NEC
NEC Electronics Inc.

μPD70320/70322 (V25) 16-Bit Microcomputers: Single-Chip, CMOS

Description

The μPD70320 and μPD70322 (V25™) are high-performance, 16-bit, single-chip microcomputers with an 8-bit external data bus. They combine the instruction set of the μPD70108 (V20®) with many of the on-chip peripherals in NEC's 78000 series.

The μPD70320/322 processor has software compatibility with the V20 (and subsequently the 8086/8088), faster memory accessing, superior interrupt processing ability, and enhanced control of internal peripherals.

A variety of on-chip components, including 16K bytes of mask programmable ROM (μPD70322 only), 256 bytes of RAM, serial and parallel I/O, comparator port lines, timers, and a DMA controller make the μPD70320/322 a sophisticated microsystem.

Eight banks of registers are mapped into internal RAM below an additional 256-byte special function register (SFR) area that is used to control on-chip peripherals. Internal RAM and the SFR area are together relocatable to anywhere in the 1M-byte address space. This maintains compatibility with existing system memory maps.

The μPD70322 is the mask ROM version, the μPD70320 is the ROM-less version, and the μPD70P322 is the EPROM version.

Features

- ☐ Complete single-chip microcomputer
  - — 16-bit ALU
  - — 16K bytes of ROM (μPD70322)
  - — 256 bytes of RAM
- ☐ 6-byte instruction prefetch queue
- ☐ 24 parallel I/O lines
- ☐ Eight analog comparator inputs with programmable threshold level
- ☐ Two independent DMA channels
- ☐ Two 16-bit timers
- ☐ Programmable time base counter
- ☐ Two full-duplex UARTs
- ☐ Programmable interrupt controller
  - — Eight priority levels
  - — Five external, 12 internal sources
  - — Register bank (eight) context switching
  - — Eight macro service function channels
- ☐ DRAM refresh pulse output
- ☐ Two standby modes
  - — HALT
  - — STOP
- ☐ Internal clock generator
  - — 5-MHz maximum CPU clock frequency (0.4-μs instruction cycle time)
  - — 8-MHz maximum CPU clock frequency (0.25-μs instruction cycle time)
- ☐ Programmable wait state generation
- ☐ Separate address/data bus interface
- ☐ CMOS technology

Ordering Information

Part Number	Clock (MHz)	Package Type	ROM
μPD70320L	5	84-pin PLCC	ROM-less
L-8	8		
GJ	5	94-pin plastic QFP	
GJ-8	8		
μPD70322L-xxx	5	84-pin PLCC	Mask ROM
L-8-xxx	8		
GJ-xxx	5	94-pin plastic QFP	
GJ-8-xxx	8		
μPD70P322KE-8	8	84-pin LCC	EPROM (UV erasable)

PLCC = plastic leaded chip carrier

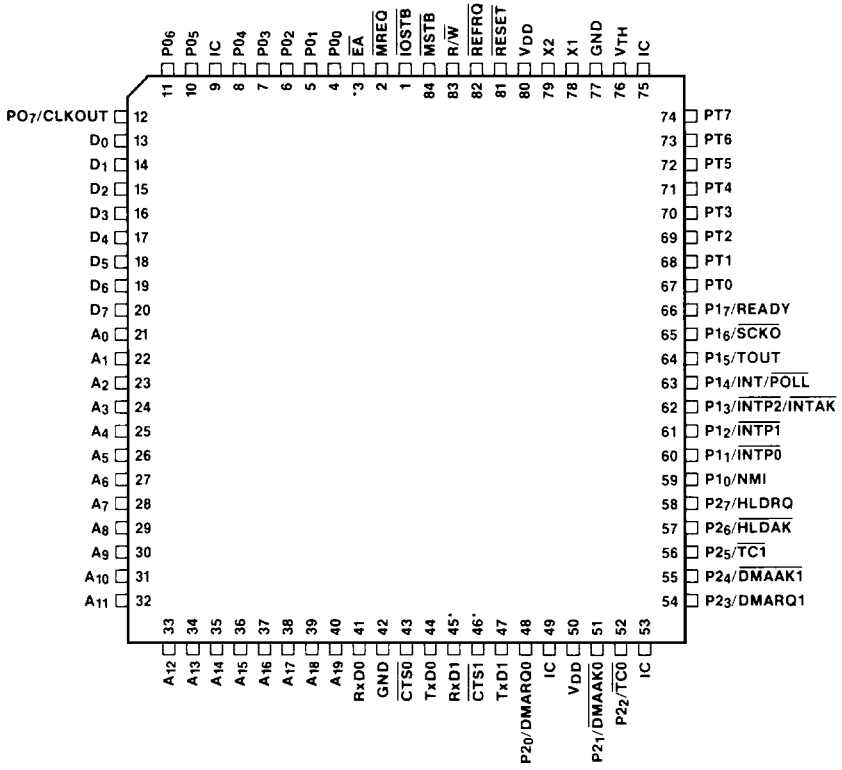
LCC = ceramic leadless chip carrier (with window)

4a

V20 is a registered trademark and V25 is a trademark of NEC Corporation.

Pin Configurations

84-Pin PLCC and LCC

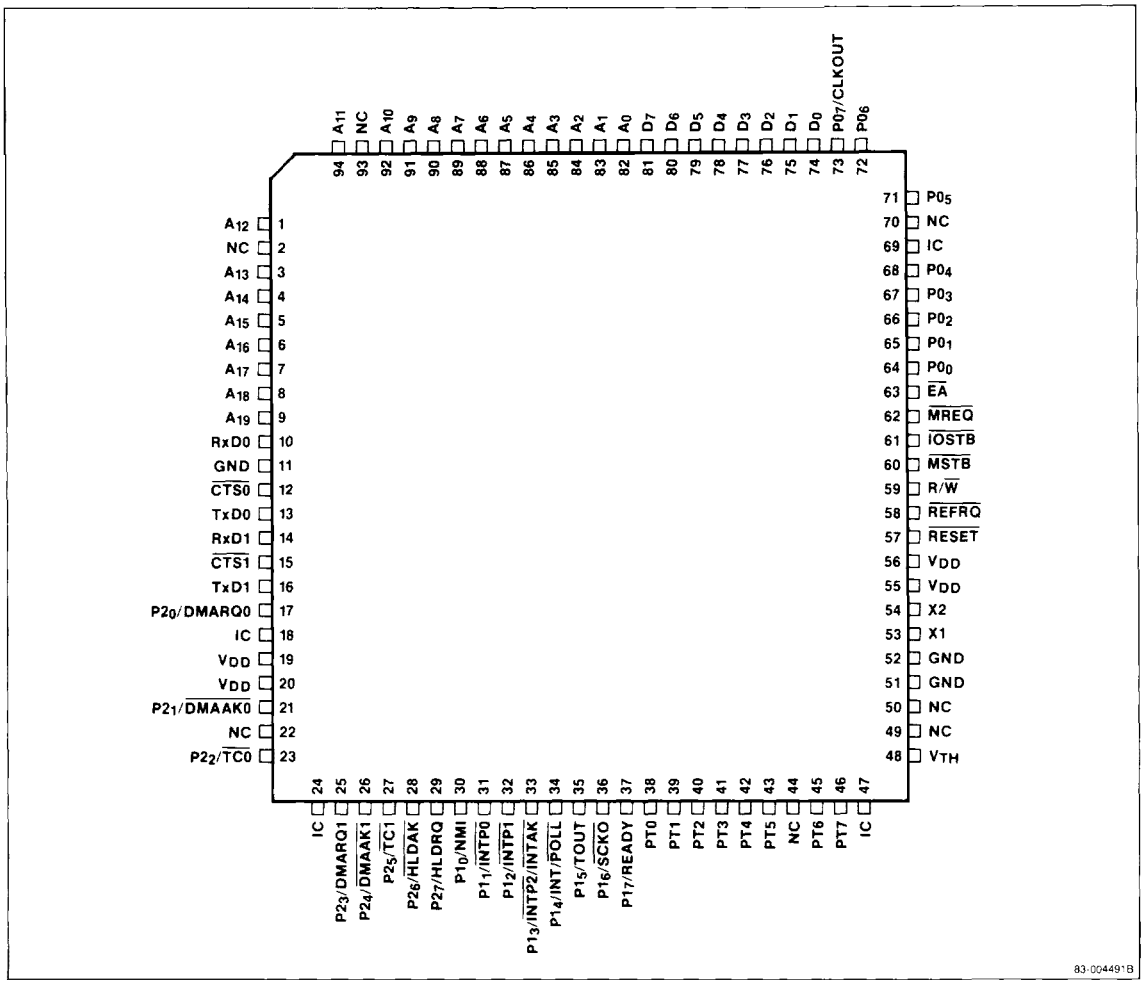


*Pin functions for normal operation of the μPD70P322 are changed as follows for programming.

Pin No.	Symbol	Function
3	V_{PP}	Write power supply input
45	$\overline{OE}$	Output enable signal input
46	$\overline{CE}$	Chip enable signal input

83-003960B

94-Pin Plastic QFP



4a

Pin Identification

Symbol	Function
A_0-A_{19}	Address bus outputs
CLKOUT	System clock output
$\overline{CTS0}$	Clear to send channel 0 input
$\overline{CTS1}$	Clear to send channel 1 input
D_0-D_7	Bidirectional data bus
$\overline{EA}$	External access
$\overline{IOSTB}$	I/O strobe output
$\overline{MREQ}$	Memory request output
$\overline{MSTB}$	Memory strobe output
$P0_0$-$P0_7$	I/O port 0
$P1_0$/NMI	Port 1 input line/Nonmaskable interrupt input
$P1_1$-$P1_2$/$\overline{INTP0}$-$\overline{INTP1}$	Port 1 input lines/External interrupt input lines
$P1_3$/$\overline{INTP2}$/$\overline{INTAK}$	Port 1 input line/External interrupt input line/Interrupt acknowledge output
$P1_4$/INT/$\overline{POLL}$	I/O port 1/Interrupt request input/I/O poll input
$P1_5$/TOUT	I/O port 1/Timer out
$P1_6$/$\overline{SCK0}$	I/O port 1/Serial clock out
$P1_7$/READY	I/O port 1/Ready input
$P2_0$/DMARQ0	I/O port 2/DMA request 0
$P2_1$/$\overline{DMAAK0}$	I/O port 2/DMA acknowledge 0
$P2_2$/$\overline{TC0}$	I/O port 2/DMA terminal count 0
$P2_3$/DMARQ1	I/O port 2/DMA request 1
$P2_4$/$\overline{DMAAK1}$	I/O port 2/DMA acknowledge 1
$P2_5$/$\overline{TC1}$	I/O port 2/DMA terminal count 1
$P2_6$/$\overline{HLDAK}$	I/O port 2/Hold acknowledge output
$P2_7$/HLDRQ	I/O port 2/Hold request input
PT0-PT7	Comparator port input lines
$\overline{REFRQ}$	Refresh pulse output
$\overline{RESET}$	Reset input
RxD0	Serial receive data, channel 0 input
RxD1	Serial receive data, channel 1 input
R/$\overline{W}$	Read/Write output
TxD0	Serial transmit data, channel 0 output
TxD1	Serial transmit data, channel 1 output
X1, X2	Crystal connection terminals
V_{DD}	Positive power supply voltage
V_{TH}	Threshold voltage input
GND	Ground
IC	Internal connection

Pin Functions

A_0-A_{19} [Address Bus]

A_0-A_{19} is the 20-bit address bus used to access all external devices.

CLKOUT [System Clock]

This is the internal system clock. It can be used to synchronize external devices to the CPU.

$\overline{CTSn}$, RxDn, TxDn, $\overline{SCKO}$ [Clear to Send, Receive Data, Transmit Data, Serial Clock Out]

The two serial ports (channels 0 and 1) use these lines for transmitting and receiving data, handshaking, and serial clock output.

D_0-D_7 [Data Bus]

D_0-D_7 is the 8-bit external data bus.

DMARQn, $\overline{DMAAKn}$, $\overline{TCn}$ [DMA Request, DMA Acknowledge, Terminal Count]

These are the control signals to and from the on-chip DMA controller.

$\overline{EA}$ [External Access]

If this pin is low on reset, the μPD70322 will execute program code from external memory instead of from internal ROM.

$\overline{HLDAK}$ [Hold Acknowledge]

The $\overline{HLDAK}$ output (active low) informs external devices that the CPU has released the system bus.

HLDRQ [Hold Request]

The HLDRQ input (active high) is used by external devices to request the CPU to release the system bus to an external bus master. The following lines go into a high-impedance state with internal 4.7-kΩ pullup resistors: A_0-A_{19}, D_0-D_7, $\overline{MREQ}$, R/$\overline{W}$, $\overline{MSTB}$, $\overline{REFRQ}$, and $\overline{IOSTB}$.

INT [Interrupt Request]

INT is a maskable, active-high, vectored interrupt request input. After assertion, external hardware must provide the interrupt vector number.

$\overline{\text{INTAK}}$ [Interrupt Acknowledge]

After INT is asserted, the CPU will respond with $\overline{\text{INTAK}}$ (active low) to inform external devices that the interrupt request has been granted.

$\overline{\text{INTP0}}$-$\overline{\text{INTP2}}$ [External Interrupt]

$\overline{\text{INTP0}}$-$\overline{\text{INTP2}}$ allow external devices to generate interrupts. Each can be programmed to be rising or falling edge triggered.

$\overline{\text{IOSTB}}$ [I/O Strobe]

$\overline{\text{IOSTB}}$ is asserted during read and write operations to external I/O.

$\overline{\text{MREQ}}$ [Memory Request]

$\overline{\text{MREQ}}$ (active low) informs external memory that the current bus cycle is a memory access bus cycle.

$\overline{\text{MSTB}}$ [Memory Strobe]

$\overline{\text{MSTB}}$ (active low) is asserted during read and write operations to external memory.

NMI [Nonmaskable Interrupt]

NMI cannot be masked through software and is typically used for emergency processing. Upon execution, the interrupt starting address is obtained from interrupt vector number 2. NMI can release the standby modes and can be programmed to be either rising or falling edge triggered.

$P0_0$-$P0_7$ [Port 0]

$P0_0$-$P0_7$ are the lines of port 0, an 8-bit bidirectional parallel I/O port.

$P1_0$-$P1_7$ [Port 1]

The status of $P1_0$-$P1_3$ can be read but these lines are always control functions. $P1_4$-$P1_7$ are the remaining lines of parallel port 1, each line individually programmable as either an input, an output, or a control function.

$P2_0$-$P2_7$ [Port 2]

$P2_0$-$P2_7$ are the lines of port 2, an 8-bit bidirectional I/O port. The lines can also be used as control signals for the on-chip DMA controller.

$\overline{\text{POLL}}$ [Poll]

Upon execution of the POLL intruction, the CPU checks the status of this pin and, if low, program execution continues. If high, the CPU will check the level of the line every five clock cycles until it is low. $\overline{\text{POLL}}$ can be used to synchronize program execution to external conditions.

PT0-PT7 [Comparator Port]

PT0-PT7 are inputs to the analog comparator port.

READY [Ready]

After READY is de-asserted low, the CPU will synchronize and insert at least two wait states into a read or write cycle to memory or I/O. This allows the processor to accommodate devices whose access times are longer than normal execution allows.

$\overline{\text{REFRQ}}$ [Refresh]

This active-low output pulse can refresh nonstatic RAM. It can be programmed to meet system specifications and is internally synchronized so that refresh cycles do not interfere with normal CPU operation.

$\overline{\text{RESET}}$ [Reset]

A low on $\overline{\text{RESET}}$ resets the CPU and all on-chip peripherals. $\overline{\text{RESET}}$ can also release the standby modes. After $\overline{\text{RESET}}$ returns high, program execution begins from address FFFF0H.

R/$\overline{\text{W}}$ [Read/Write]

An R/$\overline{\text{W}}$ output allows external hardware to determine if the current operation is a read or write cycle. It can also control the direction of bidirectional buffers.

TOUT [Timer Out]

TOUT is the square-wave output signal from the internal timer.

X1, X2 [Crystal Connections]

The internal clock generator requires an external crystal across these terminals as shown in figure 36. By programming the PRC register, the system clock frequency can be selected as the oscillator frequency (f_{OSC}) divided by 2, 4, or 8.

V_{DD} [Power Supply]

Two positive power supply pins (V_{DD}) reduce internal noise.

4a

V_{TH} [Threshold Voltage]

The comparator port uses this pin to determine the analog reference point. The actual threshold to each comparator line is programmable to V_{TH} x n/16, where n = 1 to 16.

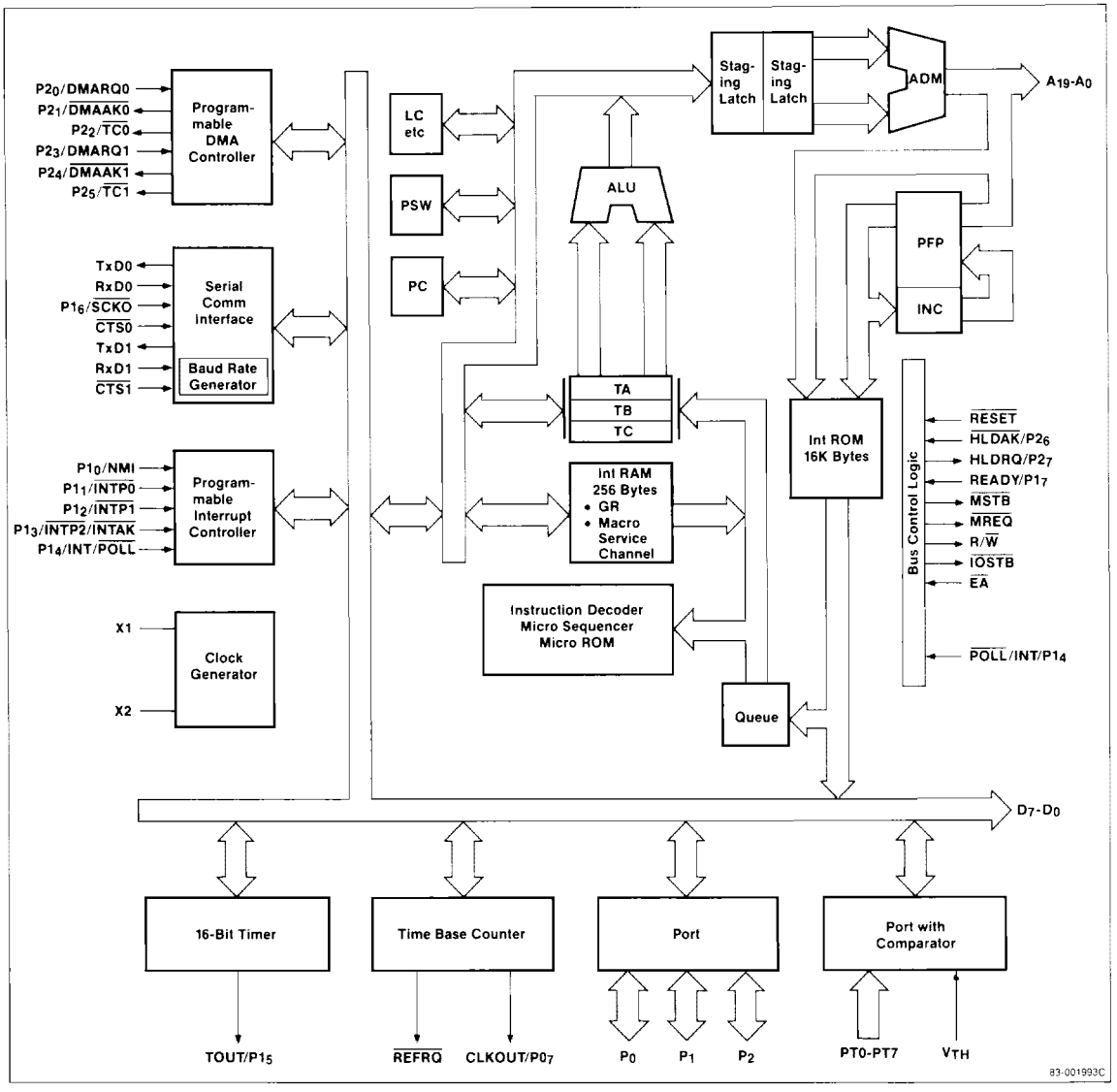
GND

Two ground connections reduce internal noise.

IC [Internal Connection]

All IC pins should be together and pulled up to V_{DD} with a 10K-20K resistor.

Block Diagram



Functional Description

Architectural Enhancements

The following features enable the μPD70320/322 to perform high-speed execution of instructions:

- Dual data bus
- 16-/32-bit temporary registers/shifters (TA, TB, TA + TB)
- 16-bit loop counter (LC)
- Program counter (PC) and prefetch pointer (PFP)
- Internal ROM pass bus (μPD70322 only)

Dual Data Bus. The μPD70320/322 has two internal 16-bit data buses: the main data bus and a subdata bus. This reduces the processing time required for addition/subtraction and logical comparison instructions by one-third over single-bus systems. The dual data bus method allows two operands to be fetched simultaneously from the general-purpose registers and transferred to the ALU.

16-/32-Bit Temporary Registers/Shifters. The 16-bit temporary registers/shifters (TA, TB) allow high-speed execution of multiplication/ division and shift/rotation instructions. By using the temporary registers/shifters, the μPD70320/322 can execute multiplication/division instructions about four times faster than with the microprogramming method.

Loop Counter [LC]. The dedicated hardware loop counter counts the number of loops for string operations and the number of shifts performed for multiple bit shift/rotation instructions. The loop counter works with internal dedicated shifters to speed the processing of multiplication/division instructions.

Program Counter and Prefetch Pointer [PC and PFP]. The hardware PC addresses the memory location of the instruction to be executed next. The hardware PFP addresses the program memory location to be accessed next. Several clocks are saved for branch, call, return, and break instructions compared with processors having only one instruction pointer.

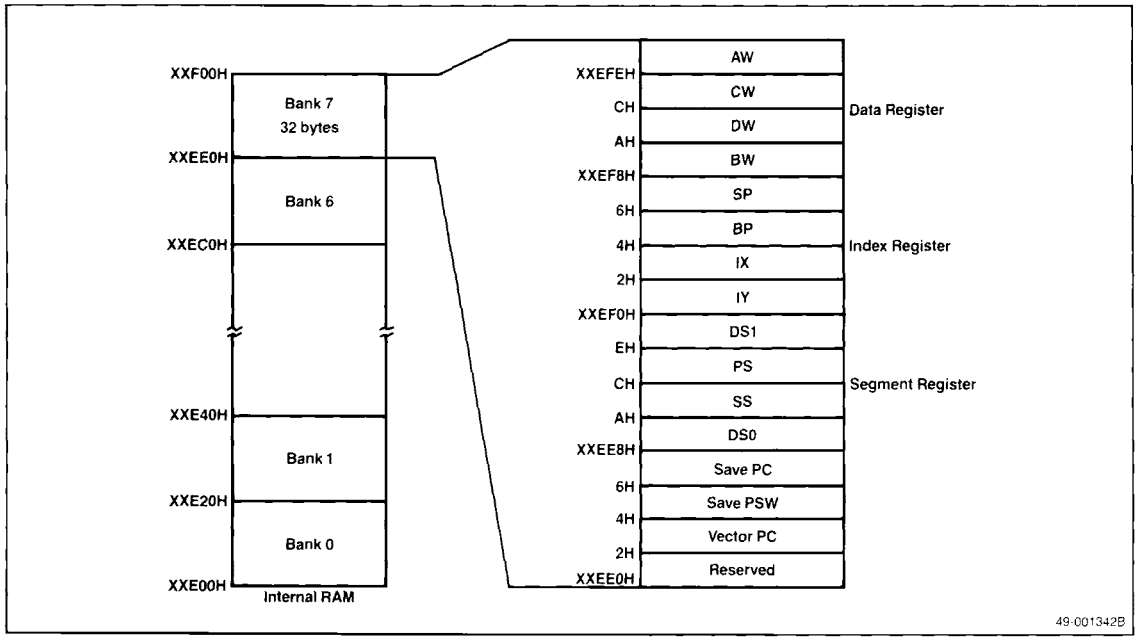
Internal ROM Pass Bus. The μPD70322 features a dedicated data bus between the internal ROM and the instruction pre-fetch queue. This allows internal ROM opcode fetches to be performed in a single clock cycle (200 ns at 5 MHz); it also makes it possible for opcode fetches to be performed while the external data bus is busy. This feature gives the V25 a 10-20% performance increase when executing from the internal ROM.

4a

Register Set

Figure 1 shows the μPD70320/322 has eight banks of registers functionally mapped into internal RAM. Each bank contains general-purpose registers, pointer and index registers, segment registers, and save areas.

Figure 1. Register Banks in Internal RAM



General-Purpose Registers [AW, BW, CW, DW]. There are four 16-bit general-purpose registers that can each serve as individual 16-bit registers or two independent 8-bit registers (AH, AL, BH, BL, CH, CL, DH, DL). The following instructions use the general-purpose registers for default:

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

Pointers [SP, BP] and Index Registers [IX, IY]. These registers are used as 16-bit base pointers or index registers in based addressing, indexed addressing, and based indexed addressing. The registers are used as default registers under the following conditions:

- SP Stack operations
- IX Block transfer (source), BCD string operations
- IY Block transfer (destination), BCD string operations

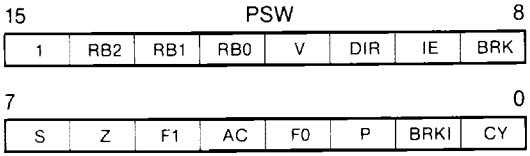
Segment Registers. The segment registers divide the 1M-byte address space into 64K-byte blocks. Each segment register functions as a base address to a block; the effective address is an offset from that base. Physical addresses are generated by shifting the associated segment register left four binary digits and then adding the effective address. The segment registers are:

Segment Register	Default Offset
PS (Program segment)	PC
SS (Stack segment)	SP, Effective address
DS0 (Data segment-0)	IX, Effective address
DS1 (Data segment-1)	IY, Effective address

Save Registers. Save PC and Save PSW are used as save areas during register bank context switching. The Vector PC save location contains the effective address of the interrupt service routine when register bank switching is used to service interrupts.

Program Counter [PC]. The PC is a 16-bit binary counter that contains the offset address from the program segment of the next instruction to be executed. It is incremented every time an instruction is received from the queue. It is loaded with a new location whenever a branch, call, return, break, or interrupt is executed.

Program Status Word [PSW]. The PSW contains the following status and control flags.



Status Flags

- V Overflow bit
- S Sign
- Z Zero
- AC Auxiliary carry
- P Parity
- CY Carry

Control Flags

- DIR Direction of string processing
- IE Interrupt enable
- BRK Break (after every instruction)
- RBn Current register bank flags
- BRKI I/O trap enable (see software interrupts)
- F0, F1 General-purpose user flags (accessed through the Flag special function register)

The eight low-order bits of the PSW can be stored in the AH register and restored by a MOV instruction execution. The only way to alter the RBn bits via software is to execute an RETRBI or RETI instruction.

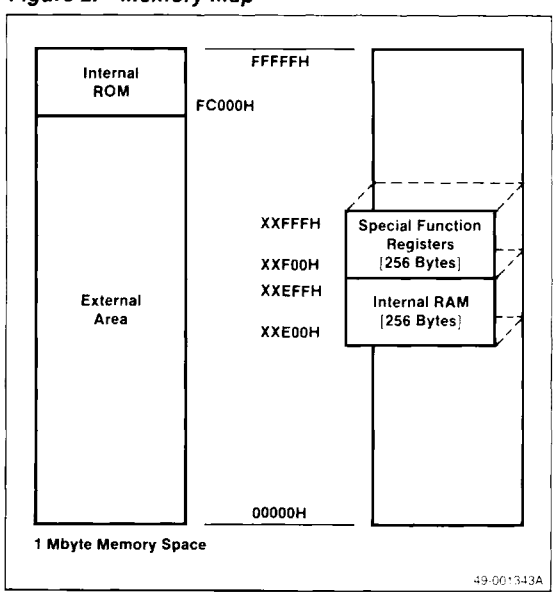
Memory Map

The μPD70320/322 has a 20-bit address bus that can directly access 1M bytes of memory. Figure 2 shows that the 16K bytes of internal ROM (μPD70322 only) are located at the top of the address space from FC000H to FFFFFH.

Figure 2 shows the internal data area (IDA) is a 256-byte internal RAM area followed consecutively by a 256-byte special function register (SFR) area. All the data and control registers for on-chip peripherals and I/O are mapped into the SFR area and accessed as RAM. For a description of these functions, see table 6. The IDA is dynamically relocatable in 4K-byte increments by changing the value in the internal data base (IDB) register. Whatever value is in this register will be assigned as the uppermost eight bits of the IDA address. The IDB register can be accessed from two different memory locations, FFFFFH and XXFFFH, where XX is the value in the IDB register.

On reset, the internal data base register is set to FFH which maps the IDA into the internal ROM space. However, since the μPD70322 has a separate bus to internal ROM, this does not present a problem. When these address spaces overlap, program code cannot be executed from the IDA and internal ROM locations cannot be accessed as data. You can select any of the eight possible register banks, which occupy the entire internal RAM space. Multiple register bank selection allows faster interrupt processing and facilitates multitasking.

Figure 2. Memory Map



In larger-scale systems where internal RAM is not required for data memory, the internal RAM can be removed completely from the address space and dedicated entirely to registers and control functions such as macro service and DMA channels. Clearing the RAMEN bit in the processor control register achieves this. When the RAMEN bit is cleared, internal RAM can only be accessed by register addressing or internal control processes. Many instructions are executed faster when the internal RAM is disabled.

Instruction Set

The μPD70320/322 instruction set is fully compatible with the V20 native mode instruction set. The V20 instruction set is a superset of the μPD8086/8088 instruction set with different execution times and mnemonics.

