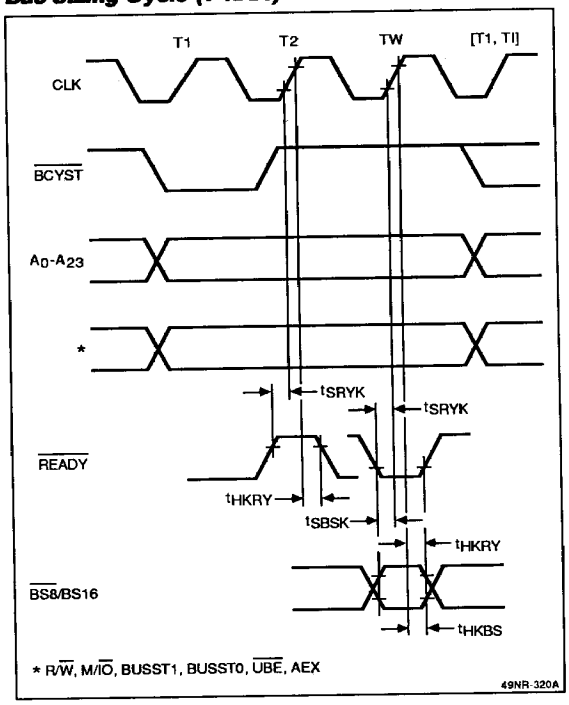


Bus Sizing Cycle (0 WAIT)

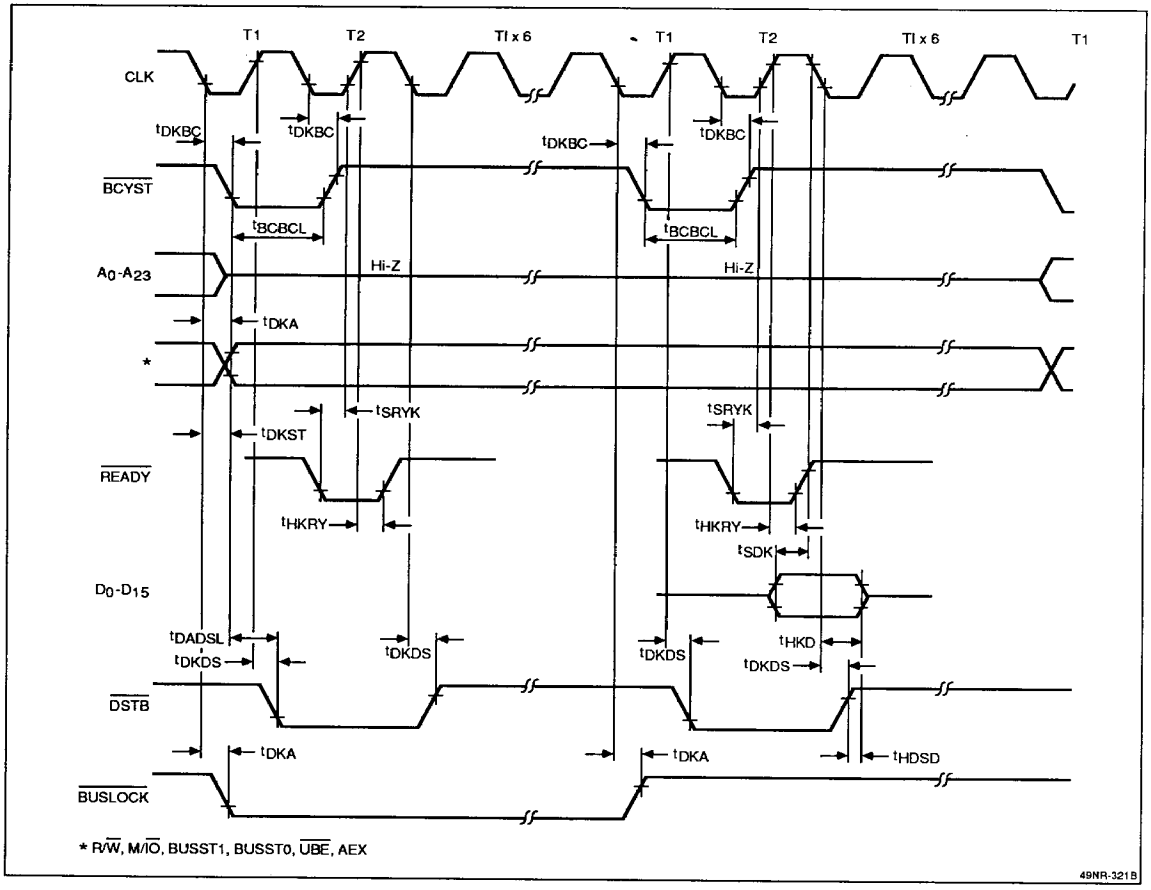


3e

Bus Sizing Cycle (1 WAIT)

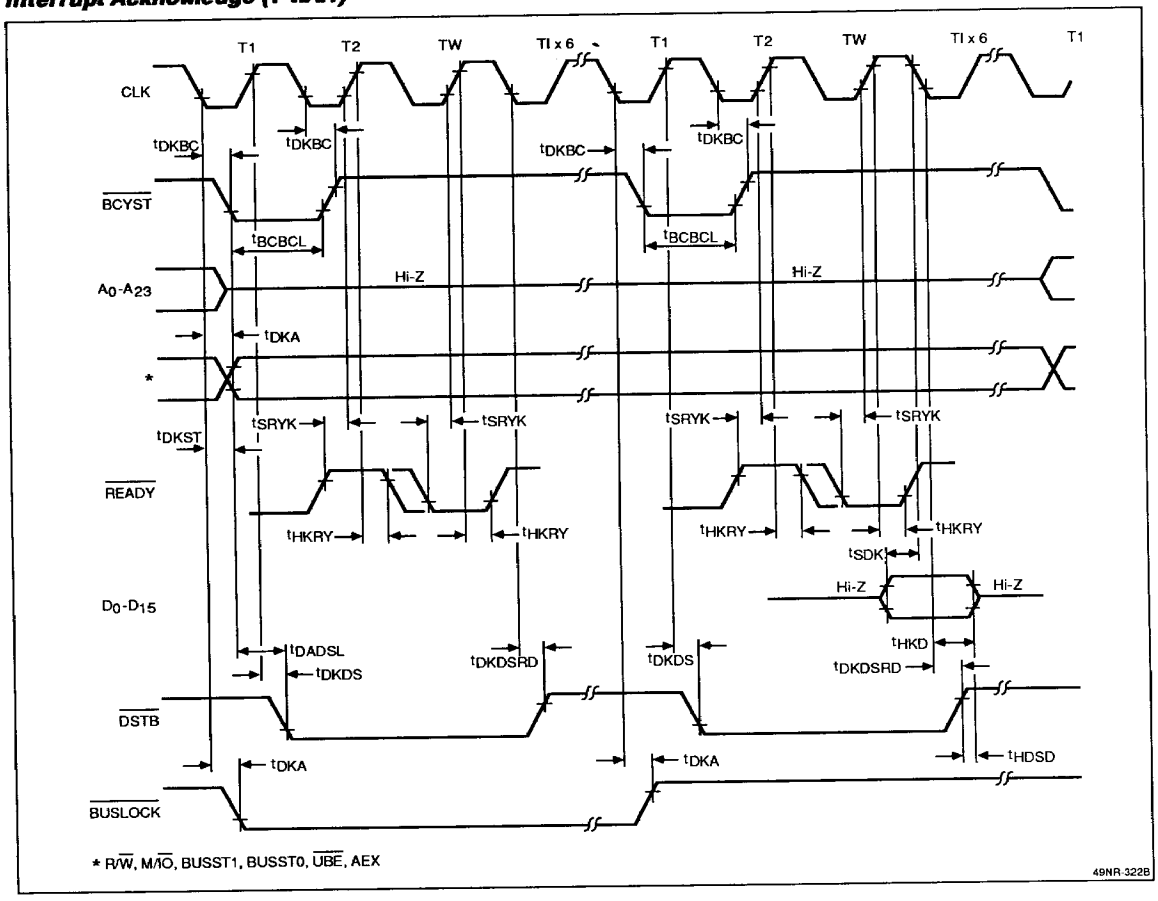


Interrupt Acknowledge (0 WAIT)

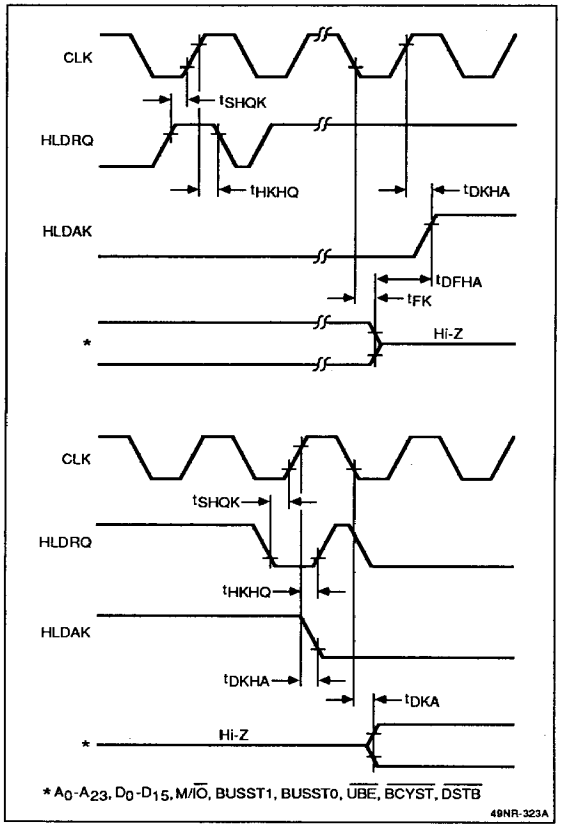


3e

Interrupt Acknowledge (1 WAIT)



Bus Hold



FUNCTIONAL DESCRIPTION

Architecture

A unique hardware architecture feature of the μPD70136 is that there is no microcode. Instruction decode and data path control are implemented using logic and small independent state machines. This greatly enhances the speed with which instructions can be executed, in the same way that programs written in assembly language can be faster and more efficient than high-level language code. The μPD70136 is four times faster than the μPD70116.

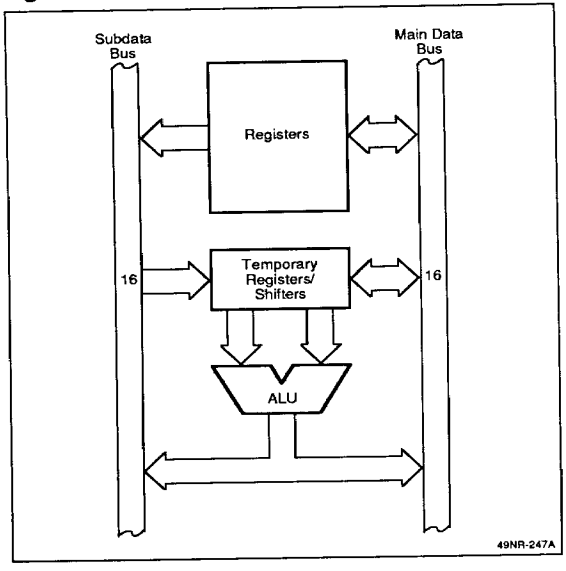
The μPD70136 hardware comprises the execution unit, a bus interface, and the address generator. See the μPD70136 Block Diagram.

Execution Unit

The μPD70136 execution unit consists of a register file, an ALU, instruction decode, and execution control logic.

Besides the hardware control logic, the most significant feature of the execution unit is a dual-bus internal data path. See figure 1. The ALU and many registers are dual-ported, with a data bus on each port. This allows two operands to be transferred in one clock cycle instead of two. Performance is improved by as much as 30 percent using the dual data bus concept.

Figure 1. Dual Data Buses



Register File. There are 12 registers in the internal RAM. Four are temporary registers used in the execution of certain instructions (LC, TA, TB, TC). The other eight are general-purpose registers (AW, BW, CW, DW, IX, IY, BP, SP) and either contain operand data or point to operand data in memory.

The temporary registers speed up instruction execution by serving as scratch pad registers during complex operations.

The loop counter (LC) is used during primitive block transfer operations. It contains the count value. It is also used as a shift counter for multiple-bit shift and rotate instructions.

Temporary registers TA, TB, and TC are the inputs to the ALU. They are used as temporary registers/shifters during multiply, divide, shift/rotate, and BCD rotate operations.

ALU. The ALU consists of a complete adder and logical operation unit. It executes arithmetic (ADD, SUB, MUL, DIV, INC, DEC, NEG, etc.) and logical (TEST, AND, OR, XOR, NOT, SET1, CLR1, etc.) instructions.

Data Path Control Logic. This logic comprises the main instruction decoder and the execution control blocks. Its purpose is to decide what operations must be done and to schedule them. It transfers operands as needed and controls the ALU. State machines are used to implement long, complex instructions.

Bus Interface

The bus interface comprises bus control logic, an operand data register (ODR), an 8-byte instruction prefetch queue (IPQ), and an effective address generator.

The bus control state machines implement the μPD70136 bus interface. To allow the bus machine to run independent of the execution unit, an operand data register is used. During a CPU write cycle, the write data is placed in the ODR and the execution of the next instruction proceeds without waiting. The bus interface finishes the write cycle when the bus is available. During a read cycle, if the operand requires two bus cycles (as in a read from an odd address), the full 16-bit value is assembled in the ODR, one byte at a time.

