NEC
NEC Electronics Inc.

μPD79011 16-Bit Microcomputer: Single-Chip, CMOS, With Built-In RTOS

Description

The μPD79011 is an upgraded μPD70322 (V25™) single-chip microcomputer with a built-in real-time operating system (RTOS).

The μPD79011 provides high-speed multitask processing particularly suited for real-time event processing and as a kernel of an embedded control system for process control and data processing applications.

The RTOS kernel provides extensive system calls for task synchronization, control, and communication as well as interrupt and time management.

The μPD79011 instruction set is the same as the V25 instruction set. The μPD79011 hardware is also identical to the standard V25, but uses 6K of the internal ROM for RTOS system code. Refer to the V25 Data Sheet.

Features

□ Real-time multitask processing

□ Supports five types of system calls
— Task management
— Communication management
— Memory management
— Time management
— Interrupt management

□ High-speed response to events
— System call processing shortens time to 41 μs (minimum) when operated at 8 MHz
— High-speed task switching using V25 register banks

□ Flexibility to perform status changes by event driven task scheduling function

□ System clock: 8 MHz maximum

□ V25 hardware compatibility

□ CMOS technology

□ Development tools
— V25 software can be used without modification
— Relocatable assembler (RA70320)
— C compiler (CC70116)
— Concurrent CP/M®, MS-DOS®, VMS™, and UNIX™ base

Ordering Information

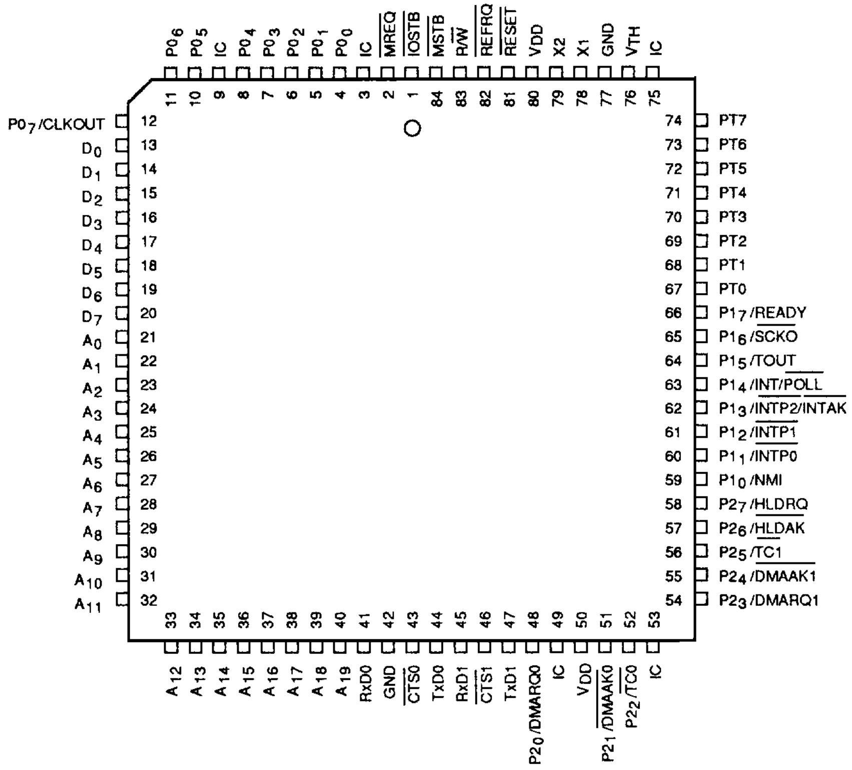
Part Number	Clock	Package
μPD79011L-8	8 MHz	84-pin PLCC
GJ-8	8 MHz	94-pin plastic QFP

4h

V25 is a trademark of NEC Corporation.
CP/M is a registered trademark of Digital Research, Inc.
MS-DOS is a registered trademark of Microsoft Corporation.
VMS is a trademark of Digital Equipment Corporation.
UNIX is a trademark of AT&T Bell Laboratories.

Pin Configurations

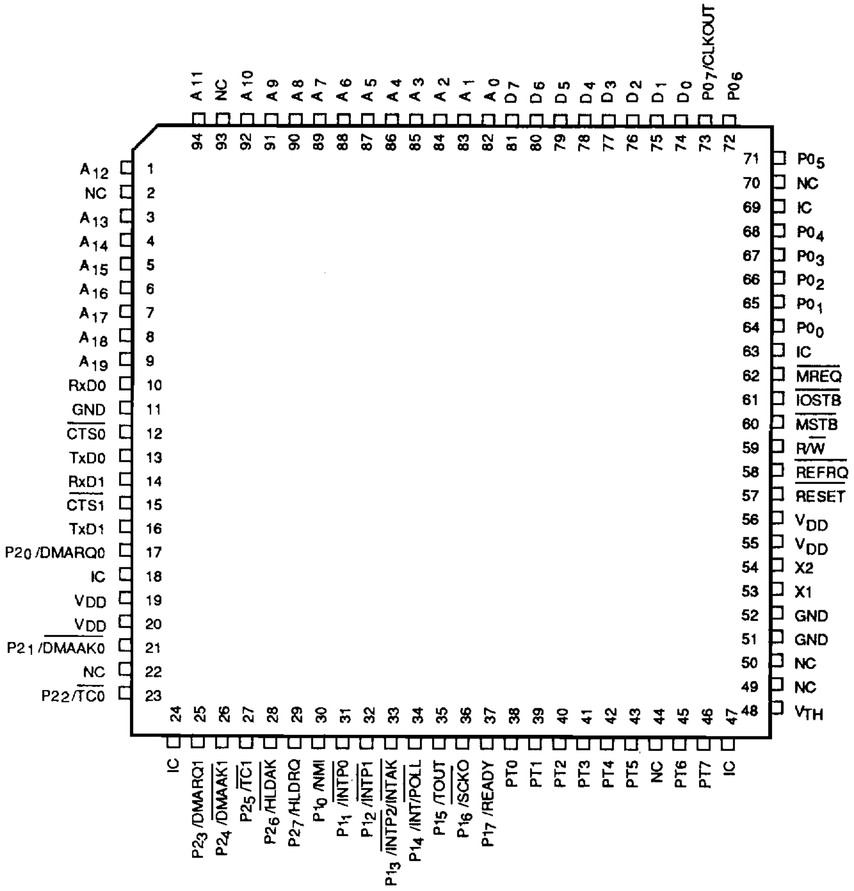
84-Pin PLCC



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

83SL-6719B

94-Pin Plastic QFP



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

83SL-6720B

Pin Identification

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

$\overline{\text{CTSn}}$, RxDn, TxDn, $\overline{\text{SCKO}}$ (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, $\overline{\text{DMAAKn}}$, $\overline{\text{TCn}}$ (DMA Request, DMA Acknowledge, Terminal Count)

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

$\overline{\text{HLDAK}}$ (Hold Acknowledge)

The $\overline{\text{HLDAK}}$ 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, $\overline{\text{MREQ}}$, R/$\overline{\text{W}}$, $\overline{\text{MSTB}}$, $\overline{\text{REFRQ}}$, and $\overline{\text{IOSTB}}$.

INT (Interrupt Request)

INT is a maskable, active-high, vectored request interrupt. 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.

INTP0-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 is 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 parallel 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 instruction, the CPU checks the status of this pin and, if low, program execution continues. If high, the CPU checks 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 synchronizes and inserts 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.

$\overline{\text{REFRQ}}$ (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)

R/$\overline{\text{W}}$ output allows external hardware to determine if the current operation is a read or a 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. 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.

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.

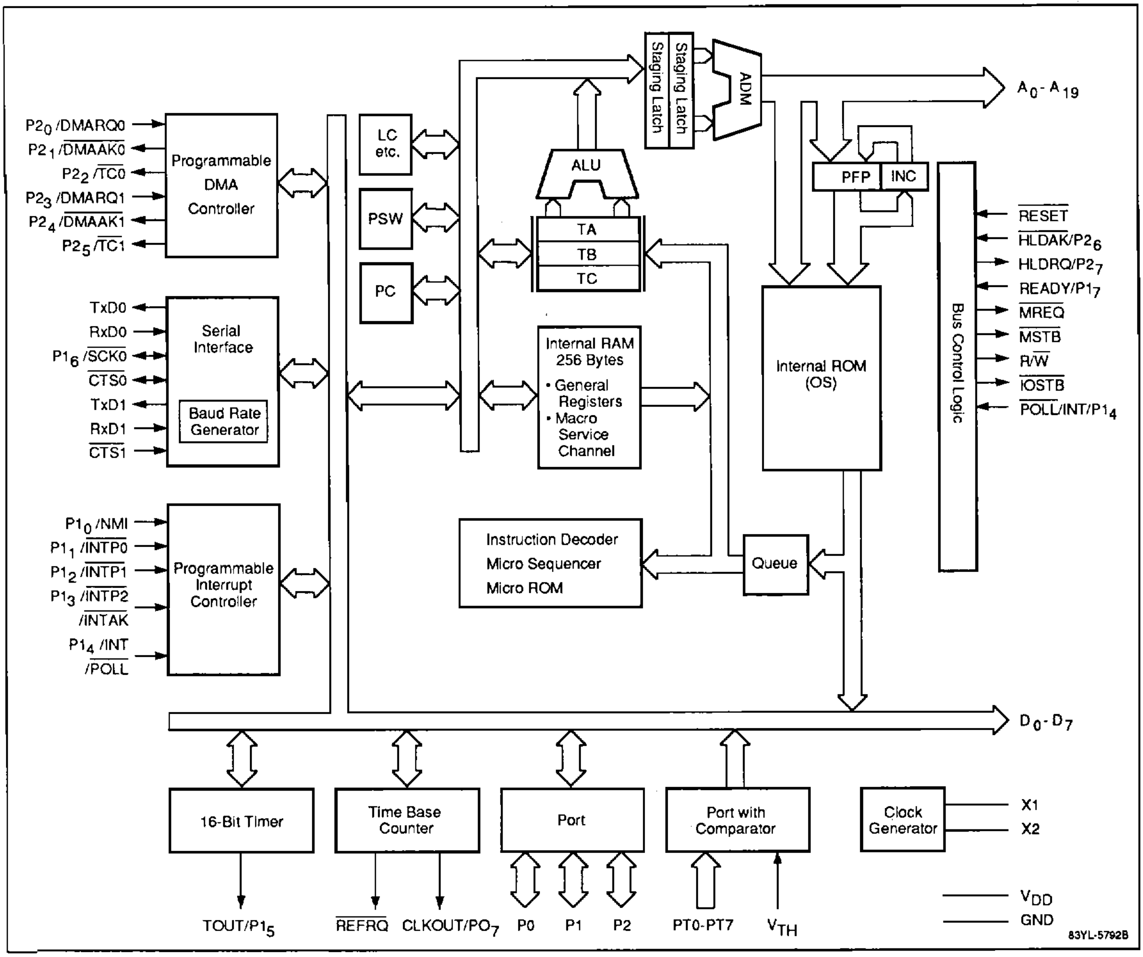
GND (Ground)

Two ground connections reduce internal noise.