The μPD70320/322 does not support the V20 8080 emulation mode. All of the instructions pertaining to this have been deleted from the μPD70320/322 instruction set.

4a

Enhanced Instructions

In addition to the μPD8086/88 instructions, the μPD70320/322 has the following enhanced instructions.

Instruction	Function
PUSH imm	Pushes immediate data onto stack
PUSH R	Pushes eight general registers onto stack
POP R	Pops eight general registers from stack
MUL 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

Unique Instructions

The μPD70320/322 has the following unique instructions.

Instruction	Function
INS	Inserts bit field
EXT	Extracts bit field
ADD4S	Performs packed BCD string addition
SUB4S	Performs packed BCD string subtraction
CMP4S	Performs packed BCD string comparison
ROL4	Rotates BCD digit left
ROR4	Rotates BCD digit right
TEST1	Tests bit
SET1	Sets bit
CLR1	Clears bit
NOT1	Complements bit
BTCLR	Tests bit; if true, clear and branch
REPC	Repeat while carry set
REPNC	Repeat while carry cleared

Variable Length Bit Field Operation Instructions

Bit fields are a variable length data structure that can range in length from 1 to 16 bits. The μPD70320/322 supports two separate operations on bit fields: insertion (INS) and extraction (EXT). There are no restrictions on the position of the bit field in memory. Separate segment, byte offset, and bit offset registers are used for insertion and extraction. Following the execution of these instructions, both the byte offset and bit offset are left pointing to the start of the next bit field, ready for the next operation. Bit field operation instructions are powerful and flexible and are therefore highly effective for graphics, high-level languages, and packing/unpacking applications.

Bit field insertion copies the bit field of specified length from the AW register to the bit field addressed by DS1:IY:reg8 (8-bit general-purpose register). The bit field length can be located in any byte register or supplied as immediate data. Following execution, both the IY and reg8 are updated to point to the start of the next bit field.

Bit field extraction copies the bit field of specified length from the bit field addressed by DS0:IX:reg8 to the AW register. If the length of the bit field is less than 16 bits, the bit field is right justified with a zero fill. The bit field length can be located in any byte register or supplied as immediate data. Following execution, both IX and reg8 are updated to point to the start of the next bit field.

Figures 3 and 4 show bit field insertion and bit field extraction.

Packed BCD Instructions

Packed BCD instructions process packed BCD data either as strings (ADD4S, SUB4S, CMP4S) or byte format operands (ROR4, ROL4). Packed BCD strings may be 1 to 254 digits in length. The two BCD rotation instructions perform rotation of a single BCD digit in the lower half of the AL register through the register or the memory operand.

Bit Manipulation Instructions

The μPD70320/322 has five unique bit manipulation instructions. The ability to test, set, clear, or complement a single bit in a register or memory operand increases code readability as well as performance over the logical operations traditionally used to manipulate bit data. This feature further enhances control over on-chip peripherals.

Figure 3. Bit Field Insertion

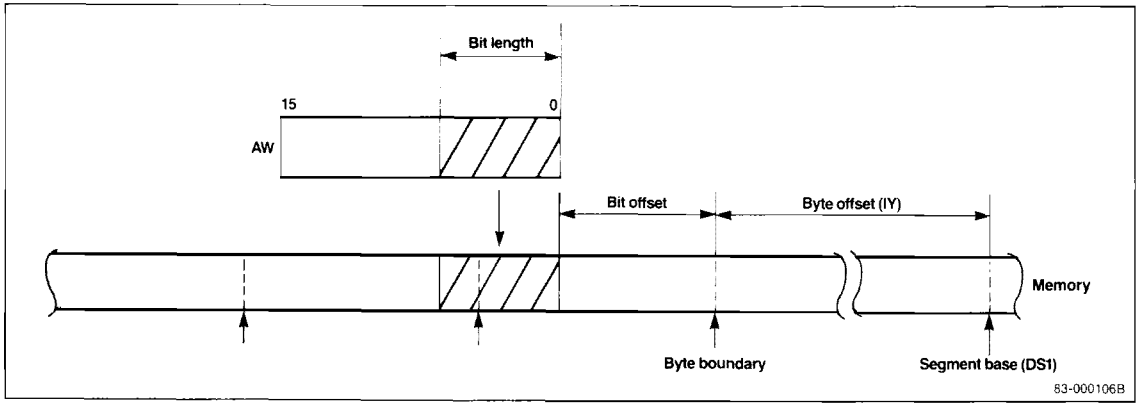
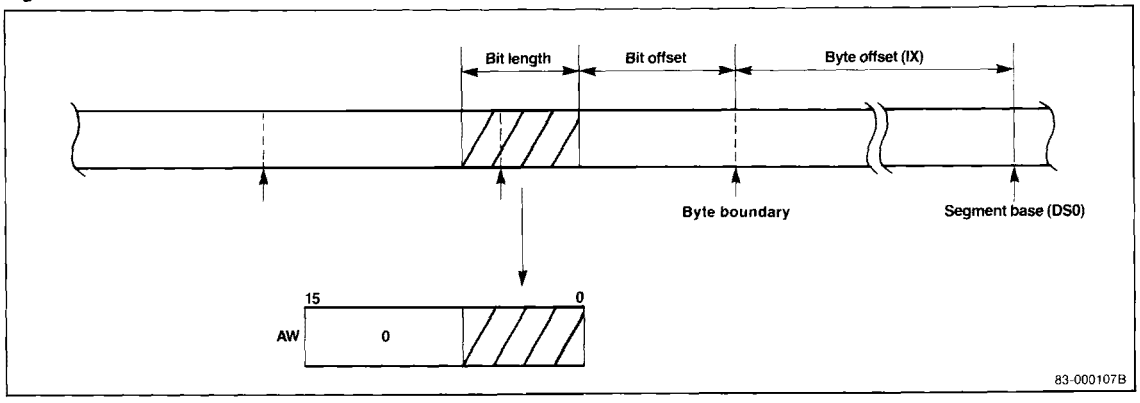


Figure 4. Bit Field Extraction



Additional Instructions

Besides the V20 instruction set, the μPD70320/322 has the four additional instructions described in table 1.

Table 1. Additional Instructions

Instruction	Function
BTCLR var,imm8, short label	Bit test and if true, clear and branch; otherwise, no operation
STOP (no operand)	Power down instruction, stops oscillator
RETRBI (no operand)	Return from register bank context switch interrupt
FINT (no operand)	Finished interrupt. After completion of a hardware interrupt request, this instruction must be used to reset the current priority bit in the in-service priority register (ISPR).*

*Do not use with NMI or INTR interrupt service routines.

Repeat Prefixes

Two new repeat prefixes (REPC, REPNC) allow conditional block transfer instructions to use the state of the CY flag as the termination condition. This allows inequalities to be used when working on ordered data, thus increasing performance when searching and sorting algorithms.

Bank Switch Instructions

The V25 has four new instructions that allow the effective use of the register banks for software interrupts and multitasking. These instructions are shown in table 2. Also, see figures 8 and 10.

Interrupt Structure

The μPD70320/322 can service interrupts generated both by hardware and by software. Software interrupts are serviced through vectored interrupt processing. See table 3 for the various types of software interrupts.

Table 2. Bank Switch Instructions

Instruction	Function
BRKCS reg 16	Performs a high-speed software interrupt with context switch to the register bank indicated by the lower 3-bits of reg 16. This operation is identical to the interrupt operation shown in figure 9.
TSKSW reg 16	Performs a high-speed task switch to the register bank indicated by the lower 3-bits of reg 16. The PC and PSW are saved in the old banks. PC and PSW save registers and the new PC and PSW values are retrieved from the new register bank's save areas. See figure 10.
MOVSPA	Transfers both the SS and SP of the old register bank to the new register bank after the bank has been switched by an interrupt or BRKCS instruction.
MOVSPB	Transfers the SS and the SP of the current register bank before the switch to the SS and SP of the new register bank indicated by the lower 3-bits of reg 16.

Table 3. Software Interrupts

Interrupt	Description
Divide error	The CPU will trap if a divide error occurs as the result of a DIV or DIVU instruction.
Single step	The interrupt is generated after every instruction if the BRK bit in the PSW is set.
Overflow	By using the BRKV instruction, an interrupt can be generated as the result of an overflow.
Interrupt instructions	The BRK 3 and BRK imm8 instructions can generate interrupts.
Array bounds	The CHKIND instruction will generate an interrupt if specified array bounds have been exceeded.
Escape trap	The CPU will trap on an FP01,2 instruction to allow software to emulate the floating point processor.
I/O trap	If the I/O trap bit in the PSW is cleared, a trap will be generated on every IN or OUT instruction. Software can then provide an updated peripheral address. This feature allows software interchangeability between different systems.

4a

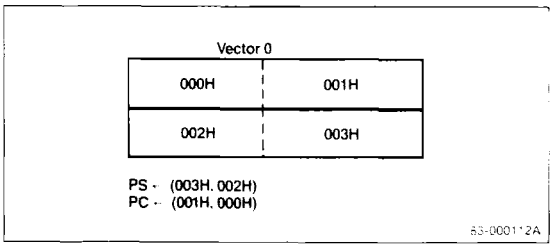
When executing software written for another system, it is better to implement I/O with on-chip peripherals to reduce external hardware requirements. However, since μPD70320/322 internal peripherals are memory mapped, software conversion could be difficult. The I/O trap feature allows easy conversion from external peripherals to on-chip peripherals.

Interrupt Vectors

The starting address of the interrupt processing routines may be obtained from table 3. The table begins at physical address 00H, which is outside the internal ROM space. Therefore, external memory is required to service these routines. By servicing interrupts via the macro service function or context switching, this requirement can be eliminated.

Each interrupt vector is four bytes wide. To service a vectored interrupt, the lower addressed word is transferred to the PC and the upper word to the PS. See figure 5.

Figure 5. Interrupt Vector 0



Execution of a vectored interrupt occurs as follows:

(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

Hardware Interrupt Configuration

The V25 features a high-performance on-chip controller capable of controlling multiple processing for interrupts from up to 17 different sources (5 external, 12 internal). The interrupt configuration includes system interrupts that are functionally compatible with those of the V20/V30 and unique high-performance microcontroller interrupts.

Table 4. Interrupt Vectors

Address	Vector No.	Assigned Use
00	0	Divide error
04	1	Break flag
08	2	NMI
0C	3	BRK3 instruction
10	4	BRKV instruction
14	5	CHKIND instruction
18	6	General purpose
1C	7	FPO instructions
20-2C	8-11	General purpose
30	12	INTSER0 (Interrupt serial error, channel 0)
34	13	INTSR0 (Interrupt serial receive, channel 0)
38	14	INTST0 (Interrupt serial transmit, channel 0)
3C	15	General purpose
40	16	INTSER1 (Interrupt serial error, channel 1)
44	17	INTSR1 (Interrupt serial receive, channel 1)
48	18	INTST1 (Interrupt serial transmit, channel 1)
4C	19	I/O trap
50	20	INTD0 (Interrupt from DMA, channel 0)
54	21	INTD1 (Interrupt from DMA, channel 1)
58	22	General purpose
5C	23	General purpose
60	24	INTP0 (Interrupt from peripheral 0)
64	25	INTP1 (Interrupt from peripheral 1)
68	26	INTP2 (Interrupt from peripheral 2)
6C	27	General purpose
70	28	INTTU0 (Interrupt from timer unit 0)
74	29	INTTU1 (Interrupt from timer unit 1)
78	30	INTTU2 (Interrupt from timer unit 2)
7C	31	INTTB (Interrupt from time base counter)
080-3FF	32-255	General purpose

Interrupt Sources

The 17 interrupt sources (table 5) are divided into groups for management by the interrupt controller. Using software, each of the groups can be assigned a priority from 0 (highest) to 7 (lowest). The priority of individual interrupts within a group is fixed in hardware. If interrupts from different groups occur simultaneously and the groups have the same assigned priority level, the priority followed will be as shown in the Default Priority column of table 5.

The ISPR is an 8-bit special function register; bits PR_0-PR_7 correspond to the eight possible interrupt request priorities. The ISPR keeps track of the priority of the interrupt currently being serviced by setting the appropriate bit. The address of the ISPR is XXFFCH. The ISPR format is shown below.

PR_7	PR_6	PR_5	PR_4	PR_3	PR_2	PR_1	PR_0

NMI and INTR are system-type external vectored interrupts. NMI is not maskable via software. INTR is maskable (IE bit in PSW) and requires that an external device provide the interrupt vector number. It allows expansion by the addition of an external interrupt controller (μPD71059).

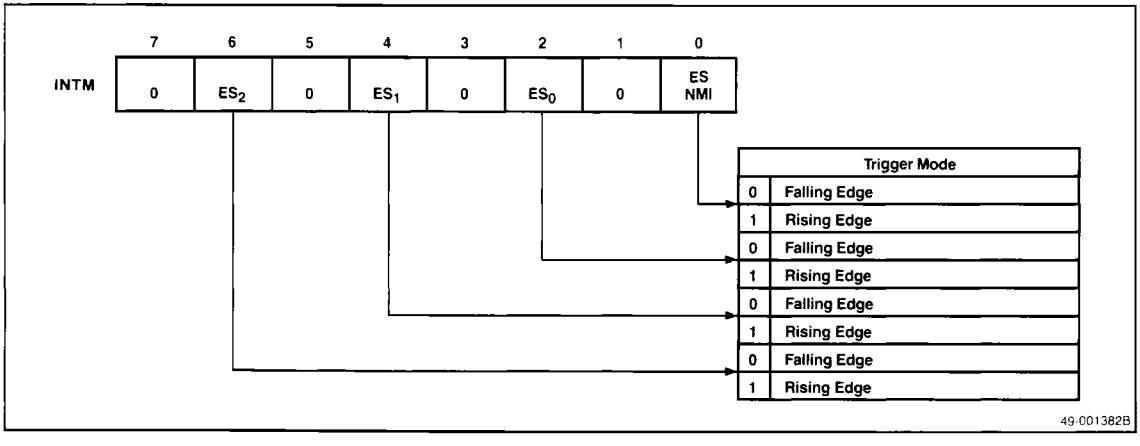
NMI, INTP0, and INTP1 are edge-sensitive interrupt inputs. By selecting the appropriate bits in the interrupt mode register, these inputs can be programmed to be either rising or falling edge triggered. ES_0-ES_2 correspond to INTP0-INTP2, respectively. See figure 6.

Table 5. Interrupt Sources

Group	Interrupt Source (Priority Within Group) 1	2	3	Default Priority
Non-maskable interrupt	NMI	—	—	0
Timer unit	INTTU0	INTTU1	INTTU2	1
DMA controller	INTD0	INTD1	—	2
External peripheral interrupt	INTP0	INTP1	INTP2	3
Serial channel 0	INTSER0	INTSR0	INTST0	4
Serial channel 1	INTSER1	INTSR1	INTST1	5
Time base counter	INTTB	—	—	6
Interrupt request	INTR	—	—	7

4a

Figure 6. Interrupt Mode Register (INTM)



Interrupt Processing Modes

Interrupts, with the exception of NMI, INT, and INTTB, have high-performance capability and can be processed in any of three modes: standard vector interrupt, register bank context switching, or macro service function. The processing mode for a given interrupt can be chosen by enabling the appropriate bits in the corresponding interrupt request control register. As shown in table 6, each individual interrupt, with the exception of INTR and NMI, has its own associated IRC register. The format for all IRC registers is shown in figure 7.

All interrupt processing routines other than those for NMI and INT must end with the execution of an FINT instruction. Otherwise, subsequently, only interrupts of a higher priority will be accepted.

In the vectored interrupt mode, the CPU traps to the vector location shown in table 4.

Register Bank Switching

Register bank context switching allows interrupts to be processed rapidly by switching register banks. After an interrupt, the new register bank selected is that which has the same register bank number (0-7) as the priority of the interrupt to be serviced. The PC and PSW are automatically stored in the save areas of the new register bank and the address of the interrupt routine is loaded from the vector PC storage location in the new register bank. As in the vectored mode, the IE and BRK bits in the PSW are cleared to zero. After interrupt processing, execution of the RETRBI (return from register bank interrupt) returns control to the former register bank and restores the former PC and PSW. Figures 8 and 9 show register bank context switching and register bank return.

Figure 7. Interrupt Request Control Registers (IRC)

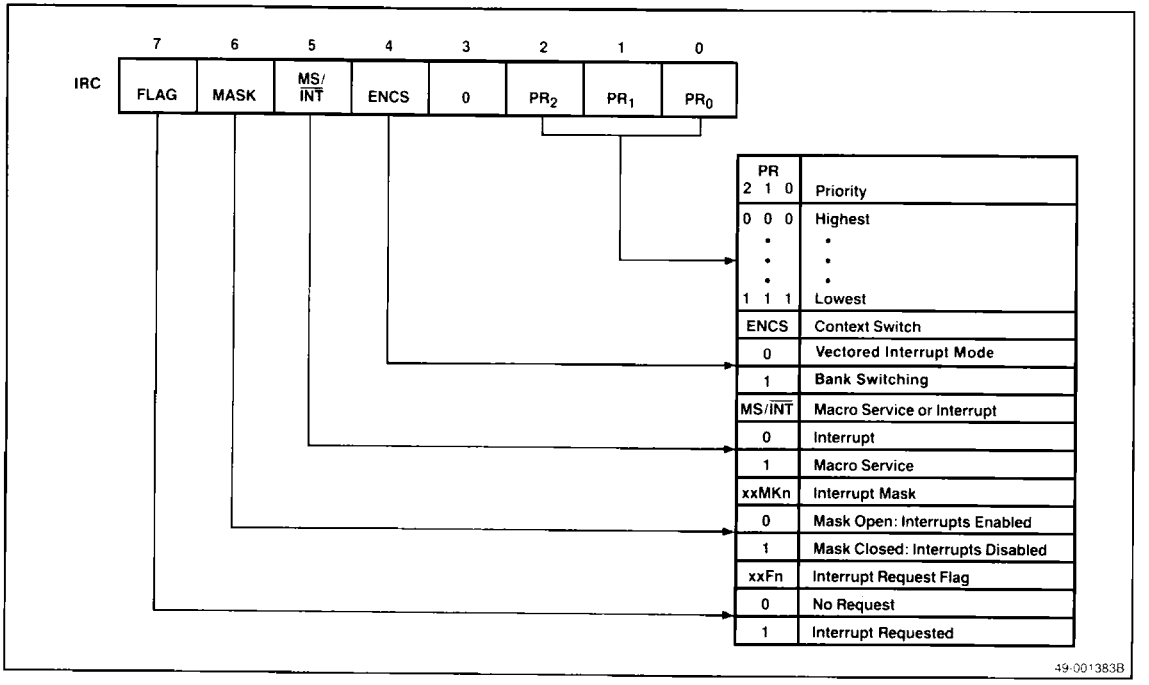


Figure 8. Register Bank Context Switching

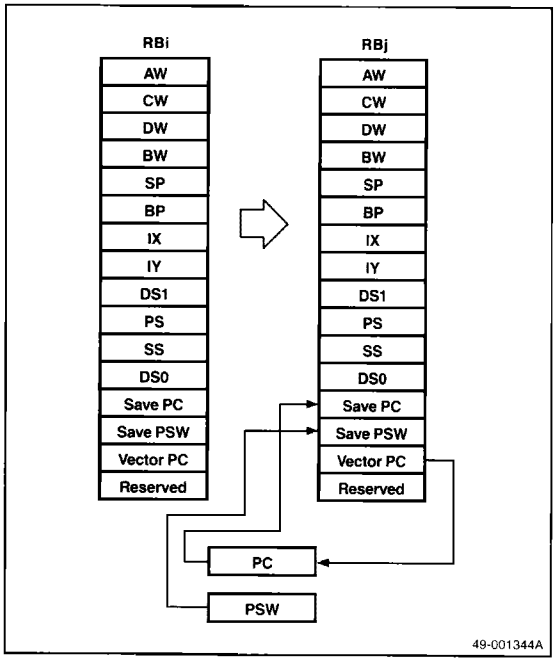
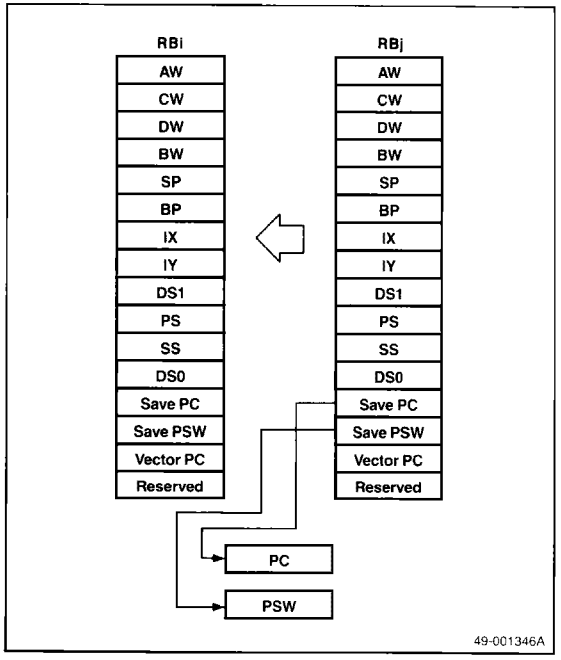


Figure 9. Register Bank Return



Macro Service Function

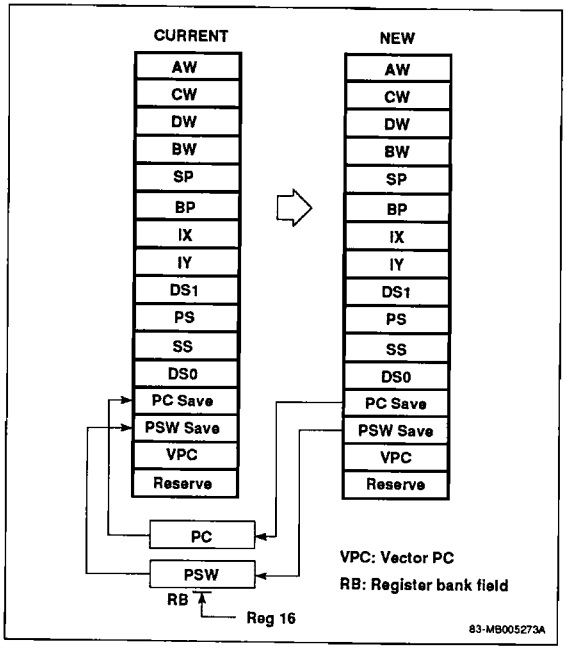
The macro service function (MSF) is a special microprogram that acts as an internal DMA controller between on-chip peripherals (special function registers, SFR) and memory. The MSF greatly reduces the software overhead and CPU time that other processors would require for register save processing, register returns, and other handling associated with interrupt processing.

If the MSF is selected for a particular interrupt, each time the request is received, a byte or word of data will be transferred between the SFR and memory without interrupting the CPU. Each time a request occurs, the macro service counter is decremented. When the counter reaches zero, an interrupt to the CPU is generated. The MSF also has a character search option. When selected, every byte transferred will be compared to an 8-bit search character and an interrupt will be generated if a match occurs or if the macro service counter counts out.

Like the NMI, INT and INTTB, the two DMA controller interrupts (INTD0, INTD1) do not have MSF capability.

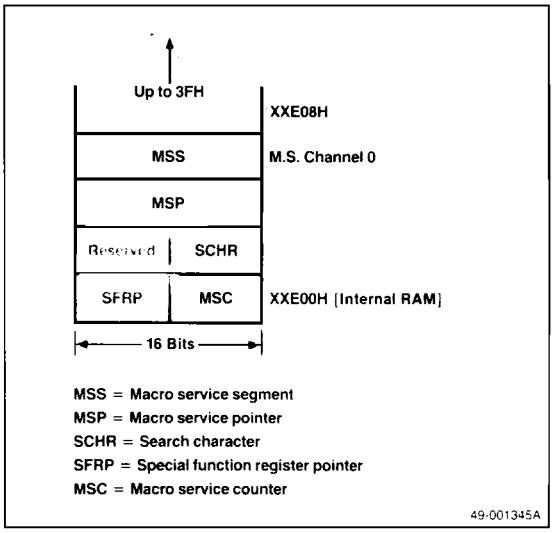
There are eight 8-byte macro service channels mapped into internal RAM from XXE00H to XXE3FH. Each macro service channel contains all of the necessary information to execute the macro service process. Figure 11 shows the components of each channel.

Figure 10. Task Switching



4a

Figure 11. Macro Service Channels



Setting the macro service mode for a given interrupt requires programming the corresponding macro service control register. Each individual interrupt, excluding INTR, NMI and TBC, has its own associated MSC register. See table 6. Format for all MSC registers is shown in figure 12.

On-Chip Peripherals

Timer Unit

The μPD70320/322 (figure 13) has two programmable 16-bit interval timers (TM0, TM1) on-chip, each with variable input clock frequencies. Each of the two 16-bit timer registers has an associated 16-bit modulus register (MD0, MD1). Timer 0 operates in the interval timer mode or one-shot mode; timer 1 has only the interval timer mode.

Interval Timer Mode. In this mode, TM0/TM1 are decremented by the selected input clock and, after counting out, the registers are automatically reloaded from the modulus registers and counting continues. Each time TM1 counts out, interrupts are generated through TF1 and TF2 (Timer Flags 1, 2). When TM0 counts out, an interrupt is generated through TF0. The timer-out signal can be used as a square-wave output whose half-cycle is equal to the count time. There are two selectable input clocks (SCLK: system clock = $f_{OSC}/2$; f_{OSC} = 10 MHz).

Clock	Timer Resolution	Full Count
SCLK/6	1.2 μs	78.643 ms
SCLK/128	25.6 μs	1.678 s

Figure 12. Macro Service Control Registers (MSC)

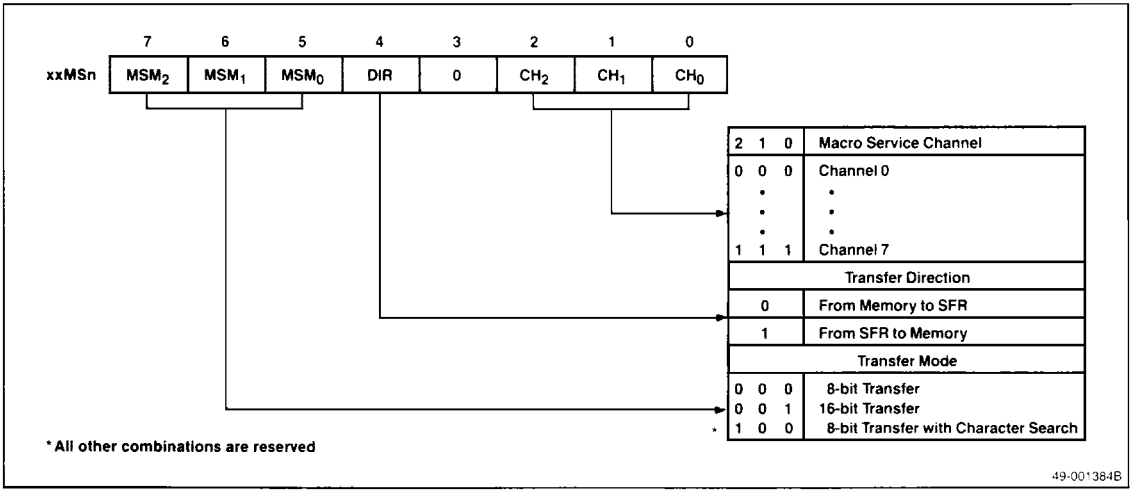
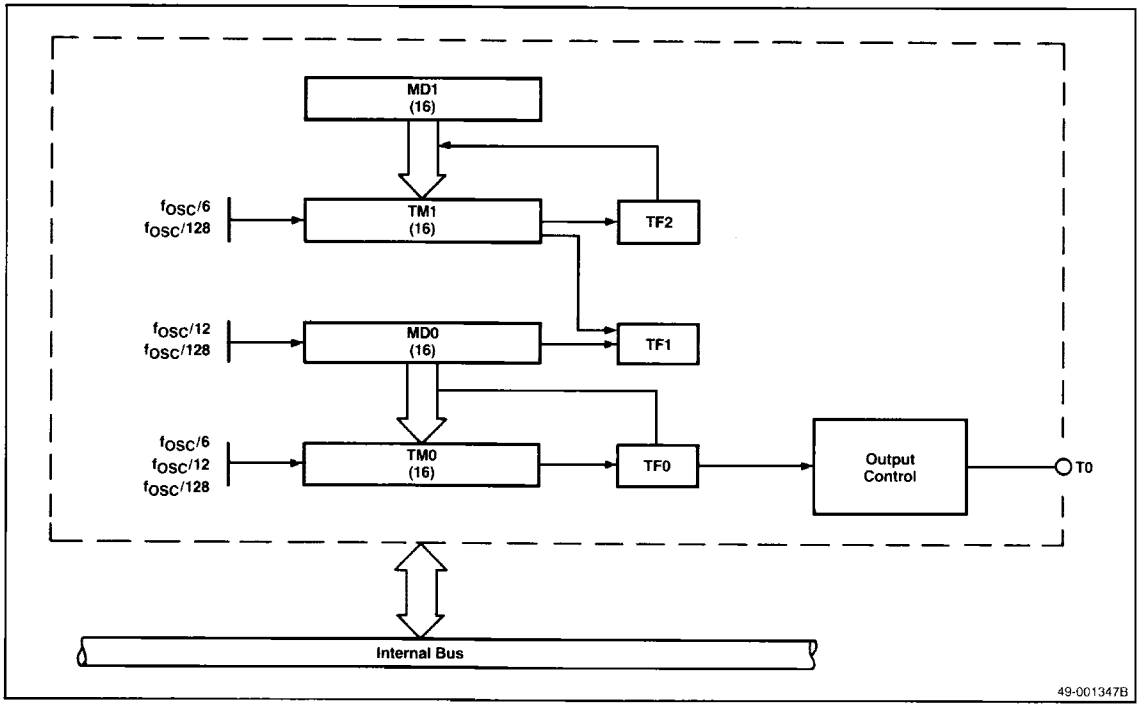


Figure 13. Timer Unit Block Diagram



One-Shot Mode. In the one-shot mode, TM0 and MD0 operate as independent one-shot timers. Starting with a preset value, each is decremented to zero. At zero, counting ceases and an interrupt is generated by TF0 (from TM0) or TF1 (from MD0). One-shot mode allows two selectable input clocks (f_{OSC} = 10 MHz).