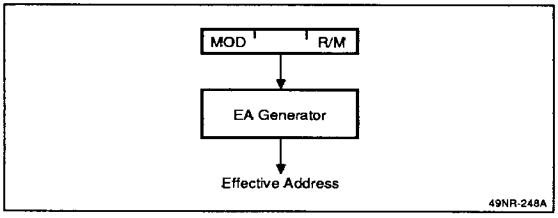
Instruction Prefetching. The μPD70136 is a pipelined machine. To keep the pipeline running efficiently, it should be kept full of instructions in various stages of execution. Instructions are fetched before they are needed and placed in the IPQ. Data in the IPQ is broken out by the pre-decoder logic to determine what addressing modes will be used and what CPU resources will be required to execute the prefetched instruction. To keep

the 8-byte IPQ full, the bus control logic will schedule an instruction prefetch cycle whenever there are at least 2 unused bytes in the IPQ.

The IPQ is cleared whenever a control transfer instruction (any branch, call, return, or break) is executed. This is done because a different instruction stream will be used following a control transfer, and the IPQ will then contain instruction data that will never be used. When this happens, the μPD70136's pipeline will empty out, hampering performance. To maximize performance, the number of control transfers should be minimized.

Effective Address Generator. The EAG logic computes a 16-bit effective address for each operand, which is an offset into one of the four segments. This effective address is passed on to the address modifier adder. The EAG decodes the first byte(s) of each instruction to determine the addressing mode and initiates any bus cycles required to fetch pointers/offsets from memory. Effective addresses are calculated in a maximum of 1 clock period, compared to 5 to 12 clocks for a microprogrammed machine. See figure 2.

Figure 2. Effective Address Generator



Address Generator

The address generator comprises the address register file, the address modifier (ADM), the address translation table, and the needed control logic.

The registers in the address register file are PS, SS, DS0, DS1, PC, and PFP. The ADM is a dedicated adder that adds one of the segment registers to the effective address to produce the 20-bit normal address. The ADM also increments the prefetch pointer. If expanded addressing is enabled, the address translation table is accessed to map the 20-bit address into a 24-bit expanded address.

For instruction stream data, addresses are generated differently. The prefetch pointer contains a 16-bit offset into the PS segment that points to the next instruction word to be prefetched. The program counter contains an offset into the PS segment that points to the instruction that is currently being executed. As part of all control transfers, the PFP is set to the same value as the PC.

ADDRESSING MECHANISM

The μPD70136 is completely compatible with the μPD70108/116 in its addressing modes, and in the way that addresses are computed. It offers a method of expanding the memory address space to 16M bytes.

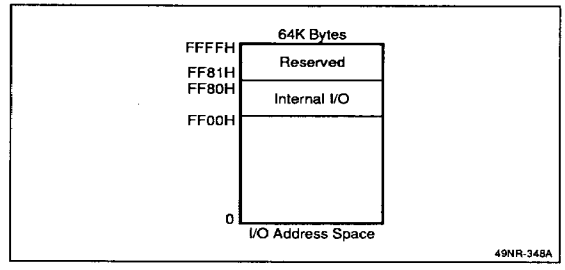
The I/O space is 64K bytes (16-bit address). The normal memory address space is 1M byte (20-bit address), and the expanded address space is 16M bytes (24-bit address). Expanded addressing is enabled or disabled using the BRKXA and RETXA instructions.

3e

The memory space is accessed when an instruction uses a memory addressing mode. Memory addresses are calculated as described below. The I/O space can only be accessed through the IN, OUT, INM, and OUTM instruction.

Certain areas of the μPD70136 address (physical for normal mode, and logical for expanded addressing mode) spaces are reserved. These areas are shown in figures 3 and 4. Memory addresses 0-3FCH are used for the Interrupt Vector table located in the Interrupt Operation section. Memory addresses FFFF0H-FFFFFH must contain a branch to boot code; PC, PFP, and PS are initialized at RESET to point to this area. I/O addresses FF00H-FF80H are reserved for the address translation registers.

Figure 3. I/O Address Space

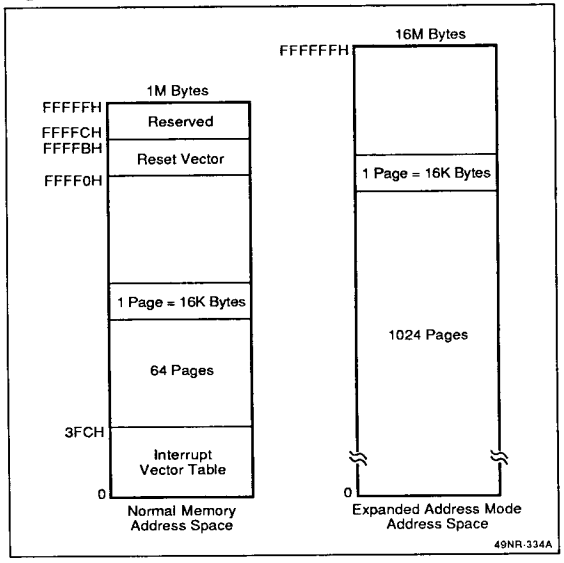


I/O Addresses

I/O addresses are always taken from 8-bit immediate data or the DW register. DW is always used as a direct pointer into the I/O address space. If I/O operations require the use of other more complex addressing modes, the I/O devices must be placed in the memory address space (using memory-mapped I/O techniques). For memory-mapped I/O devices, there are no restrictions on instruction or addressing mode usage. However,

the μPD70136 will not automatically insert 6 clock cycles after memory-mapped I/O operations; external logic must provide the needed I/O device recovery time.

Figure 4. Address Space

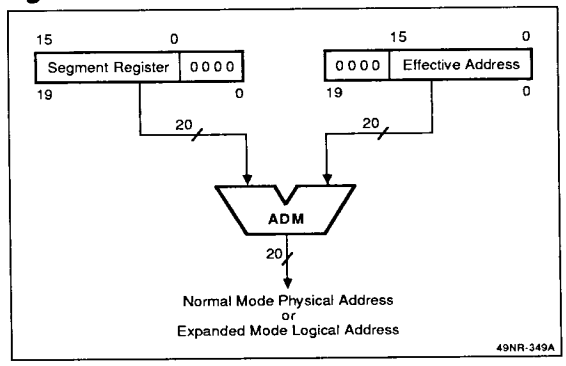


Normal Memory Addresses

The μPD70136 is a 16-bit device with 16-bit registers. To allow a memory address space larger than 64K bytes, memory segmentation is used. The 1M-byte memory address space is divided into 64K byte segments. Up to four segments can be in use at any given time. The base addresses of the four active segments (program segment, stack segment, data segment 0, and data segment 1) are contained in four 16-bit segment registers (PS, SS, DS0, and DS1, respectively). The 16-bit value in each register is the upper 16 bits of the 20-bit memory address. Thus, segments always start on 16-byte boundaries.

As described above, the μPD70136 hardware generates a 16-bit effective address for each memory operation. This effective address is an offset into one of the four active segments. The actual 20-bit memory address is computed by adding the EA to the segment register value expanded with zeros to 20 bits. Figure 5 shows this process.

Figure 5. 20-Bit Address



If normal addressing mode is enabled, then this 20-bit result is presented on the address bus during the bus cycle. If expanded addressing mode is enabled, this address is used as a logical address.

Expanded Addresses

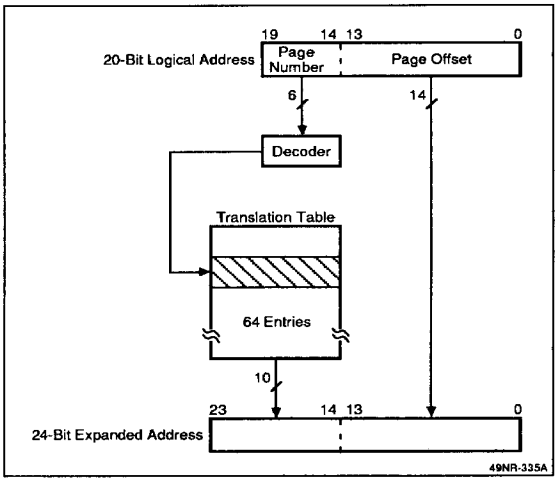
In the expanded addressing mode, the memory space is divided into 1024 pages. Each page is 16K bytes. Each page of the normal 20-bit address space is mapped to a page in the expanded address space using a 64-entry address translation table. The table is made up of 64 page registers that reside in the I/O space (figure 4).

The programming model of this mode is the same as for the normal mode. Address expansion is a layer added to the normal mode that is transparent to executing code. The program still sees a 20-bit contiguous logical memory address space, but the hardware sees 64 pages mapped into a set of 1024 physical pages.

The I/O space is not affected by the expanded addressing mode.

The address translation mechanism is shown in figure 6. The upper 6 bits of the logical 20-bit address select one of the entries in the address translation table, which supplies a 10-bit value. This value is substituted for the original 6 bits in the normal address to create a 24-bit expanded address.

Figure 6. Address Translation Mechanism



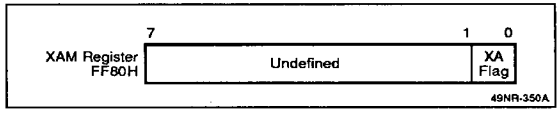
Address Expansion Registers

These are the page and XAM registers. Word IN and OUT instructions are used to access these registers. The table below shows page register usage and I/O addresses. The page registers contain the 10-bit physical page base address.

A_{19}-A_{14} Logical Address	PGR Selected	PGR I/O Address
0	PGR1	FF00
1	PGR2	FF02
2	PGR3	FF04
3	PGR4	FF06
:	:	:
63	PGR64	FF7E

The XAM register (figure 7) is a read-only status flag that indicates whether expanded addressing is enabled. Unused data bits in the XAM register are undefined. Expanded addressing must be disabled before accessing any of the page registers. I/O operations to these internal registers are not passed to the bus interface and will not be seen by external logic.

Figure 7. XAM Register

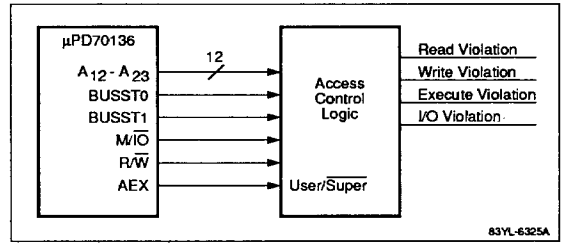


Memory Protection Mechanism

The μPD70136 expanded mode provides a harware memory protection mechanism (figure 8) that does not sacrifice software compatibility with existing μPD8088/8086 or V20/V30/V40/V50 programs. In expanded mode, the XAM and PGR registers cannot be accessed. This provides simple two-level protection.

A supervisory system task running in normal mode can set up restricted memory spaces for less privileged user tasks by programming the PGR registers and then starting up the user task in expanded mode. The user task will not be able to change its memory map to access privileged memory areas. External access control logic can monitor the AEX output to determine at which privilege level the CPU is currently running (AEX = 0 is supervisor mode, AEX = 1 is user mode) and permit or prevent each bus cycle, thereby providing additional memory and I/O protection. This scheme provides the basic hardware protection needed for most operating systems without forgoing full software compatibility.

Figure 8. Expanded Mode Protection Mechanism



OPERAND ADDRESSING MODES

For operand addressing, the μPD70136 offers 9 modes:

- Register
- Immediate
- Direct
- Register indirect
- Indexed
- Based
- Based index
- Bit
- Autoincrement/autodecrement

Register

The operand is in a μPD70136 register pointed to by the instruction.

Immediate

The operand is in the instruction stream following the opcode of the instruction. This data will have been prefetched. Immediate data uses the μPD70136 pipeline efficiently.

Direct

Immediate data in the instruction stream points directly to the operand. This data can be a 16-bit effective address or a 4-bit bit field length.

Register Indirect

A 16-bit register (IX, IY, or BW) contains a 16-bit effective address.

Indexed

One or two bytes of immediate data are treated as a signed displacement that is added to the contents of a 16-bit index register (IX or IY) to obtain a 16-bit effective address.

Based

One or two bytes of immediate data are treated as a signed displacement that is added to the contents of a 16-bit base register (BP or BW) to form a 16-bit effective address.

Based Indexed

One or two bytes of immediate data are treated as a signed displacement that is added to two 16-bit registers (one of BP or BW with one of IX or IY) to form the effective address. This mode is useful for array addressing.

Bit

Used with NOT1, SET1, CLR1, or TEST1. A 4-bit immediate data value is used to select a bit in a 16-bit operand. For 8-bit operands, only 3 bits are used.

Autoincrement/Autodecrement

Some interactive operations (such as MOVBK or INS) will automatically increment or decrement index registers after each iteration. Specifically, IX is used in addressing a source pointer, and/or IY is used in addressing a destination pointer. After the operation, both will be incremented or decremented (according to the PSW DIR control flag) to point to the next operand in the array.

INSTRUCTION ADDRESSING MODES

These modes are basically the same as the operand addressing modes, but the PC is always used as the register. The seven modes are used in control transfer instructions:

- Direct
- Relative
- Register
- Register indirect
- Indexed
- Based
- Based indexed

Direct

Four bytes of immediate data are taken as an absolute address and loaded directly into the PS and PC (and PFP).