IC (Internal Connection)

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

μPD79011 Block Diagram



ELECTRICAL SPECIFICATIONS

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 (≤ +7.0 V)
Output voltage, V_O	−0.5 to V_{DD}+0.5 (≤ +7.0 V)
Threshold voltage, V_{TH}	−0.5 to V_{DD}+0.5 (≤ +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

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

Comparator Characteristics

T_A = −10 to +70°C; V_{DD} = +5.0 V ±10%

Parameter	Symbol	Min	Max	Unit
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

Capacitance

T_A = 25°C; V_{DD} = 0 V

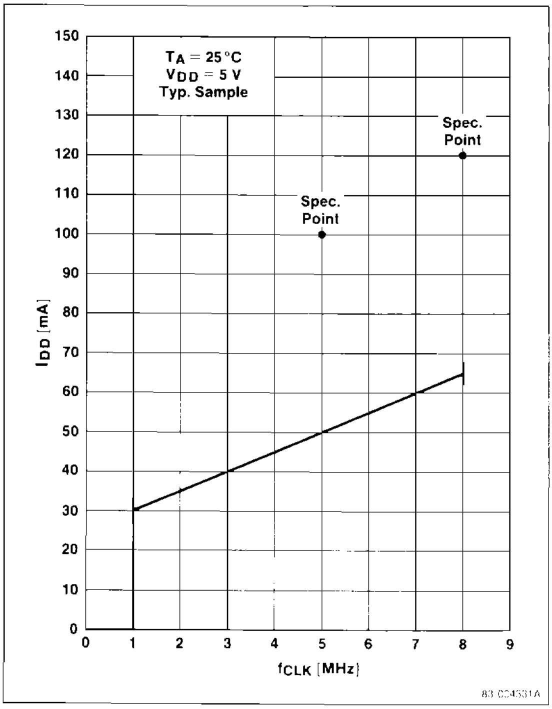
Parameter	Symbol	Min	Max	Unit	Conditions
Input capacitance	C_I		10	pF	f = 1 MHz; unmeasured pins returned to ground
Output capacitance	C_O		20	pF	
I/O capacitance	C_{IO}		20	pF	

DC Characteristics

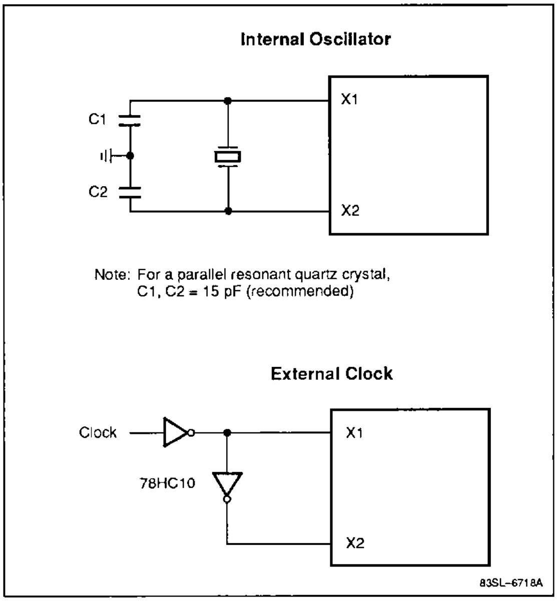
T_A = −10 to +70°C; V_{DD} = +5.0 V ±10%

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

Supply Current vs Clock Frequency



External System Clock Control Source



Note: For a parallel resonant quartz crystal, C1, C2 = 15 pF (recommended)

Recommended Oscillator Components

Ceramic Resonator		Capacitors	
Manufacturer	**Product No.**	**C1 (pF)**	**C2 (pF)**
Kyocera	KBR-10.0M	33	33
Murata Mfg.	CSA.10.0MT	47	47
	CSA16.0MX040	30	30
TDK	FCR10.M2S	30	30
	FCR16.0M2S	15	6

AC Characteristics

T_A = −10 to +70°C; V_{DD} = +5.0 V ±10%

Parameter	Symbol	Min	Max	Unit	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, $\overline{\text{RESET}}$, NMI
Input rise, fall time (Schmitt)	t_{IRS}, t_{IFS}		30	ns	$\overline{\text{RESET}}$, NMI
Output rise, fall time	t_{OR}, t_{OF}		20	ns	Except CLKOUT
X1 cycle time	t_{CYX}	98	250	ns	5-MHz CPU clock
		62	250	ns	8-MHz CPU clock
X1 width, low	t_{WXL}	35		ns	5-MHz CPU clock
		20		ns	8-MHz CPU clock
X1 width, high	t_{WXH}	20		ns	5-MHz CPU clock
		20		ns	8-MHz CPU clock
X1 rise, fall time	t_{XR}, t_{XF}		20	ns	8-MHz CPU clock
CLKOUT cycle time	t_{CYK}	125	2000	ns	fx/2, T = t_{CYK}

AC Characteristics (cont)

Parameter	Symbol	Min	Max	Unit	Conditions
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 valid to input data valid	t_{DADR}		T(n+1.5) − 90	ns	Note 2
$\overline{\text{MREQ}}$ to data delay time	t_{DMRD}		T(n+1) − 75	ns	
$\overline{\text{MSTB}}$ to data delay time	t_{DMSD}		T(n+0.5) − 75	ns	
$\overline{\text{MREQ}}$ to $\overline{\text{TC}}$ delay time	t_{DMRTC}		0.5T+50	ns	
$\overline{\text{MREQ}}$ to $\overline{\text{MSTB}}$ delay time	t_{DMRMS}	0.5T − 35	0.5+35	ns	
$\overline{\text{MREQ}}$ width, low	t_{WMRL}	T(n+1) − 30		ns	
Address hold time	t_{HMA}	0.5T − 30		ns	
Input data hold time	t_{HMDR}	0		ns	
Next control setup time	t_{SCC}	T − 25		ns	
$\overline{\text{TC}}$ width, low	t_{WTCL}	2T − 30		ns	
Address data output	t_{DADW}	0.5T+50		ns	
$\overline{\text{MREQ}}$ delay time	t_{DAMR}	0.5T − 30		ns	
$\overline{\text{MSTB}}$ delay time	t_{DAMS}	T − 30		ns	
$\overline{\text{MSTB}}$ 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	
$\overline{\text{IOSTB}}$ delay time	t_{DAIS}	0.5T − 30		ns	
$\overline{\text{IOSTB}}$ to data input	t_{DISD}		T(n+1) − 90	ns	
$\overline{\text{IOSTB}}$ width, low	t_{WISL}	T(n+1) − 30		ns	
Address hold time	t_{HISA}	0.5T − 30		ns	
Data input 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
$\overline{\text{DMAAK}}$ read width, low	t_{WDMRL}	T(n+1.5) − 30		ns	
$\overline{\text{DMAAK}}$ to TC delay time	t_{DDATC}		0.5T+50	ns	
$\overline{\text{DMAAK}}$ write width, low	t_{WDMWL}	T(n+1) − 30		ns	
$\overline{\text{REFRQ}}$ delay time	t_{DARF}	0.5T − 30		ns	
$\overline{\text{REFRQ}}$ width, low	t_{WRFL}	(n+1)T − 30		ns	
Address hold time	t_{HRFA}	0.5T − 30		ns	
$\overline{\text{RESET}}$ width, low	t_{WRSL1}	30		ms	STOP mode release; power-on reset
$\overline{\text{RESET}}$ width, low	t_{WRSL2}	5		μs	System warm reset
$\overline{\text{MREQ}}$, $\overline{\text{IOSTB}}$ to READY setup time	t_{SCRY}		T(n − 1) − 100	ns	n ≥ 2
$\overline{\text{MREQ}}$, $\overline{\text{IOSTB}}$ to READY hold time	t_{HCRY}	T(n − 1)		ns	n ≥ 2

AC Characteristics (cont)