Clock	Timer Resolution	Full Count
SCLK/12	2.4 μs	157.283 ms
SCLK/128	25.6 μs	1.678 s

Setting the desired timer mode requires programming the timer control register. See figures 14 and 15 for format.

Figure 14. Timer Control Register 0

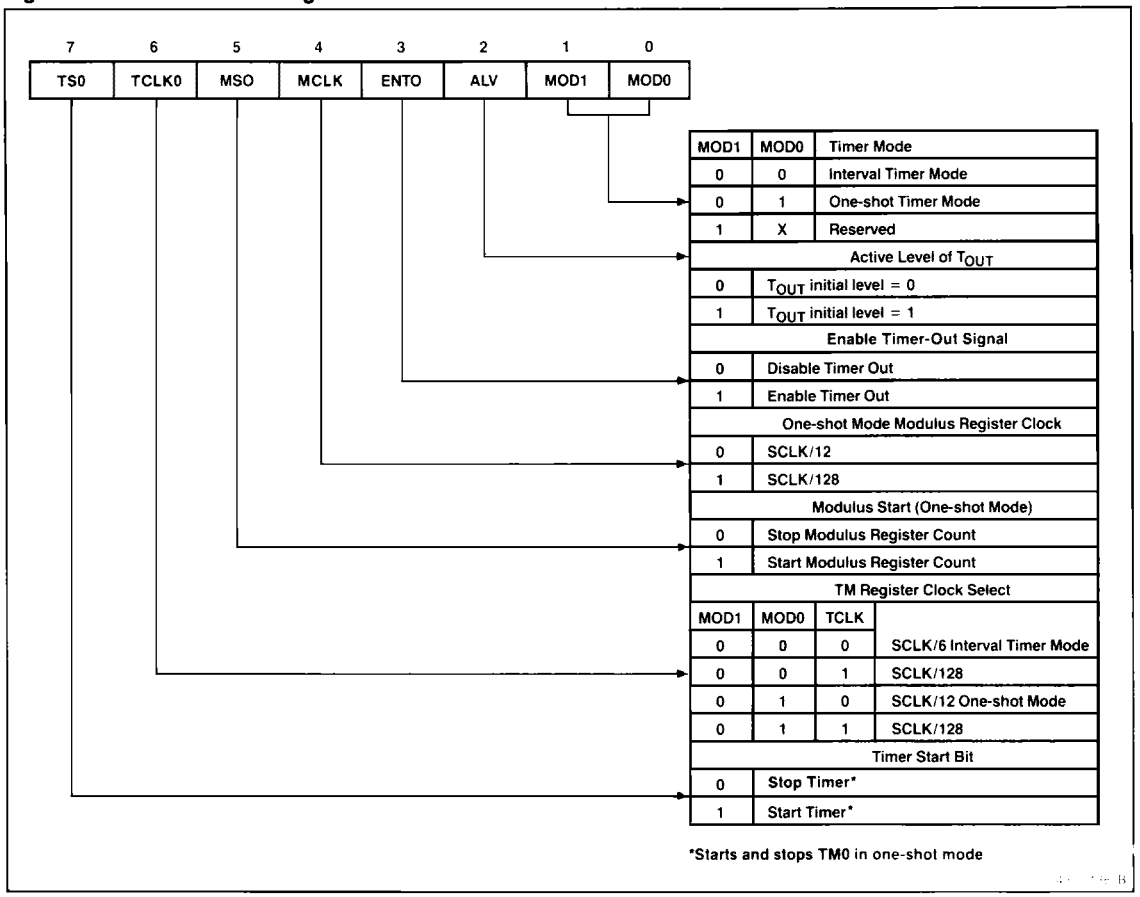
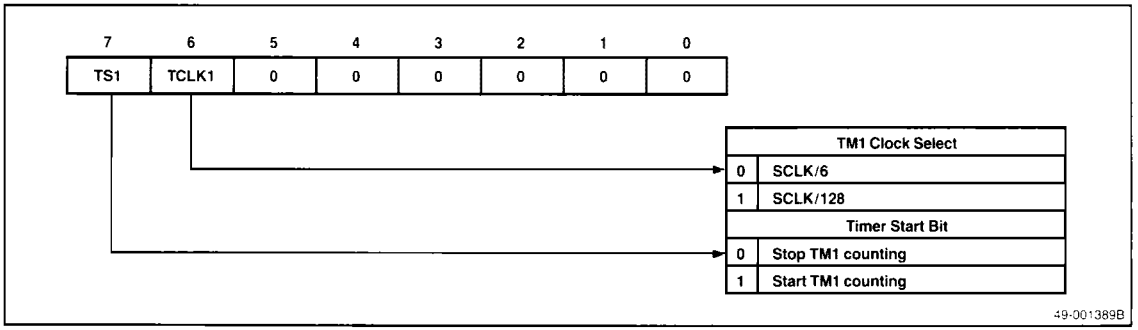


Figure 15. Timer Control Register 1



Time Base Counter/Processor Control Register

The 20-bit free-running time base counter controls internal timing sequences and is available to the user as the source of periodic interrupts at lengthy intervals. One of four interrupt periods can be selected by programming the TB0 and TB1 bits in the processor control register (PRC). The TBC interrupt is unlike the others in that it is fixed as a level 7 vectored interrupt. Macro service and register bank switching cannot be used to service this interrupt. See figures 16 and 17.

The RAMEN bit in the PRC register allows the internal RAM to be removed from the memory address space to implement faster instruction execution.

The TBC (figure 18) uses the system clock as the input frequency. The system clock can be changed by programming the PCK0 and PCK1 bits in the processor control register (PRC). Reset initializes the system clock to $f_{OSC}/8$ (f_{OSC} = external oscillator frequency).

Figure 18. Time Base Counter (TBC) Block Diagram

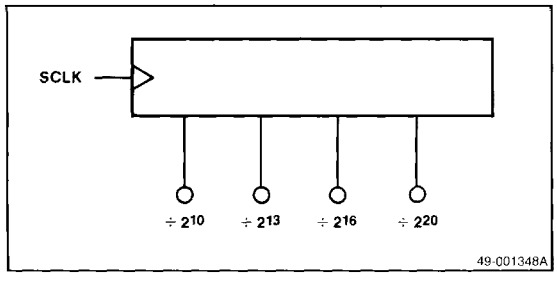


Figure 16. Time Base Interrupt Request Control Register

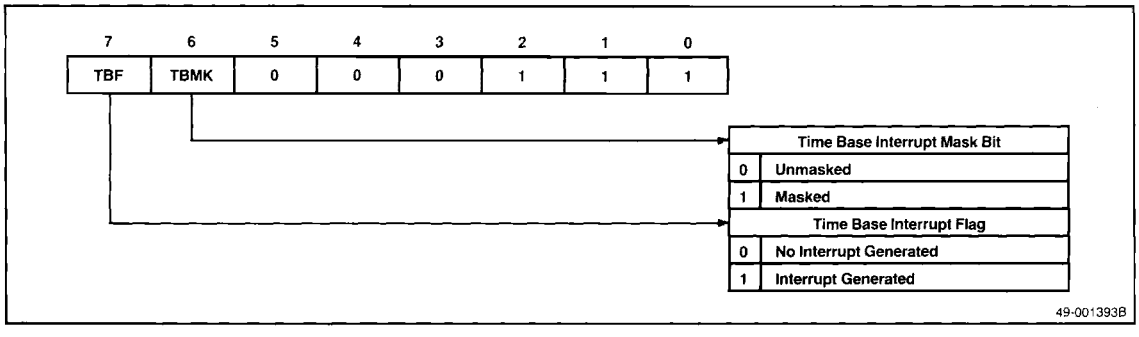
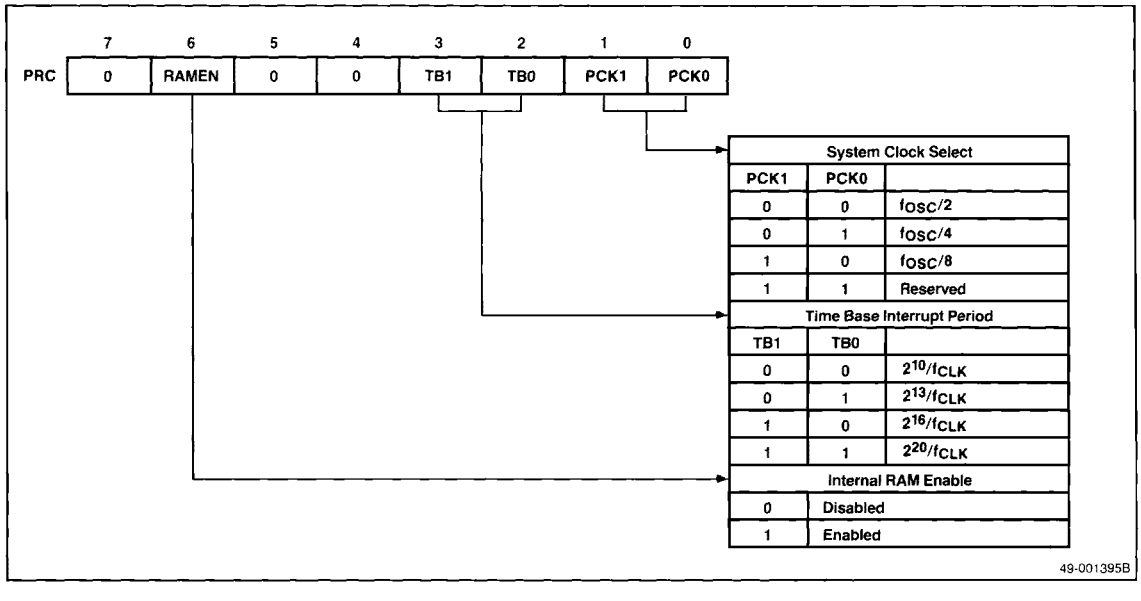


Figure 17. Processor Control Register (PRC)



Refresh Controller

The μPD70320/322 has an on-chip refresh controller for dynamic and pseudostatic RAM mass storage memories. The refresh controller generates refresh addresses and refresh pulses. It inserts refresh cycles between the normal CPU bus cycles according to refresh specifications.

The refresh controller outputs a 9-bit refresh address on address bits A_0-A_8 during the refresh bus cycle. Address bits A_9-A_{19} are all 1's. The 9-bit refresh address is automatically incremented at every refresh timing for 512 row addresses. The 8-bit refresh mode (RFM) register (figure 19) specifies the refresh operation and allows refresh during both CPU HALT and HOLD modes. Refresh cycles are automatically timed to $\overline{\text{REFRQ}}$ following read/write cycles to minimize the effect on system thoughput.

The following shows the $\overline{\text{REFRQ}}$ pin level in relation to bits 4 (RFEN) and 7 (RFLV) of the refresh mode register.

RFEN	RFLV	$\overline{\text{REFRQ}}$ Level
0	0	0
0	1	1
1	0	0
1	1	Refresh pulse output

Serial Interface

The μPD70320/322 has two full-duplex UARTs, channel 0 and channel 1. Each serial port channel has a transmit line (TxDn), a receive line (RxDn), and a clear to send (CTSn) input line for handshaking. Communication is synchronized by a start bit, and you can program the ports for even, odd, or no parity, character lengths of 7 or 8 bits, and 1 or 2 stop bits.

The μPD70320/322 has dedicated baud rate generators for each serial channel. This eliminates the need to obligate the on-chip timers. The baud rate generator allows a wide range of data transfer rates (up to 1.25 Mb/s). This includes all of the standard baud rates without being restricted by the value of the particular external crystal.

Each baud rate generator has an 8-bit baud rate generator (BRGn) data register, which functions as a prescaler to a programmable input clock selected by the serial communication control (SCCn) register. Together these must be set to generate a frequency equivalent to the desired baud rate.

The baud rate generator can be set to obtain the desired transmission rate according to the following formula:

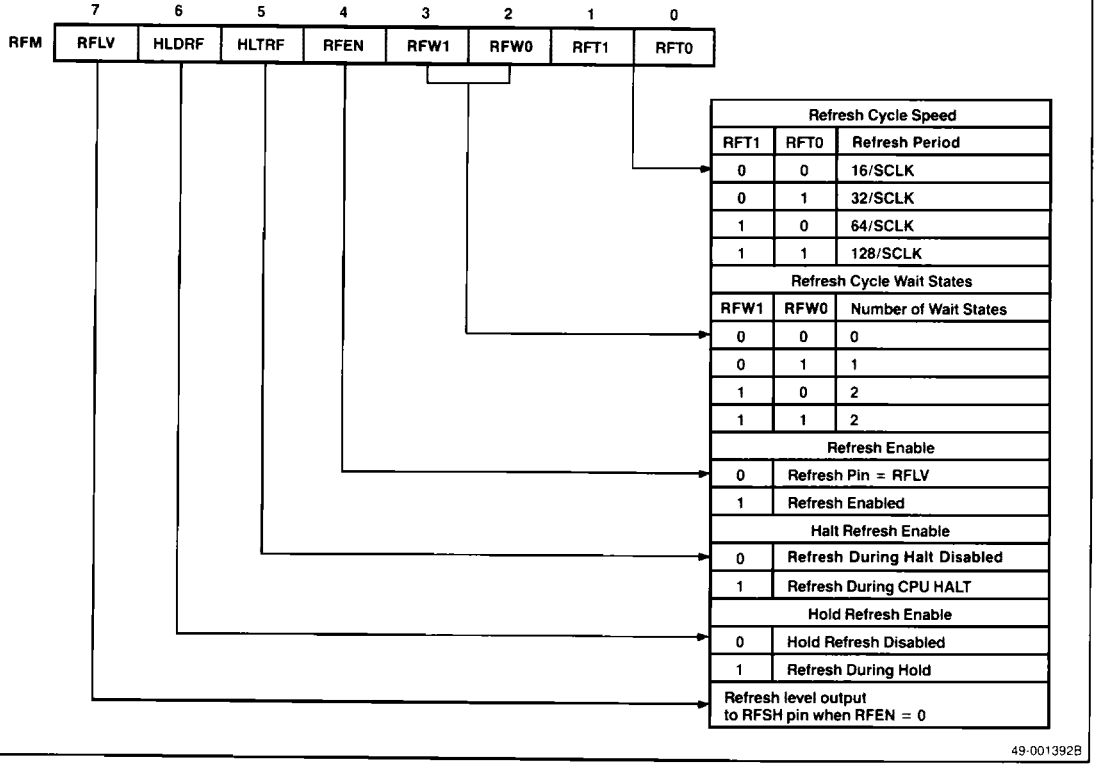
$$B \times G = \frac{SCLK \times 10^6}{2^{n+1}}$$

where B = baud rate
G = baud rate generator register (BRGn) value
n = input clock specifications (n between 0 and 8) This is the value that is loaded into the SCCn register (see figure 23).
SCLK = system clock frequency (MHz)

Based on the above expression, the following table shows the baud rate generator values used to obtain standard transmission rates when SCLK = 5 MHz.

Baud Rate	n	BRGn Value	Error (%)
110	7	178	0.25
150	7	130	0.16
300	6	130	0.16
600	5	130	0.16
1200	4	130	0.16
2400	3	130	0.16
4800	2	130	0.16
9600	1	130	0.16
19,200	0	130	0.16
38.400	0	65	0.16
1.25M	0	2	0

Figure 19. Refresh Mode Register (RFM)



Refresh Cycle Speed		
RFT1	RFT0	Refresh Period
0	0	16/SCLK
0	1	32/SCLK
1	0	64/SCLK
1	1	128/SCLK

Refresh Cycle Wait States		
RFW1	RFW0	Number of Wait States
0	0	0
0	1	1
1	0	2
1	1	2

Refresh Enable	
0	Refresh Pin = RFLV
1	Refresh Enabled

Halt Refresh Enable	
0	Refresh During Halt Disabled
1	Refresh During CPU HALT

Hold Refresh Enable	
0	Hold Refresh Disabled
1	Refresh During Hold

Refresh level output to RFSH pin when RFEN = 0

4a

In addition to the asynchronous mode, channel 0 has a synchronous I/O interface mode. In this mode, each bit of data tranferred is synchronized to a serial clock ($\overline{SCKO}$). This is the same as the NEC μCOM75 and μCOM87 series, and allows easy interfacing to these devices. Figure 20 is the serial interface block diagram; figures 21, 22, and 23 show the three serial communication registers.

Figure 20. Serial Interface Block Diagram

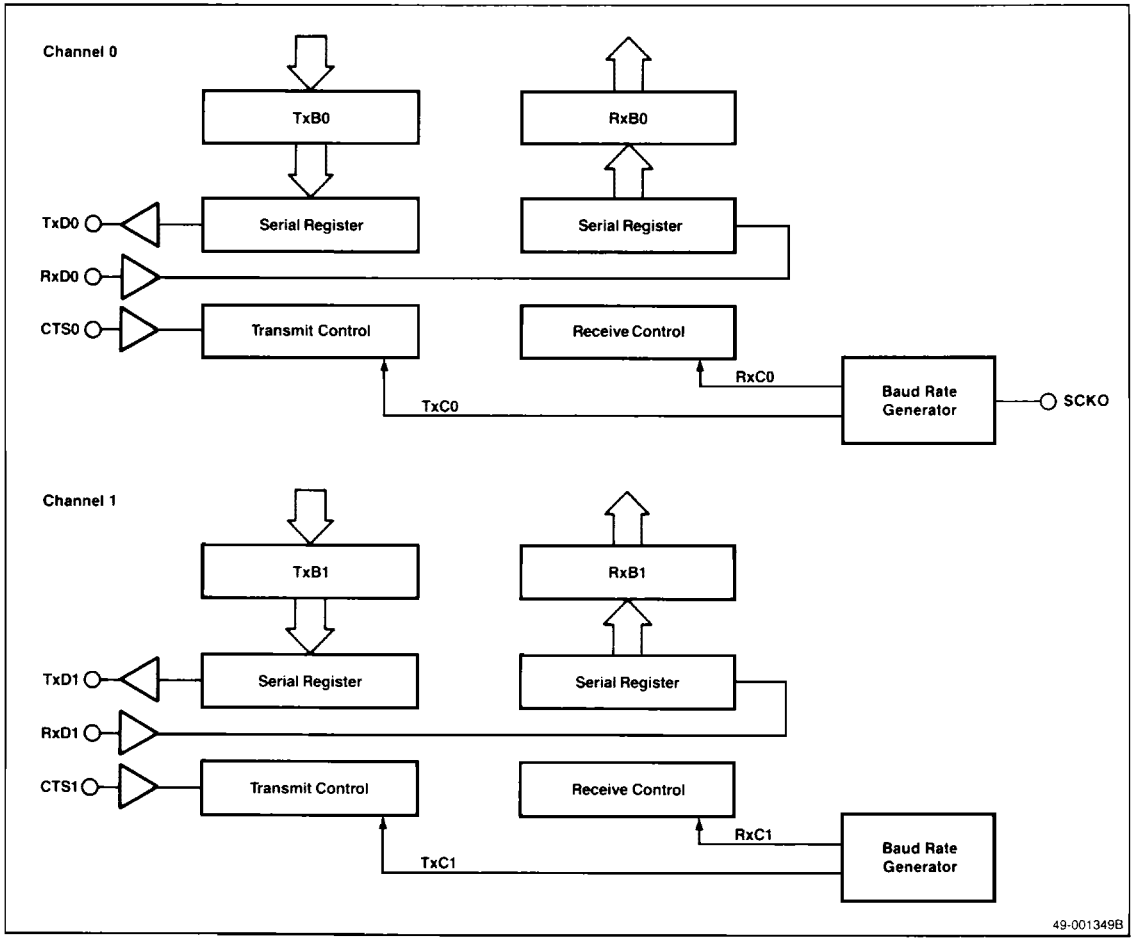


Figure 21. Serial Communication Mode Register (SCM)

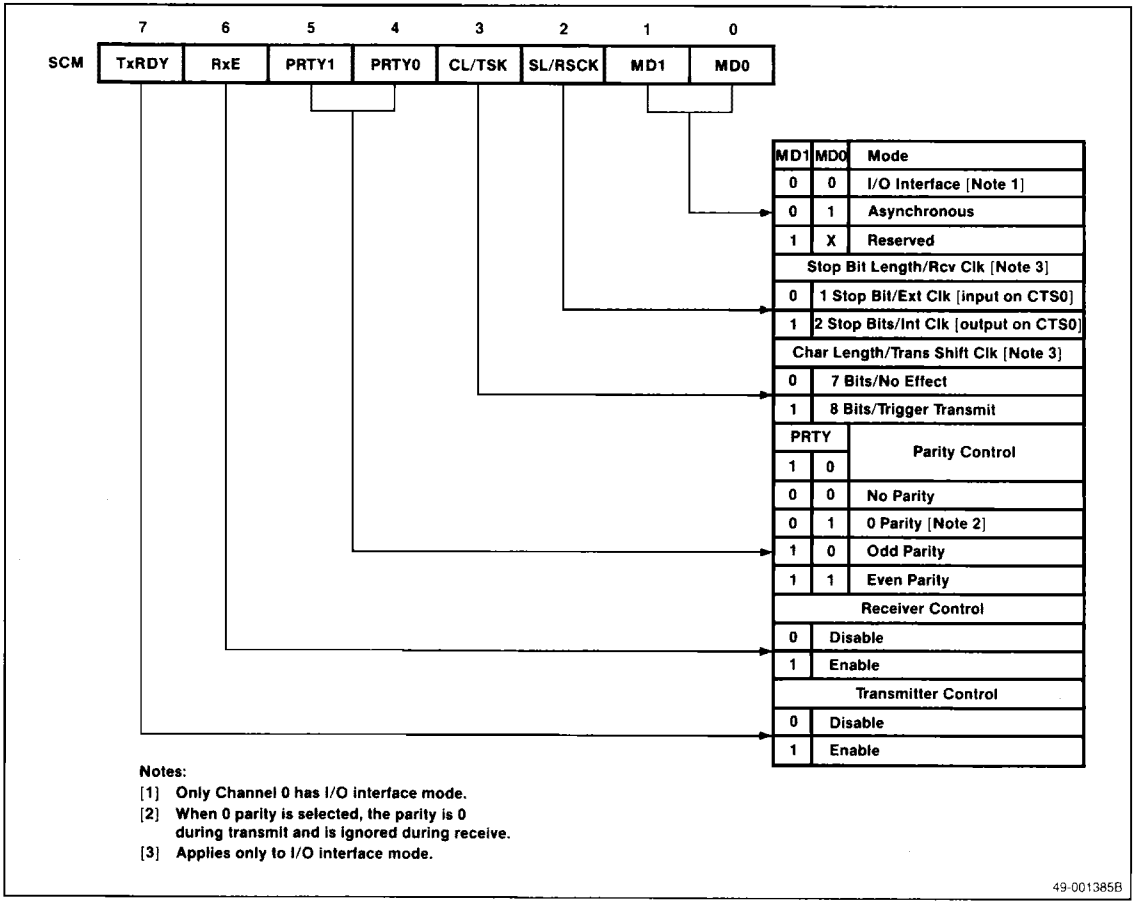


Figure 22. Serial Communication Error Registers (SCE)

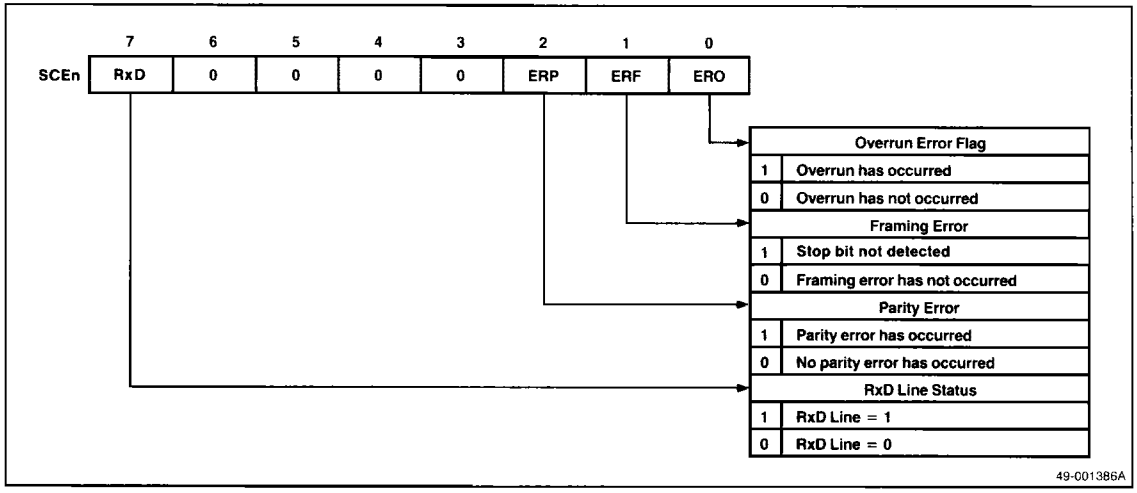
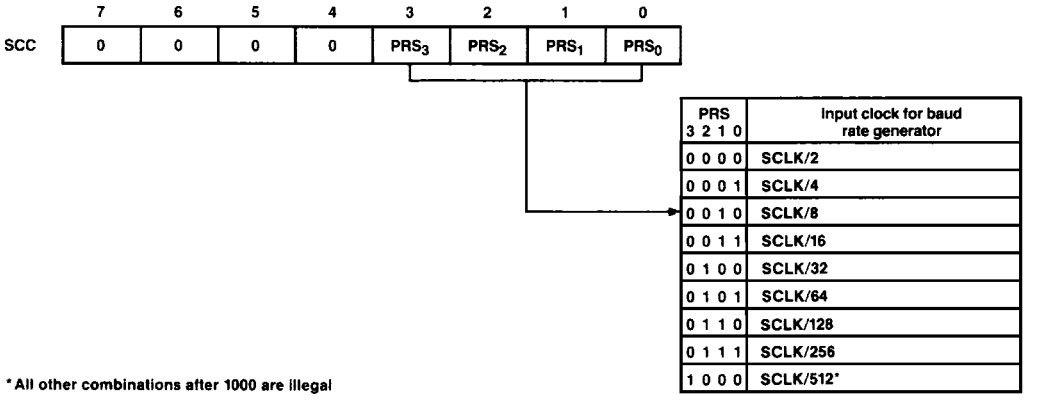


Figure 23. Serial Communication Control Register (SCC)

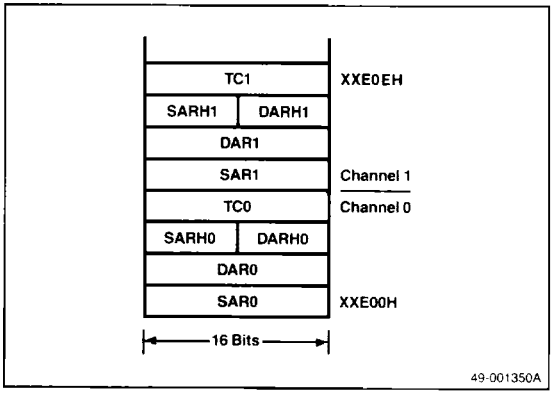


DMA Controller

The μPD70320/322 has a two-channel, on-chip DMA controller. This allows rapid data transfer between memory and auxiliary storage devices. The DMA controller supports four modes of operation, two for memory-to-memory transfers and two for transfers between I/O and memory. See figures 24, 25, and 26 for a graphic representation of the DMA registers.

Memory-to-Memory Transfers. In the single-step mode, when one DMA request is made, execution of one instruction and one DMA transfer are repeated alternately until the prescribed number of DMA transfers has occurred. Interrupts can be accepted while in this mode. In burst mode, one DMA request causes DMA transfer cycles to continue until the DMA terminal counter decrements to zero. Software can also initiate memory-to-memory transfers.

Figure 24. DMA Channels



Transfers Between I/O and Memory. In single-transfer mode, one DMA transfer occurs after each rising edge of DMARQ. After the transfer, the bus is returned to the CPU. In demand release mode, the rising edge of DMARQ enables DMA cycles, which continue as long as DMARQ is high.

In all modes, the $\overline{TC}$ (terminal count) output pin will pulse low and a DMA completion interrupt request will be generated after the predetermined number of DMA cycles has been completed.