Relative

One or two bytes of immediate data are a signed displacement that is added to the contents of the PC and then placed in the PC (and PFP). This mode is used to create position-independent code.

Register

The register selected by the instruction (AW, BW, etc.) contains an effective address, which is loaded into the PC (and PFP).

Register Indirect

An index register (IX, IY, or BW) points to a memory location that contains an effective address (short pointer) or a segment register value and an effective address (far pointer). This effective address is read from memory and loaded into the PS and/or PC (and PFP).

Indexed

One or two bytes of immediate data are a signed displacement that is added to the contents of a 16-bit index register (IX or IY) to form an effective address. This address is used to fetch another effective address, which is loaded into the PC (and PFP).

Based

One or two bytes of immediate data are a signed displacement that is added to the contents of a 16-bit base register (BP or BW) to form an effective address. This address is used to fetch another effective address from memory, which is then loaded into the PC (and PFP).

Based Indexed

One or two bytes of immediate data are a signed displacement that is added to the contents of two 16-bit register (one of BP or BW with one of IX or IY) to form an

effective address. This address is used to fetch another effective address from memory, which is then loaded into the PC (and PFP).

REGISTER CONFIGURATION

Program Counter (PC)

The PC is a 16-bit register that contains the effective address of the instruction that is currently being executed. The PC is incremented each time the instruction decoder accepts a new instruction from the prefetch queue. The PC is loaded with a new value during execution of a branch, call, return, or break instruction and during interrupt processing.

Segment Registers (PS, SS, DS0, DS1)

There are four segment registers, each of which contains the upper 16 bits of the base address of a 64K logical segment. Since logical segments reside on 16-byte boundaries, the lower 4 bits of the base address are always 0. Normal 20-bit memory addresses are formed by adding the 16-bit effective address to the base address of one of the segments. When performing this operation, certain types of effective addresses will be paired with specific 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

Program instructions will always be fetched from the program segment. Whenever the IY index register is used to address an operand, the DS1 segment register will be used. DS0 is usually used with IX. Stack operations using the SP will always use the stack segment. For other effective addresses, the preceding table shows the default segment used, but another segment may be selected by using a segment override prefix instruction.

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

The four 16-bit general-purpose registers can be accessed as 16-bit or 8-bit quantities. When the AW, BW, CW, or DW designations are used, the register will be 16 bits. When AL, AH, BL, BH, CL, CH, DL, or DH is used, the register will be 8 bits. AL will be the low byte of AW, and AH the high byte, and so on.

Some operations require the use of specific registers:

- AW Word multiplication/division, word I/O, data conversion
- AL Byte multiplication/division, byte I/O, BCD rotation, data conversion, translation
- AH Byte multiplication/division
- BW Translation
- CW Shift instructions, rotation instructions, BCD operations
- DW Word multiplication/division, indirect addressing I/O

Pointer (SP, BP) and Index (IX, IY) Registers

These registers are used as base pointers and index registers when based, indexed, or based-indexed addressing modes are used. They may also be used as general-purpose registers in data transfer, arithmetic, and logical instructions. They can only be accessed as 16-bit registers.

Some operations use these registers in specific ways:

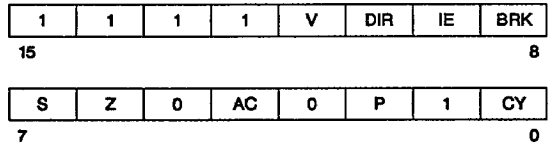
- SP Stack operations
- IX Source pointer for block transfer, bit field, and BCD string operations
- IY Destination pointer for block transfer, bit field, and BCD string operations

Program Status Word (PSW)

The program status word reflects the status of the CPU with six status flags, and affects the operation of the CPU through three control flags:

Status Flags		Control Flags	
V	Overflow	DIR	Direction
S	Sign	IE	Interrupt enable
Z	Zero	BRK	Break
AC	Auxiliary carry		
P	Parity		
CY	Carry		

The PSW cannot be accessed directly as a 16-bit register. Specific instructions are used to set/reset the control flags. When the PSW is pushed on the stack (as during interrupt processing), the following image is used.



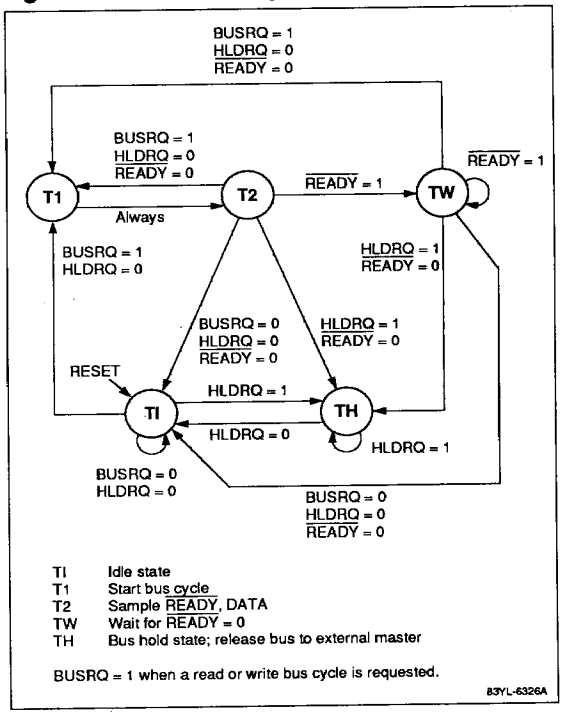
BUS OPERATION OVERVIEW

The μPD70136 uses a synchronous bus interface. The CLK input supplies the μPD70136 with a clock. All

μPD70136 bus timings and instruction execution clock counts are specified relative to this clock. Bus cycles start on the falling edge of CLK. Each bus cycle is two clock periods long, and may be extended by adding wait states.

Figure 9 is the state diagram of the bus control state machine. The first state of every bus cycle is T1, followed immediately by T2. $\overline{\text{READY}}$ is sampled on the rising (middle) edge of T2. If $\overline{\text{READY}}$ is not asserted, then the next bus state will be a TW wait state. TWs will be inserted until $\overline{\text{READY}}$ is sampled low, after which the bus cycle will finish. The dynamic bus sizing input, $\overline{\text{BS8}}$/BS16, is sampled at the same time as $\overline{\text{READY}}$.

Figure 9. Bus State Diagram



Address and bus status are output after the leading edge of T1 and are maintained until after the cycle is completed. A strobe, $\overline{\text{BCYST}}$, is asserted during T1 to indicate the beginning of a bus cycle. $\overline{\text{BCYST}}$ is output following the leading edge of T1 and is deasserted after the leading edge of T2.

Write data is driven on D_0-D_{15} following the rising (middle) edge of T1 and is maintained until after the rising edge of the cycle following T2 or the last TW. The read data is sampled on the trailing edge of T2 or the last TW state. A strobe ($\overline{\text{DSTB}}$) gives the status of the μPD70136 data bus. $\overline{\text{DSTB}}$ is asserted after the rising edge (middle) of T1. $\overline{\text{DSTB}}$ is deasserted after the rising edge of T2 or the last TW for a write cycle, and after the trailing edge of T2 or the last TW for a read cycle.

I/O cycles are identical to memory cycles except for the encoding of the bus status lines. However, six idle states are inserted after every I/O bus cycle to provide a recovery time for slow I/O devices.

Dynamic Bus Sizing

The μPD70136 supports dynamic bus sizing. On a cycle by cycle basis, the width of the data bus can be changed from 16 to 8 bits. This simplifies the connection of 8-bit I/O devices that may have internal registers at consecutive byte addresses. Other 16-bit CPUs require two ROMs for startup code, but the μPD70136 dynamic bus sizing makes it possible to use a single 8-bit wide ROM.

External logic requests an 8-bit data path by driving $\overline{\text{BS8}}$/BS16 low in time for the μPD70136 to sample it on the rising edge of T2 (or TW). The μPD70136 will perform an additional bus cycle if needed to finish the operation in byte-wide pieces.

Referring to tables 5 and 6, if the bus operation is 8 bits wide, no further bus cycles will occur. For a read cycle, the data will be sampled on D_0-D_7. For a write cycle to an even address, data will be driven on D_0-D_7. On all byte writes to an odd address, the μPD70136 will put the byte data on both upper and lower data buses; the write data will be on D_0-D_7 as well as D_8-D_{15}.

If the bus operation is 16-bit, then two bus cycles will be required. The first one, in which $\overline{\text{BS8}}$/BS16 is sampled low, will handle the low byte. The second cycle will take the form of a byte read or write using D_0-D_7.

Bus Cycle Types

The 11 different types of μPD70136 bus cycles are classified as read, write, and acknowledge cycles.

Read Cycles

The read cycles are memory, I/O, coprocessor, data reads, and instruction fetch. All have the general timing described above. Coprocessor reads are used to access the internal registers of a coprocessor. Coprocessor data reads are used to transfer data from memory to an internal coprocessor register.

Table 5. Write Cycle Bus Sizing

					16-Bit Bus ($\overline{BS8}$/BS16 = 1)		8-Bit Bus ($\overline{BS8}$/BS16 = 0)	
Type	Address	A_0	$\overline{UBE}$	Cycle	D_{15}-D_8	D_7-D_0	D_{15}-D_8	D_7-D_0
Byte	Even	0	1	1st	Invalid	Byte	Invalid	Byte
	Odd	1	0	1st	Byte	Byte	Byte	Byte
Word	Even	0	1	1st	Upper	Lower	Upper	Lower
		1	0	2nd	Not needed for 16-bit bus		Upper	Upper
	Odd	1	0	1st	Lower	Lower	Lower	Lower
		0	1	2nd	Upper	Upper	Invalid	Upper

Table 6. Read Cycle Bus Sizing

					16-Bit Bus ($\overline{BS8}$/BS16 = 1)		8-Bit Bus ($\overline{BS8}$/BS16 = 0)	
Type	Address	A_0	$\overline{UBE}$	Cycle	D_{15}-D_8	D_7-D_0	D_{15}-D_8	D_7-D_0
Byte	Even	0	1	1st	Not used	Byte	Not used	Byte
	Odd	1	0	1st	Byte	Not used	Not used	Byte
Word	Even	0	1	1st	Upper	Lower	Not used	Lower
		1	0	2nd	Not needed for 16-bit bus		Not used	Upper
	Odd	1	0	1st	Lower	Lower	Not used	Lower
		0	1	2nd	Not used	Upper	Not used	Upper

3e

I/O and memory reads are used to transfer data to the μPD70136 from an I/O device or a memory location, respectively. Instruction fetches are used to fill the μPD70136's 8-byte instruction queue from the memory space.

Write Cycles

There are four types of write cycles. Memory writes transfer data from the μPD70136 to a memory location. I/O writes transfer data from the μPD70136 to an I/O device. Coprocessor data writes transfer data from the coprocessor to a memory location. Coprocessor writes transfer data from the μPD70136 directly to a coprocessor internal register.

Interrupt Acknowledge Cycle

The interrupt acknowledge operation takes two consecutive INTAK bus cycles. The first cycle is used to freeze the state of an external interrupt controller, such as the μPD71059. The second INTAK bus cycle reads an 8-bit vector number on D_0-D_7 supplied by the μPD71059. This vector number is used to index into the interrupt vector table to select an interrupt handler.

Halt Acknowledge Cycle

When a HALT instruction is executed, a halt acknowledge cycle is issued to notify external logic that the μPD70136 is entering standby mode. This cycle is always two clocks long; $\overline{READY}$ is ignored and $\overline{DSTB}$ is not asserted.

Hold Request and Hold Acknowledge

At times, an external bus master will need to use the μPD70136 bus. When the HLDRQ input is asserted by external logic, the μPD70136 recognizes this as a request for external bus mastership. The μPD70136 will finish the current bus operation, stop driving its address, data, and control buses, and assert HLDAK. The external device, such as the μPD71071 or μPD71037 DMA controller, may then drive the μPD70136 bus. Note that if the current bus operation involves more than one bus cycle, such as a 16-bit access to an odd address or due to dynamic bus sizing, the μPD70136 will finish both cycles before releasing the bus.

If the current instruction uses the $\overline{BUSLOCK}$ prefix, HLDRQ will be ignored. This will be indicated by the $\overline{BUSLOCK}$ output. Also, during interrupt acknowledge, $\overline{BUSLOCK}$ is asserted between the two INTAK cycles so that HLDRQ is ignored until after the second INTAK.

SYSTEM INTERFACING

System Memory Access Time

Table 7 shows the system memory access time required for 12.5-MHz and 16-MHz μPD70136 systems to run with

zero, one, two, and three wait states. This is the time from when the address bus is valid to when the external system must present the read data on the data bus. These numbers are based on the preliminary ac timing given in this document and are subject to change.

Table 7. Performance vs. Wait States

	12.5 MHz			16 MHz		
Number of Wait States	Memory Cycle Time (ns)	System Access Time (ns)	Relative Performance (%)	Memory Cycle Time (ns)	System Access Time (ns)	Relative Performance (%)
0	160	113	78	125	78	100
1	240	193	64	187.5	140.5	82
2	320	273	52	250	203	67
3	400	353	43	312.5	265.5	56

Note: Performance is relative to the 0 wait state, 16 MHz.