Parameter	Symbol	Min	Max	Unit	Conditions
$\overline{\text{HLDAK}}$ output delay time	t_{DKHA}		80	ns	
Bus control float to $\overline{\text{HLDAK}}$ ↓	t_{CFHA}	T − 50		ns	
$\overline{\text{HLDAK}}$ ↑ to control output time	t_{DHAC}	T − 50		ns	
HLDRQ ↓ to control output time	t_{DHQC}	3T +30		ns	
$\overline{\text{HLDAK}}$ width, low	t_{WHAL}		T	ns	
HLDRQ setup time	t_{SHQK}	30		ns	
HLDRQ to $\overline{\text{HLDAK}}$ delay time	t_{DHQHA}		3T+160	ns	
HLDRQ width, low	t_{WHQL}	1.5T		ns	
$\overline{\text{INTP}}$, DMARQ setup time	t_{SIQK}	30		ns	
$\overline{\text{INTP}}$, DMARQ width, high	t_{WIQH}	8T		ns	
$\overline{\text{INTP}}$, DMARQ width, low	t_{WIQL}	8T		ns	
$\overline{\text{POLL}}$ setup time	t_{SPLK}	30		ns	
NMI width, high	t_{WNIH}	5		μs	
NMI width, low	t_{WNIL}	5		μs	
$\overline{\text{CTS}}$ width, low	t_{WCTL}	2T		ns	
INTR setup time	t_{SIRK}	30		ns	
INTR hold time	t_{HIAIQ}	0		ns	
$\overline{\text{INTAK}}$ width, low	t_{WIAL}	2T − 30		ns	
$\overline{\text{INTAK}}$ delay time	t_{DKIA}		80	ns	
$\overline{\text{INTAK}}$ width, high	t_{WIAH}	T − 30		ns	
$\overline{\text{INTAK}}$ to data delay time	t_{DIAD}		2T − 130	ns	
$\overline{\text{INTAK}}$ to data hold time	t_{HIAD}	0	0.5T	ns	
$\overline{\text{SCKO}}$ cycle time	t_{CYTK}	1000		ns	
$\overline{\text{SCKO}}$ (TSCK) width, high	t_{WSTH}	450		ns	
$\overline{\text{SCKO}}$ (TSCK) width, low	t_{WSTL}	450		ns	
TxD delay time	t_{DTKD}		210	ns	
TxD hold time	t_{HTKD}	20		ns	
$\overline{\text{CTS0}}$ (RSCK) cycle time	t_{CYRK}	1000		ns	
$\overline{\text{CTS0}}$ (RSCK) width, high	t_{WSRH}	420		ns	
$\overline{\text{CTS0}}$ (RSCK) width, low	t_{WSRL}	420		ns	
RxD setup time	t_{SRDK}	80		ns	
RxD hold time	t_{HKRD}	80		ns	

Notes: (1) T = CPU clock period (t_{CYK}) (2) n = number of wait states inserted

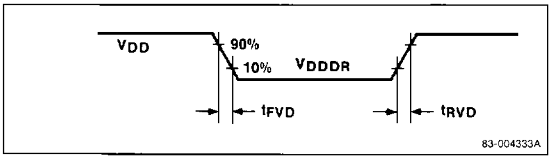
STOP Mode Data Retention Characteristics

T_A = −10 to +70°C

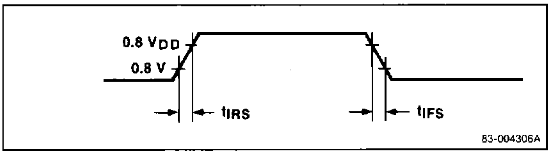
Parameter	Symbol	Min	Max	Unit
Data retention voltage	V_{DDDE}	2.5	5.5	V
V_{DD} rise time	t_{LFVD}	200		μs
V_{DD} fall time	$t_{\text{FVD°C}}$	200		μs

Timing Waveforms

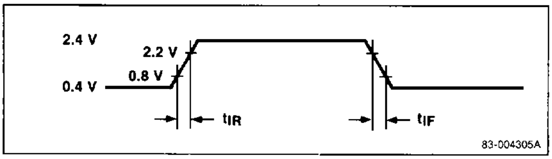
Stop Mode Data Retention Timing



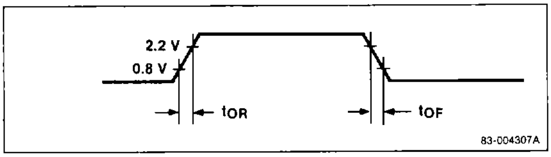
AC Input Waveform 2 ($\overline{RESET}$, NMI)



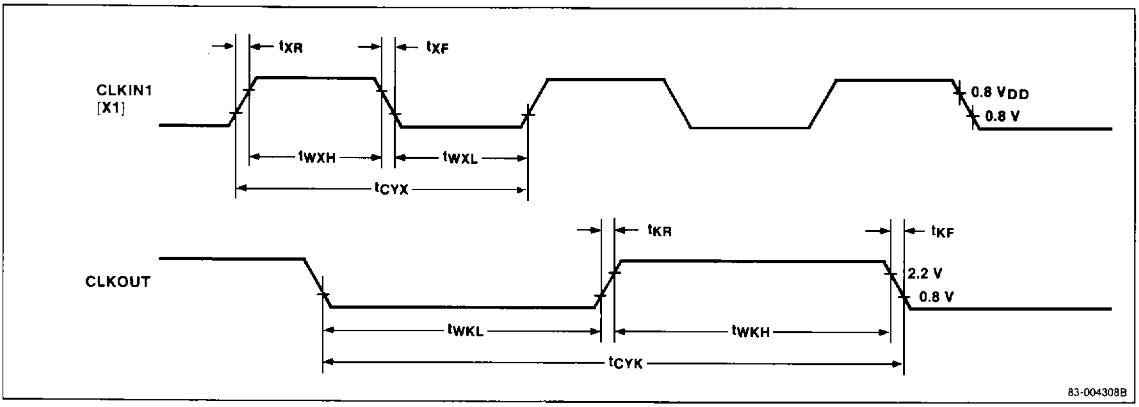
AC Input Waveform 1 (Except X1, X2, $\overline{RESET}$, NMI)



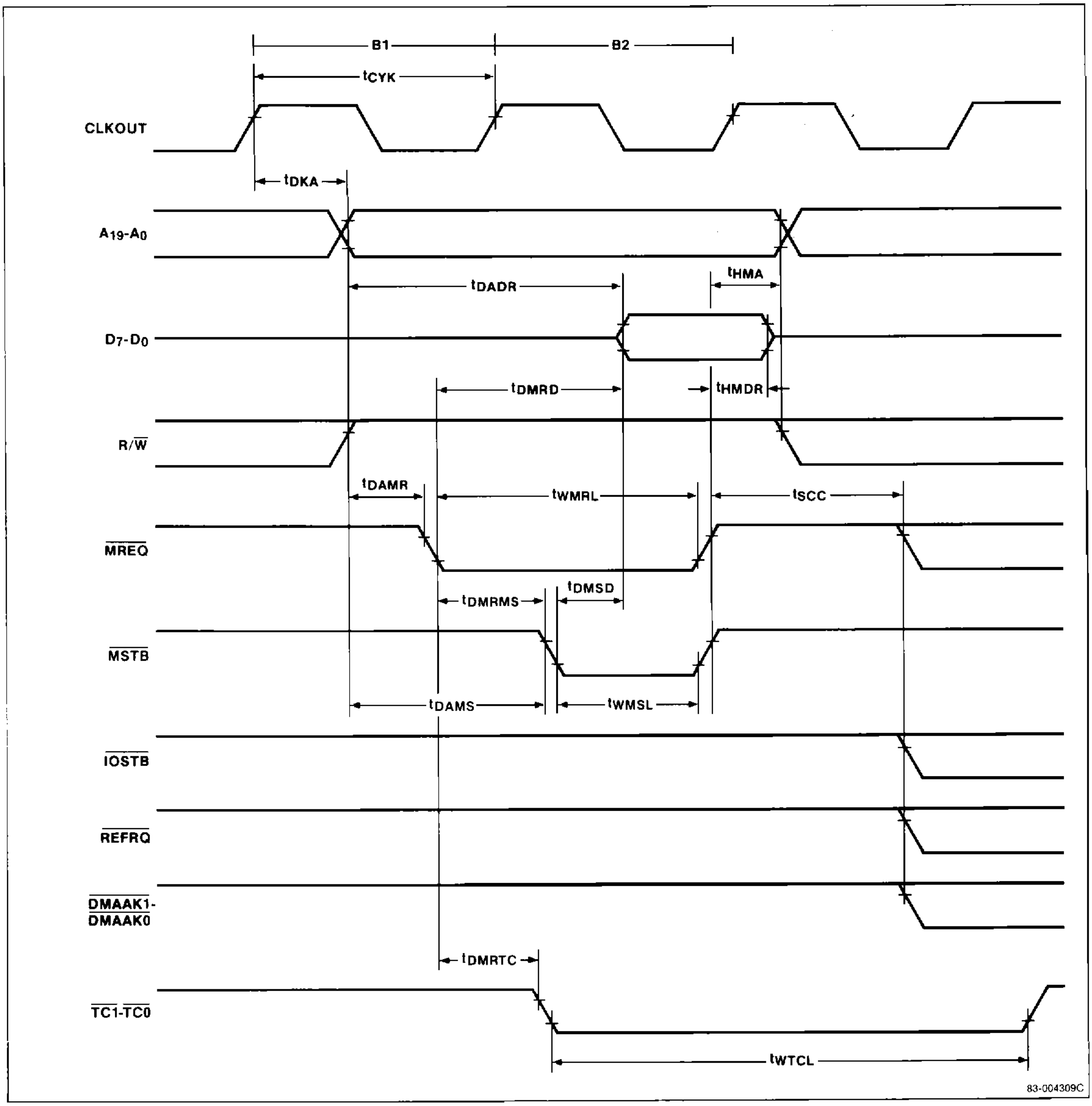
AC Output Test Point (Except CLKOUT)