The bottom of internal RAM contains all the necessary address information for the designated DMA channels. The DMA channel mnemonics are as follows:

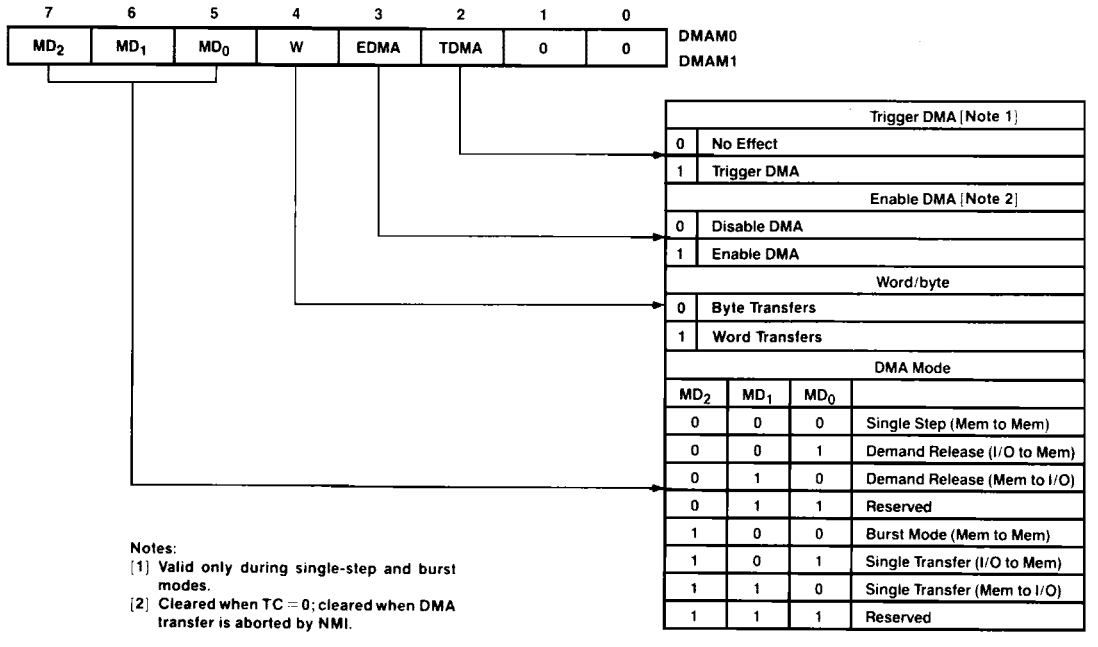
TC	Terminal counter
SAR	Source address register
SARH	Source address register high
DAR	Destination address register
DARH	Destination address register high

The DMA controller generates physical source addresses by offsetting SARH 12 bits to the left and then adding the SAR. The same procedure is also used to generate physical destination addresses. You can program the controller to increment or decrement source and/or destination addresses independently during DMA transfers.

When the EDMA bit is set, the internal DMARQ flag is cleared. Therefore, DMARQs are only recognized after the EDMA bit has been set.

See Execution Clock Counts for Operation and Bus Controller Latency tables for DMA latency and transfer rate information.

Figure 25. DMA Mode Registers (DMAM)



Trigger DMA [Note 1]	
0	No Effect
1	Trigger DMA

Enable DMA [Note 2]	
0	Disable DMA
1	Enable DMA

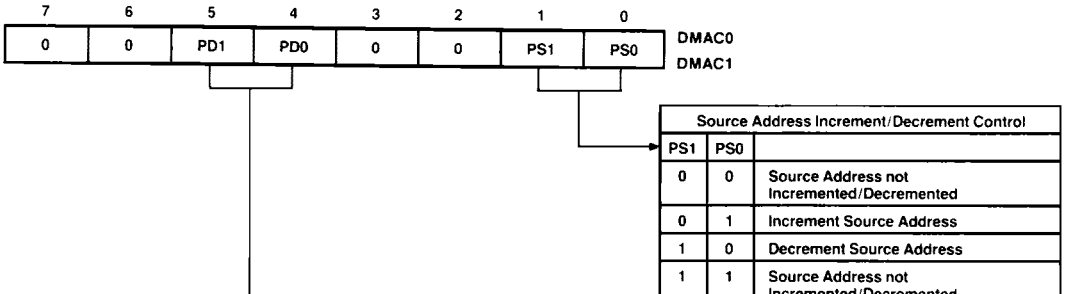
Word/byte	
0	Byte Transfers
1	Word Transfers

DMA Mode			
MD_2	MD_1	MD_0	
0	0	0	Single Step (Mem to Mem)
0	0	1	Demand Release (I/O to Mem)
0	1	0	Demand Release (Mem to I/O)
0	1	1	Reserved
1	0	0	Burst Mode (Mem to Mem)
1	0	1	Single Transfer (I/O to Mem)
1	1	0	Single Transfer (Mem to I/O)
1	1	1	Reserved

Notes:
[1] Valid only during single-step and burst modes.
[2] Cleared when TC = 0; cleared when DMA transfer is aborted by NMI.

49-001390B

Figure 26. DMA Address Control Registers (DMAC)



Source Address Increment/Decrement Control		
PS1	PS0	
0	0	Source Address not Incremented/Decremented
0	1	Increment Source Address
1	0	Decrement Source Address
1	1	Source Address not Incremented/Decremented

Destination Address Increment/Decrement Control		
PD1	PD0	
0	0	Destination Address not Incremented/Decremented
0	1	Increment Destination Address
1	0	Decrement Destination Address
1	1	Destination Address not Incremented/Decremented

49-001391B

4a

Parallel Ports

The μPD70320/322 has three 8-bit parallel I/O ports: P0, P1, and P2. Refer to figures 27 through 31. Special function register (SFR) locations can access these ports. The port lines are individually programmable as inputs or outputs. Many of the port lines have dual functions as port or control lines.

Use the associated port mode and port mode control registers to select the mode for a given I/O line.

Figure 27. Port Mode Registers 0 and 2 (PM0, PM2)

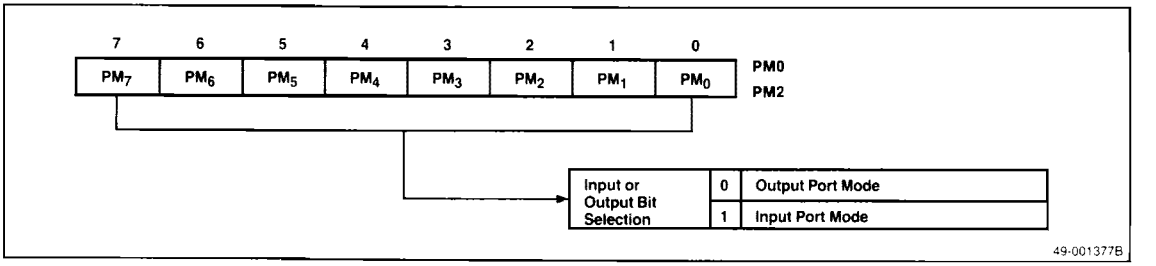


Figure 28. Port Mode Register 1 (PM1)

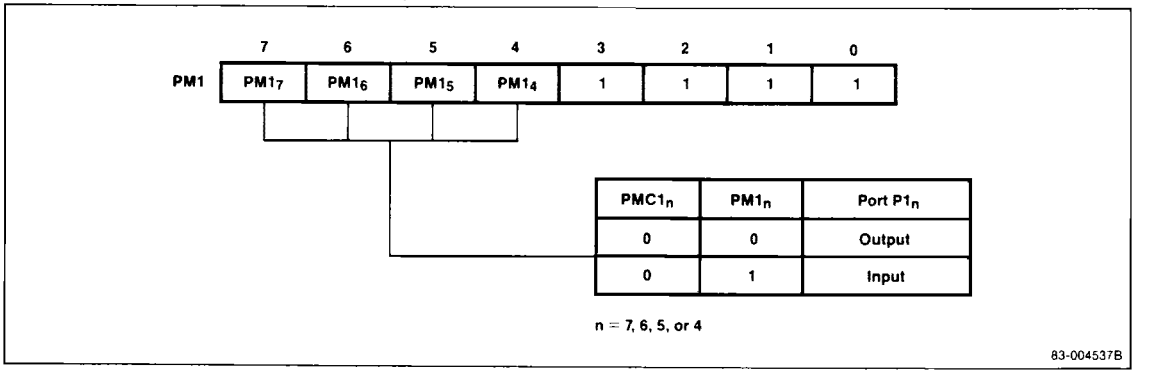


Figure 29. Port Mode Control Register 0 (PMC0)

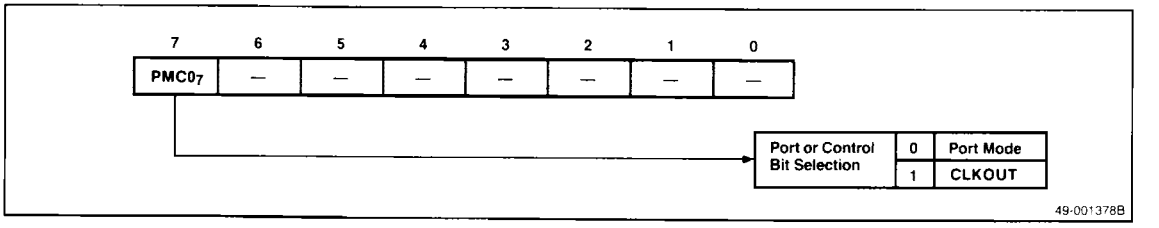


Figure 30. Port Mode Control Register 1 (PMC1)

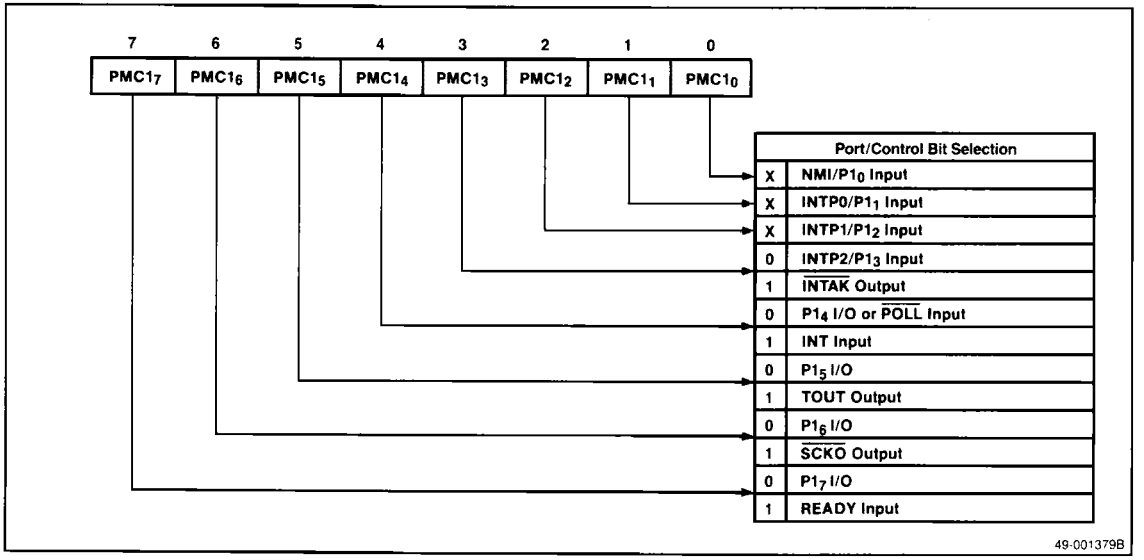
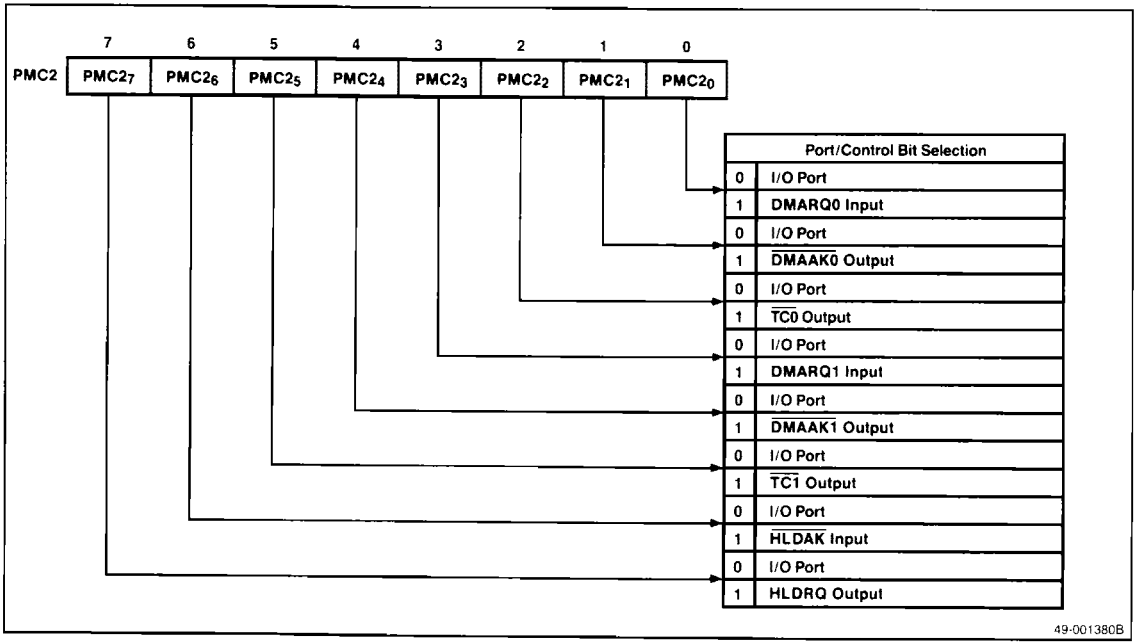


Figure 31. Port Mode Control Register 2 (PMC2)



The analog comparator port (PT) compares each input line to a reference voltage. The reference voltage is programmable to be the V_{TH} input x n/16, where n = 1 to 16. See figure 32.

Programmable Wait State Generation

You can generate wait states internally to further reduce the necessity for external hardware. Insertion of these wait states allows direct interface to devices whose access times cannot meet the CPU read/write timing requirements.

When using this function, the entire 1M-byte memory address space is divided into 128K-blocks. Each block can be programmed for zero, one, or two wait states, or two plus those added by the extenal READY signal. The top two blocks are programmed together as one unit.

The appropriate bits in the wait control word (WTC) control wait state generation. Programming the upper two bits in the wait control word will set the wait state conditions for the entire I/O address space. Figure 33 shows the memory map for programmable wait state generation; see figure 34 for a graphic representation of the wait control word.

Figure 33. Programmable Wait State Generation

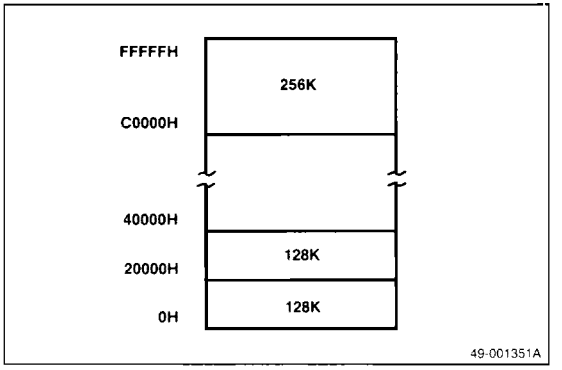


Figure 32. Port Mode Register T (PMT)

7	6	5	4	3	2	1	0	
0	0	0	0	PMT_3	PMT_2	PMT_1	PMT_0	PMT

Comparator Port Threshold Selection				
0	0	0	0	V_{TH} x 16/16
0	0	0	1	V_{TH} x 1/16
0	0	1	0	V_{TH} x 2/16
0	0	1	1	V_{TH} x 3/16
0	1	0	0	V_{TH} x 4/16
0	1	0	1	V_{TH} x 5/16
0	1	1	0	V_{TH} x 6/16
0	1	1	1	V_{TH} x 7/16
1	0	0	0	V_{TH} x 8/16
1	0	0	1	V_{TH} x 9/16
1	0	1	0	V_{TH} x 10/16
1	0	1	1	V_{TH} x 11/16
1	1	0	0	V_{TH} x 12/16
1	1	0	1	V_{TH} x 13/16
1	1	1	0	V_{TH} x 14/16
1	1	1	1	V_{TH} x 15/16

49-001381B

Standby Modes

The two low-power standby modes are HALT and STOP. Software causes the processor to enter either mode.

HALT Mode.

In the HALT mode, the processor is inactive and the chip consumes much less power than when operational. The external oscillator remains functional and all peripherals are active. Internal status and output port line conditions are maintained. Any unmasked interrupt can release this mode. In the EI state, interrupts subsequently will be processed in vector mode. In the DI state, program execution is restarted with the instruction following the HALT instruction.

STOP Mode.

The STOP mode allows the largest power reduction while maintaining RAM. The oscillator is stopped, halting all internal peripherals. Internal status is maintained. Only a reset or NMI can release this mode.

A standby flag in the SFR area is reset by rises in the supply voltage. Its status is maintained during normal operation and standby. The STBC register (figure 35) is not initialized by RESET. Use the standby flag to determine whether program execution is returning from standby or from a cold start by setting this flag before entering the STOP mode.

Figure 34. Wait Control Word

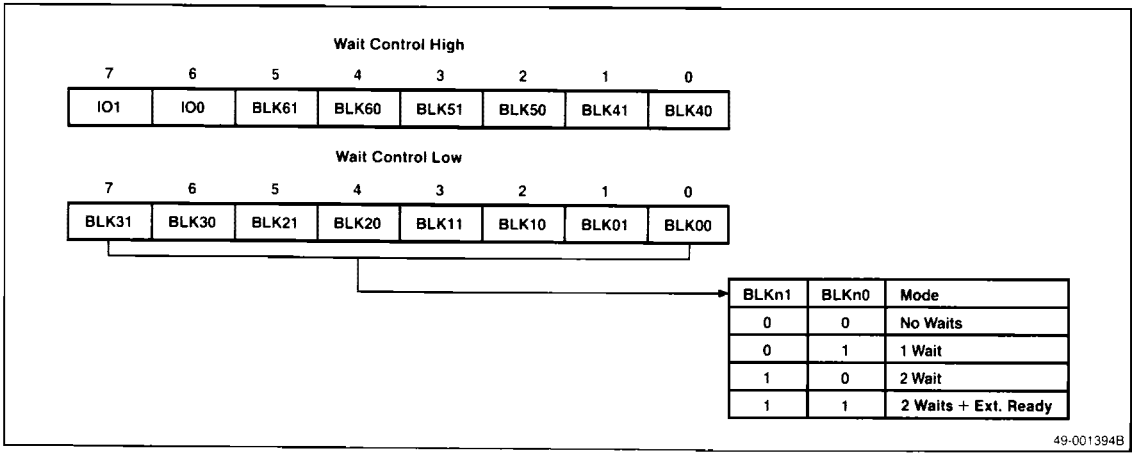
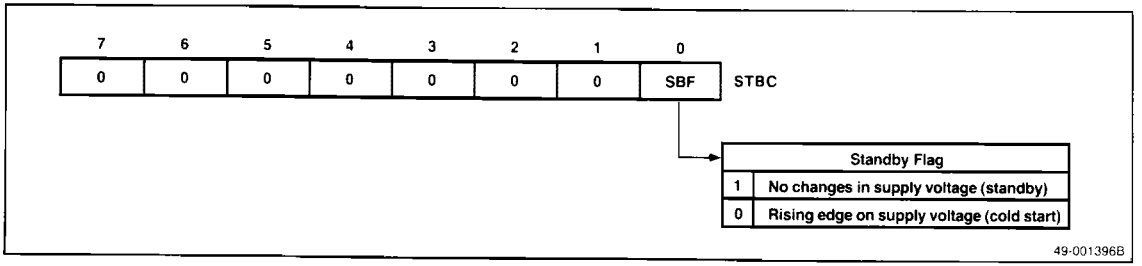


Figure 35. Standby Register



Special Function Registers

Table 6 shows the special function register mnemonic, type, address, reset value, and function. The 8 high-order bits of each address (xx) are specified by the IDB register.

SFR area addresses not listed in table 6 are reserved. If read, the contents of these addresses are undefined, and any write operation will be meaningless.

Table 6. Special Function Registers

Name	Byte/ Word	Address	Reset Value (Note 2)	R/W (Note 1)	Function
P0	B	xxF00H		R/W	Port 0
PM0	B	xxF01H	FFH	W	Port mode 0
PMC0	B	xxF02H	00H	W	Port mode control 0
P1	B	xxF08H		R/W	Port 1
PM1	B	xxF09H	FFH	W	Port mode 1
PMC1	B	xxF0AH	00H	W	Port mode control 1
P2	B	xxF10H		R/W	Port 2
PM2	B	xxF11H	FFH	W	Port mode 2
PMC2	B	xxF12H	00H	W	Port mode control 2
PT	B	xxF38H		R	Port T
PMT	B	xxF3BH	00H	R/W	Port mode T
INTM	B	xxF40H	00H	R/W	Interrupt mode
EMS0	B	xxF44H		R/W	External interrupt macro service 0
EMS1	B	xxF45H		R/W	External interrupt macro service 1
EMS2	B	xxF46H		R/W	External interrupt macro service 2
EXIC0	B	xxF4CH	47H	R/W	External interrupt control 0
EXIC1	B	xxF4DH	47H	R/W	External interrupt control 1
EXIC2	B	xxF4EH	47H	R/W	External interrupt control 2
RXB0	B	xxF60H		R	Receive buffer 0
TXB0	B	xxF62H		W	Transfer buffer 0
SRMS0	B	xxF65H		R/W	Serial receive macro service 0
STMS1	B	xxF66H		R/W	Serial transmit macro service 1
SCM0	B	xxF68H	00H	R/W	Serial communication mode 0
SCC0	B	xxF69H	00H	R/W	Serial communication control 0
BRG0	B	xxF6AH	00H	R/W	Baud rate generator 0
SCE0	B	xxF6BH	00H	R	Serial communication error 0
SEIC0	B	xxF6CH	47H	R/W	Serial error interrupt control 0
SRIC0	B	xxF6DH	47H	R/W	Serial receive interrupt control 0
STIC0	B	xxF6EH	47H	R/W	Serial transmit interrupt control 0
RXB1	B	xxF70H		R	Receive buffer 1
TXB1	B	xxF72H		W	Transmit buffer 1
SRMS1	B	xxF75H		R/W	Serial receive macro service 1
STMS1	B	xxF76H		R/W	Serial transmit macro service 1
SCM1	B	xxF78H	00H	R/W	Serial communication mode 1
SCC1	B	xxF79H	00H	R/W	Serial communication control 1
BRG1	B	xxF7AH	00H	R/W	Baud rate generator register 1
SCE1	B	xxF7BH	00H	R	Serial communication error 1
SEIC1	B	xxF7CH	47H	R/W	Serial error interrupt control 1
SRIC1	B	xxF7DH	47H	R/W	Serial receive interrupt control 1
STIC1	B	xxF7EH	47H	R/W	Serial transmit interrupt control 1
TM0	W	xxF80H		R/W	Timer register 0
TM0L	B	xxF80H		R/W	Timer register 0 low
TM0H	B	xxF81H		R/W	Timer register 0 high
MD0	W	xxF82H		R/W	Modulo register 0

Notes:

(1) Indicates if register is available for read/write operations.

(2) Reset values not specified are undefined.

Table 6. Special Function Registers (cont)

Name	Byte/ Word	Address	Reset Value (Note 2)	R/W (Note 1)	Function
MD0L	B	xxF82H		R/W	Modulo register 0 low
MD0H	B	xxF83H		R/W	Modulo register 0 high
TM1	W	xxF88H		R/W	Timer register 1
TM1L	B	xxF88H		R/W	Timer register 1 low
TM1H	B	xxF89H		R/W	Timer register 1 high
MD1	W	xxF8AH		R/W	Modulo register 1
MD1L	B	xxF8AH		R/W	Modulo register 1 low
MD1H	B	xxF8BH		R/W	Modulo register 1 high
TMC0	B	xxF90H	00H	R/W	Timer control 0
TMC1	B	xxF91H	00H	R/W	Timer control 1
TMMS0	B	xxF94H		R/W	Timer macro service 0
TMMS1	B	xxF95H		R/W	Timer macro service 1
TMMS2	B	xxF96H		R/W	Timer macro service 2
TMIC0	B	xxF9CH	47H	R/W	Timer interrupt control 0
TMIC1	B	xxF9DH	47H	R/W	Timer interrupt control 1
TMIC2	B	xxF9EH	47H	R/W	Timer interrupt control 2
DMAC0	B	xxFA0H		R/W	DMA control 0
DMAM0	B	xxFA1H	00H	R/W	DMA mode 0
DMAC1	B	xxFA2H		R/W	DMA control 1
DMAM1	B	xxFA3H	00H	R/W	DMA mode 1
DIC0	B	xxFACH	47H	R/W	DMA interrupt control 0
DIC1	B	xxFADH	47H	R/W	DMA interrupt control 1
STBC	B	xxFE0H		R/W	Standby control
RFM	B	xxFE1H	0FCH	R/W	Refresh mode
WTC	W	xxFE8H	FFH	R/W	Wait control
WTCL	B	xxFE8H	FFH	R/W	Wait control low
WTCH	B	xxFE9H	FFH	R/W	Wait control high
FLAG	B	xxFEAH	00H	R/W	Flag register
PRC	B	xxFEBH	4EH	R/W	Processor control
TBIC	B	xxFECH	47H	R/W	Time base IRC register
ISPR	B	xxFFCH		R	In service priority register
IDB	B	xxFFFH FFFFFH		R/W	Internal data area base

Absolute Maximum Ratings

$T_A = 25°C$

Supply voltage, V_{DD}	−0.5 to +7.0 V
Input voltage, V_I	−0.5 to V_{DD} + 0.5 V (≤ +7.0 V)
Output voltage, V_O	−0.5 to V_{DD} + 0.5 V (≤ +7.0 V)
Threshold voltage, V_{TH}	−0.5 to V_{DD} + 0.5 V (≤ +7.0 V)
Output current, low; I_{OL}	
Each output pin	4.0 mA
Total	50 mA
Output current, high; I_{OH}	
Each output pin	−2.0 mA
Total	−20 mA
Operating temperature range, T_{OPT}	−40 to +85°C
Storage temperature range, T_{STG}	−65 to +150°C

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

DC Characteristics

$V_{DD} = +5\ V \pm 10\%$; $T_A = -10$ to $+70°C$ (Note 1)

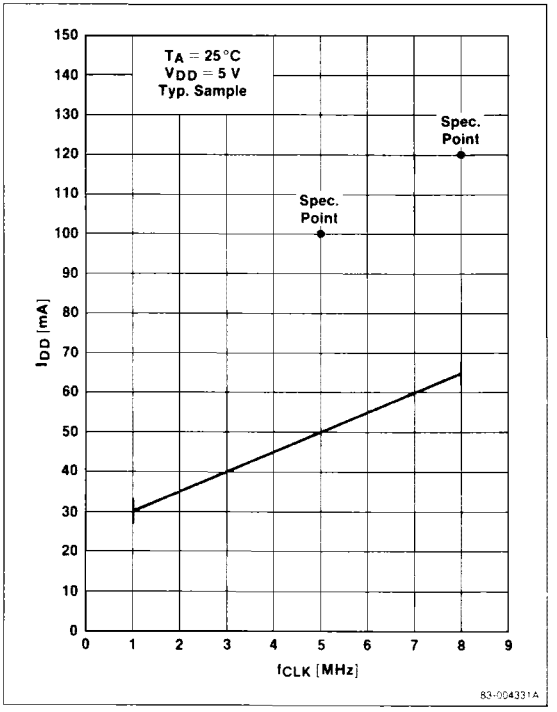
Parameter	Symbol	Limits Min	Typ	Max	Unit	Test Conditions
Supply current, operating	I_{DD1}		43 58	100 120	mA mA	$f_{CLK} = 5$ MHz $f_{CLK} = 8$ MHz
Supply current, HALT mode	I_{DD2}		17 21	40 50	mA mA	$f_{CLK} = 5$ MHz $f_{CLK} = 8$ MHz
Supply current, STOP mode	I_{DD3}		10	30	μA	
Threshold current	I_{TH}		0.5	1.0	mA	$V_{TH} = 0$ to V_{DD}
Input voltage, low	V_{IL}	0		0.8	V	
Input voltage, high	V_{IH1}	2.2		V_{DD}	V	All inputs except $\overline{RESET}$, $P1_0$/NMI, X1, X2
	V_{IH2}	0.8 x V_{DD}		V_{DD}	V	$\overline{RESET}$, $P1_0$/NMI, X1, X2
Output voltage, low	V_{OL}			0.45	V	$I_{OL} = 1.6$ mA
Output voltage, high	V_{OH}	V_{DD} − 1.0			V	$I_{OH} = -0.4$ mA
Input current	I_{IN}			±20	μA	$\overline{EA}$, $P1_0$/NMI; $V_I = 0$ to V_{DD}
Input leakage current	I_{LI}			±10	μA	All except $\overline{EA}$, $P1_0$/NMI; $V_I = 0$ to V_{DD}
Output leakage current	I_{LO}			±10	μA	$V_O = 0$ to V_{DD}
Data retention voltage	V_{DDDR}	2.5		5.5	V	

Notes:

(1) The standard operating temperature range is −10 to +70°C. However, extended temperature range parts (−40 to +85°C) are available.