Wait States

Table 7 also shows the effect of wait states on performance. The μPD70136 overlaps bus interface operations in time with instruction execution. This greatly reduces the effect of wait states on performance. Each bus cycle is nominally two clocks long, while the minimum instruction time is two clocks, with many instructions taking longer. There is some idle bus time when the CPU is processing a long instruction and the prefetch queue is full. Wait states can often fill these idle states.

However, adding wait states to bus cycles reduces the bus bandwidth available for other bus masters, such as DMA controllers, since some of the idle time that would have been available to them is used for CPU cycles.

Note that in all cases, a 16-MHz μPD70136 with N+1 wait states is faster than a 12.5-MHz device with N wait states while using slower memories.

Please note also that these numbers were measured using a particular set of benchmarks and should be used for comparison purposes only. Different results will be obtained for other program mixes.

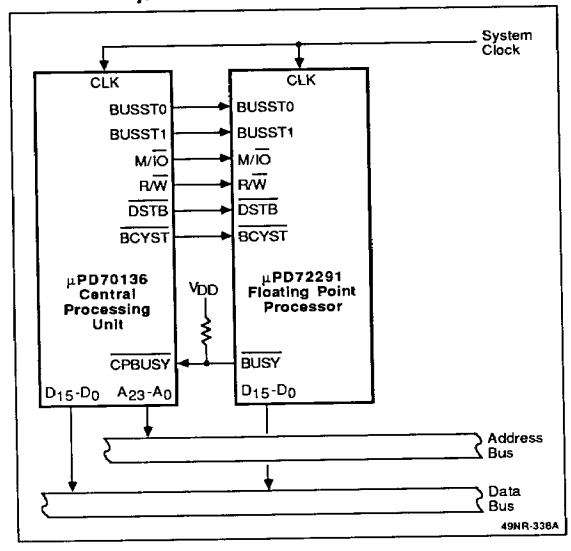
Interfacing to the μPD72291 Floating-Point Coprocessor

The μPD72291 (AFPP) is a very-high-performance floating-point coprocessor for the μPD70136 offering in excess of 530K floating-point operations per second at 16 MHz. The AFPP is programmed as an extension of the μPD70136 instruction set. The AFPP executes floating-point operations, computes transcendental functions, and performs vector multiplications.

AFPP instructions use the FP01 and FP02 formats. When one of these opcodes is encountered and an AFPP is connected, a coprocessor protocol routine is entered. The μPD70136 will compute any effective addresses required, read or write the operands for the AFPP, and instruct the AFPP as to what operation should be performed. The AFPP reponds by asserting its BUSY output when it starts the operation. The μPD70136 will not start another AFFP operation until BUSY is deasserted, but may execute CPU instructions. When BUSY is deasserted, the μPD70136 will transfer the AFPP status to the AW register.

Figure 10 shows how to connect a μPD70136 CPU to a μPD72291 AFPP. Figure 11 shows a typical system. The CPU reads and writes status and commands to the AFPP using coprocessor read and write cycles, which always take two clocks. AFPP operands are written/read using coprocessor memory write/read cycles, which always require one wait state. External $\overline{\text{READY}}$ logic must take care to include this wait state.

Figure 10. Connections Between μPD70136 and μPD72291

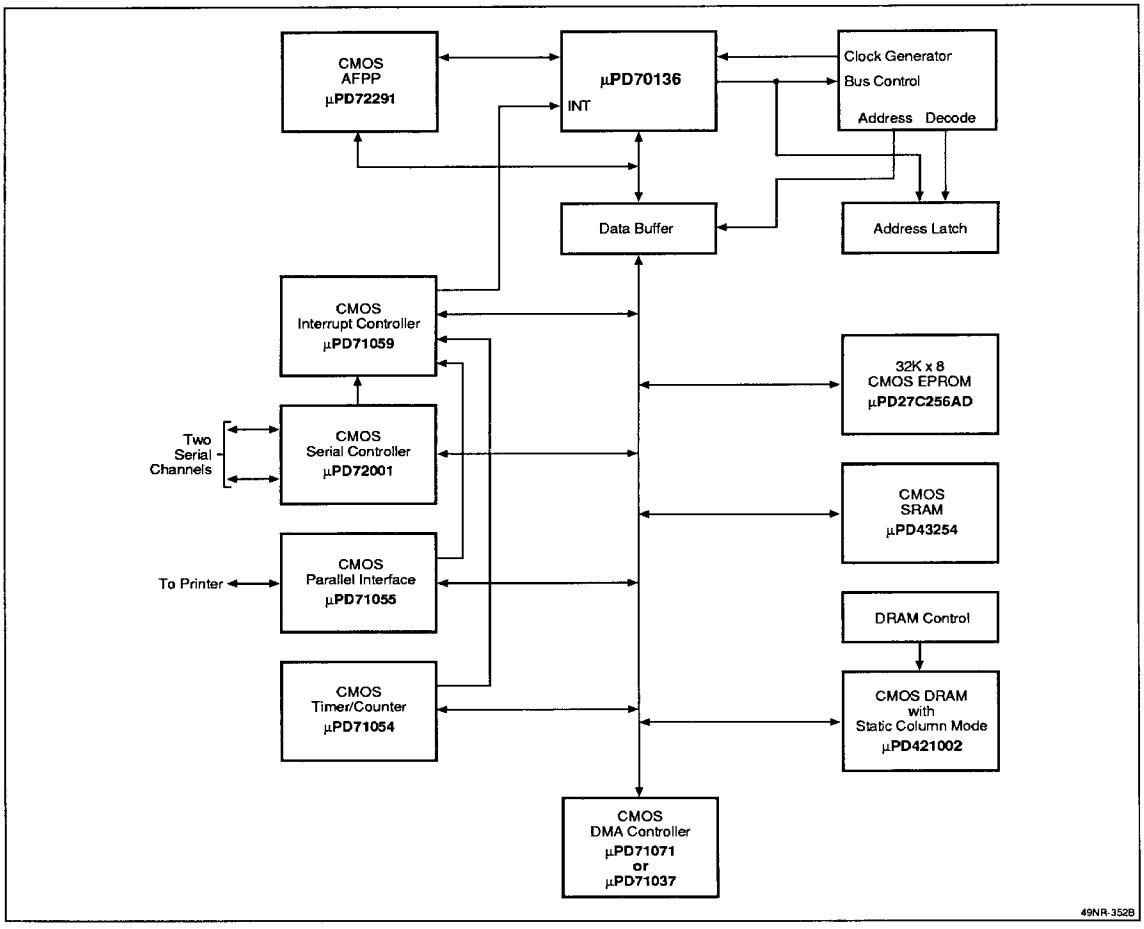


On RESET, $\overline{CPBUSY}$ is sampled. If it is low, the μPD70136 assumes that a coprocessor is connected. $\overline{CPERR}$ is also sampled to determine what kind of coprocessor is connected, as follows.

$\overline{CPBUSY}$	$\overline{CPERR}$	Coprocessor
1	X	None
0	GND	μPD72291
0	V_{DD}	Other

AFPP memory operands must always begin on even addresses and may not reside in 8-bit wide memory. Dynamic bus sizing may not be used for AFPP operands.

Figure 11. Typical μPD70136 System



3e

INTERRUPT OPERATION

The interrupts supported by the μPD70136 can be divided into two types: interrupts generated by external interrupt requests and traps generated by software processing. They are:

External Interrupts

- $\overline{\text{NMI}}$ input (nonmaskable)
- INT input (maskable)

Software Traps

- Divide error during DIV or DIVU instruction
- Array bound error during CHKIND
- Single-step (PSW BRK flag = 1)
- Undefined instruction
- Coprocessor error
- Coprocessor not connected
- Break instructions
  BRKV BRK imm8
  BRK3 BRKXA

Interrupt Priorities

Interrupts are prioritized as follows:

$\overline{\text{NMI}}$ > INT > BRK flag > others at same level

Interrupts are not accepted during certain times. $\overline{\text{NMI}}$, INT and BRK flags are not accepted in these cases:

(1) Between execution of MOV or POP that uses a segment register as an operand and the next instruction.

(2) Between a segment override prefix and the next instruction

(3) Between a repeat or $\overline{\text{BUSLOCK}}$ prefix and the next instruction

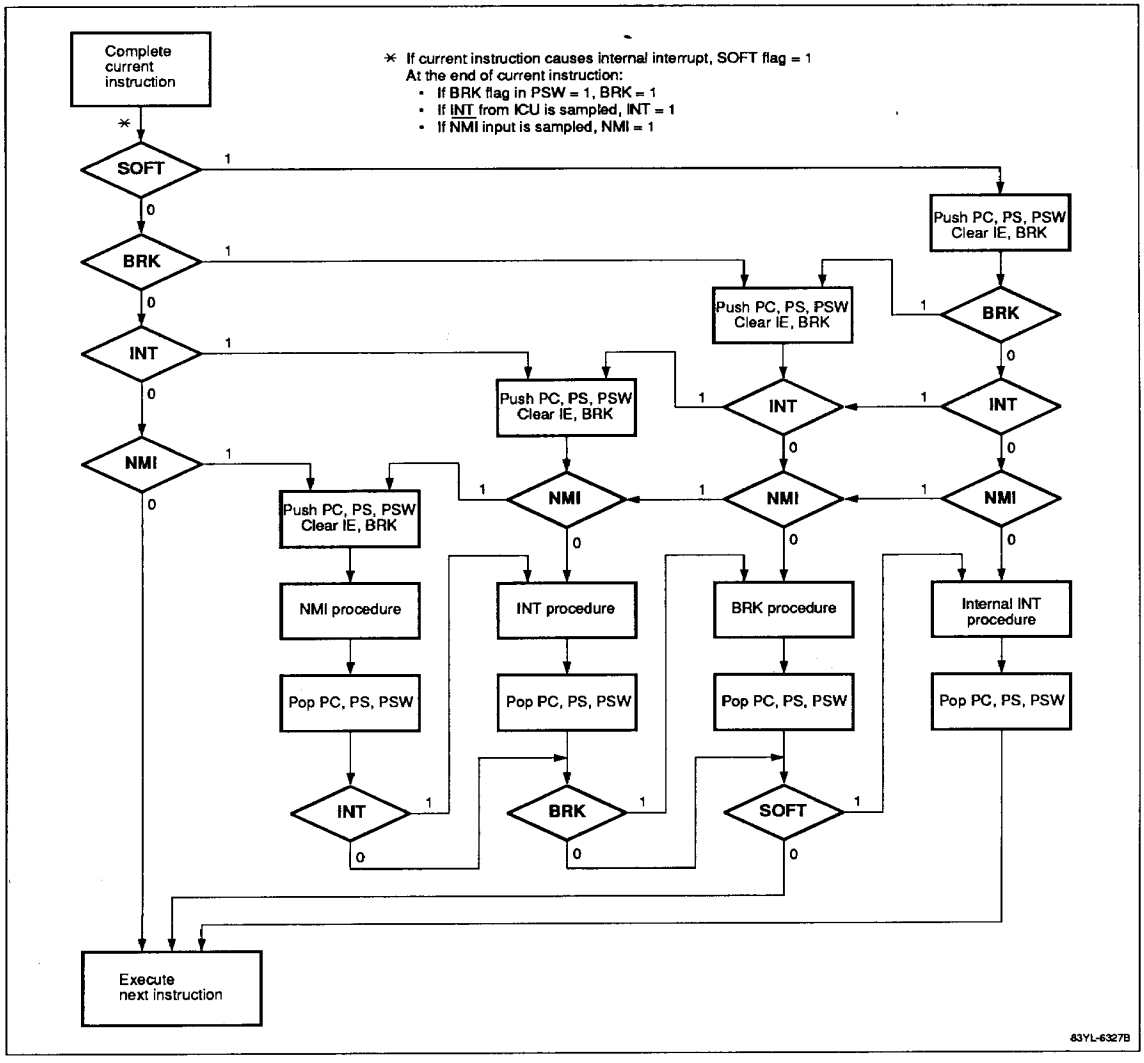
INT is not accepted when the PSW IE flag is 0, or between an RETI or POP PSW and the next instruction. Figure 12 is a flow diagram for processing interrupt requests.

Interrupt Vectors

Once an interrupt has been accepted, an interrupt service routine will be entered. The address of this routine is specified by an interrupt vector, which is stored in the interrupt vector table. For most interrupts, the vector used depends on what interrupt is being processed (e.g., $\overline{\text{NMI}}$ always uses vector 2). For INT and BRK imm8 interrupts, any vector may be used; the vector number is supplied by an external device in the case of INT (e.g., a μPD71059), or by immediate data in the case of BRK.

Figure 13 is the interrupt vector table. The table uses 1K bytes of memory—addresses 000H to 3FFH—and stores up to 256 vectors (4 bytes per vector).

Figure 12. Interrupt Prioritization Flow Diagram



3e

Each interrupt vector consists of 4 bytes. The 2 bytes in the low addresses of memory are loaded into PC as the offset, and the 2 high bytes are loaded into PS as the base address. Interrupt vector 0 in figure 14 is an example. The bytes are combined in reverse order. The lower-order bytes in the vector become the most significant bytes in the PC and PS, and the higher-order bytes become the least significant.