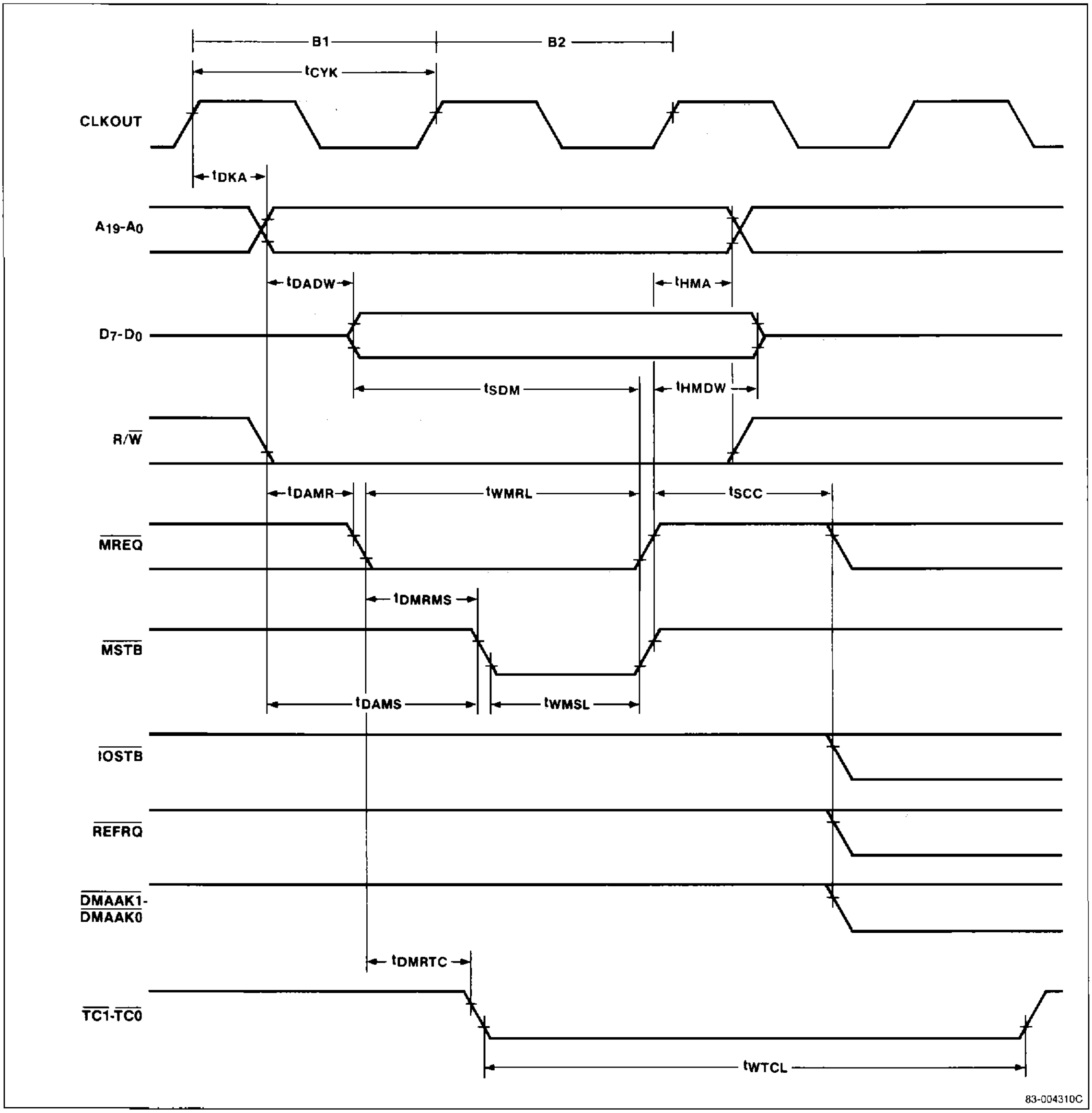
Clock In and Clock Out



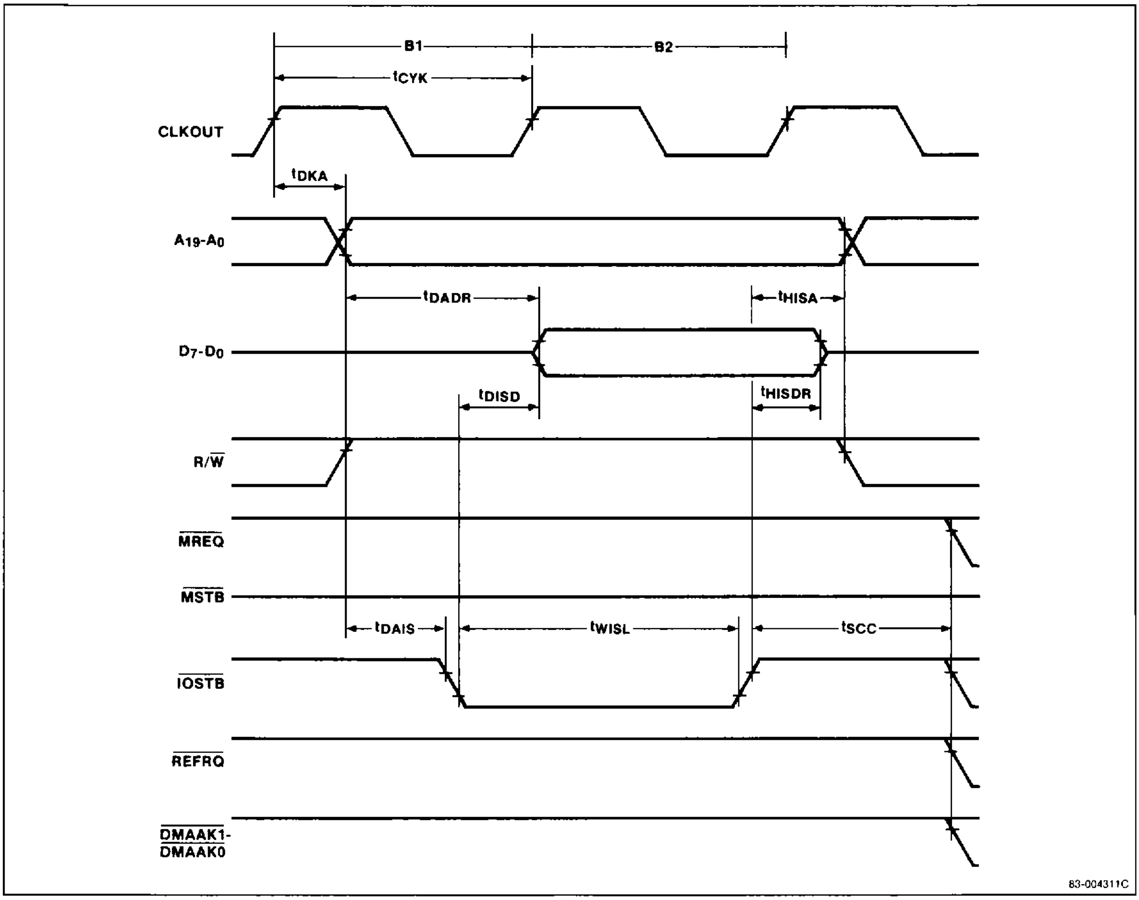
Memory Read


B1
B2
tCYK
CLKOUT
tDKA
A19-A0
tDADR
tHMA
D7-D0
tDMRD
tHMDR
R/W̄
tDAMR
tWMRL
tSCC
MREQ
tDMRMS
tDMSD
MSTB
tDAMS
tWMSL
IOSTB
REFRQ
DMAAK1-
DMAAK0
tDMRTC
TC1-TC0
tWTCL
83-004309C