4a

Supply Current vs Clock Frequency



Comparator Characteristics

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

Parameter	Symbol	Limits Min	Limits Max	Unit	Test Conditions
Accuracy	VA_{COMP}		±100	mV	
Threshold voltage	V_{TH}	0	V_{DD} + 0.1	V	
Comparison time	t_{COMP}	64	65	t_{CYK}	
PT input voltage	V_{IPT}	0	V_{DD}	V	
PTn leakage current	I_{LC}		±10	μA	

Capacitance Characteristics

V_{DD} = 0 V; T_A = 25°C

Parameter	Symbol	Limits Min	Limits Max	Unit	Test Conditions
Input capacitance	C_I		10	pF	f_C = 1 MHz; Unmeasured pins returned to 0 V
Output capacitance	C_O		20	pF	
I/O capacitance	C_{IO}		20	pF	

AC Characteristics

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

Parameter	Symbol	Limits Min	Limits Max	Unit	Test Conditions
V_{DD} rise, fall time	t_{RVD}, t_{FVD}	200		μs	STOP mode
Input rise, fall time	t_{IR}, t_{IF}		20	ns	Except X1, X2, $\overline{RESET}$, NMI
Input rise, fall time	t_{IRS}, t_{IFS}		30	ns	$\overline{RESET}$, NMI (Schmitt)
Output rise, fall time	t_{OR}, t_{OF}		20	ns	Except CLKOUT
X1 cycle time	t_{CYX}	98	250	ns	Note 3
		62	250	ns	Note 4
X1 width, low	t_{WXL}	35		ns	Note 3
		20		ns	Note 4
X1 width, high	t_{WXH}	35		ns	Note 3
		20		ns	Note 4
X1 rise, fall time	t_{XR}, t_{XF}		20	ns	
CLKOUT cycle time	t_{CYK}	200	2000	ns	Note 3
		125	2000	ns	Note 4
CLKOUT width, low	t_{WKL}	0.5T − 15		ns	Note 1
CLKOUT width, high	t_{WKH}	0.5T − 15		ns	
CLKOUT rise, fall time	t_{KR}, t_{KF}		15	ns	
Address delay time	t_{DKA}	15	90	ns	
Address hold time	t_{HMA}	0.5T − 30		ns	
Address valid to input data valid	t_{DADR}		T(n + 1.5) − 90	ns	Note 2
$\overline{MREQ}$ to data delay	t_{DMRD}		T(n + 1) − 75	ns	
$\overline{MSTB}$ to data delay	t_{DMSD}		T(n + 0.5) − 75	ns	
$\overline{MREQ}$ to MSTB delay	t_{DMRMS}	0.5T − 35	0.5T + 35	ns	
$\overline{MREQ}$ width, low	t_{WMRL}	T(n + 1) − 30		ns	
Input data hold time	t_{HMDR}	0		ns	
Next control setup time	t_{SCC}	T − 25		ns	

Notes:

(1) T = CPU clock period (t_{CYK}).

(2) n = number of wait states inserted.

(3) For 5 MHz parts (μPD70320/322).

(4) For 8 MHz parts (μPD70320/322-8).

AC Characteristics (cont)

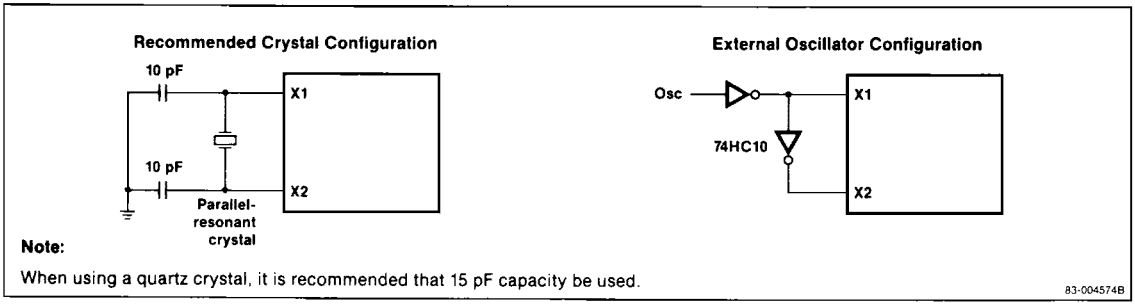
Parameter	Symbol	Limits Min	Limits Max	Unit	Test Conditions
$\overline{MREQ}$ to TC delay time	t_{DMRTC}		0.5T + 50	ns	
Address data output	t_{DADW}		0.5T + 50	ns	
$\overline{MREQ}$ delay time	t_{DAMR}	0.5T − 30	0.5T + 30	ns	
$\overline{MSTB}$ delay time	t_{DAMS}	T − 30	T + 30	ns	
$\overline{MSTB}$ width, low	t_{WMSL}	T(n + 0.5) − 30		ns	
Data output setup time	t_{SDM}	T(n + 1) − 50		ns	
Data output hold time	t_{HMDW}	0.5T − 30		ns	
$\overline{IOSTB}$ delay time	t_{DAIS}	0.5T − 30	0.5T + 30	ns	
$\overline{IOSTB}$ to data input	t_{DISD}		T(n + 1) − 90	ns	
$\overline{IOSTB}$ width, low	t_{WISL}	T(n + 1) − 30		ns	
Address hold time	t_{HISA}	0.5T − 30		ns	
Input data hold time	t_{HISDR}	0		ns	
Output data setup time	t_{SDIS}	T(n + 1) − 50		ns	
Output data hold time	t_{HISDW}	0.5T − 30		ns	
Next DMARQ setup time	t_{SDADQ}		T	ns	Demand mode
DMARQ hold time	t_{HDADQ}	0		ns	Demand mode
$\overline{DMAAK}$ read width, low	t_{WDMRL}	T(n + 1.5) − 30		ns	
$\overline{DMAAK}$ write width, low	t_{WDMWL}	T(n + 1) − 30		ns	
$\overline{DMAAK}$ to TC delay time	t_{DDATC}		0.5T + 50	ns	
$\overline{TC}$ width, low	t_{WTCL}	2T − 30		ns	
$\overline{REFRQ}$ delay time	t_{DARF}	0.5T − 30		ns	
$\overline{REFRQ}$ width, low	t_{WRFL}	T(n + 1) − 30		ns	
Address hold time	t_{HRFA}	0.5T − 30		ns	

AC Characteristics (cont)

Parameter	Symbol	Limits Min	Limits Max	Unit	Test Conditions
$\overline{RESET}$ width low	t_{WRSL1}	30		ms	STOP/ POR (Power-on reset)
	t_{WRSL2}	5		μs	System reset
$\overline{MREQ}$, $\overline{IOSTB}$ to READY setup time	t_{SCRY}		T(n − 1) − 100	ns	n ≥ 2
$\overline{MREQ}$, $\overline{IOSTB}$ to READY hold time	t_{HCRY}	T(n − 1)		ns	n ≥ 2
HLDRQ setup time	t_{SHQK}	30		ns	
$\overline{HLDAK}$ output delay	t_{DKHA}		80	ns	
Bus control float to $\overline{HLDAK}$↓	t_{CFHA}	T − 50		ns	
$\overline{HLDAK}$↑ to control output time	t_{DHAC}	T − 50		ns	
HLDRQ to $\overline{HLDAK}$ delay	t_{DHQHA}		3T + 160	ns	
HLDRQ↓ to control float	t_{DHQC}	3T + 30		ns	
HLDRQ width, low	t_{WHQL}	1.5T		ns	
$\overline{HLDAK}$ width, low	t_{WHAL}		T	ns	
$\overline{INTP}$, DMARQ setup	t_{SIQK}	30		ns	
$\overline{INTP}$, DMARQ width, high	t_{WIQH}	8T		ns	
$\overline{INTP}$, DMARQ width, low	t_{WIQL}	8T		ns	
$\overline{POLL}$ setup time	t_{SPLK}	30		ns	
NMI width, high	t_{WNIH}	5		μs	
NMI width, low	t_{WNIL}	5		μs	
$\overline{CTS}$ width, low	t_{WCTL}	2T		ns	
INTR setup time	t_{SIRK}	30		ns	
$\overline{INTAK}$ delay time	t_{DKIA}		80	ns	
INTR hold time	t_{HIAIQ}	0		ns	
$\overline{INTAK}$ width, low	t_{WIAL}	2T − 30		ns	
$\overline{INTAK}$ width, high	t_{WIAH}	T − 30		ns	
$\overline{INTAK}$ to data delay	t_{DIAD}		2T − 130	ns	
$\overline{INTAK}$ to data hold	t_{HIAD}	0	0.5T	ns	
$\overline{SCK0}$ (TSCK) cycle time	t_{CYTK}	1000		ns	
$\overline{SCK0}$ (TSCK) width, high	t_{WSTH}	450		ns	
$\overline{SCK0}$ (TSCK) width, low	t_{WSTL}	450		ns	
TxD delay time	t_{DTKD}		210	ns	
TxD hold time	t_{HTKD}	20		ns	
$\overline{CTS0}$ (RSCK) cycle time	t_{CYRK}	1000		ns	
$\overline{CTS0}$ (RSCK) width, high	t_{WSRH}	420		ns	
$\overline{CTS0}$ (RSCK) width, low	t_{WSRL}	420		ns	
RxD setup time	t_{SRDK}	80		ns	
RxD hold time	t_{HKRD}	80		ns	

4a

Figure 36. External System Clock Control Source



Note:

When using a quartz crystal, it is recommended that 15 pF capacity be used.

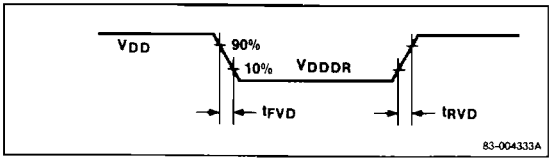
83-004574B

Recommended Ceramic Resonator and Capacitance Requirements

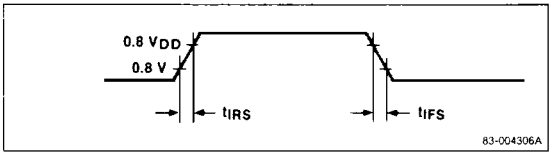
Manufacturer	Product Number	Recommended Constants C1 (pF)	C2 (pF)	Product Number	Recommended Constants C1 (pF)	C2 (pF)
Kyocera	KBR-10.0M	33	33			
Murata Manufacturing	CSA.10.0MT	47	47	CSA16.0MX040	30	30
TDK	FCR10.0M2S	30	30	FCR16.0M2S	15	6

Timing Waveforms

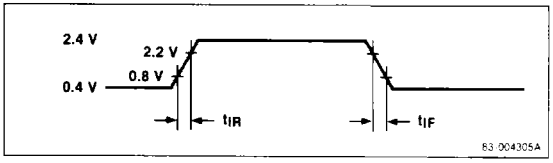
Stop Mode Data Retention Timing



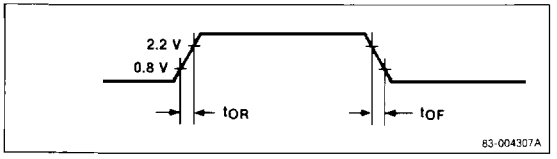
AC Input Waveform 2 (RESET, NMI)



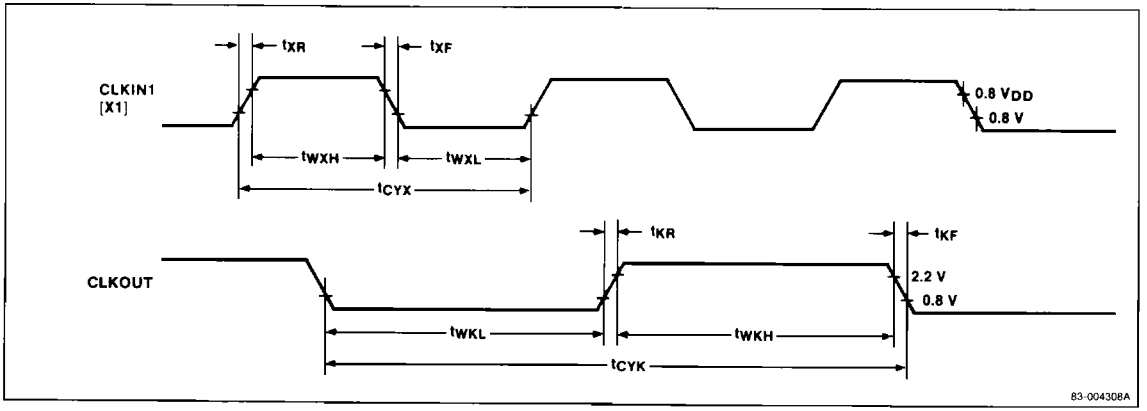
AC Input Waveform 1 (Except X1, X2, RESET, NMI)



AC Output Test Point (Except CLKOUT)

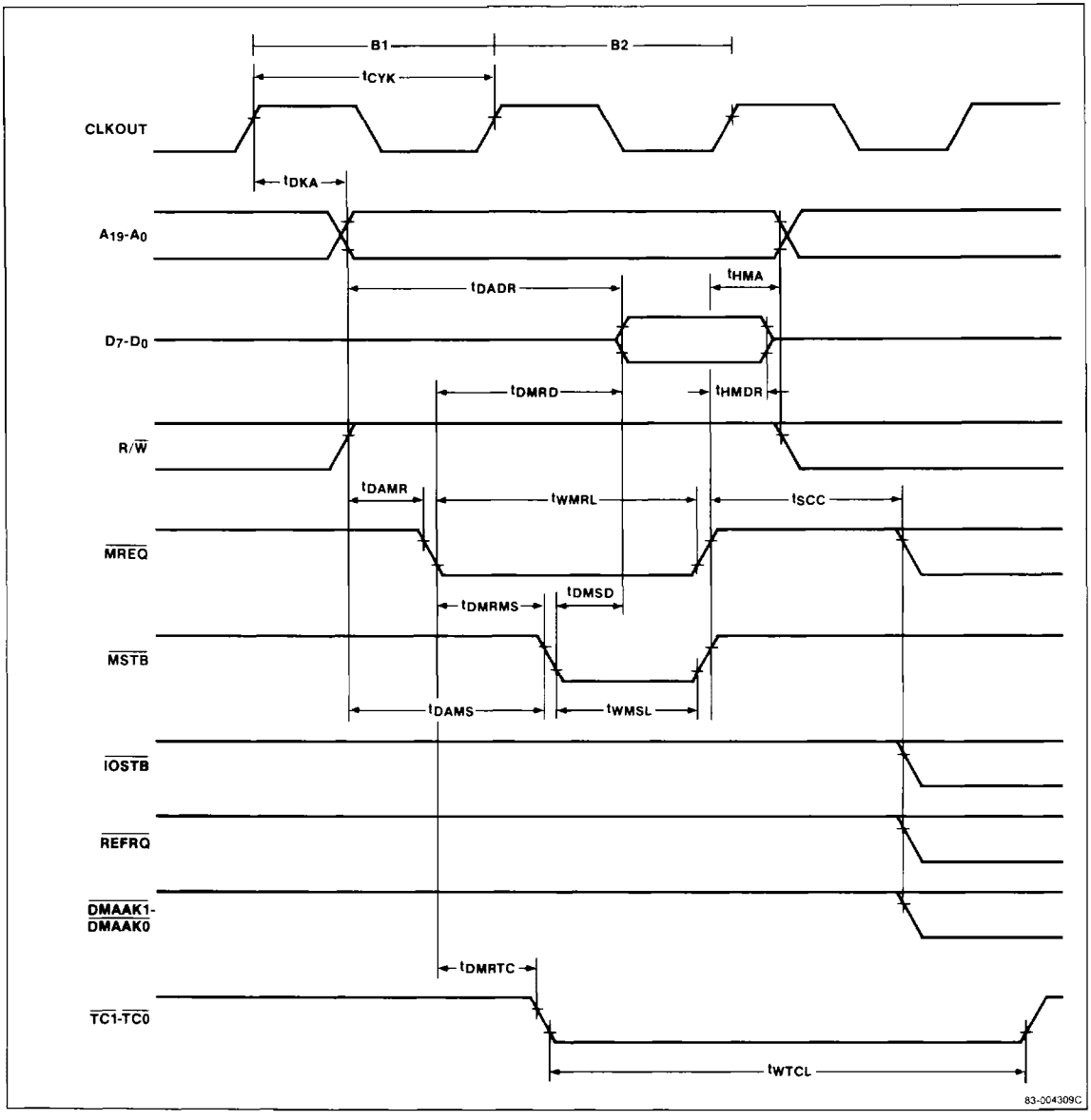


Clock In and Clock Out



Timing Waveforms (cont)

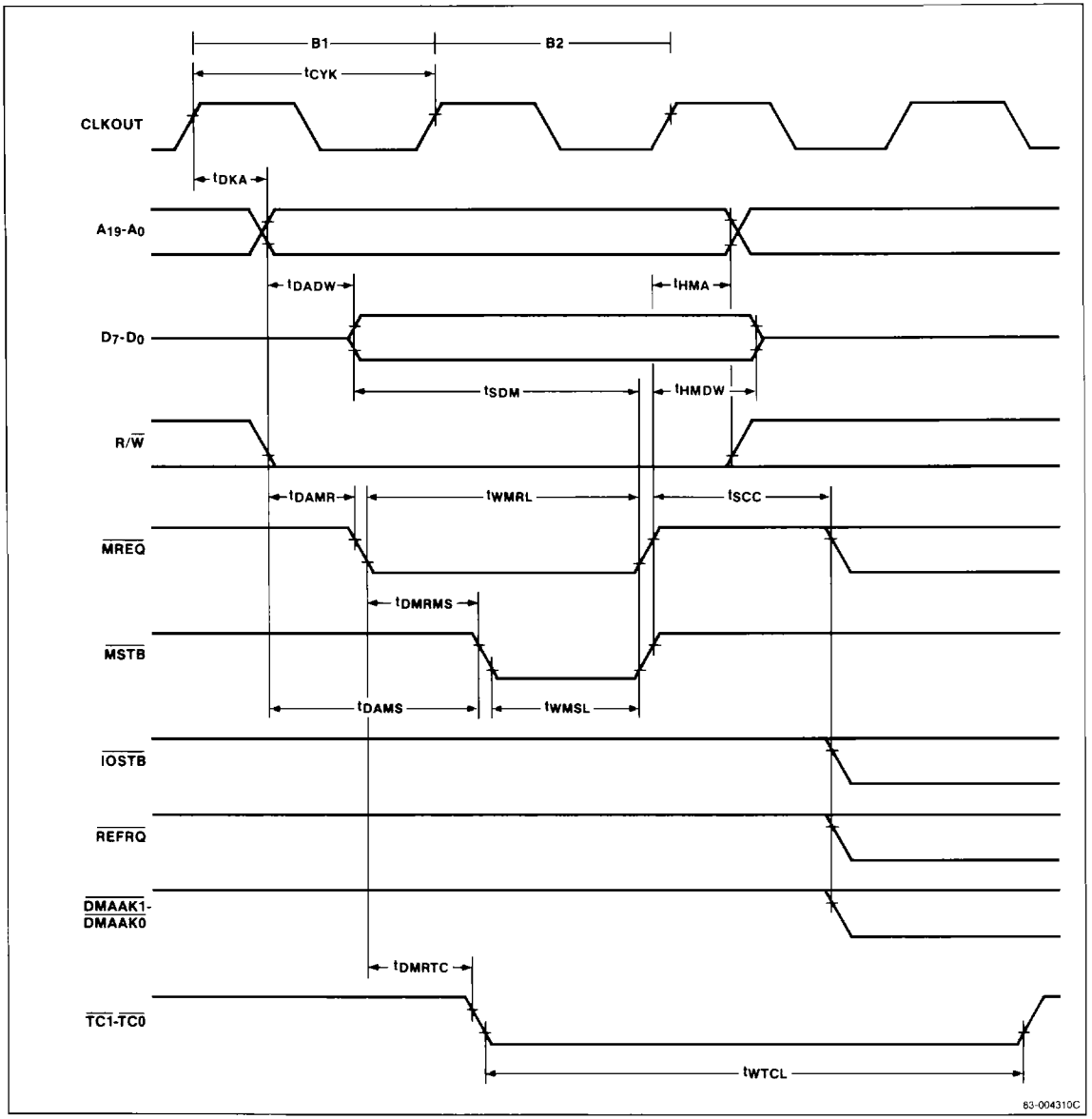
Memory Read



4a

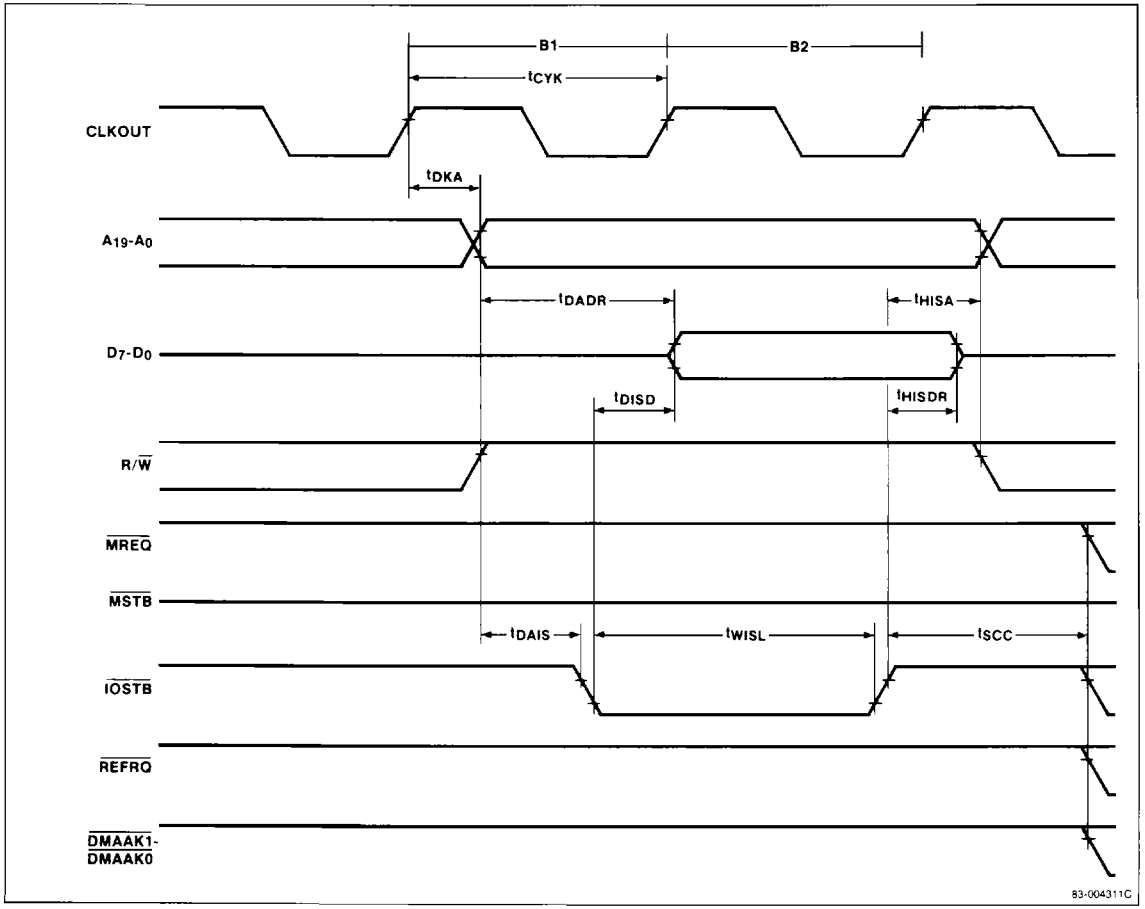
Timing Waveforms (cont)

Memory Write



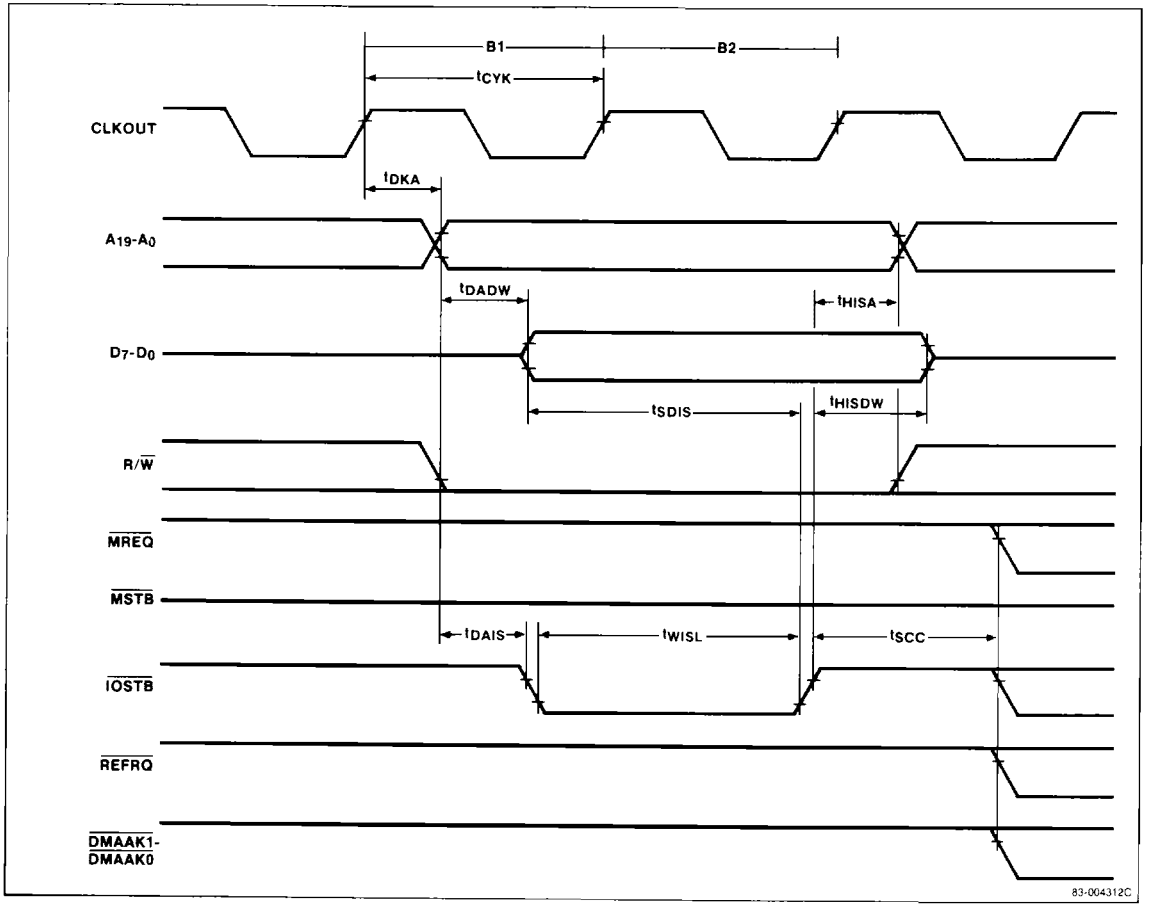
Timing Waveforms (cont)

I/O Read



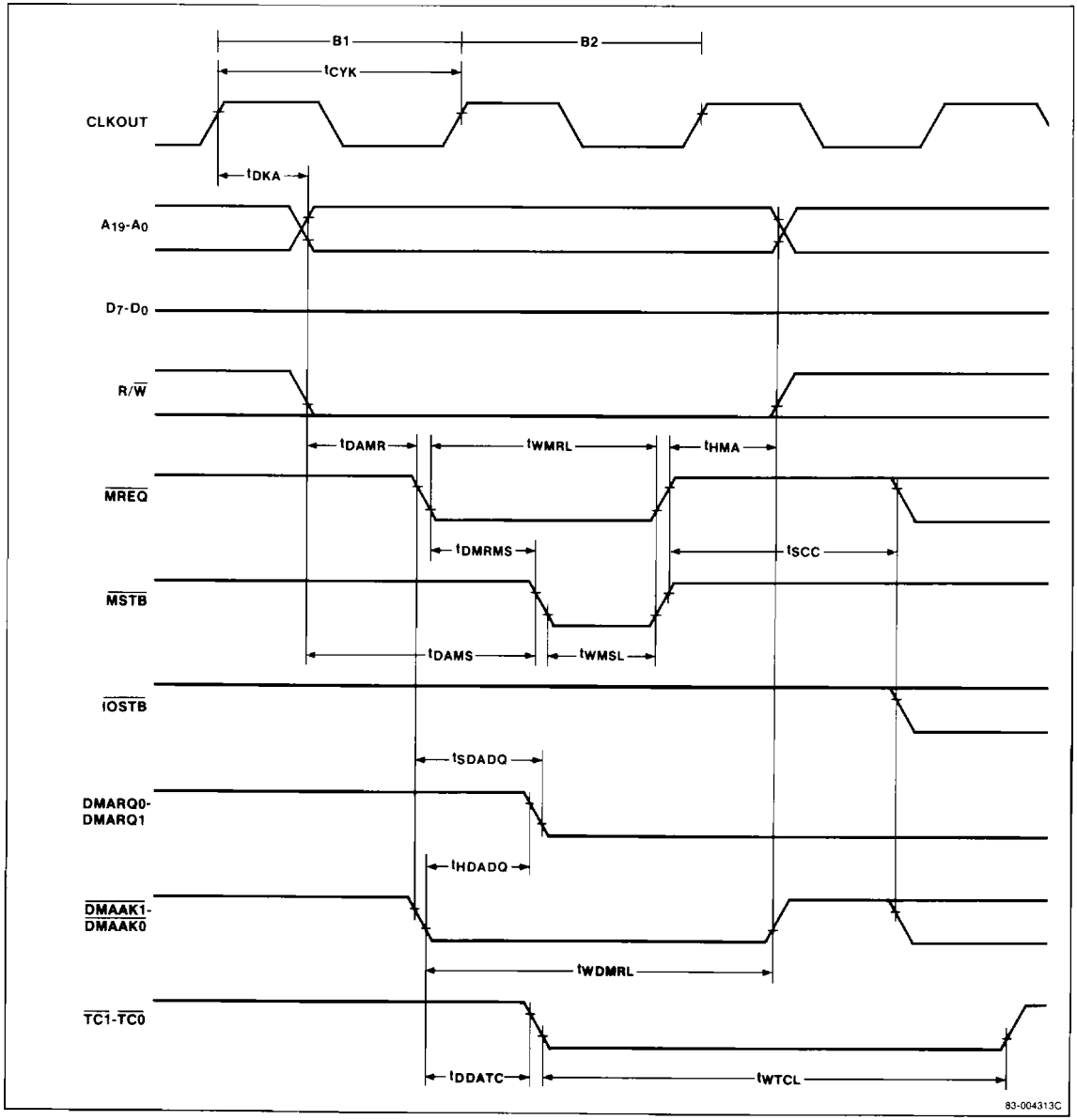
Timing Waveforms (cont)

I/O Write



Timing Waveforms (cont)