Based on this format, the contents of each vector should be initialized at the beginning of the program. The basic mechanism for servicing an interrupt is:

(SP – 1, SP – 2) ← PSW
(SP – 3, SP – 4) ← PS
(SP – 5, SP – 6) ← PC
SP ← SP – 6
IE ← 0, BRK ← 0
PS ← vector high bytes
PC ← vector low bytes

Figure 13. Interrupt Vector Table

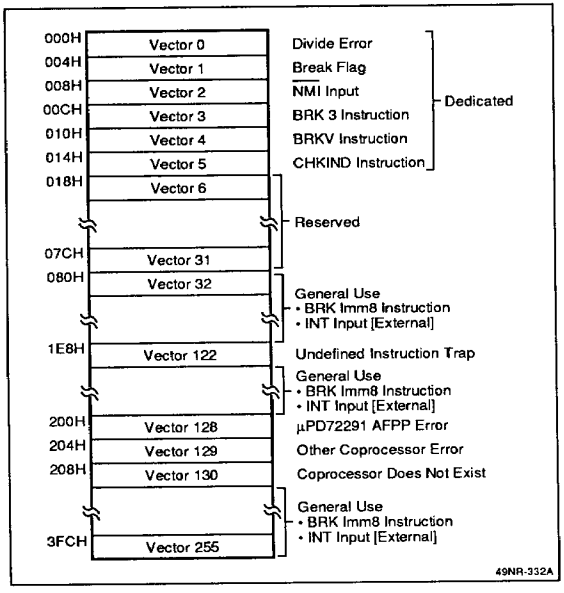
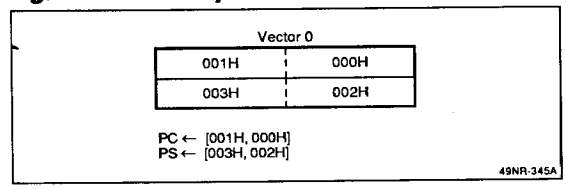


Figure 14. Interrupt Vector 0



During interrupt servicing, the third item pushed on the stack is the return PC value. For some types of traps (divide error, CHKIND, illegal opcode, AFPP error, coprocesor not present, or other CP error), this value points to the instruction that generated the trap. For the other interrupts (single-step, BRK3, BRKV, $\overline{\text{NMI}}$, or INT), this value points to the next instruction. Trap handlers for error traps can thus easily find the offending opcode, and other handlers can simply return after processing the interrupt.

STANDBY FUNCTION

The μPD70136 offers two standby modes to reduce power consumption: HALT and STOP. Both are entered after executing a HALT instruction.

HALT Standby Mode

In the HALT 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 to one-fifth the level of normal operation.

The HALT standby mode is exited when RESET or an external interrupt ($\overline{\text{NMI}}$, INT) is received. If INT is used and interrupts were enabled before the HALT state was entered, an INTAK cycle will be performed to fetch a vector number. The interrupt service routine will be executed. After RETI, execution will resume with the instruction following the HALT. If interrupts were disabled, the interrupt service routine will not be entered, but execution will resume with the instruction following the HALT.

If $\overline{\text{NMI}}$ is used to exit the HALT standby mode, the $\overline{\text{NMI}}$ service routine will always be entered.

The bus hold (HLDRQ/HLDAK) function still operates during HALT standby mode. The CPU returns to HALT standby mode when the bus hold request is removed.

During HALT standby mode, when all control outputs go low, the address and data buses will be either high or low. Refer to table 1 for information about the states of other outputs in the standby mode.

STOP Standby Mode

In the STOP standby mode, the μPD70136 clock is stopped for maximum power reduction. To enter this mode, special steps must be taken to prepare the μPD70136 for having its clock stopped.

INT, $\overline{NMI}$ and HLDRQ must not be asserted while the μPD70136 is in STOP mode, or for at least 10 clock periods before STOP is entered, or for at least 10 clock periods after STOP mode is exited. External hardware must ensure that these intputs are not asserted during this time.

STOP mode is entered by disabling $\overline{NMI}$, INT, and HLDRQ, entering the HALT standby mode, and stopping the clock input 10 clock periods after the HALT acknowledge but cycle is issued. The CLK input must be stopped during the low phase of the clock. STOP mode is exited when external logic starts the clock, waits 10 clock periods, and enable $\overline{NMI}$, INT, and HLDRQ; the μPD70136 will return to the HALT standby mode.

All output pins in STOP mode are in the same state as in HALT standby mode. Refer to table 1.

INSTRUCTION SET HIGHLIGHTS

Enhanced Instructions

In addition to the μPD8088/86 instructions, the μPD70136 has enhanced instructions listed in table 8.

Table 8. Enhanced Instruction

Instruction	Function
PUSH imm	Pushes immediate data onto stack
PUSH R	Pushes 8 general registers onto stack
POP R	Pops 8 general registers onto stack
MULL imm	Executes 16-bit multiply of register or memory contents by immediate data
SHL imm8 SHR imm8 SHRA imm8 ROL imm8 ROR imm8 ROLC imm8 RORC imm8	Shifts/rotates register or memory by immediate value
CHKIND	Checks array index against designated boundaries
INM	Moves a string from an I/O port to memory
OUTM	Moves a string from memory to an I/O port
PREPARE	Allocates an area for a stack frame and copies previous frame pointers
DISPOSE	Frees the current stack frame on a procedure exit

Enhanced Stack Operation Instructions

PUSH imm. This instruction allows immediate data to be pushed onto the stack.

PUSH R; POP R. These instructions allow the contents of the eight general registers to be pushed onto or popped from the stack with a single instruction.

Enhanced Multiplication Instructions

MUL reg16, imm16; MUL mem16, imm16. These instructions allow the contents of a register or memory location to be multiplied by immediate data.

Enhanced Shift and Rotate Instructions

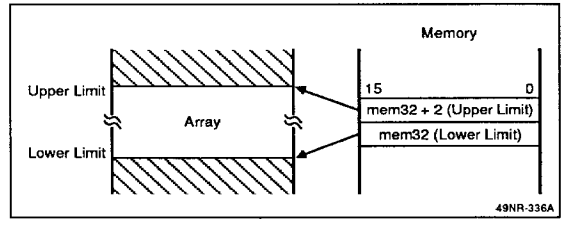
SHL reg, imm8; SHR reg, imm8; SHRA reg, imm8. These instructions allow the contents of a register to be shifted by the number of bits defined by the immediate data.

ROL reg, imm8; ROR reg imm8; ROLC reg, imm8; RORC reg, imm8. These instructions allow the contents of a register to be rotated by the number of bits defined by the immediate data.

Check Array Boundary Instruction

CHKIND reg16, mem32. This instruction is used to verify that index values pointing to the elements of an array data structure are within the defined range. See figure 15. The lower limit of the array should be in memory location mem32, the upper limit in mem32 + 2. If the index value in reg16 is not between these limits when CHKIND is executed, a BRK 5 will occur. This causes a jump to the location in interrupt vector 5.

Figure 15. Check Array Boundary



Block I/O Instruction

OUTM DW, src-block; INM dist-block, DW. These instructions are used to output or input a string to or from memory, when preceded by a repeat prefix.

Stack Frame Instruction

PREPARE imm16,imm8. This instruction is used to generate the stack frames required by block-structured languages, such as PASCAL and Ada. The stack frame consists of two areas. One area has a pointer that points to another frame which has variables that the current frame can access. The other is a local variable area for the current procedure.

DISPOSE. This instruction releases that last stack frame generated by the PREPARE instruction. It returns the stack and base pointers to the values they had before the PREPARE instruction was used to call a procedure.

Unique Instructions

In addition to the μPD8088/86 instructions and the enhanced instructions, the μPD70136 has the unique instructions listed in table 9.

Table 9. Unique Instructions

Instruction	Function
INS	Insert bit field
EXT	Extract bit field
ADD4S	Adds packed decimal strings
SUB4S	Subtracts one packed decimal string from another
CMP4S	Compares two packed decimal strings
ROL4	Rotates one BCD digit left through AL lower 4 bits
ROR4	Rotates one BCD digit right through AL lower 4 bits
BRKXA	Break and enable expanded addressing
RETXA	Return from break and disable expanded addressing
TEST1	Tests a specified bit and sets/resets Z flag
NOT1	Inverts a specified bit
CLR1	Clears a specified bit
SET1	Sets a specified bit
REPC	Repeats next instruction until CY flag is cleared
REPNC	Repeats next instruction until CY flag is set
FP02	Additional floating-point processor call

Variable Length Bit Field Operation Instructions

This category has two instructions: INS (Insert Bit Field) and EXT (Extract Bit Field). These instructions are highly effective for computer graphics and high-level languages. They can, for example, be used for data structures such as packed arrays and record type data used in PASCAL.

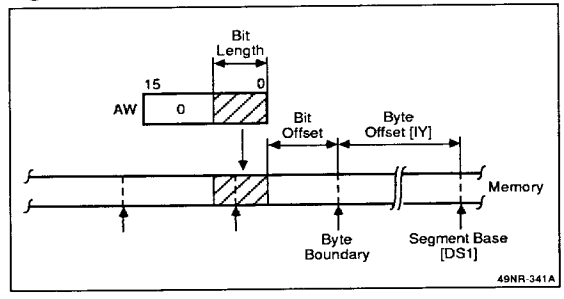
INS reg8, reg8; INS reg8, imm4. This instruction transfers low bits from the 16-bit AW register (the number of bits is specified by the second operand) to the memory location specified by the segment base (DS1 register) plus the byte offset (IY register). The starting bit position within this byte is specified as an offset by the lower 4 bits of the first operand. See figure 16.

After each complete data transfer, the IY register and the register specified by the first operand are automatically updated to point to the next bit field.

Either immediate data or a register may specify the number of bits transferred (second operand). Because the maximum transferable bit length is 16 bits, only the lower 4 bits of the specified register (00H to 0FH) will be valid.

Bit field data may overlap the byte boundary of memory.

Figure 16. Bit Field Insertion



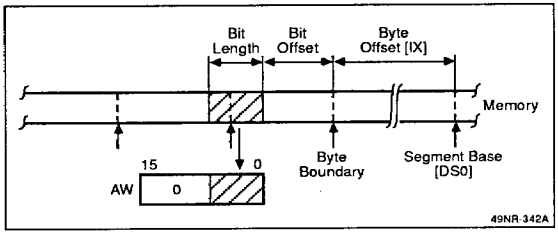
EXT reg8, reg8; EXT reg8, imm4. This instruction loads to the AW registers the bit field data whose bit length is specified by the second operand of the instruction from the memory location that is specified by the DS0 segment register (segment base), the IX index register (byte offset), and the lower 4 bits of the first operand (bit offset). See figure 17.

After the transfer is complete, the IX register and the lower 4 bits of the first operand are automatically updated to point to the next bit field.

Either immediate data or a register may be specified for the second operand. Because the maximum transferable bit length is 16 bits, however, only the lower 4 bits of the specified register (00H to 0FH) will be valid.

Bit field data may overlap the byte boundary of memory.

Figure 17. Bit Field Extraction



Packed BCD Operation Instructions

The instructions described here process packed BCD data either as strings (ADD4S, SUB4S, CMP4S) or byte-format operands (ROR4, ROL4). Packed BCD strings may be from 1 to 254 digits in length.

When the number of digits is even, the zero (Z) and carry (CY) flags will be set according to the result of the operation. When the number of digits is odd, the Z and CY flags may not be set correctly. In this case (CL = odd), the Z flag will not be set unless the upper 4 bits of the highest byte are all 0s. The CY flag will not be set unless there is a carry out of the upper 4 bits of the highest byte. When CL is odd, the contents of the upper 4 bits of the highest byte of the result are undefined.

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 result of the operation will affect the V (overflow), CY, and Z flags .

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

SUB4S. This instruction subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY 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 result of the operation will affect the V, CY, and Z flags.

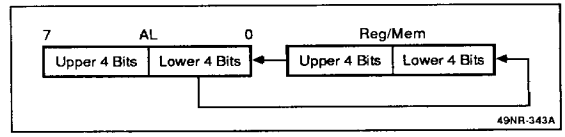
BCD string (IY, CL) ← BCD string (IY, CL) – BCD string (IX, CL)

CMP4S. This instruction performs the same operation as SUB4S except that the result is not stored and only the V, CY, and Z flags are affected.

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

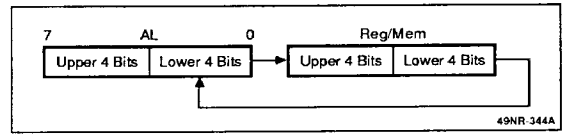
ROL4. This instruction treats the byte data of the register or memory operand specified by the instruction as BCD data and uses the lower 4 bits of the AL register (AL_L) to rotate that data one BCD digit to the left. See figure 18.

Figure 18. BCD Rotate Left



ROR4. This instruction treats the byte data of the register or memory specified by the instruction as BCD data and uses the lower 4 bits of the AL register (AL_L) to rotate that data one BCD digit to the right. See figure 19.

Figure 19. BCD Rotate Right



Bit Manipulation Instructions

TEST1. This instruction tests a specific bit in a register or memory location. If the bit is 1, the Z flag is reset to 0. If the bit is 0, the Z flag is set to 1.

NOT1. This instruction inverts a specific bit in a register or memory location.

CLR1. This instruction clears a specific bit in a register or memory location.