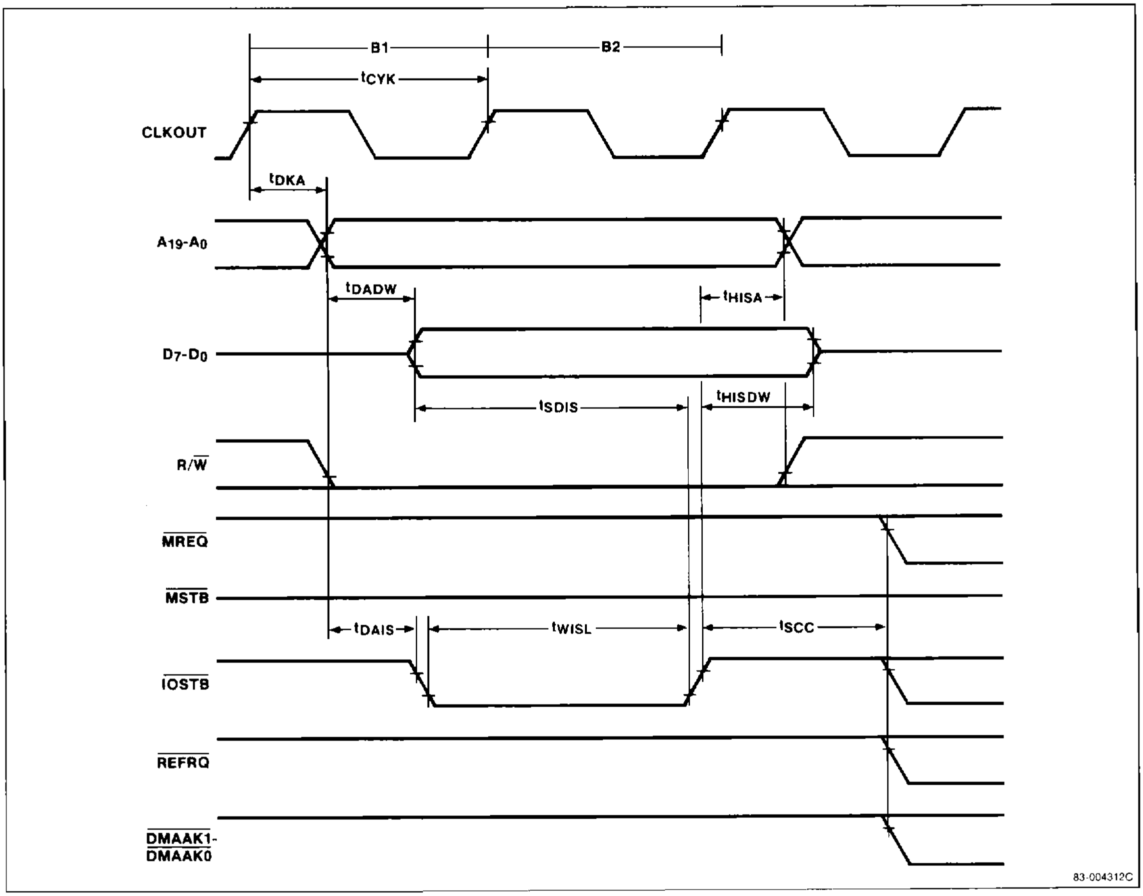
Memory Write



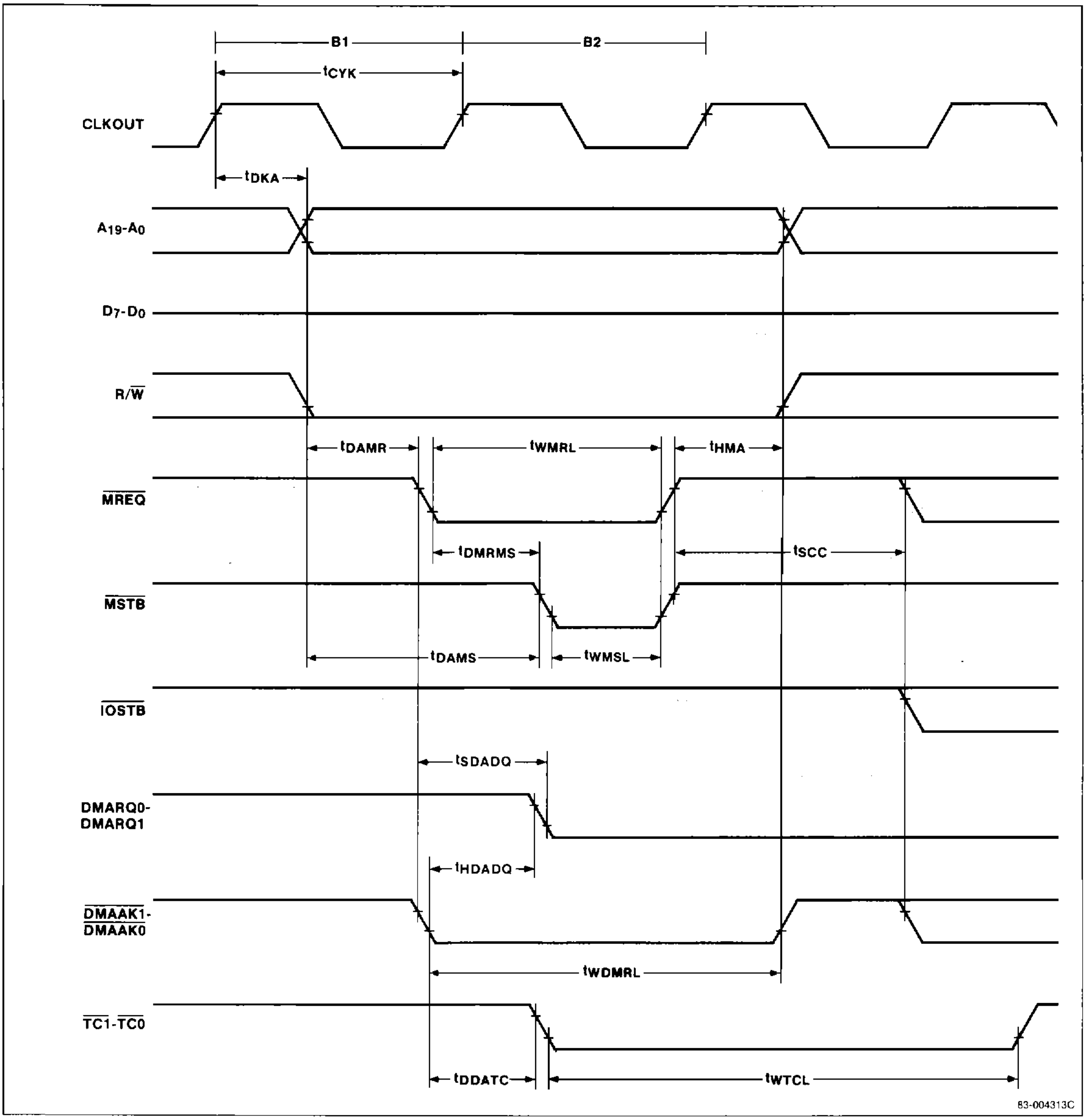
I/O Read


B1
B2
tCYK
CLKOUT
tDKA
A19-A0
tDADR
tHISA
D7-D0
tDISD
tHISDR
R/W
MREQ
MSTB
tDAIS
tWISL
tSCC
IOSTB
REFRQ
DMAAK1-
DMAAK0
83-004311C

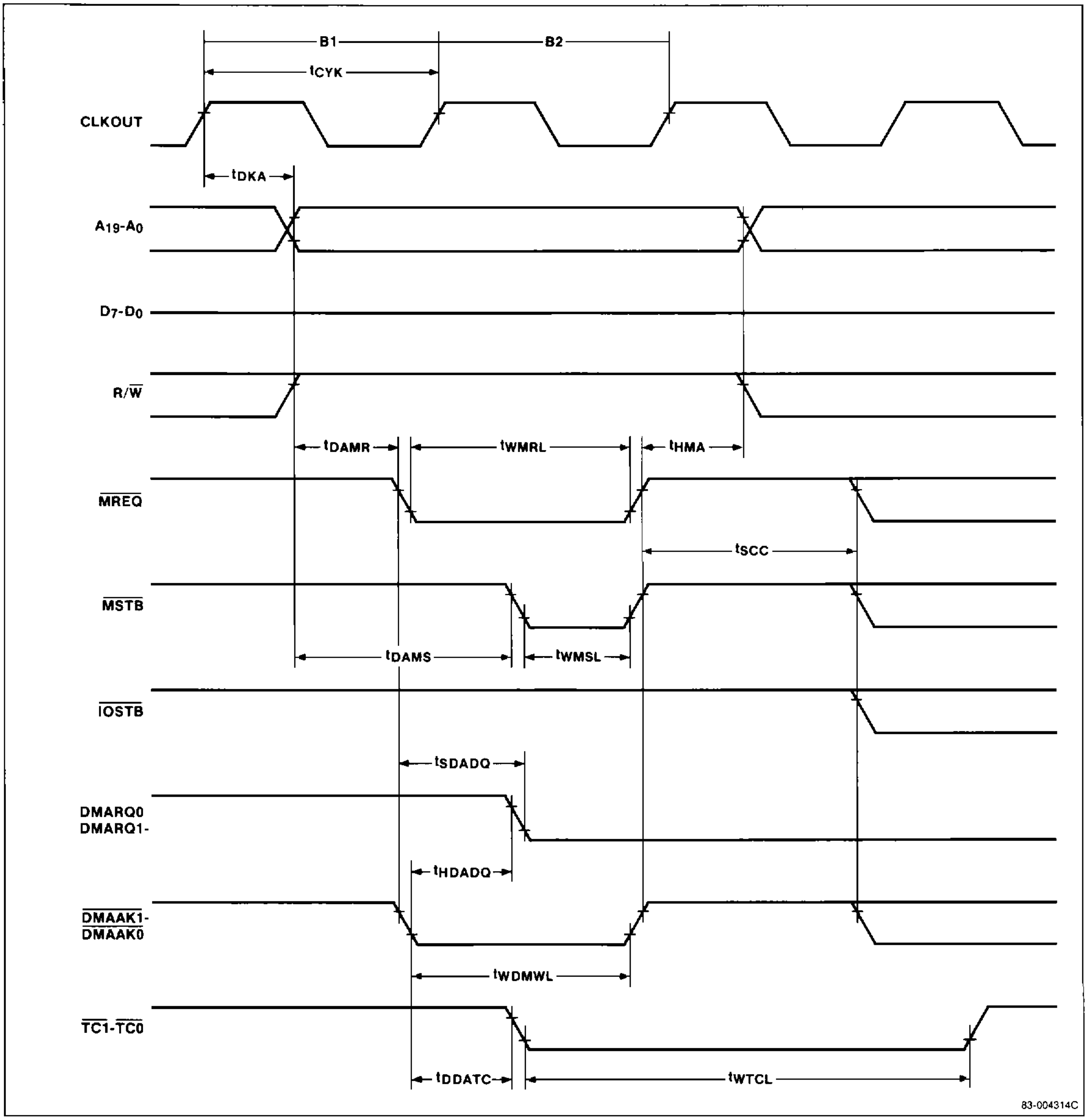
I/O Write


B1
B2
tCYK
CLKOUT
tDKA
A19-A0
tDADW
tHISA
D7-D0
tSDIS
tHISDW
R/W̄
M̄R̄Ē
M̄S̄T̄B̄
tDAIS
tWISL
tSCC
ĪŌS̄T̄B̄
R̄ĒF̄R̄Q̄
D̄M̄ĀĀK̄1̄-
D̄M̄ĀĀK̄0̄
83-004312C