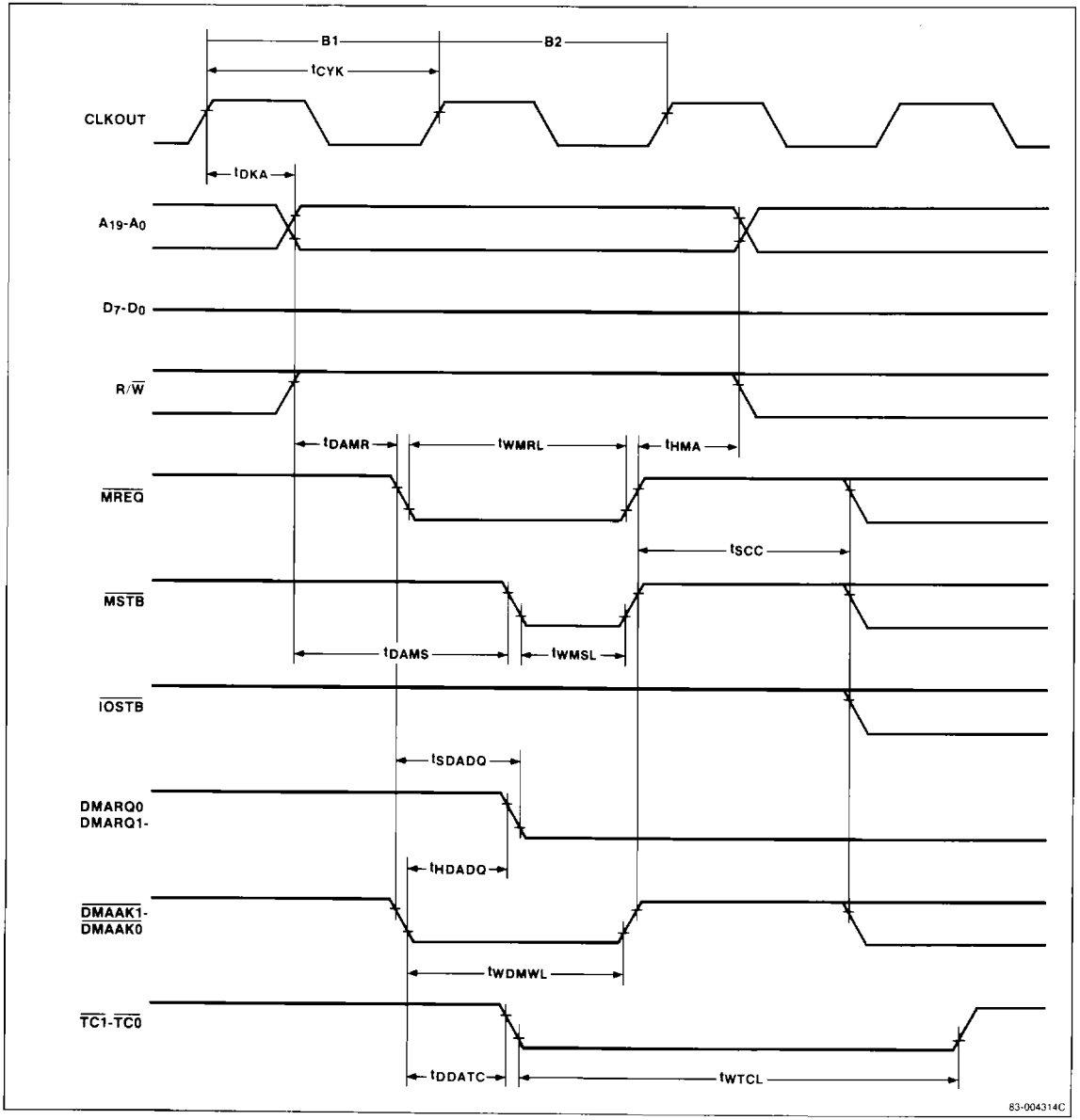
DMA, I/O to Memory



4a

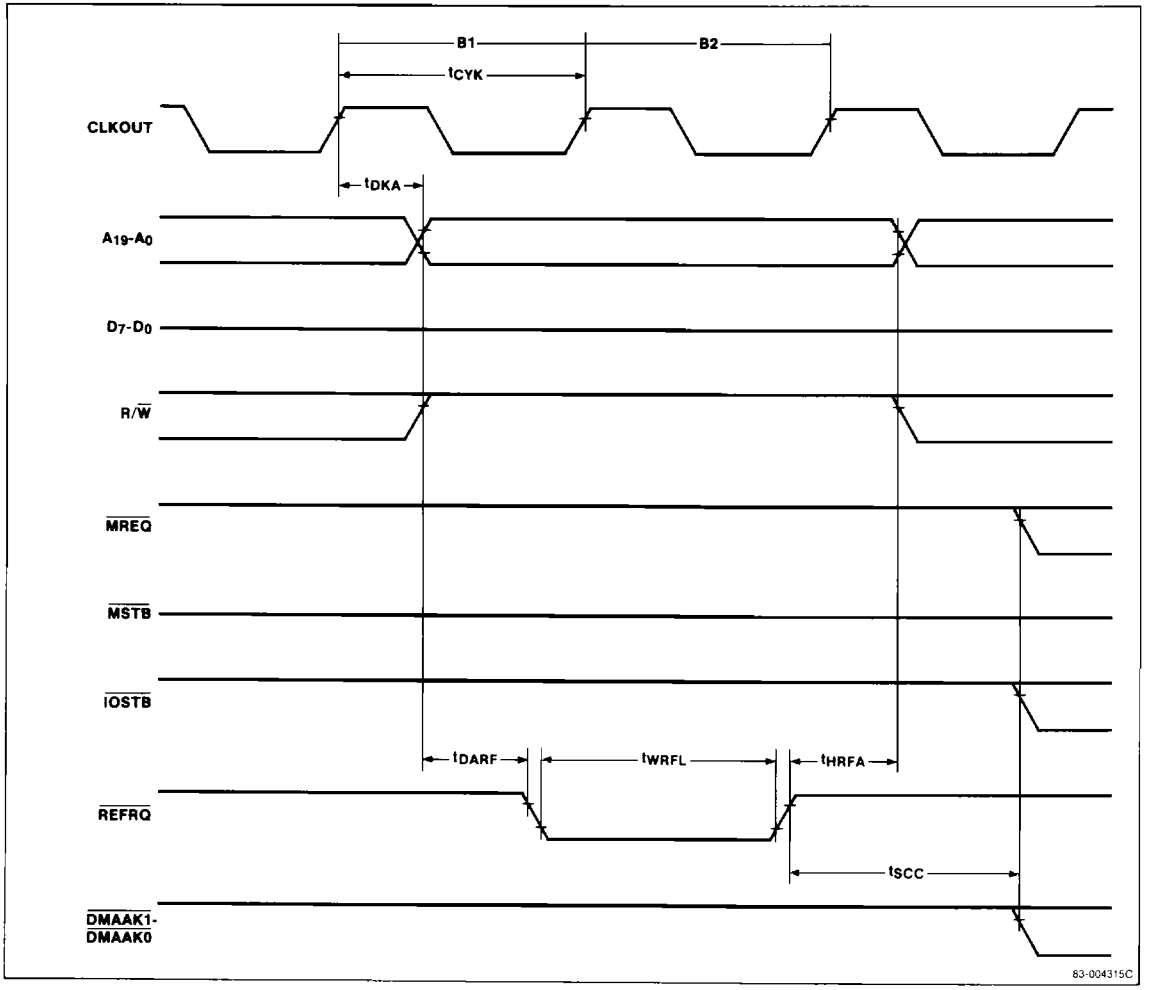
Timing Waveforms (cont)

DMA, Memory to I/O

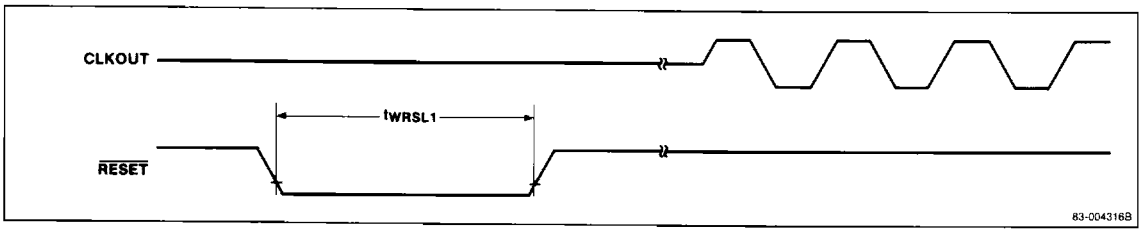


Timing Waveforms (cont)

Refresh



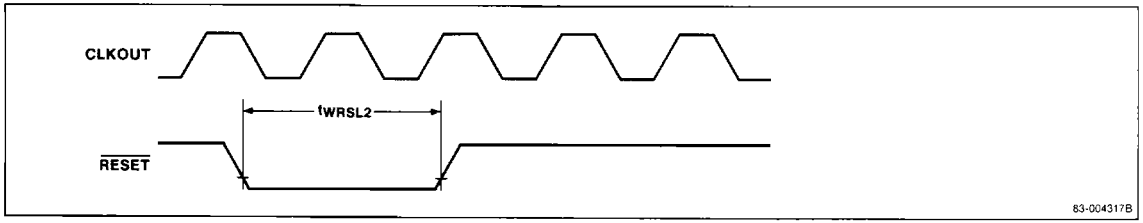
RESET 1



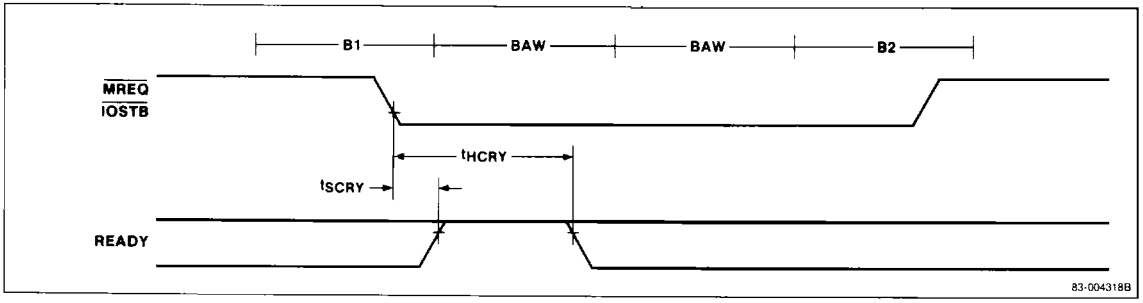
4a

Timing Waveforms (cont)

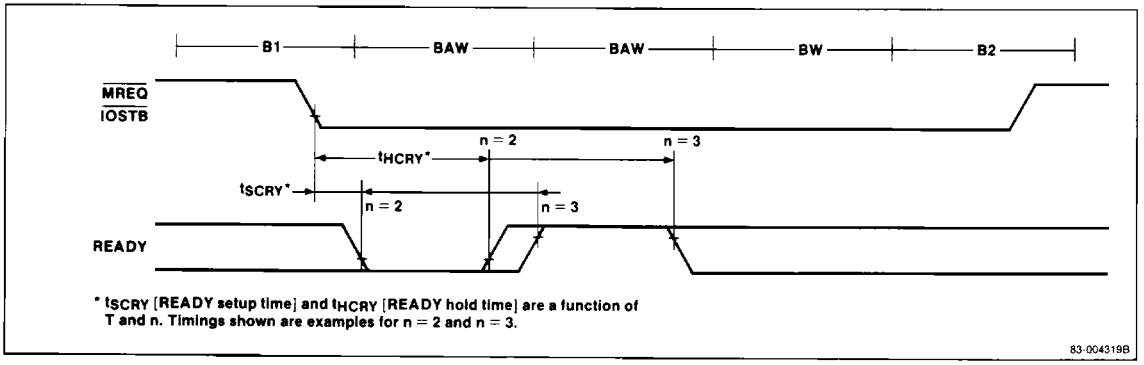
$\overline{RESET}$ 2



READY 1

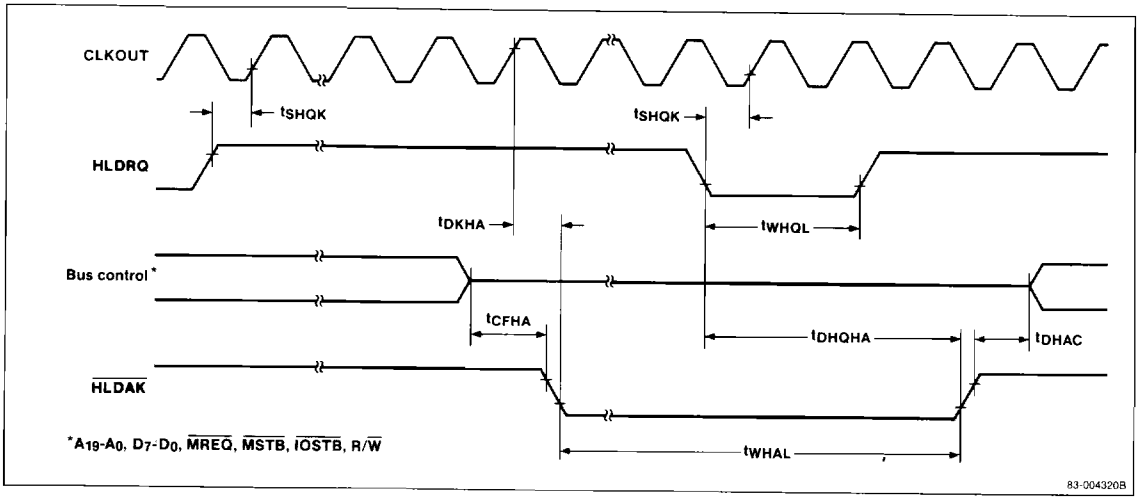


READY 2



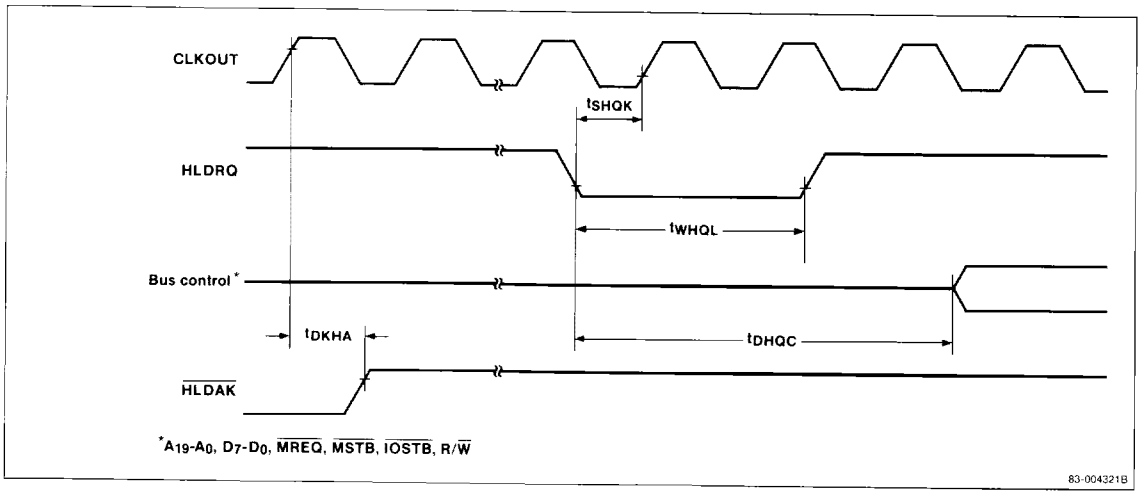
Timing Waveforms (cont)

HLDRQ/$\overline{\text{HLDAK}}$ 1

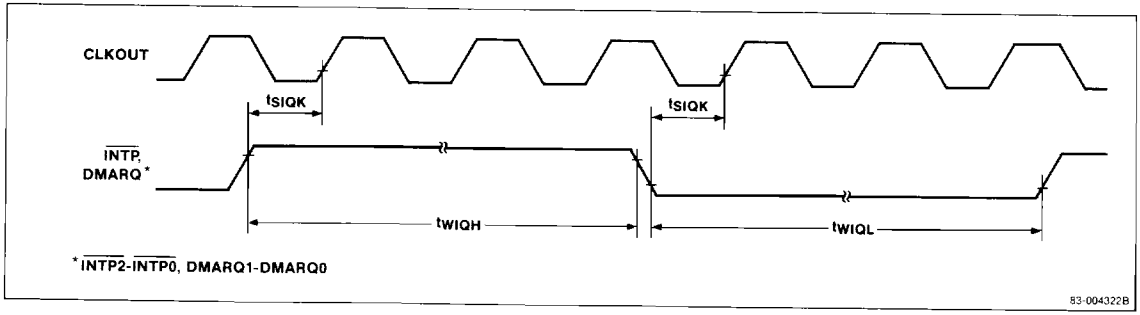


4a

HLDRQ/$\overline{\text{HLDAK}}$ 2

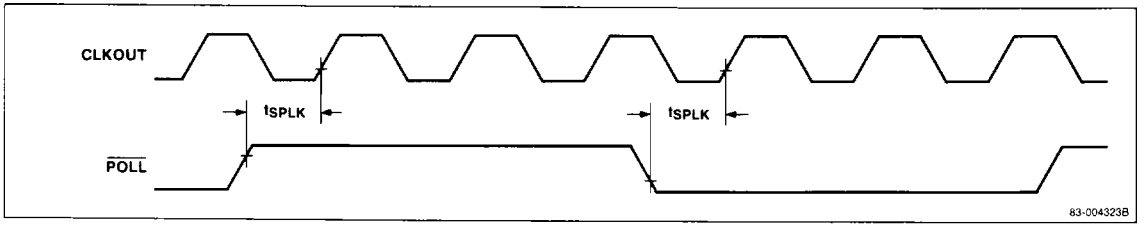


INTP, DMARQ Input

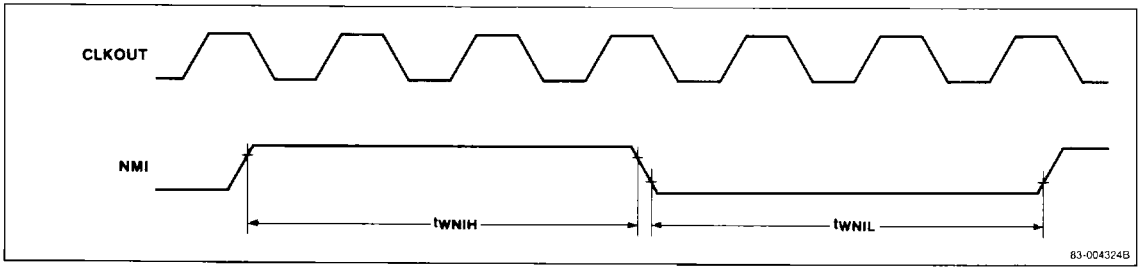


Timing Waveforms (cont)

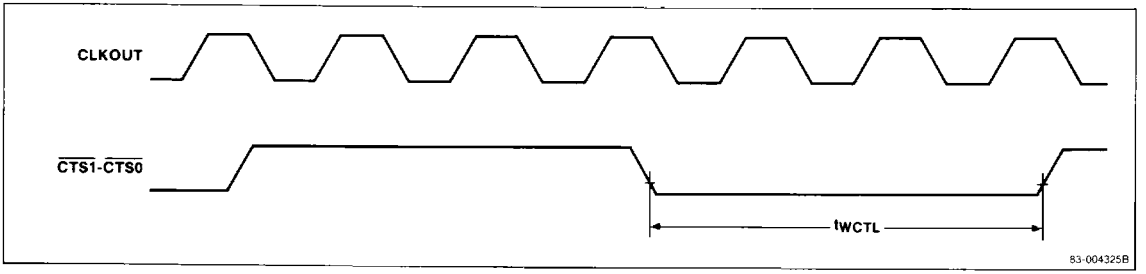
$\overline{POLL}$ Input



NMI Input

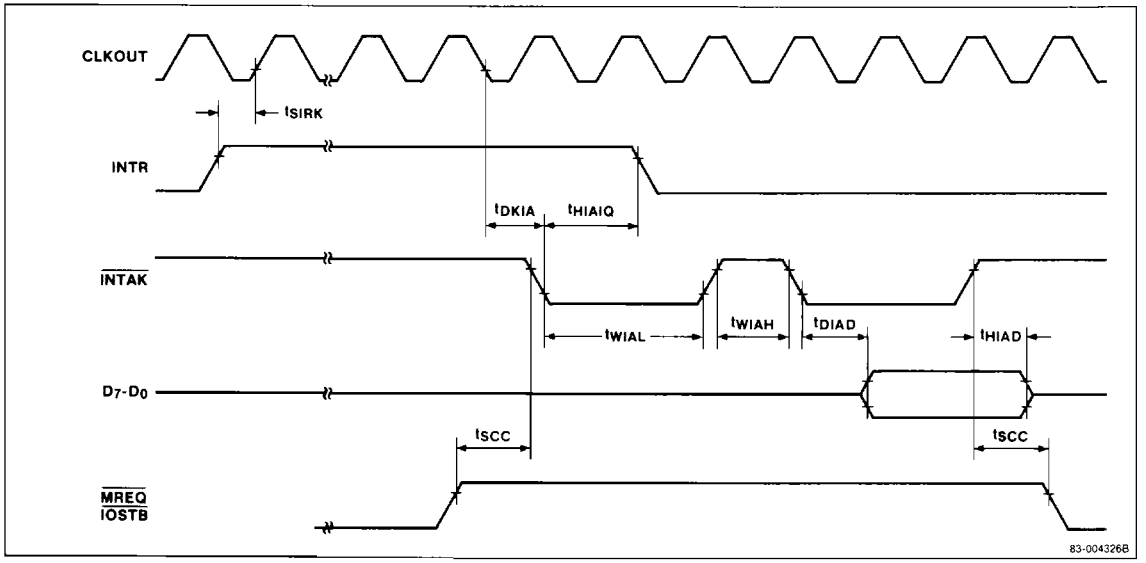


$\overline{CTS}$ Input

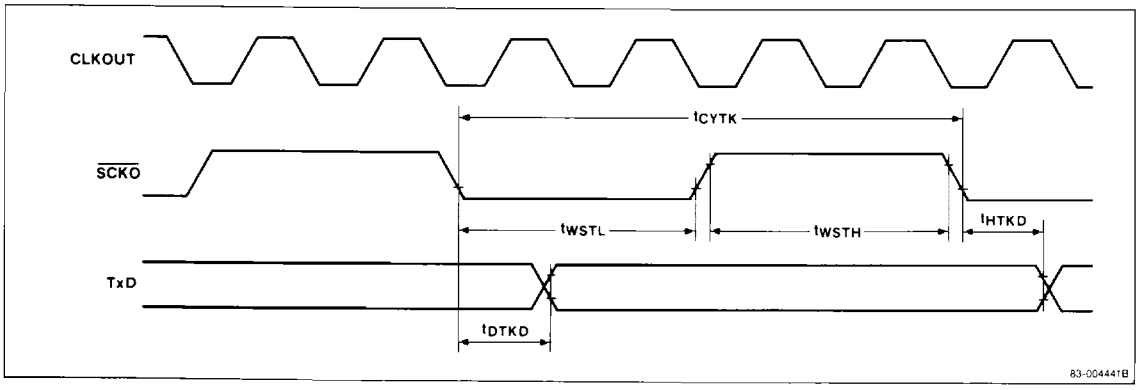


Timing Waveforms (cont)

INTR/$\overline{INTAK}$



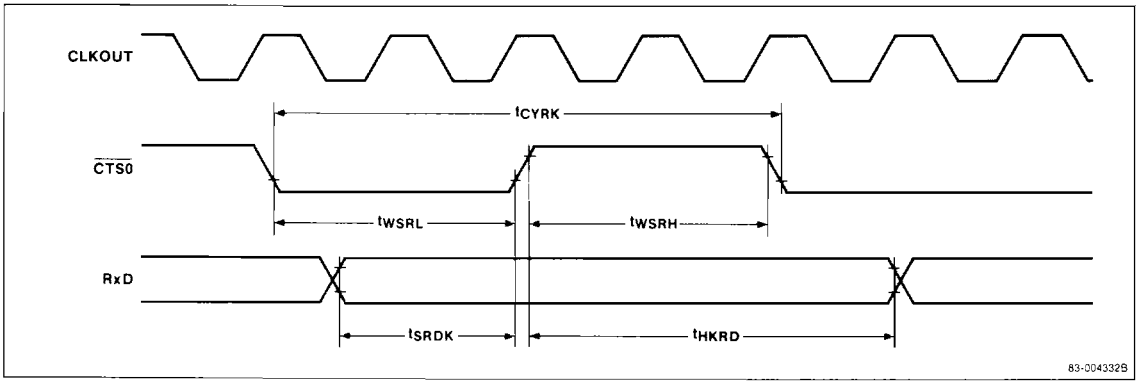
Serial Transmit



4a

Timing Waveforms (cont)

Serial Receive



Instruction Set

Instructions, grouped according to function, are described in a table near the end of this data sheet. Descriptions include source code, operation, opcode, number of bytes, and flag status. Supplementary information applicable to the instruction set is contained in the following tables.

- Symbols and Abbreviations
- Flag Symbols
- 8- and 16-Bit Registers. When mod = 11, the register is specified in the operation code by the byte/word operand (W = 0/1) and reg (000 to 111).
- Segment Registers. The segment register is specified in the operation code by sreg (00, 01, 10, or 11).
- Memory Addressing. The memory addressing mode is specified in the operation code by mod (00, 01, or 10) and mem (000 through 111).
- Instruction Clock Count. This table gives formulas for calculating the number of clock cycles occupied by each type of instruction. The formulas, which depend on byte/word operand and RAM enable/disable, have variables such as EA (effective address), W (wait states), and n (iterations or string instructions).

Symbols and Abbreviations

Identifier	Description
reg	8- or 16-bit general-purpose register
reg8	8-bit general-purpose register
reg16	16-bit general-purpose register
dmem	8- or 16-bit direct memory location
mem	8- or 16-bit memory location
mem8	8-bit memory location
mem16	16-bit memory location
mem32	32-bit memory location
sfr	8-bit special function register location
imm	Constant (0 to FFFFH)
imm16	Constant (0 to FFFFH)
imm8	Constant (0 to FFH)
imm4	Constant (0 to FH)
imm3	Constant (0 to 7)
acc	AW or AL register
sreg	Segment register
src-table	Name of 256-byte translation table
src-block	Name of block addressed by the IX register
dst-block	Name of block addressed by the IY register
near-proc	Procedure within the current program segment
far-proc	Procedure located in another program segment
near-label	Label in the current program segment
short-label	Label between −128 and +127 bytes from the end of instruction
far-label	Label in another program segment
memptr16	Word containing the offset of the memory location within the current program segment to which control is to be transferred
memptr32	Double word containing the offset and segment base address of the memory location to which control is to be transferred
regptr16	16-bit register containing the offset of the memory location within the program segment to which control is to be transferred
pop-value	Number of bytes of the stack to be discarded (0 to 64K bytes, usually even addresses)
fp-op	Immediate data to identify the instruction code of the external floating point operation
R	Register set
W	Word/byte field (0 to 1)
reg	Register field (000 to 111)
mem	Memory field (000 to 111)
mod	Mode field (00 to 10)
S:W	When S:W = 01 or 11, data = 16 bits. At all other times, data = 8 bits.
X, XXX, YYY, ZZZ	Data to identify the instruction code of the external floating point arithmetic chip
AW	Accumulator (16 bits)
AH	Accumulator (high byte)
AL	Accumulator (low byte)
BP	Base pointer register (16 bits)
BW	BW register (16 bits)
BH	BW register (high byte)
BL	BW register (low byte)
CW	CW register (16 bits)
CH	CW register (high byte)
CL	CW register (low byte)
DW	DW register (16 bits)
DH	DW register (high byte)
DL	DW register (low byte)
SP	Stack pointer (16 bits)
PC	Program counter (16 bits)
PSW	Program status word (16 bits)

Symbols and Abbreviations (cont)

Identifier	Description
IX	Index register (source) (16 bits)
IY	Index register (destination) (16 bits)
PS	Program segment register (16 bits)
SS	Stack segment register (16 bits)
DS_0	Data segment 0 register (16 bits)
DS_1	Data segment 1 register (16 bits)
AC	Auxiliary carry flag
CY	Carry flag
P	Parity flag
S	Sign flag
Z	Zero flag
DIR	Direction flag
IE	Interrupt enable flag
V	Overflow flag
BRK	Break flag
MD	Mode flag
(...)	Values in parentheses are memory contents
disp	Displacement (8 or 16 bits)
ext-disp8	16-bit displacement (sign-extension byte + 8-bit displacement)
temp	Temporary register (8/16/32 bits)
tmpcy	Temporary carry flag (1-bit)
seg	Immediate segment data (16 bits)
offset	Immediate offset data (16 bits)
←	Transfer direction
+	Addition
−	Subtraction
x	Multiplication
÷	Division
%	Modulo
AND	Logical product
OR	Logical sum
XOR	Exclusive logical sum
XXH	Two-digit hexadecimal value
XXXXH	Four-digit hexadecimal value

Flag Symbols

Identifier	Description
(blank)	No change
0	Cleared to 0
1	Set to 1
X	Set or cleared according to the result
U	Undefined
R	Value saved earlier is restored

8- and 16-Bit Registers (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 Registers

sreg	Register
00	DS_1
01	PS
10	SS
11	DS_0

Memory Addressing

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

Instruction Clock Count

Mnemonic	Operand	Clocks
ADD	reg8, reg8	2
	reg16, reg16	2
	reg8, mem8	EA+6+W
	reg16, mem16	EA+8+2W
	mem8, reg8	EA+8+2W [EA+6+W]
	mem16, reg16	EA+12+4W [EA+8+2W]
	reg8, imm8	5
	reg16, imm8	5
	reg16, imm16	6
	mem8, imm8	EA+9+2W [EA+7+2W]
	mem16, imm8	EA+9+2W [EA+7+2W]
	mem16, imm16	EA+14+4W [EA+10+4W]
	AL, imm8	5
	AW, imm16	6
ADD4S		22+(27+3W)n [22+(25+3W)n]
ADDC		Same as ADD
ADJ4A		9
ADJ4S		9
ADJBA		17
ADJBS		17
AND	reg8, reg8	2
	reg16, reg16	2
	reg8, mem8	EA+6+W
	reg16, mem16	EA+8+2W
	mem8, reg8	EA+8+2W [EA+6+W]
	mem16, reg16	EA+12+4W [EA+8+2W]
	reg8, imm8	5
	reg16, imm16	6
	mem8, imm8	EA+9+2W [EA+7+2W]
	mem16, imm16	EA+14+4W [EA+10+4W]
Bcond (conditional branch)		8 or 15
BCWZ		8 or 15
BR	near-label	12
	short-label	12
	regptr16	13
	memptr16	EA+17+2W
	far-label	15
	memptr32	EA+25+4W

Mnemonic	Operand	Clocks
BRK	3	55+10W [43+10W]
	imm8	56+10W [44+10W]
BRKCS		15
BRKV		55+10W [43+10W]
BTCLR		29
BUSLOCK		2
CALL	near-proc	22+2W [18+2W]
	regptr16	22+2W [18+2W]
	memptr16	EA+26+4W [EA+24+4W]
	far-proc	36+4W [34+4W]
	memptr32	EA+36+8W [EA+24+8W]
CHKIND		EA+26+4W
CLR1	CY	2
	DIR	2
	reg8, CL	8
	reg16, CL	8
	mem8, CL	EA+14+2W [EA+12+W]
	mem16, CL	EA+18+4W [EA+14+2W]
	reg8, imm3	7
	reg16, imm4	7
	mem8, imm3	EA+11+2W [EA+9+W]
	mem16, imm4	EA+15+4W [EA+10+2W]
CMP	reg8, reg8	2
	reg16, reg16	2
	reg8, mem8	EA+6+W
	reg16, mem16	EA+8+2W
	mem8, reg8	EA+6+W
	mem16, reg16	EA+8+2W
	reg8, imm8	5
	reg16, imm8	5
	reg16, imm16	6
	mem8, imm8	EA+7+W
	mem16, imm8	EA+10+2W
	mem16, imm16	EA+10+2W
	AL, imm8	5
	AW, imm16	6
CMP4S		22+(23+2W)n
CMPBK	mem8, mem8	23+2W [19+2W]
	mem16, mem16	27+4W [21+2W]

Notes:

(1) If the number of clocks is not the same for RAM enabled and RAM disabled conditions, the RAM enabled value is listed first, followed by the RAM disabled value in brackets; for example, EA+8+2W [EA+6+W].