SET1. This instruction sets a specific bit in a register or memory location.

Repeat Prefix Instructions

REPC. This instruction causes the μPD70136 to repeat the following primitive block transfer instruction until the CY flag becomes cleared or the CW register becomes zero.

REPNC. This instruction causes the μPD70136 to repeat the following primitive block transfer instruction until the CY flag becomes set or the CW register becomes zero.

Address Expansion Control Instructions

BRKXA Imm8. This instruction is used to turn on expanded addressing. The 8-bit immediate data specifies an interrupt vector. The PC field of this vector is loaded into the PC (and PFP). The XA flag in the XAM register is set to 1, thereby enabling the expanded addressing

mode. The μPD70136 will begin fetching from the new PFP through the address translation table. That is, the new PC is treated as a logical address and is translated to the new, larger physical address space.

This instruction does not save any return address information, such as PC, PS, or PSW to the stack.

RETXA Imm8. This instruction is used to turn off expanded addressing. It is identical in operation to BRKXA, except that the expanded addressing mode is turned off before fetching from the new address. That is, the XA flag in the XAM register is set to 0, and the PC is loaded with the value of the PC field in the interrupt vector selected by the immediate data.

This instruction does not save any return address information such as PC, PS, or PSW to the stack.

Porting μPD70116/70108 Code to μPD70136

The μPD70136 is completely software compatible with the μPD70116/70108. However, the μPD70136 offers some improvements that may affect the porting of μPD70116 code to the μPD70136. These improvements are:

(1) The μPD70116 does not trap on undefined opcodes. The μPD70136 will trap, and also will trap when a register addressing mode is used for any of these instructions:

CHKIND LDEA
MOV DS0/DS1 BR 1,id
CALL 1,id

(2) During signed division (DIV), if the quotient is 80H (byte operation) or 8000H (word), the μPD70116 will take a Divide By 0 trap. The μPD70136 will perform the calculation.

(3) When the μPD70116 executes the POLL instruction, it will wait for the POLL input signal to be asserted. The μPD70136 has no POLL input; instead, when this instruction is executed, if a coprocessor is not connected, then a Coprocessor Not Present trap will be taken. If a coprocessor is attached, then no operation takes place.

The μPD70116 accepts FP01 and FP02 as opcodes for the iAPX8087 coprocessor. The μPD70136 accepts these as opcodes for the μPD72291 coprocessor, which is not compatible with the iAPX8087.

(4) During the POP R instruction, the μPD70116 does not restore the SP register. The μPD70136 does restore the SP.

(5) When processing a divide error, the μPD70116 saves the address of the next instruction. The μPD70136 saves the address of the current instruction (the divide instruction).

(6) The μPD70116 allows up to 3 prefix instructions in any combination. The μPD70136 also allows 3 prefixes, but only one of each type can be used. The μPD70136 could operate incorrectly if there are two prefixes of the same type. For example, consider:

REP
REPC
CMPBK SS: src-block, dst-block

If the compare operation is interrupted, then when it resumes following the interrupt service, execution will begin at the REPC instruction, not the REP instruction, because two repeat prefixes were used.

(7) The μPD70116 accepts $\overline{\text{NMI}}$ requests even while processing an $\overline{\text{NMI}}$. The μPD70136 does not allow nesting of $\overline{\text{NMI}}$s; the $\overline{\text{NMI}}$ input will be ignored until the $\overline{\text{NMI}}$ interrupt handler is exited.

INSTRUCTION SET

Symbols

Preceding the instruction set, several tables explain symbols, abbreviations, and codes.

Clocks

In the Clocks column of the instruction set, the numbers cover these operations: instruction decoding, effective address calculation, operand fetch, and instruction execution.

Clock timings assume the instruction has been prefetched and is present in the 8-byte instruction queue. Otherwise, add two clocks for each pair of bytes not present.

Word operands require two additional clocks for each transfer to an unaligned (odd address) memory operand. These times are shown on the right side of the slash (/).

For conditional control transfer or branch instructions, the number on the left side of the slash is applicable if the transfer or branch takes place. The number on the right side is applicable if it does not take place.

If a range of numbers is given, the execution time depends on the operands involved.

Symbols

Symbol	Meaning
acc	Accumulator(AW or AL)
duso	Displacement (8 or 16 bits)
dmem	Direct memory address
dst	Destination operand or address
ext-disp8	16-bit displacement (sign-extension byte + 8-bit displacement)
far_label	Label within a different program segment
far_proc	Procedure within a different program segment
fp_op	Floating-point instruction operation
imm	8- or 16-bit immediate operand
imm3/4	3- or 4-bit immediate bit offset
imm8	8-bit immediate operand
imm16	16-bit immediate operand
mem	Memory field (000 to 111); 8- or 16-bit memory location
mem8	8-bit memory location
mem16	16-bit memory location
mem32	32-bit memory location
memptr16	Word containing the destination address within the current segment
memptr32	Double word containing a destination address in another segment
mod	Mode field (00 to 10)
near_label	Label within the current segment
near_proc	Procedure within the current segment
offset	Immediate offset data (16 bits)
pop_value	Number of bytes to discard from the stack
reg	Register field (000 to 111); 8- or 16-bit general-purpose register
reg8	8-bit general-purpose register
reg16	16-bit general-purpose register
regptr	16-bit register containing a destination address within the current segment
regptr16	Register containing a destination address within the current segment
seg	Immediate segment data (16 bits)
short_label	Label between −128 and +127 bytes from the end of the current instruction
sr	Segment register
src	Source operand or address
temp	Temporary register (8/16/32 bits)
AC	Auxiliary carry flag
AH	Accumulator (high byte)
AL	Accumulator (low byte)

Symbol	Meaning
AW	Accumulator (16 bits)
BH	BW register (high byte)
BL	BW register (low byte)
BP	BP register
BRK	Break flag
BW	BW register (16 bits)
CH	CW register (high byte)
CL	CW register (low byte)
CW	CW register (16 bits)
CY	Carry flag
DH	DW register (high byte)
DIR	Direction flag
DL	DW register (low byte)
DS0	Data segment 0 register (16 bits)
DS1	Data segment 1 register (16 bits)
DW	DW register (16 bits)
IE	Interrupt enable flag
IX	Index register (source) (16 bits)
IY	Index register (destination) (16 bits)
MD	Mode flag
P	Parity flag
PC	Program counter (16 bits)
PS	Program segment register (16 bits)
PSW	Program status word (16 bits)
R	Register set
S	Sign extend operand field S = No sign extension S = Sign extend immediate byte operand
S	Sign flag
SP	Stack pointer (16 bits)
SS	Stack segment register (16 bits)
V	Overflow flag
W	Word/byte field (0 to 1)
X, XXX, YYY, ZZZ	Data to identify the instruction code of the external floating-point arithmetic chip
XXH	Two-digit hexadecimal value
XXXXH	Four-digit hexadecimal value
Z	Zero flag

3e

Flag Operations

Symbol	Meaning
(blank)	No change
0	Cleared to 0
1	Set to 1
x	Set or cleared according to result
u	Undefined
R	Restored to previous state

Memory Addressing Modes

mem	mod = 00	mod = 01	mod = 10
000	BW + IX	BW + IX + disp8	BW + IX + disp16
001	BW + IY	BW + IY + disp8	BW + IY + disp16
010	BP + IX	BP + IX + disp8	BP + IX + disp16
011	BP + IY	BP + IY + disp8	BP + IY + disp16
100	IX	IX + disp8	IX + disp16
101	IY	IY + disp8	IY + disp16
110	Direct	BP + disp8	BP + disp16
111	BW	BW + disp8	BW + disp16

Register Selection (mod = 11)

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

Segment Register Selection

sr	Segment Register
00	DS1
01	PS
10	SS
11	DS0

Instruction Set

		Opcode																	Flags					
Mnemonic	Operand	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Clocks	Bytes	AC	CY	V	P	S	Z
Data Transfer Instructions																									
MOV	reg, reg	1	0	0	0	1	0	1	W	1	1		reg		reg			2	2						
	mem, reg	1	0	0	0	1	0	0	W	mod			reg		mem			3/5	2-4						
	reg, mem	1	0	0	0	1	0	1	W	mod			reg		mem			5/7	2-4						
	mem, imm	1	1	0	0	0	1	1	W	mod			000		mem			3/5	3-6						
	reg, imm	1	0	1	1	W		reg										2	2-3						
	acc, dmem	1	0	1	0	0	0	0	W									5/7	3						
	dmem, acc	1	0	1	0	0	0	1	W									3/5	3						
	sr, reg16	1	0	0	0	1	1	1	0	1	1	0	sr		reg			2	2						
	sr, mem16	1	0	0	0	1	1	1	0	mod		0	sr		mem			5/7	2-4						
	reg16, sr	1	0	0	0	1	1	0	0	1	1	0	sr		reg			2	2						
	mem16, sr	1	0	0	0	1	1	0	0	mod		0	sr		mem			3/5	2-4						
	DS0, reg16, mem32	1	1	0	0	0	1	0	1	mod			reg		mem			10/14	2-4						
	DS1, reg16, mem32	1	1	0	0	0	1	0	0	mod			reg		mem			10/14	2-4						
	AH, PSW	1	0	0	1	1	1	1	1									2	1						
	PSW, AH	1	0	0	1	1	1	1	0									2	1	x	x		x	x	x
LDEA	reg16, mem16	1	0	0	0	1	1	0	0	mod			reg		mem			2	2-4						
TRANS	src_table	1	1	0	1	0	1	1	1									5	1						
XCH	reg, reg	1	0	0	0	0	1	1	W	1	1		reg		reg			3	2						
	mem, reg	1	0	0	0	0	1	1	W	mod			reg		mem			8/12	2-4						

Instruction Set (cont)

Mnemonic	Operand	Opcode																Clocks	Bytes	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 Instructions (cont)																									
XCH (cont)	AW, reg16	1	0	0	1	0		reg										3	1						
Repeat Prefixes																									
REPC		0	1	1	0	0	1	0	1									2	1						
REPNC		0	1	1	0	0	1	0	0									2	1						
REP REPE REPZ		1	1	1	1	0	0	1	1									2	1						
REPNE REPNZ		1	1	1	1	0	0	1	0									2	1						
Block Transfer Instructions																									
MOVBK	dst, src	1	0	1	0	0	1	0	W										1						
																		3 + 4n (W = 0) 3 + 4n (W = 1, even addresses) 3 + 8n (W = 1, odd addresses) 3 + 6n (W = 1, odd/even addresses)							
CMPBK	dst, src	1	0	1	0	0	1	1	W										1	x	x	x	x	x	x
																		3 + 7n (W = 0) 3 + 7n (W = 1, even addresses) 3 + 11n (W = 1, odd addresses) 3 + 9n (W = 1, odd/even addresses)							
CMPM	dst	1	0	1	0	1	1	1	W										1	x	x	x	x	x	x
																		3 + 5n (W = 0) 3 + 5n (W = 1, even addresses) 3 + 7n (W = 1, odd addresses)							
LDM	src	1	0	1	0	1	1	0	W										1						
																		5 + 2n (W = 0) 5 + 2n (W = 1, even addresses) 5 + 4n (W = 1, odd addresses)							
STM	dst	1	0	1	0	1	0	1	W										1						
																		3 + 2n (W = 0) 3 + 2n (W = 1, even addresses) 3 + 4n (W = 1, odd addresses)							
	n = number of returns String instruction execution clocks for a single-instruction execution are in parentheses.																								
I/O Instructions																									
IN	acc, imm8	1	1	1	0	0	1	0	W									5/7	2						
	acc, DW	1	1	1	0	1	1	0	W									3/5	1						
OUT	imm8, acc	1	1	1	0	0	1	1	W									3/5	2						
	DW, acc	1	1	1	0	1	1	1	W									3/5	1						

3e

Instruction Set (cont)

Mnemonic	Operand	Opcode																Clocks	Bytes	Flags					
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			AC	CY	V	P	S	Z
I/O Instructions (cont)																									
INM	dst, DW	0	1	1	0	1	1	0	W									1 3 + 11n (W = 0) 3 + 8n (W = 1, even addresses) 3 + 22n (W = 1, odd addresses) 3 +20n (W = 1, odd/even addresses; odd for I/O) 3 + 13n (W = 1, odd/even addresses; odd for memory)							
OUTM	DW, src	0	1	1	0	1	1	1	W									1 3 + 11n (W = 0) 3 + 8n (W = 1, even addresses) 3 + 22n (W = 1, odd addresses) 3 + 20n (W = 1, odd addresses; odd for I/O) 3 + 13n (W = 1, odd addresses; odd for memory)							
	n = number of transfers String instruction execution clocks for a single-instruction execution are in parentheses. Use the right side of the slash (/) for DMA I/O accesses.																								
BCD Instructions																									
ADJBA		0	0	1	1	0	1	1	1									4	1	x	x	u	u	u	u
ADJ4A		0	0	1	0	0	1	1	1									2	1	x	x	u	x	x	x
ADJBS		0	0	1	1	1	1	1	1									4	1	x	x	u	u	u	u
ADJ4S		0	0	1	0	1	1	1	1									2	1	x	x	u	x	x	x
ADD4S	dst, src	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0	2 + 18n	2	u	x	u	u	u	x
SUB4S	dst, src	0	0	0	0	1	1	1	1	0	0	1	0	0	0	1	0	2 + 18n	2	u	x	u	u	u	x
CMP4S	dst, src	0	0	0	0	1	1	1	1	0	0	1	0	0	1	1	0	7 + 14n	2	u	x	u	u	u	x
ROL4	reg8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0	9	3						
		1	1	0	0	0	reg																		
	mem8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0	15	3-5						
		mod	0	0	0	mem																			
ROR4	reg8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0	13	3						
		1	1	0	0	0	reg																		
	mem8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0	19	3-5						
		mod	0	0	0	mem																			
	n = number of BCD digits divided by 2																								
Data Type Conversion Instructions																									
CVTBD		1	1	0	1	0	1	0	0	0	0	0	0	1	0	1	0	12	2	u	u	u	x	x	x
CVTDB		1	1	0	1	0	1	0	1	0	0	0	0	1	0	1	0	8	2	u	u	u	x	x	x
CVTBW		1	0	0	1	1	0	0	0									2	1						
CVTWL		1	0	0	1	1	0	0	1									2	1						