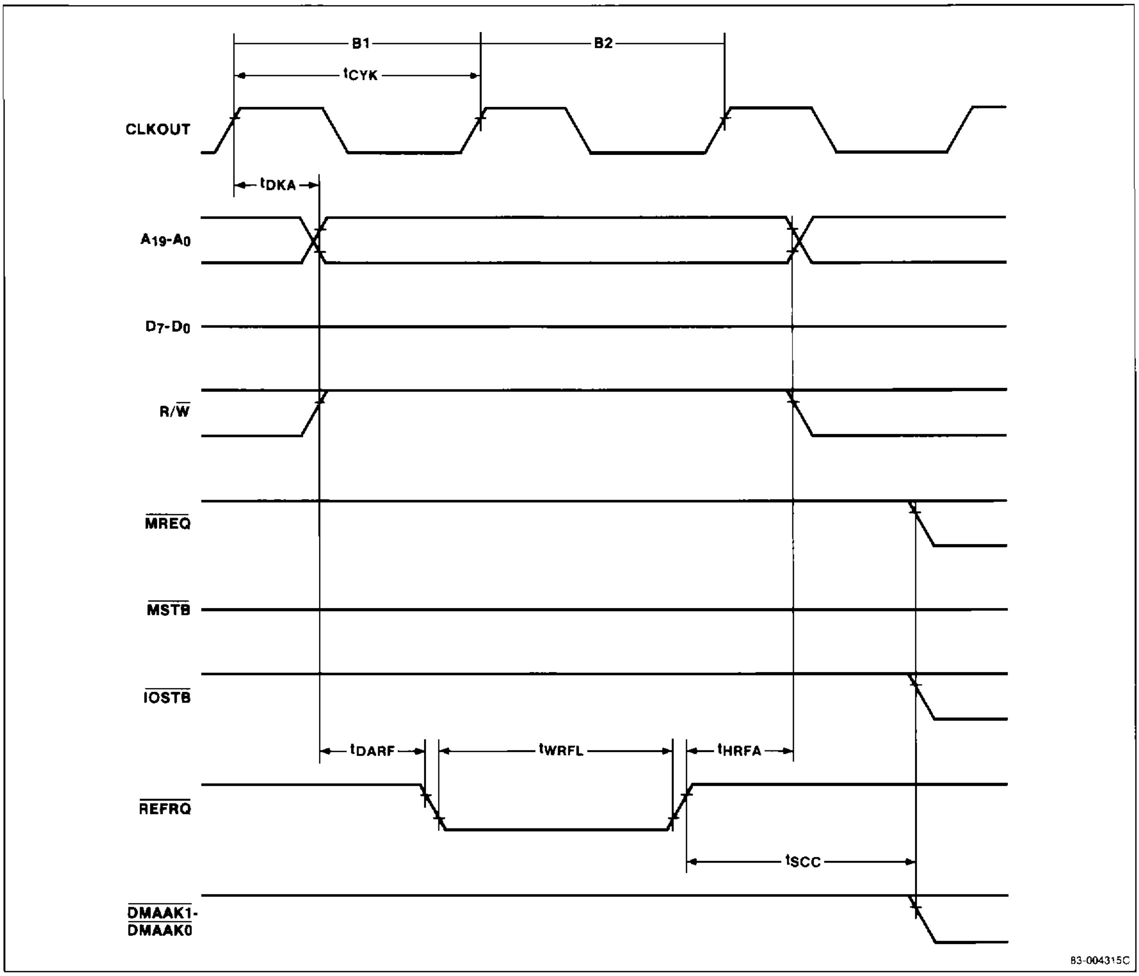
DMA, I/O to Memory



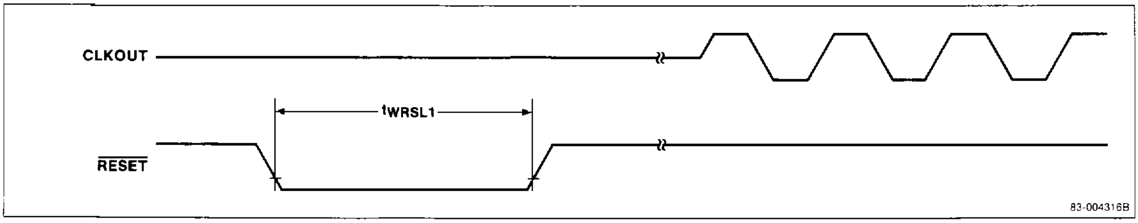
DMA, Memory to I/O



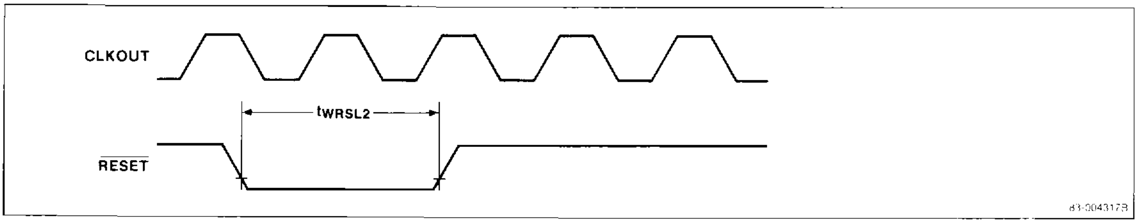
Refresh



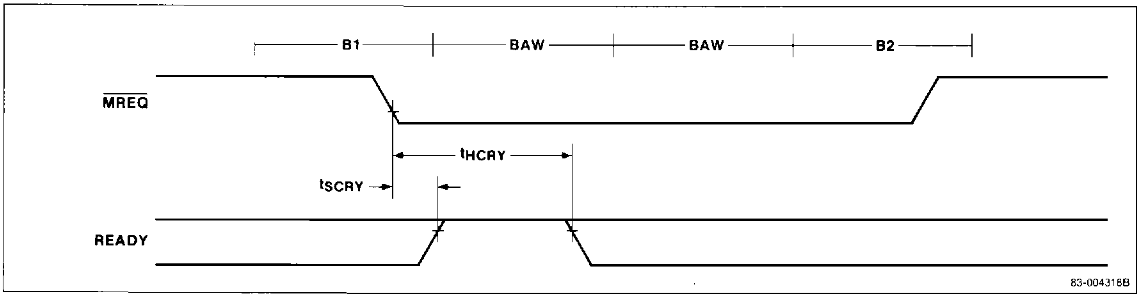
RESET 1



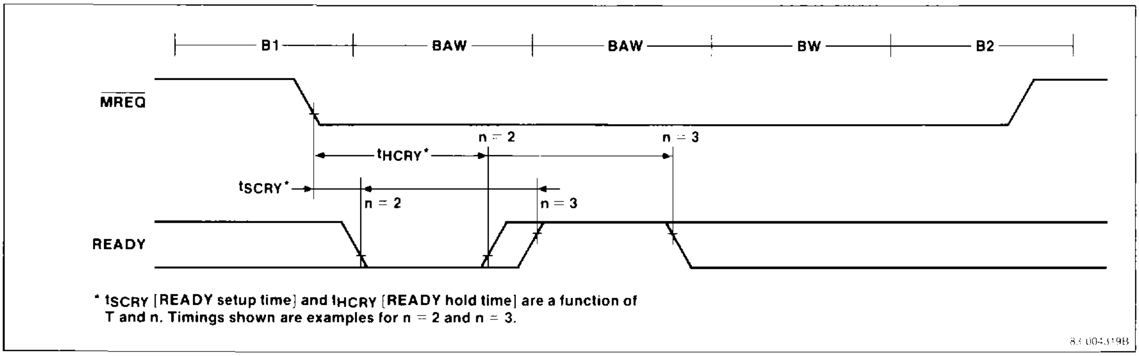
RESET 2



READY 1

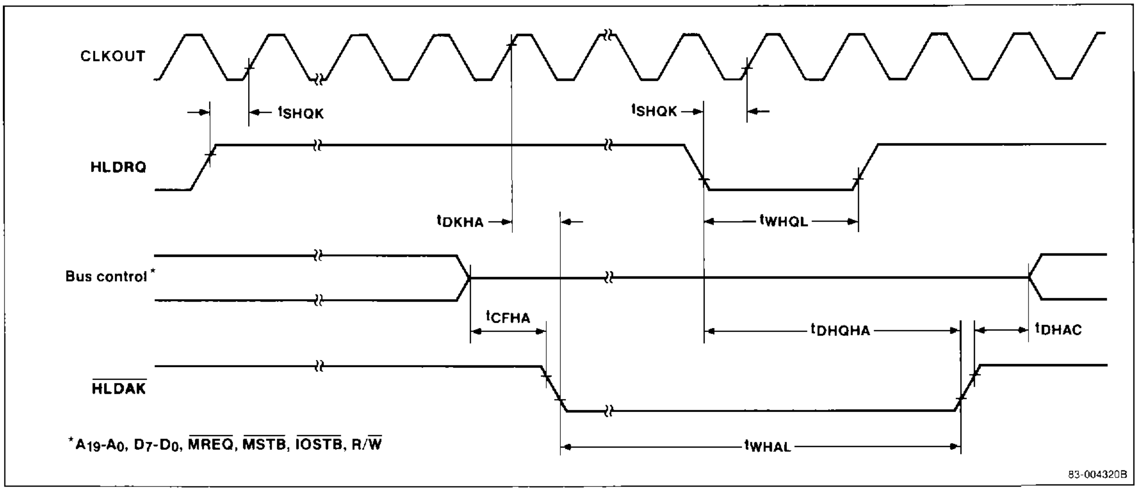


READY 2



* t_{SCRY} [READY setup time] and t_{HCRY} [READY hold time] are a function of T and n. Timings shown are examples for n = 2 and n = 3.

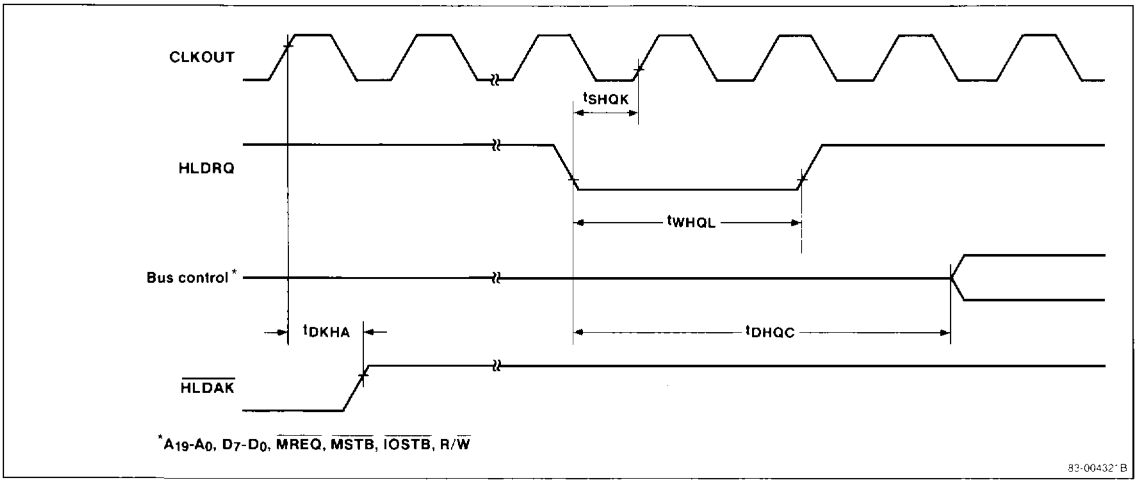
HLDRQ/$\overline{\text{HLDAK}}$ 1



*A_{19}-A_0, D_7-D_0, $\overline{\text{MREQ}}$, $\overline{\text{MSTB}}$, $\overline{\text{IOSTB}}$, R/$\overline{\text{W}}$