(2) Symbols in the Clocks column are defined as follows.

EA = additional clock cycles required for calculation of the effective address

= 3 (mod 00 or 01) or 4 (mod 10)

W = number of wait states selected by the WTC register

n = number of iterations or string instructions

Instruction Clock Count (cont)

Mnemonic	Operand	Clocks
CMPBKB		16+(21+2W)n
CMPBKW		16+(25+4W)n
CMPM	mem8	17+W
	mem16	19+2W
CMPMB		16+(15+W)n
CMPMW		16+(17+2W)n
CVTBD		19
CVTBW		3
CVTDB		20
CVTWL		8
DBNZ		8 or 17
DBNZE		8 or 17
DBNZNE		8 or 17
DEC	reg8	5
	reg16	2
	mem8	EA+11+2W [EA+9+2W]
	mem16	EA+15+4W [EA+11+4W]
DI		4
DISPOSE		12+2W
DIV	AW, reg8	46-56
	AW, mem8	EA+48+W to EA+58+W
	DW:AW, reg16	54-64
	DW:AW, mem16	EA+58+2W to EA+68+2W
DIVU	AW, reg8	31
	AW, mem8	EA+33+W
	DW:AW, reg16	39
	DW:AW, mem16	EA+43+2W
DS0:		2
DS1:		2
EI		12
EXT	reg8, reg8	41-121
	reg8, imm4	42-122
FINT		2
FPO1		60+10W [48+10W]
FPO2		60+10W [48+10W]
HALT		0
IN	AL, imm8	14+W
	AW, imm8	16+2W
	AL, DW	13+W
	AW, DW	15+2W
INC	reg8	5
	reg16	2
	mem8	EA+11+2W [EA+9+2W]
	mem16	EA+15+4W [EA+11+4W]

Mnemonic	Operand	Clocks
INM	mem8, DW	19+2W [17+2W]
	mem16, DW	21+4W [17+4W]
	mem8, DW	18+(13+2W)n [18+(11+2W)n]
	mem16, DW	18+(15+4W)n [18+(11+4W)n]
INS	reg8, reg8	63-155
	reg8, imm4	64-156
LDEA		EA+2
LDM	mem8	12+W
	mem16	16+(12+2W)n
LDMB	mem16	14+2W
LDMW	mem8	16+(10+W)n
MOV	reg8, reg8	2
	reg16, reg16	2
	reg8, mem8	EA+6+W
	reg16, mem16	EA+8+2W
	mem8, reg8	EA+4+W [EA+2]
	mem16, reg16	EA+6+2W [EA+2]
	reg8, imm8	5
	reg16, imm16	6
	mem8, imm8	EA+5+W
	mem16, imm16	EA+5+2W
	AL, dmem8	9+W
	AW, dmem16	11+2W
	dmem8, AL	7+W [5]
	dmem16, AW	9+2W [5]
	sreg, reg16	4
	sreg, mem16	EA+10+2W
	reg16, sreg	3
	mem16, sreg	EA+7+2W [EA+3]
	AH, PSW	2
	PSW, AH	3
	DS0, reg16, memptr32	EA+19+4W
	DS1, reg16, memptr32	EA+19+4W
MOVBK	mem8, mem8	20+2W [16+W]
	mem16, mem16	16+(20+4W)n [16+(12+2W)n]
MOVBKB	mem8, mem8	16+(16+2W)n [16+(12+W)n]
MOVBKW	mem16, mem16	24+4W [20+2W]
MOVSPA		16
MOVSPB		11
MUL	AW, AL, reg8	31-40
	AW, AL, mem8	EA+33+W to EA+42+W
	DW:AW, AW, reg16	39-48
	DW:AW, AW, mem16	EA+43+2W to EA+52+2W
	reg16, reg16, imm8	39-49
	reg16, mem16, imm8	EA+43+2W to EA+53+2W
	reg16, reg16, imm16	40-50
	reg16, mem16, imm16	EA+44+2W to EA+54+2W

Instruction Clock Count (cont)

Mnemonic	Operand	Clocks
MULU	reg8	24
	mem8	EA+26+W
	reg16	32
	mem16	EA+34+2W
NEG	reg8	5
	reg16	5
	mem8	EA+11+2W [EA+9+W]
	mem16	EA+15+4W [EA+11+2W]
NOP		4
NOT	reg8	5
	reg16	5
	mem8	EA+11+2W [EA+9+W]
	mem16	EA+15+4W [EA+11+2W]
NOT1	CY	2
	reg8, CL	7
	reg16, CL	7
	mem8, CL	EA+13+2W [EA+11+W]
	mem16, CL	EA+17+4W [EA+13+2W]
	reg8, imm3	6
	reg16, imm4	6
	mem8, imm3	EA+10+2W [EA+8+W]
	mem16, imm4	EA+14+4W [EA+10+2W]
OR	reg8, reg8	2
	reg16, reg16	2
	reg8, mem8	EA+6+W
	reg16, mem16	EA+8+2W
	mem8, reg8	EA+8+2W [EA+6+W]
	mem16, reg16	EA+12+4W [EA+8+2W]
	reg8, imm8	5
	reg16, imm16	6
	mem8, imm8	EA+9+2W [EA+7+2W]
	mem16, imm16	EA+14+4W [EA+10+4W]
	AL, imm8	5
	AW, imm16	6
OUT	imm8, AL	10+W
	imm8, AW	10+2W
	DW, AL	9+W
	DW, AW	9+2W
OUTM	DW, mem8	19+2W [17+2W]
	DW, mem16	21+4W [17+4W]
	DW, mem8	18+(13+2W)n [18+(11+2W)n]
	DW, mem16	18+(15+4W)n [18+11+4W)n]
POLL		0
POP	reg16	12+2W
	mem16	EA+16+4W [EA+12+2W]
	DS1	13+2W
	SS	13+2W
	DS0	13+2W
	PSW	14+2W
	R	82+16W [58]
PREPARE	imm16, imm8	imm8 = 0: 27+2W imm8 = 1: 39+4W imm8 = n > 1: 46+19 (n−1)+4W
PS:		2
PUSH	reg16	10+2W [6]
	mem16	EA+18+4W [EA+14+4W]
	DS1	11+2W [7]
	PS	11+2W [7]
	SS	11+2W [7]
	DS0	11+2W [7]
	PSW	10+2W [6]
	R	82+16W [50]
	imm8	13+2W [9]
	imm16	14+2W [10]
REP		2
REPE		2
REPZ		2
REPC		2
REPNC		2
REPNE		2
REPNZ		2
RET	null	20+2W
	pop-value	20+2W
	null	29+4W
	pop-value	30+4W
RETI		43+6W [35+2W]
RETRBI		12
ROL	reg8, 1	8
	reg16, 1	8
	mem8, 1	EA+14+2W [EA+12+W]
	mem16, 1	EA+18+4W [EA+14+2W]
	reg8, CL	11+2n
	reg16, CL	11+2n
	mem8, CL	EA+17+2W+2n [EA+15+W+2n]
	mem16, CL	EA+21+4W+2n [EA+17+2W+2n]
	reg8, imm8	9+2n
	reg16, imm8	9+2n
	mem8, imm8	EA+13+2W+2n [EA+11+W+2n]
	mem16, imm8	EA+17+4W+2n [EA+13+2W+2n]
ROL4	reg8	17
	mem8	EA+18+2W [EA+16+2W]
ROLC		Same as ROL
ROR		Same as ROL
ROR4	reg8	21
	mem8	EA+24+2W [EA+22+2W]
RORC		Same as ROL
SET1	CY	2
	DIR	2

4a

Instruction Clock Count (cont)

Mnemonic	Operand	Clocks
SET1 (cont)	reg8, CL	7
	reg16, CL	7
	mem8, CL	EA+13+2W [EA+11+W]
	mem16, CL	EA+17+4W [EA+13+2W]
	reg8, imm3	6
	reg16, imm4	6
	mem8, imm3	EA+10+2W [EA+8+W]
	mem16, imm4	EA+14+4W [EA+10+2W]
SHL		Same as ROL
SHR		Same as ROL
SHRA		Same as ROL
SS:		2
STM	mem8	12+2 [10]
	mem16	16+(10+2W)n [16+(6+2W)n]
STMB	mem8	16+(8+W)n [16+(6+W)n]
STMW	mem16	14+2W [10]
STOP		0
SUB		Same as ADD
SUB4S		22+(27+3W)n [22+(25+3W)n]
SUBC		Same as ADD
TEST	reg8, reg8	4
	reg16, reg16	4
	reg8, mem8	EA+8+W
	reg16, mem16	EA+10+2W
	mem8, reg8	EA+8+W
	mem16, reg16	EA+10+2W
	reg8, imm8	7
	reg16, imm16	8
	mem8, imm8	EA+11+W
	mem16, imm16	EA+11+2W
	AL, imm8	5
	AW, imm16	6
TEST1	reg8, CL	7
	reg16, CL	7
	mem8, CL	EA+11+W
	mem16, CL	EA+13+2W
	reg8, imm3	6
	reg16, imm4	6
	mem8, imm3	EA+8+W
	mem16, imm4	EA+10+2W
TRANS		10+W
TRANSB		10+W
TSKSW		11
XCH	reg8, reg8	3
	reg16, reg16	3
	reg8, mem8	EA+10+2W [EA+8+2W]
	reg16, mem16	EA+14+4W [EA+10+4W]
	mem8, reg8	EA+10+2W [EA+8+2W]
	mem16, reg16	EA+14+4W [EA+10+4W]
	AW, reg16	4
	reg16, AW	4
XOR		Same as AND

Execution Clock Counts for Operations

	Byte		Word	
	RAM Enable	RAM Disable	RAM Enable	RAM Disable
Context switch interrupt (Note 1)	—	—	27	27
DMA (Single-step mode) (Note 2)	20 + 2W	20 + 2W	24 + 4W	24 + 4W
DMA (Demand release mode)	15 + W	15 + W	17 + 2W	17 + 2W
DMA (Burst mode)	(12 + 2W)n	(12 + 2W)n	(12 + 4W)n	(12 + 4W)n
DMA (Single-transfer mode)	33 + W + N	33 + W + N	35 + 2W + N	35 + 2W + N
Interrupt (INT pin)	—	—	62 + 6W	62 + 6W
Macro service, sfr ← mem (Note 2)	24 + W	19 + W	26 + 2W	21 + 2W
Macro service, mem ← sfr	22 + W	20 + W	22 + 2W	22 + 2W
Macro service (Search char mode), sfr ← mem	27 + W	27 + W	—	—
Macro service (Search char mode), mem ← sfr	37 + W	34 + W	—	—
Priority vectored interrupt, including NMI (Note 1)	—	—	58 + 10W	58 + 10W

N = number of clocks to complete the instruction currently executing.

Notes:

(1) Every interrupt (except NMI) has an additional associated latency time of 27 + N clocks. During the 27 clocks, the interrupt controller performs some overhead tasks such as arbitrating priority. This time should be added to the above listed interrupt and macro service execution times. NMI latency time is 18 + N clocks.

(2) The DMA and macro service clock counts listed are the required number of CPU clocks for each transfer.

(3) When an external interrupt is asserted, a maximum of 6 clocks is required for internal synchronization before the interrupt request flag is set. For an internal interrupt, a maximum of 2 clocks is required.

4a

Bus Controller Latency

	Mode	Clocks
HLDRQ latency		7 + 2W
DMA request latency (Note 1)	Burst	29 + N
	Single step	29 + N
	Demand release	29 + N
	Single transfer	31 + N

Notes:

(1) The listed DMA latency times are the maximum number of clocks when a DMA request is asserted until DMAAK or MREQ goes low in the corresponding DMA cycles. The test conditions are no wait states, no interrupts, no macro service requests, and no hold requests.

Instruction Set

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0								No. of Bytes	Flags AC	CY	V	P	S	Z
Data Transfer																	
MOV	reg, reg	reg ← reg	1 0 0 0 1 0 1 W	1 1	reg	reg					2						
	mem, reg	(mem) ← reg	1 0 0 0 1 0 0 W	mod	reg	mem					2-4						
	reg, mem	reg ← (mem)	1 0 0 0 1 0 1 W	mod	reg	mem					2-4						
	mem, imm	(mem) ← imm	1 1 0 0 0 1 1 W	mod	0 0 0	mem					3-6						
	reg, imm	reg ← imm	1 0 1 1 W	reg							2-3						
	acc, dmem	When W = 0 AL ← (dmem) When W = 1 AH ← (dmem + 1), AL ← (dmem)	1 0 1 0 0 0 0 W								3						
	dmem, acc	When W = 0 (dmem) ← AL When W = 1 (dmem + 1) ← AH, (dmem) ← AL	1 0 1 0 0 0 1 W								3						
	sreg, reg16	sreg ← reg16 sreg : SS, DS0, DS1	1 0 0 0 1 1 1 0	1 1 0	sreg	reg					2						
	sreg, mem16	sreg ← (mem16) sreg : SS, DS0, DS1	1 0 0 0 1 1 1 0	mod 0	sreg	mem					2-4						
	reg16, sreg	reg16 ← sreg	1 0 0 0 1 1 0 0	1 1 0	sreg	reg					2						
	mem16, sreg	(mem16) ← sreg	1 0 0 0 1 1 0 0	mod 0	sreg	mem					2-4						
	DS0, reg16, mem32	reg16 ← (mem32) DS0 ← (mem32 + 2)	1 1 0 0 0 1 0 1	mod	reg	mem					2-4						
	DS1, reg16, mem32	reg16 ← (mem32) DS1 ← (mem32 + 2)	1 1 0 0 0 1 0 0	mod	reg	mem					2-4						
	AH, PSW	AH ← S, Z, x, AC, x, P, x, CY	1 0 0 1 1 1 1 1								1						
	PSW, AH	S, Z, x, AC, x, P, x, CY ← AH	1 0 0 1 1 1 1 0								1	x	x		x	x	x
LDEA	reg16, mem16	reg16 ← mem16	1 0 0 0 1 1 0 1	mod	reg	mem					2-4						
TRANS	src-table	AL ← (BW + AL)	1 1 0 1 0 1 1 1								1						
XCH	reg, reg	reg ⟷ reg	1 0 0 0 0 1 1 W	1 1	reg	reg					2						
	mem, reg or reg, mem	(mem) ⟷ reg	1 0 0 0 0 1 1 W	mod	reg	mem					2-4						
	AW, reg16 or reg16, AW	AW ⟷ reg16	1 0 0 1 0	reg							1						
Repeat Prefixes																	
REPC		While CW ≠ 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (− 1). If there is a waiting interrupt, it is processed. When CY ≠ 1, exit the loop.	0 1 1 0 0 1 0 1								1						
REPNC		While CW ≠ 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (− 1). If there is a waiting interrupt, it is processed. When CY ≠ 0, exit the loop.	0 1 1 0 0 1 0 0								1						

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Repeat Prefixes (cont)										
REP REPE REPZ		While CW ≠ 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (− 1). If there is a waiting interrupt, it is processed. If the primitive block transfer instruction is CMPBK or CMPM and Z ≠ 1, exit the loop.	1 1 1 1 0 0 1 1	1						
REPNE REPNZ		While CW ≠ 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (− 1). If there is a waiting interrupt, it is processed. If the primitive block transfer instruction is CMPBK or CMPM and Z ≠ 0, exit the loop.	1 1 1 1 0 0 1 0	1						
Primitive Block Transfer										
MOVBK	dst-block, src-block	When W = 0 (IY) ← (IX) DIR = 0: IX ← IX + 1, IY ← IY + 1 DIR = 1: IX ← IX − 1, IY ← IY − 1 When W = 1 (IY + 1, IY) ← (IX + 1, IX) DIR = 0: IX ← IX + 2, IY ← IY + 2 DIR = 1: IX ← IX − 2, IY ← IY − 2	1 0 1 0 0 1 0 W	1						
CMPBK	src-block, dst-block	When W = 0 (IX) − (IY) DIR = 0: IX ← IX + 1, IY ← IY + 1 DIR = 1: IX ← IX − 1, IY ← IY − 1 When W = 1 (IX + 1, IX) − (IY + 1, IY) DIR = 0: IX ← IX + 2, IY ← IY + 2 DIR = 1: IX ← IX − 2, IY ← IY − 2	1 0 1 0 0 1 1 W	1	x	x	x	x	x	x
CMPM	dst-block	When W = 0 AL − (IY) DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1 AW − (IY + 1, IY) DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	1 0 1 0 1 1 1 W	1	x	x	x	x	x	x
LDM	src-block	When W = 0 AL ← (IX) DIR = 0: IX ← IX + 1; DIR = 1: IX ← IX − 1 When W = 1 AW ← (IX + 1, IX) DIR = 0: IX ← IX + 2; DIR = 1: IX ← IX − 2	1 0 1 0 1 1 0 W	1						
STM	dst-block	When W = 0 (IY) ← AL DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1 (IY + 1, IY) ← AW DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	1 0 1 0 1 0 1 W	1						
Bit Field Transfer										
INS	reg8, reg8	16-Bit field ← AW	0 0 0 0 1 1 1 1 0 0 1 1 0 0 0 1 1 1 reg reg	3						
	reg8, imm4	16-Bit field ← AW	0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 reg	4						

Instruction Set (cont)

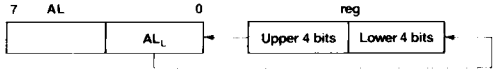
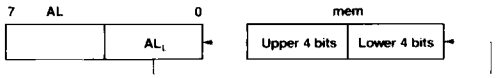
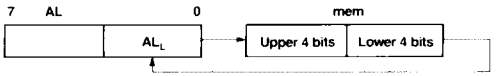
Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Bit Field Transfer (cont)											
EXT	reg8, reg8	AW ← 16-Bit field	0 0 0 0 1 1 1 1	0 0 1 1 reg reg	3						
	reg8, imm4	AW ← 16-Bit field	0 0 0 0 1 1 1 1	0 0 1 1 1 0 1 1 1 0 0 reg	4						
I/O											
IN	acc, imm8	When W = 0 AL ← (imm8) When W = 1 AH ← (imm8 + 1), AL ← (imm8)	1 1 1 0 0 1 0 W		2						
	acc, DW	When W = 0 AL ← (DW) When W = 1 AH ← (DW + 1), AL ← (DW)	1 1 1 0 1 1 0 W		1						
OUT	imm8, acc	When W = 0 (imm8) ← AL When W = 1 (imm8 + 1) ← AH, (imm8) ← AL	1 1 1 0 0 1 1 W		2						
	DW, acc	When W = 0 (DW) ← AL When W = 1 (DW + 1) ← AH, (DW) ← AL	1 1 1 0 1 1 1 W		1						
Primitive Block I/O Transfer											
INM	dst-block, DW	When W = 0 (IY) ← (DW) DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1 (IY + 1, IY) ← (DW + 1, DW) DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	0 1 1 0 1 1 0 W		1						
OUTM	DW, src-block	When W = 0 (DW) ← (IX) DIR = 0: IX ← IX + 1; DIR = 1: IX ← IX − 1 When W = 1 (DW + 1, DW) ← (IX + 1, IX) DIR = 0: IX ← IX + 2; DIR = 1: IX ← IX − 2	0 1 1 0 1 1 1 W		1						
Addition/Subtraction											
ADD	reg, reg	reg ← reg + reg	0 0 0 0 0 0 1 W	1 1 reg reg	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) + reg	0 0 0 0 0 0 0 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg + (mem)	0 0 0 0 0 0 1 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg + imm	1 0 0 0 0 0 S W	1 1 0 0 0 reg	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) + imm	1 0 0 0 0 0 S W	mod 0 0 0 mem	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL + imm When W = 1 AW ← AW + imm	0 0 0 0 0 1 0 W		2-3	x	x	x	x	x	x

NEC

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Addition/Subtraction (cont)											
ADDC	reg, reg	reg ← reg + reg + CY	0 0 0 1 0 0 1 W	1 1 reg reg	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) + reg + CY	0 0 0 1 0 0 0 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg + (mem) + CY	0 0 0 1 0 0 1 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg + imm + CY	1 0 0 0 0 0 S W	1 1 0 1 0 reg	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) + imm + CY	1 0 0 0 0 0 S W	mod 0 1 0 mem	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL + imm + CY When W = 1 AW ← AW + imm + CY	0 0 0 1 0 1 0 W		2-3	x	x	x	x	x	x
SUB	reg, reg	reg ← reg − reg	0 0 1 0 1 0 1 W	1 1 reg reg	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) − reg	0 0 1 0 1 0 0 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg − (mem)	0 0 1 0 1 0 1 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg − imm	1 0 0 0 0 0 S W	1 1 1 0 1 reg	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) − imm	1 0 0 0 0 0 S W	mod 1 0 1 mem	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL − imm When W = 1 AW ← AW − imm	0 0 1 0 1 1 0 W		2-3	x	x	x	x	x	x
SUBC	reg, reg	reg ← reg − reg − CY	0 0 0 1 1 0 1 W	1 1 reg reg	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) − reg − CY	0 0 0 1 1 0 0 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg − (mem) − CY	0 0 0 1 1 0 1 W	mod reg mem	2-4	x	x	x	x	x	
	reg, imm	reg ← reg − imm − CY	1 0 0 0 0 0 S W	1 1 0 1 1 reg	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) − imm − CY	1 0 0 0 0 0 S W	mod 0 1 1 mem	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL − imm − CY When W = 1 AW ← AW − imm − CY	0 0 0 1 1 1 0 W		2-3	x	x	x	x	x	x

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
BCD Operation										
ADD4S		dst BCD string ← dst BCD string + src BCD string	0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0	2	u	x	u	u	u	x
SUB4S		dst BCD string ← dst BCD string − src BCD string	0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 0	2	u	x	u	u	u	x
CMP4S		dst BCD string − src BCD string	0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0	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 1 1 0 0 0 reg	3						
	mem8		0 0 0 0 1 1 1 1 0 0 1 0 1 0 0 0 mod 0 0 0 mem	3-5						
ROR4	reg8		0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0 1 1 0 0 0 reg	3						
	mem8		0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0 mod 0 0 0 mem	3-5						
BCD Adjust										
ADJBA		When (AL AND 0FH) >9 or AC = 1, AL ← AL + 6, AH ← AH + 1, AC ← 1, CY ← AC, AL ← AL AND 0FH	0 0 1 1 0 1 1 1	1	x	x	u	u	u	u
ADJ4A		When (AL AND 0FH) >9 or AC = 1, AL ← AL + 6, CY ← CY OR AC, AC ← 1, When AL > 9FH, or CY = 1 AL ← AL + 60H, CY ← 1	0 0 1 0 0 1 1 1	1	x	x	u	x	x	x
ADJBS		When (AL AND 0FH) >9 or AC = 1, CY ← AC, AL ← AL AND 0FH	0 0 1 1 1 1 1 1	1	x	x	u	u	u	u
ADJ4S		When (AL AND 0FH) >9 or AC = 1, AL ← AL − 6, CY ← CY OR AC, AC ← 1, When AL > 9FH, or CY = 1 AL ← AL + 60H, CY ← 1	0 0 1 0 1 1 1 1	1	x	x	u	x	x	x

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Increment/Decrement											
INC	reg8	reg8 ← reg8 + 1	1 1 1 1 1 1 1 0	1 1 0 0 0 reg	2	x		x	x	x	x
	mem	(mem) ← (mem) + 1	1 1 1 1 1 1 1 W	mod 0 0 0 mem	2-4	x		x	x	x	x
	reg16	reg16 ← reg16 + 1	0 1 0 0 0 reg		1	x		x	x	x	x
DEC	reg8	reg8 ← reg8 − 1	1 1 1 1 1 1 1 0	1 1 0 0 1 reg	2	x		x	x	x	x
	mem	(mem) ← (mem) − 1	1 1 1 1 1 1 1 W	mod 0 0 1 mem	2-4	x		x	x	x	x
	reg16	reg16 ← reg16 − 1	0 1 0 0 1 reg		1	x		x	x	x	x
Multiplication											
MULU	reg8	AW ← AL x reg8 AH = 0: CY ← 0, V ← 0 AH ≠ 0: CY ← 1, V ← 1	1 1 1 1 0 1 1 0	1 1 1 0 0 reg	2	u	x	x	u	u	u
	mem8	AW ← AL x (mem8) AH = 0: CY ← 0, V ← 0 AH ≠ 0: CY ← 1, V ← 1	1 1 1 1 0 1 1 0	mod 1 0 0 mem	2-4	u	x	x	u	u	u
	reg16	DW, AW ← AW x reg16 DW = 0: CY ← 0, V ← 0 DW ≠ 0: CY ← 1, V ← 1	1 1 1 1 0 1 1 1	1 1 1 0 0 reg	2	u	x	x	u	u	u
	mem16	DW, AW ← AW x (mem16) DW = 0: CY ← 0, V ← 0 DW ≠ 0: CY ← 1, V ← 1	1 1 1 1 0 1 1 1	mod 1 0 0 mem	2-4	u	x	x	u	u	u
MUL	reg8	AW ← AL x reg8 AH = AL sign expansion: CY ← 0, V ← 0 AH ≠ AL sign expansion: CY ← 1, V ← 1	1 1 1 1 0 1 1 0	1 1 1 0 1 reg	2	u	x	x	u	u	u
	mem8	AW ← AL x (mem8) AH = AL sign expansion: CY ← 0, V ← 0 AH ≠ AL sign expansion: CY ← 1, V ← 1	1 1 1 1 0 1 1 0	mod 1 0 1 mem	2-4	u	x	x	u	u	u
	reg16	DW, AW ← AW x reg16 DW = AW sign expansion: CY ← 0, V ← 0 DW ≠ AW sign expansion: CY ← 1, V ← 1	1 1 1 1 0 1 1 1	1 1 1 0 1 reg	2	u	x	x	u	u	u
	mem16	DW, AW ← AW x (mem16) DW = AW sign expansion: CY ← 0, V ← 0 DW ≠ AW sign expansion: CY ← 1, V ← 1	1 1 1 1 0 1 1 1	mod 1 0 1 mem	2-4	u	x	x	u	u	u
	reg16, reg16, imm8	reg16 ← reg16 x imm8 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0 1 1 0 1 0 1 1	1 1 reg reg	3	u	x	x	u	u	u
	reg16, mem16, imm8	reg16 ← (mem16) x imm8 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0 1 1 0 1 0 1 1	mod reg mem	3-5	u	x	x	u	u	u