Instruction Set (cont)

Mnemonic	Operand	Opcode 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Clocks	Bytes	Flags AC	CY	V	P	S	Z
Arithmetic Instructions																									
ADD	reg, reg	0	0	0	0	0	0	1	W	1	1		reg			reg		2	2	x	x	x	x	x	x
	mem, reg	0	0	0	0	0	0	0	W	mod			reg			mem		7/11	2-4	x	x	x	x	x	x
	reg, mem	0	0	0	0	0	0	1	W	mod			reg			mem		6/8	2-4	x	x	x	x	x	x
	reg, imm	1	0	0	0	0	0	S	W	1	1	0	0	0		reg		2	3-4	x	x	x	x	x	x
	mem, imm	1	0	0	0	0	0	S	W	mod		0	0	0		mem		7/11	3-6	x	x	x	x	x	x
	acc, imm	0	0	0	0	0	1	0	W									2	2-3	x	x	x	x	x	x
ADDC	reg, reg	0	0	0	1	0	0	1	W	1	1		reg			reg		2	2	x	x	x	x	x	x
	mem, reg	0	0	0	1	0	0	0	W	mod			reg			mem		7/11	2-4	x	x	x	x	x	x
	reg, mem	0	0	0	1	0	0	1	W	mod			reg			mem		6/8	2-4	x	x	x	x	x	x
	reg, imm	1	0	0	0	0	0	S	W	1	1	0	1	0		reg		2	3-4	x	x	x	x	x	x
	mem, imm	1	0	0	0	0	0	S	W	mod		0	1	0		mem		7/11	3-6	x	x	x	x	x	x
	acc, imm	0	0	0	1	0	1	0	W									2	2-3	x	x	x	x	x	x
SUB	reg, reg	0	0	1	0	1	0	1	W	1	1		reg			reg		2	2	x	x	x	x	x	x
	mem, reg	0	0	1	0	1	0	0	W	mod			reg			mem		7/11	2-4	x	x	x	x	x	x
	reg, mem	0	0	1	0	1	0	1	W	mod			reg			mem		6/8	2-4	x	x	x	x	x	x
	reg, imm	1	0	0	0	0	0	S	W	1	1	1	0	1		reg		2	3-4	x	x	x	x	x	x
	mem, imm	1	0	0	0	0	0	S	W	mod		1	0	1		mem		7/11	3-6	x	x	x	x	x	x
	acc, imm	0	0	1	0	1	1	0	W									2	2-3	x	x	x	x	x	x
SUBC	reg, reg	0	0	0	1	1	0	1	W	1	1		reg			reg		2	2	x	x	x	x	x	x
	mem, reg	0	0	0	1	1	0	0	W	mod			reg			mem		7/11	2-4	x	x	x	x	x	x
	reg, mem	0	0	0	1	1	0	1	W	mod			reg			mem		6/8	2-4	x	x	x	x	x	x
	reg, imm	1	0	0	0	0	0	S	W	1	1	0	1	1		reg		2	3-4	x	x	x	x	x	x
	mem, imm	1	0	0	0	0	0	S	W	mod		0	1	1		mem		7/11	3-6	x	x	x	x	x	x
	acc, imm	0	0	0	1	1	1	0	W									2	2-3	x	x	x	x	x	x
INC	reg8	1	1	1	1	1	1	1	0	1	1	0	0	0		reg		2	2	x		x	x	x	x
	mem	1	1	1	1	1	1	1	W	mod		0	0	0		mem		7/11	2-4	x		x	x	x	x
	reg16	0	1	0	0	0		reg										2	1	x		x	x	x	x
DEC	reg8	1	1	1	1	1	1	1	0	1	1	0	0	1		reg		2	2	x		x	x	x	x
	mem	1	1	1	1	1	1	1	W	mod		0	0	1		mem		7/11	2-4	x		x	x	x	x
	reg16	0	1	0	0	1		reg										2	1	x		x	x	x	x
MULU	reg8	1	1	1	1	0	1	1	0	1	1	1	0	0		reg		8	2	u	x	x	u	u	u
	reg16	1	1	1	1	0	1	1	1	1	1	1	0	0		reg		12	2	u	x	x	u	u	u
	mem8	1	1	1	1	0	1	1	0	mod		1	0	0		mem		12	2-4	u	x	x	u	u	u
	mem16	1	1	1	1	0	1	1	1	mod		1	0	0		mem		16/18	2-4	u	x	x	u	u	u
MUL	reg8	1	1	1	1	0	1	1	0	1	1	1	0	1		reg		8	2	u	x	x	u	u	u
	reg16	1	1	1	1	0	1	1	1	1	1	1	0	1		reg		12	2	u	x	x	u	u	u
	mem8	1	1	1	1	0	1	1	0	mod		1	0	1		mem		12	2-4	u	x	x	u	u	u
	mem16	1	1	1	1	0	1	1	1	mod		1	0	1		mem		16/18	2-4	u	x	x	u	u	u
	reg16, reg16, imm8	0	1	1	0	1	0	1	1	1	1		reg			reg		12	3	u	x	x	u	u	u

Instruction Set (cont)

Mnemonic	Operand	Opcode															Clocks	Bytes	Flags					
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			AC	CY	V	P	S	Z
Arithmetic Instructions (cont)																									
MUL (cont)	reg16, mem16, imm8	0	1	1	0	1	0	1	1	mod			reg				mem	16/18	3-5	u	x	x	u	u	u
	reg16, reg16, imm16	0	1	1	0	1	0	0	1	1	1		reg				reg	12	4	u	x	x	u	u	u
	reg16, mem16, imm16	0	1	1	0	1	0	0	1	mod			reg				mem	16/8	4-6	u	x	x	u	u	u
DIVU	reg8	1	1	1	1	0	1	1	0	1	1	1	1	0			reg	11	2	u	u	u	u	u	u
	reg16	1	1	1	1	0	1	1	1	1	1	1	1	0			reg	19	2	u	u	u	u	u	u
	mem8	1	1	1	1	0	1	1	0	mod			1	1	0		mem	15	2-4	u	u	u	u	u	u
	mem16	1	1	1	1	0	1	1	1	mod			1	1	0		mem	23/25	2-4	u	u	u	u	u	u
DIV	reg8	1	1	1	1	0	1	1	0	1	1	1	1	1			reg	16	2	u	u	u	u	u	u
	reg16	1	1	1	1	0	1	1	1	1	1	1	1	1			reg	24	2	u	u	u	u	u	u
	mem8	1	1	1	1	0	1	1	0	mod			1	1	1		mem	20	2-4	u	u	u	u	u	u
	mem16	1	1	1	1	0	1	1	1	mod			1	1	1		mem	28/30	2-4	u	u	u	u	u	u
Comparison Instructions																									
CMP	reg, reg	0	0	1	1	1	0	1	W	1	1		reg				reg	2	2	x	x	x	x	x	x
	mem, reg	0	0	1	1	1	0	0	W	mod			reg				mem	6/8	2-4	x	x	x	x	x	x
	reg, mem	0	0	1	1	1	0	1	W	mod			reg				mem	6/8	2-4	x	x	x	x	x	x
	reg, imm	1	0	0	0	0	0	S	W	1	1	1	1	1			reg	2	3-4	x	x	x	x	x	x
	mem, imm	1	0	0	0	0	0	S	W	mod			1	1	1		mem	6/8	3-6	x	x	x	x	x	x
	acc, imm	0	0	1	1	1	1	0	W									2	2-3	x	x	x	x	x	x
Logical Instructions																									
NOT	reg	1	1	1	1	0	1	1	W	1	1	0	1	0			reg	2	2						
	mem	1	1	1	1	0	1	1	W	mod			0	1	0		mem	7/11	2-4						
NEG	reg	1	1	1	1	0	1	1	W	1	1	0	1	1			reg	2	2	x	x	x	x	x	x
	mem	1	1	1	1	0	1	1	W	mod			0	1	1		mem	7/11	2-4	x	x	x	x	x	x
TEST	reg, reg	1	0	0	0	0	1	0	W	1	1		reg				reg	2	2	u	0	0	x	x	x
	mem, reg	1	0	0	0	0	1	0	W	mod			reg				mem	6/8	2-4	u	0	0	x	x	x
	reg, imm	1	1	1	1	0	1	1	W	1	1	0	0	0			reg	2	3-4	u	0	0	x	x	x
	mem, imm	1	1	1	1	0	1	1	W	mod			0	0	0		mem	6/8	3-6	u	0	0	x	x	x
	acc, imm	1	0	1	0	1	0	0	W									2	2-3	u	0	0	x	x	x
AND	reg, reg	0	0	1	0	0	0	1	W	1	1		reg				reg	2	2	u	0	0	x	x	x
	mem, reg	0	0	1	0	0	0	0	W	mod			reg				mem	7/11	2-4	u	0	0	x	x	x
	reg, mem	0	0	1	0	0	0	1	W	mod			reg				mem	6/8	2-4	u	0	0	x	x	x
	reg, imm	1	0	0	0	0	0	0	W	1	1	1	0	0			reg	2	3-4	u	0	0	x	x	x
	mem, imm	1	0	0	0	0	0	0	W	mod			1	0	0		mem	7/11	3-6	u	0	0	x	x	x
	acc, imm	0	0	0	0	1	1	0	W									2	2-3	u	0	0	x	x	x
OR	reg, reg	0	0	0	0	1	0	1	W	1	1		reg				reg	2	2	u	0	0	x	x	x
	mem, reg	0	0	0	0	1	0	0	W	mod			reg				mem	7/11	2-4	u	0	0	x	x	x

Instruction Set (cont)

Mnemonic	Operand	Opcode 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Clocks	Bytes	Flags AC	CY	V	P	S	Z
Logical Instructions (cont)																									
OR (cont)	reg, mem	0	0	0	0	1	0	1	W	mod			reg			mem		6/8	2-4	u	0	0	x	x	x
	reg, imm	1	0	0	0	0	0	0	W	1	1	0	0	1		reg		2	3-4	u	0	0	x	x	x
	mem, imm	1	0	0	0	0	0	0	W	mod			0	0	1	mem		7/11	3-6	u	0	0	x	x	x
	acc, imm	0	0	0	0	1	1	0	W									2	2-3	u	0	0	x	x	x
XOR	reg, reg	0	0	1	1	0	0	1	W	1	1		reg			reg		2	2	u	0	0	x	x	x
	mem, reg	0	0	1	1	0	0	0	W	mod			reg			mem		7/11	2-4	u	0	0	x	x	x
	reg, mem	0	0	1	1	0	0	1	W	mod			reg			mem		6/8	2-4	u	0	0	x	x	x
	reg, imm	1	0	0	0	0	0	0	W	1	1	1	1	0		reg		2	3-4	u	0	0	x	x	x
	mem, imm	1	0	0	0	0	0	0	W	mod		1	1	0		mem		7/11	3-6	u	0	0	x	x	x
	acc, imm	0	0	1	1	0	1	0	W									2	2-3	u	0	0	x	x	x
Bit Manipulation Instructions																									
INS	reg8, reg8	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	1	37-61/	3						
		1	1		reg			reg										39-77							
	reg8, imm4	0	0	0	0	1	1	1	1	0	0	1	1	1	0	0	1	37-69/	4						
		1	1	0	0	0		reg										39-77							
EXT	reg8, reg8	0	0	0	0	1	1	1	1	0	0	1	1	0	0	1	1	29-61/	3						
		1	1		reg			reg										33-63							
	reg8, imm4	0	0	0	0	1	1	1	1	0	0	1	1	1	0	1	1	29-61/	4						
		1	1	0	0	0		reg										33-63							
TEST1	reg, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	W	4	3	u	0	0	u	u	x
		1	1	0	0	0		reg																	
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	0	8	3-5	u	0	0	u	u	x
		mod		0	0	0		mem																	
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	1	8/10	3-5	u	0	0	u	u	x
		mod		0	0	0		mem																	
	reg, imm3/4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	W	4	4	u	0	0	u	u	x
		1	1	0	0	0		reg																	
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	0	13	4-6	u	0	0	u	u	x
		mod		0	0	0		mem																	
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	0	1	8/10	4-6	u	0	0	u	u	x
		mod		0	0	0		mem																	
SET1	reg, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0	W	4	3						
		1	1	0	0	0		reg																	
	mem, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0	W	9	3-5						
		mod		0	0	0		mem																	
	reg, imm3/4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	W	4	4						
		1	1	0	0	0		reg																	
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	9	4-6						
		mod		0	0	0		mem																	