83-004320B

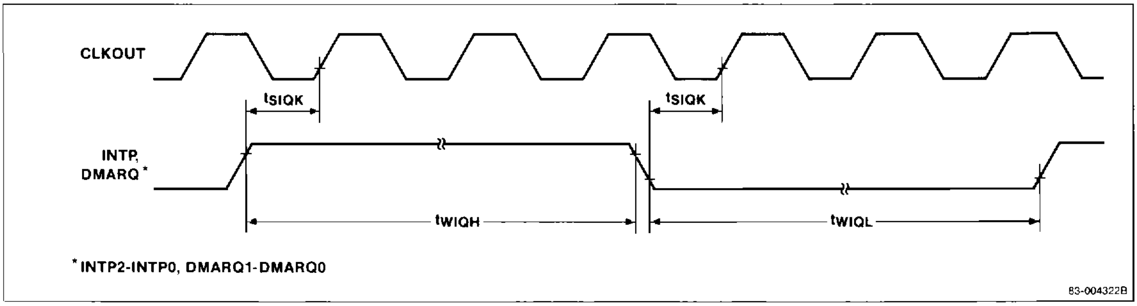
HLDRQ/$\overline{\text{HLDAK}}$ 2



*A_{19}-A_0, D_7-D_0, $\overline{\text{MREQ}}$, $\overline{\text{MSTB}}$, $\overline{\text{IOSTB}}$, R/$\overline{\text{W}}$

83-004321B

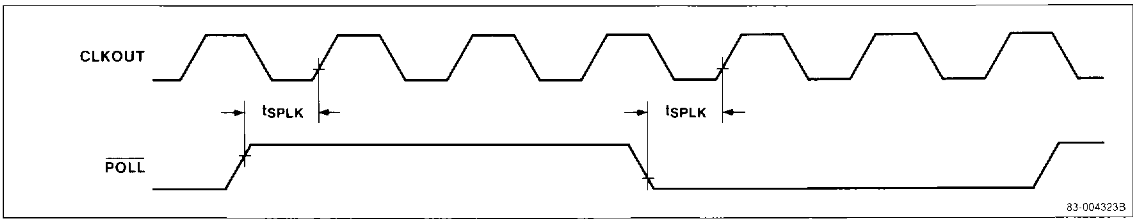
INTP, DMARQ Input



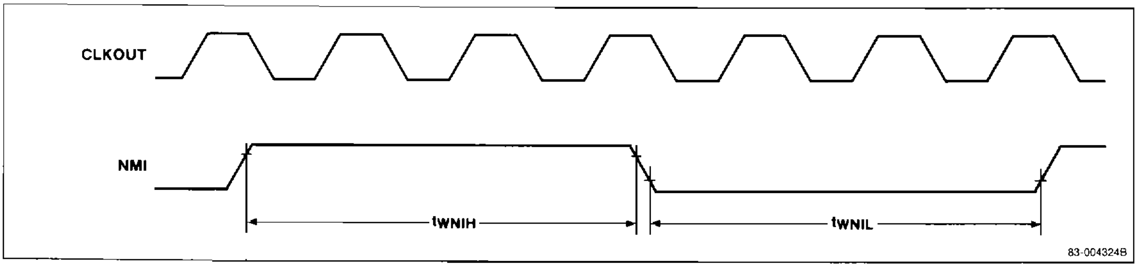
* INTP2-INTP0, DMARQ1-DMARQ0

83-004322B

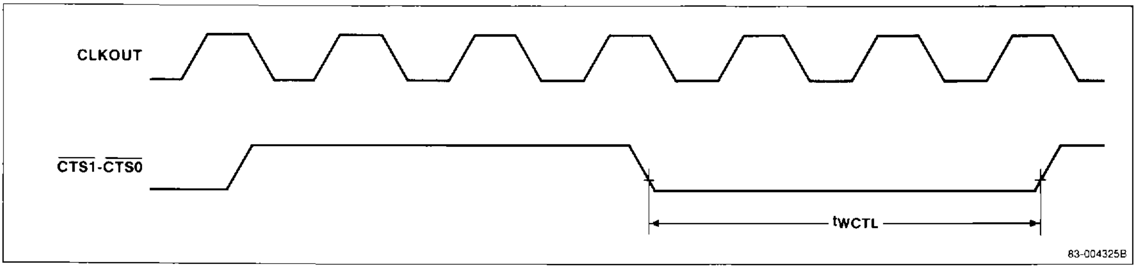
$\overline{\text{POLL}}$ Input



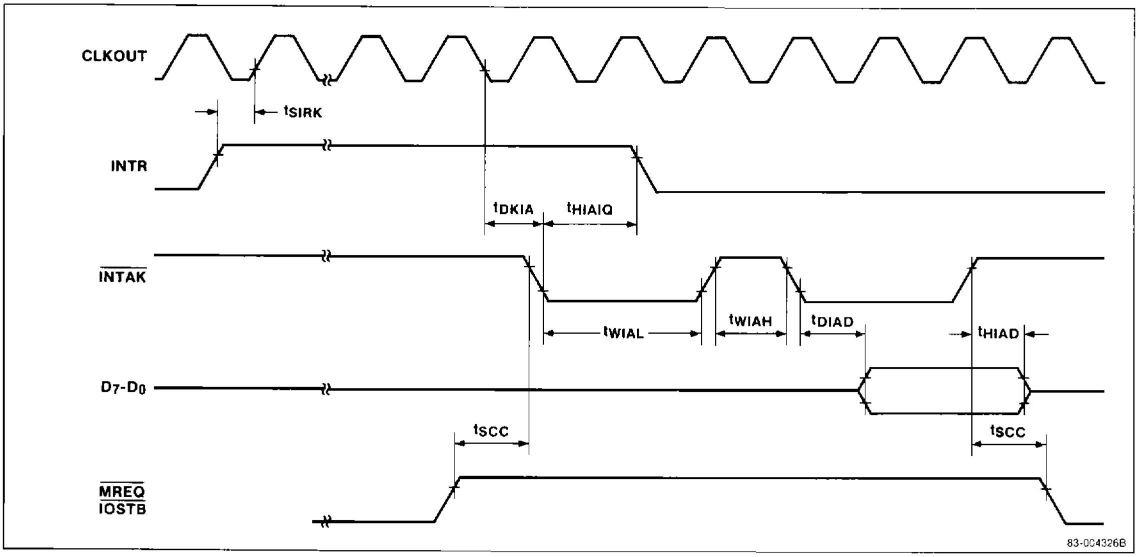
NMI Input



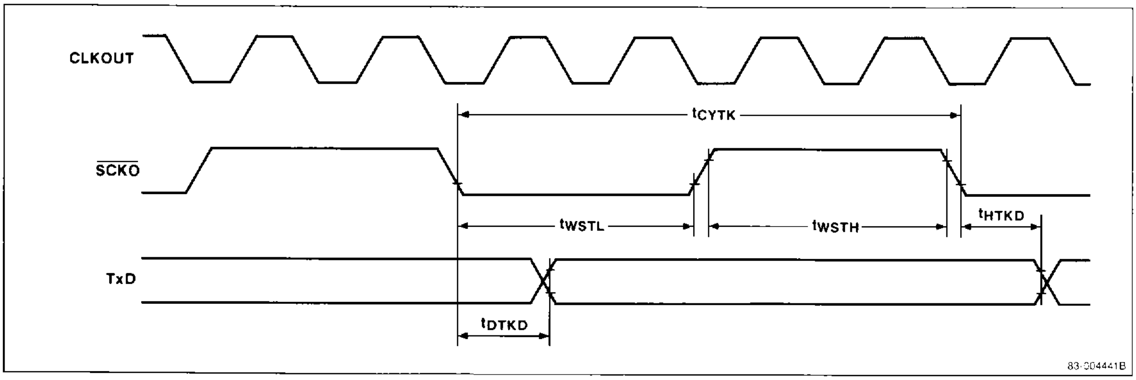
$\overline{\text{CTS}}$ Input



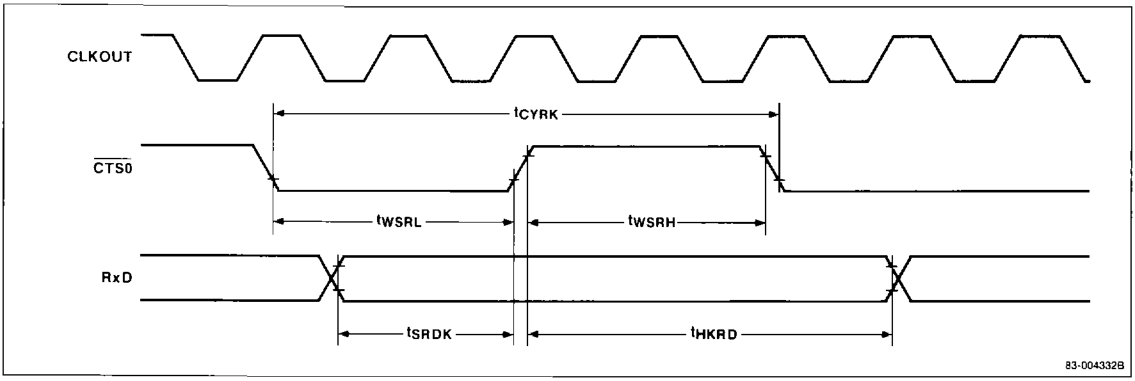
INTR/$\overline{\text{INTAK}}$



Serial Transmit



Serial Receive



ARCHITECTURAL DESCRIPTION

The µPD79011 is an upgraded version of µPD70322 (V25), NEC's original single-chip microcomputer. It has a real-time operating system built into internal ROM.