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Multiplication (cont)											
MUL (cont)	reg16, reg16, imm16	reg16 ← reg16 x imm16 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0 1 1 0 1 0 0 1	1 1 reg reg	4	u	x	x	u	u	u
	reg16, mem16, imm16	reg16 ← (mem16) x imm16 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0 1 1 0 1 0 0 1	mod reg mem	4-6	u	x	x	u	u	u
Unsigned Division											
DIVU	reg8	temp ← AW When temp ÷ reg8 > FFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg8, AL ← temp ÷ reg8	1 1 1 1 0 1 1 0	1 1 1 1 0 reg	2	u	u	u	u	u	u
	mem8	temp ← AW When temp ÷ (mem8) > FFH (SP 1, SP 2) ← PSW, (SP 3, SP 4) ← PS (SP 5, SP 6) ← PC, SP ← SP 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem8), AL ← temp ÷ (mem8)	1 1 1 1 0 1 1 0	mod 1 1 0 mem	2-4	u	u	u	u	u	u
	reg16	temp ← AW When temp ÷ reg16 > FFFFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg16, AL ← temp ÷ reg16	1 1 1 1 0 1 1 1	1 1 1 1 0 reg	2	u	u	u	u	u	u
	mem16	temp ← AW When temp ÷ (mem16) > FFFFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem16), AL ← temp ÷ (mem16)	1 1 1 1 0 1 1 1	mod 1 1 0 mem	2-4	u	u	u	u	u	u

NEC

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Signed Division											
DIV	reg8	temp ← AW When temp ÷ reg8 > 0 and temp ÷ reg8 > 7FH or temp ÷ reg8 < 0 and temp ÷ reg8 < 0 - 7FH – 1 (SP – 1, SP – 2) ← PSW, (SP – 3, SP – 4) ← PS (SP – 5, SP – 6) ← PC, SP ← SP – 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg8, AL ← temp ÷ reg8	1 1 1 1 0 1 1 0	1 1 1 1 1 reg	2	u	u	u	u	u	u
	mem8	temp ← AW When temp ÷ (mem8) > 0 and (mem8) > 7FH or temp ÷ (mem8) < 0 and temp ÷ (mem8) < 0 - 7FH – 1 (SP – 1, SP – 2) ← PSW, (SP – 3, SP – 4) ← PS (SP – 5, SP – 6) ← PC, SP ← SP – 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem8), AL ← temp ÷ (mem8)	1 1 1 1 0 1 1 0	mod 1 1 1 mem	2-4	u	u	u	u	u	u
	reg 16	temp ← DW, AW When temp ÷ reg 16 > 0 and reg 16 > 7FFFH or temp ÷ reg 16 < 0 - 7FFFH – 1 (SP – 1, SP – 2) ← PSW, (SP – 3, SP – 4) ← PS (SP – 5, SP – 6) ← PC, SP ← SP – 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg. 16, AL ← temp ÷ reg 16	1 1 1 1 0 1 1 1	1 1 1 1 1 reg	2	u	u	u	u	u	u
	mem 16	temp ← DW, AW When temp ÷ (mem 16) > 0 and (mem 16) > 7FFFH or temp ÷ (mem 16) < 0 and temp ÷ [**mem 16**] < 0 - 7FFFH – 1 (SP – 1, SP – 2)) ← PSW, (SP –3, SP – 4) ← PS (SP – 5, SP – 6) ← PC, SP ← SP – 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem 16), AL ← temp ÷ (mem 16)	1 1 1 1 0 1 1 1	mod 1 1 1 mem	2-4	u	u	u	u	u	u

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Data Conversion											
CVTBD		AH ← AL ÷ 0AH, AL ← AL % 0AH	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	2	u	u	u	x	x	x
CVTDB		AH ← 0, AL ← AH x 0AH + AL	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	2	u	u	u	x	x	x
CVTBW		When AL < 80H, AH ← 0, all other times AH ← FFH	1 0 0 1 1 0 0 0		1						
CVTWL		When AL < 8000H, DW ← 0, all other times DW ← FFFFH	1 0 0 1 1 0 0 1		1						
Comparison											
CMP	reg, reg	reg − reg	0 0 1 1 1 0 1 W	1 1 reg reg	2	x	x	x	x	x	x
	mem, reg	(mem) − reg	0 0 1 1 1 0 0 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, mem	reg − (mem)	0 0 1 1 1 0 1 W	mod reg mem	2-4	x	x	x	x	x	x
	reg, imm	reg − imm	1 0 0 0 0 0 S W	1 1 1 1 1 reg	3-4	x	x	x	x	x	x
	mem, imm	(mem) − imm	1 0 0 0 0 0 S W	mod 1 1 1 mem	3-6	x	x	x	x	x	x
	acc, imm	When W = 0, AL − imm When W = 1, AW − imm	0 0 1 1 1 1 0 W		2-3	x	x	x	x	x	x
Complement											
NOT	reg	reg ← $\overline{reg}$	1 1 1 1 0 1 1 W	1 1 0 1 0 reg	2						
	mem	(mem) ← ($\overline{mem}$)	1 1 1 1 0 1 1 W	mod 0 1 0 mem	2-4						
NEG	reg	reg ← $\overline{reg}$ + 1	1 1 1 1 0 1 1 W	1 1 0 1 1 reg	2	x	x	x	x	x	x
	mem	(mem) ← ($\overline{mem}$) + 1	1 1 1 1 0 1 1 W	mod 0 1 1 mem	2-4	x	x	x	x	x	x
Logical Operation											
TEST	reg, reg	reg AND reg	1 0 0 0 0 1 0 W	1 1 reg reg	2	u	0	0	x	x	x
	mem, reg or reg, mem	(mem) AND reg	1 0 0 0 0 1 0 W	mod reg mem	2-4	u	0	0	x	x	x
	reg, imm	reg AND imm	1 1 1 1 0 1 1 W	1 1 0 0 0 reg	3-4	u	0	0	x	x	x
	mem, imm	(mem) AND imm	1 1 1 1 0 1 1 W	mod 0 0 0 mem	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL AND imm8 When W = 1, AW AND imm8	1 0 1 0 1 0 0 W		2-3	u	0	0	x	x	x
AND	reg, reg	reg ← reg AND reg	0 0 1 0 0 0 1 W	1 1 reg reg	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) AND reg	0 0 1 0 0 0 0 W	mod reg mem	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg AND (mem)	0 0 1 0 0 0 1 W	mod reg mem	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg AND imm	1 0 0 0 0 0 0 W	1 1 1 0 0 reg	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) AND imm	1 0 0 0 0 0 0 W	mod 1 0 0 mem	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL ← AL AND imm8 When W = 1, AW ← AW AND imm16	0 0 1 0 0 1 0 W		2-3	u	0	0	x	x	x

NEC

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Logical Operation (cont)											
OR	reg, reg	reg ← reg OR reg	0 0 0 0 1 0 1 W	1 1 reg reg	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) OR reg	0 0 0 0 1 0 0 W	mod reg mem	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg OR (mem)	0 0 0 0 1 0 1 W	mod reg mem	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg OR imm	1 0 0 0 0 0 0 W	1 1 0 0 1 reg	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) OR imm	1 0 0 0 0 0 0 W	mod 0 0 1 mem	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL ← AL OR imm8 When W = 1, AW ← AW OR imm16	0 0 0 0 1 1 0 W		2-3	u	0	0	x	x	x
XOR	reg, reg	reg ← reg XOR reg	0 0 1 1 0 0 1 W	1 1 reg reg	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) XOR reg	0 0 1 1 0 0 0 W	mod reg mem	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg XOR (mem)	0 0 1 1 0 0 1 W	mod reg mem	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg XOR imm	1 0 0 0 0 0 0 W	1 1 1 1 0 reg	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) XOR imm	1 0 0 0 0 0 0 W	mod 1 1 0 mem	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL ← AL XOR imm8 When W = 1, AW ← AW XOR imm16	0 0 1 1 0 1 0 W		2-3	u	0	0	x	x	x
Bit Operation											
			2nd byte*	3rd byte*							
TEST1	reg8, CL	reg8 bit no. CL = 0: Z ← 1 reg8 bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 0	1 1 0 0 0 reg	3	u	0	0	u	u	x
	mem8, CL	(mem8) bit no. CL = 0: Z ← 1 (mem8) bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 0	mod 0 0 0 mem	3-5	u	0	0	u	u	x
	reg16, CL	reg16 bit no. CL = 0: Z ← 1 reg16 bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 1	1 1 0 0 0 reg	3	u	0	0	u	u	x
	mem16, CL	(mem16) bit no. CL = 0: Z ← 1 (mem16) bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 1	mod 0 0 0 mem	3-5	u	0	0	u	u	x
	reg8, imm3	reg8 bit no. imm3 = 0: Z ← 1 reg8 bit no. imm3 = 1: Z ← 0	0 0 0 1 1 0 0 0	1 1 0 0 0 reg	4	u	0	0	u	u	x
	mem8, imm3	(mem8) bit no. imm3 = 0: Z ← 1 (mem8) bit no. imm3 = 1: Z ← 0	0 0 0 1 1 0 0 0	mod 0 0 0 mem	4-6	u	0	0	u	u	x
	reg16, imm4	reg16 bit no. imm4 = 0: Z ← 1 reg16 bit no. imm4 = 1: Z ← 0	0 0 0 1 1 0 0 1	1 1 0 0 0 reg	4	u	0	0	u	u	x
	mem16, imm4	(mem16) bit no. imm4 = 0: Z ← 1 (mem16) bit no. imm4 = 1: Z ← 0	0 0 0 1 1 0 0 1	mod 0 0 0 mem	4-6	u	0	0	u	u	x
			2nd byte*	3rd byte*							

*Note: First byte = 0FH

Instruction Set (cont)

Bit Operation (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
			2nd byte*	3rd byte*							
NOT1	reg8, CL	reg8 bit no. CL ← $\overline{\text{reg8 bit no. CL}}$	0 0 0 1 0 1 1 0	1 1 0 0 0 reg	3						
	mem8, CL	(mem8) bit no. CL ← $\overline{\text{(mem8) bit no. CL}}$	0 0 0 1 0 1 1 0	mod 0 0 0 mem	3-5						
	reg16, CL	reg16 bit no. CL ← $\overline{\text{reg16 bit no. CL}}$	0 0 0 1 0 1 1 1	1 1 0 0 0 reg	3						
	mem16, CL	(mem16) bit no. CL ← $\overline{\text{(mem16) bit no. CL}}$	0 0 0 1 0 1 1 1	mod 0 0 0 mem	3-5						
	reg8, imm3	reg8 bit no. imm3 ← $\overline{\text{reg8 bit no. imm3}}$	0 0 0 1 1 1 1 0	1 1 0 0 0 reg	4						
	mem8, imm3	(mem8) bit no. imm3 ← $\overline{\text{(mem8) bit no. imm3}}$	0 0 0 1 1 1 1 0	mod 0 0 0 mem	4-6						
	reg16, imm4	reg16 bit no. imm4 ← (reg16) bit no. imm4	0 0 0 1 1 1 1 1	1 1 0 0 0 reg	4						
	mem16, imm4	(mem16) bit no. imm4 ← $\overline{\text{(mem16) bit no. imm4}}$	0 0 0 1 1 1 1 1	mod 0 0 0 mem	4-6						
			2nd byte*	3rd byte*							
			*Note: First byte = 0FH								
	CY	CY ← $\overline{\text{CY}}$	1 1 1 1 0 1 0 1		1		x				
			2nd byte*	3rd byte*							
CLR1	reg8, CL	reg8 bit no. CL ← 0	0 0 0 1 0 0 1 0	1 1 0 0 0 reg	3						
	mem8, CL	(mem8) bit no. CL ← 0	0 0 0 1 0 0 1 0	mod 0 0 0 mem	3-5						
	reg16, CL	reg16 bit no. CL ← 0	0 0 0 1 0 0 1 1	1 1 0 0 0 reg	3						
	mem16, CL	(mem16) bit no. CL ← 0	0 0 0 1 0 0 1 1	mod 0 0 0 mem	3-5						
	reg8, imm3	reg8 bit no. imm3 ← 0	0 0 0 1 1 0 1 0	1 1 0 0 0 reg	4						
	mem8, imm3	(mem8) bit no. imm3 ← 0	0 0 0 1 1 0 1 0	mod 0 0 0 mem	4-6						
	reg16, imm4	reg16 bit no. imm4 ← 0	0 0 0 1 1 0 1 1	1 1 0 0 0 reg	4						
	mem16, imm4	(mem16) bit no. imm4 ← 0	0 0 0 1 1 0 1 1	mod 0 0 0 mem	4-6						
			2nd byte*	3rd byte*							
			*Note: First byte = 0FH								
	CY	CY ← 0	1 1 1 1 1 0 0 0		1		0				
	DIR	DIR ← 0	1 1 1 1 1 1 0 0		1						

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Bit Operation (cont)											
SET1	reg8, CL	reg8 bit no. CL ← 1	0 0 0 1 0 1 0 0	1 1 0 0 0 reg	3						
	mem8, CL	(mem8) bit no. CL ← 1	0 0 0 1 0 1 0 0	mod 0 0 0 mem	3-5						
	reg16, CL	reg16 bit no. CL ← 1	0 0 0 1 0 1 0 1	1 1 0 0 0 reg	3						
	mem16, CL	(mem16) bit no. CL ← 1	0 0 0 1 0 1 0 1	mod 0 0 0 mem	3-5						
	reg8, imm3	reg8 bit no. imm3 ← 1	0 0 0 1 1 1 0 0	1 1 0 0 0 reg	4						
	mem8, imm3	(mem8) bit no. imm3 ← 1	0 0 0 1 1 1 0 0	mod 0 0 0 mem	4-6						
	reg16, imm4	reg16 bit no. imm4 ← 1	0 0 0 1 1 1 0 1	1 1 0 0 0 reg	4						
	mem16, imm4	(mem16) bit no. imm4 ← 1	0 0 0 1 1 1 0 1	mod 0 0 0 mem	4-6						
			2nd byte*	3rd byte*							
			*Note: First byte = 0FH								
	CY	CY ← 1	1 1 1 1 1 0 0 1		1		1				
	DIR	DIR ← 1	1 1 1 1 1 1 0 1		1						
Shift											
SHL	reg, 1	CY ← MSB of reg, reg ← reg x 2 When MSB of reg ≠ CY, V ← 1 When MSB of reg = CY, V ← 0	1 1 0 1 0 0 0 W	1 1 1 0 0 reg	2	u	x	x	x	x	x
	mem, 1	CY ← MSB of (mem), (mem) ← (mem) x 2 When MSB of (mem) ≠ CY, V ← 1 When MSB of (mem) = CY, V ← 0	1 1 0 1 0 0 0 W	mod 1 0 0 mem	2-4	u	x	x	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← MSB of reg, reg ← reg x 2, temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 1 0 0 reg	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← MSB of (mem), (mem) ← (mem) x 2, temp ← temp − 1	1 1 0 1 0 0 1 W	mod 1 0 0 mem	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← MSB of reg, reg ← reg x 2, temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 1 0 0 reg	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← MSB of (mem), (mem) ← (mem) x 2, temp ← temp − 1	1 1 0 0 0 0 0 W	mod 1 0 0 mem	3-5	u	x	u	x	x	x
SHR	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2 When MSB of reg ≠ bit following MSB of reg: V ← 1 When MSB of reg = bit following MSB of reg: V ← 0	1 1 0 1 0 0 0 W	1 1 1 0 1 reg	2	u	x	x	x	x	x

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Shift (cont)											
SHR (cont)	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2 When MSB of (mem) ≠ bit following MSB of (mem): V ← 1 When MSB of (mem) = bit following MSB of (mem): V ← 0	1 1 0 1 0 0 0 W	mod 1 0 1 mem	2-4	u	x	x	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 1 0 1 reg	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1	1 1 0 1 0 0 1 W	mod 1 0 1 mem	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 1 0 1 reg	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1	1 1 0 0 0 0 0 W	mod 1 0 1 mem	3-5	u	x	u	x	x	x
SHRA	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2, V ← 0 MSB of operand does not change	1 1 0 1 0 0 0 W	1 1 1 1 1 reg	2	u	x	0	x	x	x
	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2, V ← 0, MSB of operand does not change	1 1 0 1 0 0 0 W	mod 1 1 1 mem	2-4	u	x	0	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1 MSB of operand does not change	1 1 0 1 0 0 1 W	1 1 1 1 1 reg	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1 MSB of operand does not change	1 1 0 1 0 0 1 W	mod 1 1 1 mem	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1 MSB of operand does not change	1 1 0 0 0 0 0 W	1 1 1 1 1 reg	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1 MSB of operand does not change	1 1 0 0 0 0 0 W	mod 1 1 1 mem	3-5	u	x	u	x	x	x

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Rotation											
ROL	reg, 1	CY ← MSB of reg, reg ← reg x 2 + CY MSB of reg ≠ CY: V ← 1 MSB of reg = CY: V ← 0	1 1 0 1 0 0 0 W	1 1 0 0 0 reg	2		x	x			
	mem, 1	CY ← MSB of (mem), (mem) ← (mem) x 2 + CY MSB of (mem) ≠ CY: V ← 1 MSB of (mem) = CY: V ← 0	1 1 0 1 0 0 0 W	mod 0 0 0 mem	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← MSB of reg, reg ← reg x 2 + CY temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 0 0 reg	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← MSB of (mem), (mem) ← (mem) x 2 + CY temp ← temp − 1	1 1 0 1 0 0 1 W	mod 0 0 0 reg	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← MSB of reg, reg ← reg x 2 + CY temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 0 0 0 reg	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← MSB of (mem), (mem) ← (mem) x 2 + CY temp ← temp − 1	1 1 0 0 0 0 0 W	mod 0 0 0 mem	3-5		x	u			
ROR	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2 MSB of reg ← CY MSB of reg ≠ bit following MSB of reg: V ← 1 MSB of reg = bit following MSB of reg: V ← 0	1 1 0 1 0 0 0 W	1 1 0 0 1 reg	2		x	x			
	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← CY MSB of (mem) ≠ bit following MSB of (mem): V ← 1 MSB of (mem) = bit following MSB of (mem): V ← 0	1 1 0 1 0 0 0 W	mod 0 0 1 mem	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← CY temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 0 1 reg	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, MSB of (mem) ← CY temp ← temp − 1	1 1 0 1 0 0 1 W	mod 0 0 1 mem	2-4		x	u			

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Rotation (cont)											
ROR (cont)	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← CY temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 0 0 1 reg	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2 temp ← temp − 1	1 1 0 0 0 0 0 W	mod 0 0 1 mem	3-5		x	u			
Rotate											
ROLC	reg, 1	tmpcy ← CY, CY ← MSB of reg reg ← reg x 2 + tmpcy MSB of reg = CY: V ← 0 MSB of reg ≠ CY: V ← 1	1 1 0 1 0 0 0 W	1 1 0 1 0 reg	2		x	x			
	mem, 1	tmpcy ← CY, CY ← MSB of (mem) (mem) ← (mem) x 2 + tmpcy MSB of (mem) = CY: V ← 0 MSB of (mem) ≠ CY: V ← 1	1 1 0 1 0 0 0 W	mod 0 1 0 mem	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← MSB of reg, reg ← reg x 2 + tmpcy temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 1 0 reg	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← MSB of (mem), (mem) ← (mem) x 2 + tmpcy temp ← temp − 1	1 1 0 1 0 0 1 W	mod 0 1 0 mem	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← MSB of reg, reg ← reg x 2 + tmpcy temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 0 1 0 reg	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← MSB of (mem) (mem) ← (mem) x 2 + tmpcy temp ← temp − 1	1 1 0 0 0 0 0 W	mod 0 1 0 mem	3-5		x	u			

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Rotate (cont)											
RORC	reg, 1	tmpcy ← CY, CY ← LSB of reg reg ← reg ÷ 2, MSB of reg ← tmpcy MSB of reg ≠ bit following MSB of reg: V ← 1 MSB of reg = bit following MSB of reg: V ← 0	1 1 0 1 0 0 0 W	1 1 0 1 1 reg	2		x	x			
	mem, 1	tmpcy ← CY, CY ← LSB of (mem) (mem) ← (mem) ÷ 2, MSB of (mem) ← tmpcy MSB of (mem) ≠ bit following MSB of (mem): V ← 1 MSB of (mem) = bit following MSB of (mem): V ← 0	1 1 0 1 0 0 0 W	mod 0 1 1 mem	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← tmpcy, temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 1 1 reg	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← tmpcy, temp ← temp − 1	1 1 0 1 0 0 1 W	mod 0 1 1 mem	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0 repeat this operation, tmpcy ← CY, CY ← LSB of reg, reg ← reg ÷ 2 MSB of reg ← tmpcy, temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 0 1 1 reg	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← tmpcy, temp ← temp − 1	1 1 0 0 0 0 0 W	mod 0 1 1 mem	3-5		x	u			
Subroutine Control Transfer											
CALL	near-proc	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← PC + disp	1 1 1 0 1 0 0 0		3						
	regptr16	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← regptr16	1 1 1 1 1 1 1 1	1 1 0 1 0 reg	2						
	memptr16	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← (memptr16)	1 1 1 1 1 1 1 1	mod 0 1 0 mem	2-4						
	far-proc	(SP − 1, SP − 2) ← PS, (SP − 3, SP − 4) ← PC SP ← SP − 4, PS ← seg, PC ← offset	1 0 0 1 1 0 1 0		5						
	memptr32	(SP − 1, SP − 2) ← PS, (SP − 3, SP − 4) ← PC SP ← SP − 4, PS ← (memptr32 + 2), PC ← (memptr32)	1 1 1 1 1 1 1 1	mod 0 1 1 mem	2-4						

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Subroutine Control Transfer (cont)											
RET		PC ← (SP + 1, SP), SP ← SP + 2	1 1 0 0 0 0 1 1		1						
	pop-value	PC ← (SP + 1, SP) SP ← SP + 2, SP ← SP + pop-value	1 1 0 0 0 0 1 0		3						
		PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2) SP ← SP + 4	1 1 0 0 1 0 1 1		1						
	pop-value	PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2) SP ← SP + 4, SP ← SP + pop-value	1 1 0 0 1 0 1 0		3						
Stack Manipulation											
PUSH	mem16	(SP − 1, SP − 2) ← (mem16), SP ← SP − 2	1 1 1 1 1 1 1 1	mod 1 1 0 mem	2-4						
	reg16	(SP − 1, SP − 2) ← reg16, SP ← SP − 2	0 1 0 1 0 reg		1						
	sreg	(SP− 1, SP− 2) ← sreg, SP ← SP − 2	0 0 0 sreg 1 1 0		1						
	PSW	(SP − 1, SP − 2) ← PSW, SP ← SP − 2	1 0 0 1 1 1 0 0		1						
	R	Push registers on the stack	0 1 1 0 0 0 0 0		1						
	imm	(SP − 1, SP − 2) ← imm SP ← SP − 2, When S = 1, sign extension	0 1 1 0 1 0 S 0		2-3						
POP	mem16	(mem16) ← (SP + 1, SP), SP ← SP + 2	1 0 0 0 1 1 1 1	mod 0 0 0 mem	2-4						
	reg16	reg16 ← (SP + 1, SP), SP ← SP + 2	0 1 0 1 1 reg		1						
	sreg	sreg ← (SP + 1, SP) sreg : SS, DS0, DS1 SP ← SP + 2	0 0 0 sreg 1 1 1		1						
	PSW	PSW ← (SP + 1, SP), SP ← SP + 2	1 0 0 1 1 1 0 1		1	R	R	R	R	R	R
	R	Pop registers from the stack	0 1 1 0 0 0 0 1		1						
PREPARE	imm16, imm8	Prepare new stack frame	1 1 0 0 1 0 0 0		4						
DISPOSE		Dispose of stack frame	1 1 0 0 1 0 0 1		1						
Branch											
BR	near-label	PC ← PC + disp	1 1 1 0 1 0 0 1		3						
	short-label	PC ← PC + ext-disp8	1 1 1 0 1 0 1 1		2						
	regptr16	PC ← regptr16	1 1 1 1 1 1 1 1	1 1 1 0 0 reg	2						
	memptr16	PC ← (memptr16)	1 1 1 1 1 1 1 1	mod 1 0 0 mem	2-4						
	far-label	PS ← seg, PC ← offset	1 1 1 0 1 0 1 0		5						
	memptr32	PS ← (memptr32 + 2), PC ← (memptr32)	1 1 1 1 1 1 1 1	mod 1 0 1 mem	2-4						

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	No. of Bytes	Flags AC CY V P S Z
Conditional Branch					
BV	short-label	if V = 1, PC ← PC + ext-disp8	0 1 1 1 0 0 0 0	2	
BNV	short-label	if V = 0, PC ← PC + ext-disp8	0 1 1 1 0 0 0 1	2	
BC, BL	short-label	if CY = 1, PC ← PC + ext-disp8	0 1 1 1 0 0 1 0	2	
BNC, BNL	short-label	if CY = 0, PC ← PC + ext-disp8	0 1 1 1 0 0 1 1	2	
BE, BZ	short-label	if Z = 1, PC ← PC + ext-disp8	0 1 1 1 0 1 0 0	2	
BNE, BNZ	short-label	if Z = 0, PC ← PC + ext-disp8	0 1 1 1 0 1 0 1	2	
BNH	short-label	if CY OR Z = 1, PC ← PC + ext-disp8	0 1 1 1 0 1 1 0	2	
BH	short-label	if CY OR Z = 0, PC ← PC + ext-disp8	0 1 1 1 0 1 1 1	2	
BN	short-label	if S = 1, PC ← PC + ext-disp8	0 1 1 1 1 0 0 0	2	
BP	short-label	if S = 0, PC ← PC + ext-disp8	0 1 1 1 1 0 0 1	2	
BPE	short-label	if P = 1, PC ← PC + ext-disp8	0 1 1 1 1 0 1 0	2	
BPO	short-label	if P = 0, PC ← PC + ext-disp8	0 1 1 1 1 0 1 1	2	
BLT	short-label	if S XOR V = 1, PC ← PC + ext-disp8	0 1 1 1 1 1 0 0	2	
BGE	short-label	if S XOR V = 0, PC ← PC + ext-disp8	0 1 1 1 1 1 0 1	2	
BLE	short-label	if (S XOR V) OR Z = 1, PC ← PC + ext-disp8	0 1 1 1 1 1 1 0	2	
BGT	short-label	if (S XOR V) OR Z = 0, PC ← PC + ext-disp8	0 1 1 1 1 1 1 1	2	
DBNZNE	short-label	CW ← CW − 1 if Z = 0 and CW ≠ 0, PC ← PC + ext-disp8	1 1 1 0 0 0 0 0	2	
DBNZE	short-label	CW ← CW − 1 if Z = 1 and CW ≠ 0, PC ← PC + ext-disp8	1 1 1 0 0 0 0 1	2	
DBNZ	short-label	CW ← CW − 1 if CW ≠ 0, PC ← PC + ext-disp8	1 1 1 0 0 0 1 0	2	
BCWZ	short-label	if CW = 0, PC ← PC + ext-disp8	1 1 1 0 0 0 1 1	2	
BTCLR	sfr. imm3, short-label	if bit no. imm3 of (sfr) = 1, PC ← PC + ext − disp8, bit no. imm3 of (sfr) ← 0	0 0 0 0 1 1 1 1 1 0 0 1 1 1 0 0	5	
Interrupt					
BRK	3	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PS ← (15, 14), PC ← (13, 12)	1 1 0 0 1 1 0 0	1	
	imm8 (≠ 3)	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PC ← (n x 4, + 1, n x 4) PS ← (n x 4 + 3, n x 4 + 2) n = imm8	1 1 0 0 1 1 0 1	2	

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Bytes	Flags AC	CY	V	P	S	Z
Interrupt (cont)											
BRKV		When V = 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PS ← (19, 18), PC ← (17, 16)	1 1 0 0 1 1 1 0		1						
RETI		PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2), PSW ← (SP + 5, SP + 4), SP ← SP + 6	1 1 0 0 1 1 1 1		1	R	R	R	R	R	R
RETRBI		PC ← Save PC, PSW ← Save PSW	0 0 0 0 1 1 1 1	1 0 0 1 0 0 0 1	2	R	R	R	R	R	R
FINT		Indicates that interrupt service routine to the interrupt controller built in the CPU has been completed	0 0 0 0 1 1 1 1	1 0 0 1 0 0 1 0	2						
CHKIND	reg16, mem32	When (mem32) > reg16 or (mem32 + 2) < reg16 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (23, 22), PC ← (21, 20)	0 1 1 0 0 0 1 0	mod reg mem	2-4						
CPU Control											
HALT		CPU Halt	1 1 1 1 0 1 0 0			1					
STOP		CPU Halt	0 0 0 0 1 1 1 1	1 0 1 1 1 1 1 0		1					
BUSLOCK		Bus Lock Prefix	1 1 1 1 0 0 0 0			1					
FPO1 (Note 1)	fp-op	No Operation	1 1 0 1 1 X X X	1 1 Y Y Y Z Z Z	2						
	fp-op, mem	data bus ← (mem)	1 1 0 1 1 X X X	mod Y Y Y mem	2-4						
FPO2 (Note 1)	fp-op	No Operation	0 1 1 0 0 1 1 X	1 1 Y Y Y Z Z Z	2						
	fp-op, mem	data bus ← (mem)	0 1 1 0 0 1 1 X	mod Y Y Y mem	2-4						
POLL		Poll and wait	1 0 0 1 1 0 1 1		1						
NOP		No Operation	1 0 0 1 0 0 0 0		1						
DI		IE ← 0	1 1 1 1 1 0 1 0		1						
EI		IE ← 1	1 1 1 1 1 0 1 1		1						
DS0; DS1; PS; SS		Segment override prefix	0 0 1 sreg 1 1 0		1						

Notes:

(1) Does not execute on the V25, but does generate an interrupt.

NEC

Instruction Set (cont)

Mnemonic	Operand	Operation	Operation Code 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	No. of Bytes	Flags AC	CY	V	P	S	Z
Register Bank Switching																									
MOVSPA			0	0	0	0	1	1	1	1	0	0	1	0	0	1	0	1	2						
BRKCS	reg16		0	0	0	0	1	1	1	1	0	0	1	0	1	1	0	1	3						
MOVSPB	reg16		0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	1	3						
			1	1	1	1	1		reg																
TSKSW	reg16		0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	0	3	x	x	x	x	x	x
			1	1	1	1	1		reg																