3e

Instruction Set (cont)

Mnemonic	Operand	Opcode																Clocks	Bytes	Flags					
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			AC	CY	V	P	S	Z
Bit Manipulation Instructions (cont)																									
SET1 (cont)	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	1	9/13	4-6						
		mod			0	0	0	mem																	
	CY	1	1	1	1	1	0	0	1									2	1		1				
	DIR	1	1	1	1	1	1	0	1									2	1						
CLR1	reg, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	W	4	3						
		1	1	0	0	0	reg																		
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	0	9	3-5						
		mod			0	0	0	mem																	
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	9/13	3-5						
		mod			0	0	0	mem																	
	reg, imm3/4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	W	4	4						
		1	1	0	0	0	reg																		
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	0	9	4-6						
		mod			0	0	0	mem																	
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	1	9/13	4-6						
		mod			0	0	0	mem																	
	CY	1	1	1	1	1	0	0	0									2	1		0				
	DIR	1	1	1	1	1	1	0	0									2	1						
NOT1	reg, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	W	4	3						
		1	1	0	0	0	reg																		
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	0	9	3-5						
		mod			0	0	0	mem																	
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1	9/13	3-5						
		mod			0	0	0	mem																	
	reg, imm3/4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	W	4	4						
		1	1	0	0	0	reg																		
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	W	9	4-6						
		mod			0	0	0	mem																	
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1	9/13	4-6						
		mod			0	0	0	mem																	
	CY	1	1	1	1	0	1	0	1									2	1		x				
Shift/Rotate Instructions																									
SHL	reg, 1	1	1	0	1	0	0	0	W	1	1	1	0	0	reg			2	2	u	x	x	x	x	x
	mem, 1	1	1	0	1	0	0	0	W	mod			1	0	0	mem		7/11	2-4	u	x	x	x	x	x
	reg, CL	1	1	0	1	0	0	1	W	1	1	1	0	0	reg			2 + n	2	u	x	x	x	x	x
	mem, CL	1	1	0	1	0	0	1	W	mod			1	0	0	mem		6/10 + n	2-4	u	x	x	x	x	x
	reg, imm8	1	1	0	0	0	0	0	W	1	1	1	0	0	reg			2 + n	3	u	x	x	x	x	x
																		n = number of shifts							

Instruction Set (cont)

Mnemonic	Operand	Opcode																Clocks	Bytes	Flags					
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			AC	CY	V	P	S	Z
Shift/Rotate Instructions (cont)																									
SHL (cont)	mem, imm8	1	1	0	0	0	0	0	W	mod			1	0	0	mem		6/10 + n	3-5	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	u	x	x	x	x	x
	mem, 1	1	1	0	1	0	0	0	W	mod			1	0	1	mem		7/11	2-4	u	x	x	x	x	x
	reg, CL	1	1	0	1	0	0	1	W	1	1	1	0	1	reg			2 + n	2	u	x	u	x	x	x
	mem, CL	1	1	0	1	0	0	1	W	mod			1	0	1	mem		6/10 + n	2-4	u	x	u	x	x	x
	reg, imm8	1	1	0	0	0	0	0	W	1	1	1	0	1	reg			2 + n	3	u	x	u	x	x	x
	mem, imm8	1	1	0	0	0	0	0	W	mod			1	0	1	mem		6/10 + n	3-5	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	u	x	0	x	x	x
	mem, 1	1	1	0	1	0	0	0	W	mod			1	1	1	mem		7/11	2-4	u	x	0	x	x	x
	reg, CL	1	1	0	1	0	0	1	W	1	1	1	1	1	reg			2 + n	2	u	x	u	x	x	x
	mem, CL	1	1	0	1	0	0	1	W	mod			1	1	1	mem		6/10 + n	2-4	u	x	u	x	x	x
	reg, imm8	1	1	0	0	0	0	0	W	1	1	1	1	1	reg			2 + n	3	u	x	u	x	x	x
	mem, imm8	1	1	0	0	0	0	0	W	mod			1	1	1	mem		6/10 + n	3-5	u	x	u	x	x	x
ROL	reg, 1	1	1	0	1	0	0	0	W	1	1	0	0	0	reg			2	2		x	x			
	mem, 1	1	1	0	1	0	0	0	W	mod			0	0	0	mem		7/11	2-4		x	x			
	reg, CL	1	1	0	1	0	0	1	W	1	1	0	0	0	reg			2 + n	2		x	u			
	mem, CL	1	1	0	1	0	0	1	W	mod			0	0	0	mem		6/10 + n	2-4		x	u			
	reg, imm	1	1	0	0	0	0	0	W	1	1	0	0	0	reg			2 + n	3		x	u			
	mem, imm	1	1	0	0	0	0	0	W	mod			0	0	0	mem		6/10 + n	3-5		x	u			
ROR	reg, 1	1	1	0	1	0	0	0	W	1	1	0	0	1	reg			2 + n	2		x	u			
	mem, 1	1	1	0	1	0	0	0	W	mod			0	0	1	mem		7/11	2-4		x	x			
	reg, CL	1	1	0	1	0	0	1	W	1	1	0	0	1	reg			7 + n	2		x	u			
	mem, CL	1	1	0	1	0	0	1	W	mod			0	0	1	mem		6/10 + n	2-4		x	u			
	reg, imm8	1	1	0	0	0	0	0	W	1	1	0	0	1	reg			2 + n	3		x	u			
	mem, imm8	1	1	0	0	0	0	0	W	mod			0	0	1	mem		6/10 + n	3-5		x	u			
ROLC	reg, 1	1	1	0	1	0	0	0	W	1	1	0	1	0	reg			2	2		x	x			
	mem, 1	1	1	0	1	0	0	0	W	mod			0	1	0	mem		7/11	2-4		x	x			
	reg, CL	1	1	0	1	0	0	1	W	1	1	0	1	0	reg			2 + n	2		x	u			
	mem, CL	1	1	0	1	0	0	1	W	mod			0	1	0	mem		6/10 + n	2-4		x	u			
	reg, imm8	1	1	0	0	0	0	0	W	1	1	0	1	0	reg			2 + n	3		x	u			
	mem, imm8	1	1	0	0	0	0	0	W	mod			0	1	0	mem		6/10 + n	3-5		x	u			
RORC	reg, 1	1	1	0	1	0	0	0	W	1	1	0	1	1	reg			2	2		x	x			
	mem, 1	1	1	0	1	0	0	0	W	mod			0	1	1	mem		7/11	2-4		x	x			
	reg, CL	1	1	0	1	0	0	1	W	1	1	0	1	1	reg			2 + n	2		x	u			
	mem, CL	1	1	0	1	0	0	1	W	mod			0	1	1	mem		6/10 + n	2-4		x	u			
	reg, imm8	1	1	0	0	0	0	0	W	1	1	0	1	1	reg			2 + n	3		x	u			

n = number of shifts

3e

Instruction Set (cont)

Mnemonic	Operand	Opcode																Clocks	Bytes	Flags					
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			AC	CY	V	P	S	Z
Shift/Rotate Instructions (cont)																									
RORC (cont)	mem, imm8	1	1	0	0	0	0	0	W	mod		0	1	1	mem			6/10 + n	3-5		x	u			
Stack Manipulation Instructions																									
PUSH	mem16	1	1	1	1	1	1	1	1	mod		1	1	0	mem			5/9	2-4						
	reg16	0	1	0	1	0	reg											3/5	1						
	sr	0	0	0	sr		1	1	0									3/5	1						
	PSW	1	0	0	1	1	1	0	0									3/5	1						
	R	0	1	1	0	0	0	0	0									20/36	1						
	imm	0	1	1	0	1	0	S	0									3/5	2-3						
POP	mem16	1	0	0	0	1	1	1	1	mod		0	0	0	mem			5/9	2-4						
	reg16	0	1	0	1	1	reg											5/7	1						
	sr	0	0	0	sr		1	1	1									5/7	1						
	PSW	1	0	0	1	1	1	0	1									5/7	1	R	R	R	R	R	R
	R	0	1	1	0	0	0	0	1									22/38	1						
PREPARE	imm16, imm8	1	1	0	0	1	0	0	0									*	4						
										*imm8 = 0:15 imm8≥ 1: 17 + 12 (imm8 – 1) odd, 15 + 8 (imm8-1) even															
DISPOSE		1	1	0	0	1	0	0	1									6/10	1						
Control Transfer Instructions																									
CALL	near_proc	1	1	1	0	1	0	0	0									7/9	3						
	regptr16	1	1	1	1	1	1	1	1	1	1	0	1	0	reg			7/9	2						
	memptr16	1	1	1	1	1	1	1	1	mod		0	1	0	mem			11/15	2-4						
	far_proc	1	0	0	1	1	0	1	0									9/13	5						
	memptr32	1	1	1	1	1	1	1	1	mod		0	1	1	mem			15/23	2-4						
RET		1	1	0	0	0	0	1	1									10/12	1						
	pop_value	1	1	0	0	0	0	1	0									10/12	3						
		1	1	0	0	1	0	1	1									12/16	1						
	pop_value	1	1	0	0	1	0	1	0									12/16	3						
BR	near_label	1	1	1	0	1	0	0	1									7	3						
	short_label	1	1	1	0	1	0	1	1									7	2						
	regptr16	1	1	1	1	1	1	1	1	1	1	1	0	0	reg			7	2						
	memptr16	1	1	1	1	1	1	1	1	mod		1	0	0	mem			11/13	2-4						
	far_label	1	1	1	0	1	0	1	0									7	5						
	memptr32	1	1	1	1	1	1	1	1	mod		1	0	1	mem			13/17	2-4						
BV	short_label	0	1	1	1	0	0	0	0									3/6	2						
BNV	short_label	0	1	1	1	0	0	0	1									3/6	2						
BC, BL	short_label	0	1	1	1	0	0	1	0									3/6	2						

n = number of shifts

Instruction Set (cont)

Mnemonic	Operand	Opcode 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Clocks	Bytes	Flags AC	CY	V	P	S	Z
Control Transfer Instructions (cont)																									
BNC, BNL	short_label	0	1	1	1	0	0	1	1									3/6	2						
BE, BZ	short_label	0	1	1	1	0	1	0	0									3/6	2						
BNE, BNZ	short_label	0	1	1	1	0	1	0	1									3/6	2						
BNH	short_label	0	1	1	1	0	1	1	0									3/6	2						
BH	short_label	0	1	1	1	0	1	1	1									3/6	2						
BN	short_label	0	1	1	1	1	0	0	0									3/6	2						
BP	short_label	0	1	1	1	1	0	0	1									3/6	2						
BPE	short_label	0	1	1	1	1	0	1	0									3/6	2						
BPO	short_label	0	1	1	1	1	0	1	1									3/6	2						
Interrupt Instructions																									
BLT	short_label	0	1	1	1	1	1	0	0									3/6	2						
BGE	short_label	0	1	1	1	1	1	0	1									3/6	2						
BLE	short_label	0	1	1	1	1	1	1	0									3/6	2						
BGT	short_label	0	1	1	1	1	1	1	1									3/6	2						
DBNZNE	short_label	1	1	1	0	0	0	0	0									3/6	2						
DBNZE	short_label	1	1	1	0	0	0	0	1									3/6	2						
DBNZ	short_label	1	1	1	0	0	0	1	0									3/6	2						
BCWZ	short_label	1	1	1	0	0	0	1	1									3/6	2						
BRK	3	1	1	0	0	1	1	0	0									18/24	1						
	imm8	1	1	0	0	1	1	0	1									18/24	2						
BRKV	imm8	1	1	0	0	1	1	1	0									20/26	1						
RETI		1	1	0	0	1	1	1	1									13/19	1	R	R	R	R	R	R
CHKIND	reg16, mem32	0	1	1	0	0	0	1	0	mod			reg			mem		24-26/ 30-32	2-4						
CPU Control Instructions																									
HALT		1	1	1	1	0	1	0	0									2	1						
BUSLOCK		1	1	1	1	0	0	0	0									2	1						
FP01	fp_op	1	1	0	1	1	X	X	X	1	1	Y	Y	Y	Z	Z	Z	*	2						
	fp_op, mem	1	1	0	1	1	X	X	X	mod		Y	Y	Y	mem			*	2-4						
FP02	fp_op	0	1	1	0	0	1	1	X	1	1	Y	Y	Y	Z	Z	Z	*	2						
	fp_op, mem	0	1	1	0	0	1	1	X	mod		Y	Y	Y	mem			*	2-4						
POLL		1	0	0	1	1	0	1	1									2 + 5n	1						
		n = number of times POLL pin is sampled.																							
NOP		1	0	0	1	0	0	0	0									3	1						
DI		1	1	1	1	1	0	1	0									2	1						
EI		1	1	1	1	1	0	1	1									2	1						
DS0:, DS1:, PS:, SS: (segment override prefixes)		0	0	1	seg		1	1	0									2	1						

3e

Instruction Set (cont)

Mnemonic	Operand	Opcode																Clocks	Bytes	Flags					
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			AC	CY	V	P	S	Z
Address Expansion Control Instructions																									
BRKXA	imm8	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	12	3						
					imm8																				
RETXA	imm8	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	12	3						
					imm8																				