The µPD79011 is the same as the V25 in both hardware and software specifications except for the built-in ROM contents. For more information on the V25, refer to the µPD70320/70322 V25 Data Sheet

Memory Map

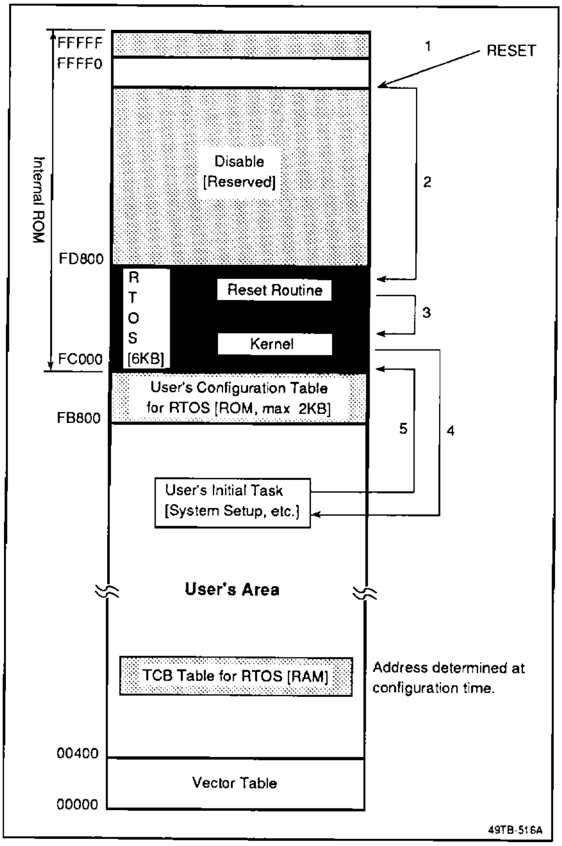
The µPD79011 can access a maximum of 1M bytes of memory via the 20-bit address bus. A 16K-byte segment of memory (FC000H to FFFFFH) is allocated to the on-chip ROM. The µPD79011 operating system is stored in this ROM area.

An external memory area of 2K bytes (FB800H to FBFFFH) contains a configuration table. When reset, the µPD79011 starts program execution at address FFFF0H, and performs the necessary initialization according to the information in this table. Then, program control is passed to each user-defined task.

A 1K-byte area (00000H to 003FFH) contains the vector tables. Thus, the total area for user tasks is from 00400H to FB7FFH.

Figure 1 is the µPD79011 memory map.

Figure 1. Memory Map



Reset Operation

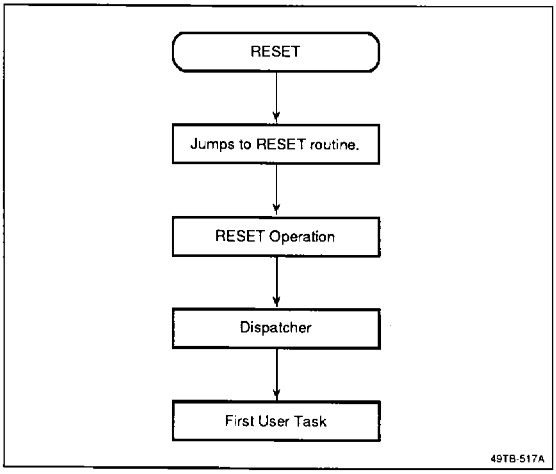
When reset, the μPD79011 begins program execution at address FFFF0H and jumps to the reset routine, which performs the following processing.

- Initializes special registers
- Initializes the interrupt vector table
- Generates the system table
- Specifies both semaphore and mailbox areas
- Generates and starts tasks

After completing the required reset processing, the μPD79011 jumps to the operating system dispatch routine, and then passes the program control to each user-defined task.

Figure 2 is a flowchart of system operation at reset time.

Figure 2. Reset Operation Flowchart



Interrupt Vectors

Up to 256 interrupt vectors (4 bytes/vector) can be stored in the vector table area. See table 1.

Table 1. Vector Table Area Assignments

Vector Number	Start Address	Use
0 to 31	00000H	Reserved for hardware as on μPD70322 (V25)
32 to 47	00080H	Available for use
48	000C0H	Operating system data table
49 to 55	000C4H	Available for use
56 to 63	000E0H	External μPD71059 (Master. Available for use)
64 to 71	00100H	External μPD71059 (Slave 0. Available for use)
72 to 79	00120H	External μPD71059 (Slave 1. Available for use)
80 to 87	00140H	External μPD71059 (Slave 2. Available for use)
88 to 95	00160H	External μPD71059 (Slave 3. Available for use)
96 to 103	00180H	External μPD71059 (Slave 4. Available for use)
104 to 111	001A0H	External μPD71059 (Slave 5. Available for use)
112 to 119	001C0H	External μPD71059 (Slave 6. Available for use)
120 to 127	001E0H	External μPD71059 (Slave 7. Available for use)
128 to 255	00200H	Available for use

Note: Vectors 56 to 127 are assigned to the master and slave interrupt controllers when added to the μPD79011. Otherwise, the area is free to be used.

Configuration Table

The configuration table resides in memory from FB800H to FBFFFH. The reset routine obtains initialization information from the configuration table. Any items not initialized by the reset routine must be initialized by the user initial task.

Table 2 is an example of a configuration table. It shows the assembler sources (described by RA70116). The input values in the table are only examples.

Table 2. Configuration Table, Filing Example

CONF_TBL	Data Type	Example Value	Notes
PTR0	DW	INTERNAL_ RAM_ BASE	1
PTR1	DW	TASK_CNT	
PTR2	DW	SMA_CNT	
PTR3	DW	MBOX_CNT	
INTERNAL_ RAM_BASE	DB	FFH	2
PRC_INFO	DB	46H	
LOW_DS	DW	1000H	3
HIGH_DS	DW	2000H	
BLK_SIZE	DW	2FC0H	

Table 2. Configuration Table, Filing Example (cont)

CONF_TBL	Data Type	Example Value	Notes
PORT0	DW	1000H	4
PORT1	DW	2000H	
PORT2	DW	0FFFFH	
PORT3	DW	0FFFFH	
PORT4	DW	0FFFFH	
PORT5	DW	0FFFFH	
PORT6	DW	0FFFFH	
PORT7	DW	0FFFFH	
PORT8	DW	0FFFFH	
TASK_CNT	DB	2BH	5
MIN_TASK_NO	DB	0	
INIT_TASK	DB	0	
IDLE_SP	DW	1000H	6
IDLE_SS	DW	0F000H	
INIT_PC0	DW	0000H	7 User Task 0
INIT_PS0	DW	4000H	
INIT_ SP0	DW	2000H	
INIT_SS0	DW	0F000H	
INIT_DS0	DW	2000H	
INIT_PC1	DW	1000H	7 User Task 1
INIT_PS1	DW	4000H	
INIT_SP1	DW	3000H	
INIT_SS1	DW	0F000H	
INIT_DS1	DW	2000H	
SMA_CNT	DW	2	8
INIT_RSC0	DW	1	
	DW	10H	
MBOX_CNT	DW	10H	
RESERVE	DW	00H	9
CONF_TBL	ENDS		
	END		

Notes:

(1) Pointers

(2) System information

(3) RAM information

(4) Interrupt controller information

(5) User task information

(6) Idle task stack information

(7) User task register information

(8) Semaphore/mailbox information

(9) Reserved area

Pointers

A pointer is an offset value obtained using a segment value of 0FB08H. The following pointers are provided. The organization of the configuration table changes according to user system status.

Pointer	Size	Points to
PTR0	1 word	INTERNAL_RAM_BASE
PTR1	1 word	TASK_CNT
PTR2	1 word	SMA_CNT
PTR3	1 word	MBOX_CNT

System Information

INTERNAL_RAM_BASE: This byte is required to set the internal RAM base segment of the µPD79011. It is specified in the internal data area base register (IDB address 0FFFFFH).

If XXH is specified as the IDB value (where X is a hexadecimal number), the internal RAM base segment is assumed to be XX00H. Therefore, each register bank and the special function register (including IDB) are assigned to the 512-byte area starting at address XXE00H.

PRC_INFO. This byte sets the processor control register (PRC), which has the following functions.

- System clock divider of oscillator frequency
- Interval of time base interrupt
- Enable/disable of internal RAM

RAM Information

The configuration table provides the following RAM information.

LOW_DS/HIGH_DS: These two words specify the user free RAM area. Because it is a continuous memory area, both the upper and lower limit segment addresses (offset 0) must be used to specify this area.

The initialize routine sets the system table and each control block in this RAM area. Any remaining control blocks are queued in the system table as memory blocks (the section System Calls provides more information). The user free RAM area must be large enough to hold all control blocks.

BLK_SIZE: This word of information specifies the memory block size in units of 16 bytes. If BLK_SIZE of zero is specified, no memory blocks are generated.

Interrupt Controller

PORT0 through PORT8 (9 words) provide the information required when one or more external interrupt controllers (μPD71059) are connected to μPD79011.

PORT0 specifies the port address for the master interrupt controller. PORT1 through PORT8 specify the port addresses corresponding to the slave interrupt controllers (0 to 7).

If fewer than nine interrupt controllers are used, 0FFFFH indicates the addresses of the unused interrupt controllers.

User Task Information

TASK_CNT: This byte of information specifies the total number of user tasks (except for idle tasks). Up to 63 tasks can be specified.

MIN_TASK_NO: User task numbers are assigned sequentially starting from this number, the mimimum task number. Only tasks with numbers greater than the minimum task number are generated.

INIT_TASK: This byte of information indicates the number of the first task that the operating system must execute when the system is initialized. All other tasks are dormant when the system is initialized.

Idle Task Stack

IDLE_SP: This word of information specifies the idle task stack pointer (SP) value.

IDLE_SS: This word of information specifies the idle task stack segment (SS) value. When a stack is set, any value can be used for the address. The stack area must be a minimum of 32 bytes.

User Task Register Initialization

INIT_PC0: This word of information specifies the initial value of the program counter (PC) in relation to the minimum user task number specified for MIN_TASK_NO.

INIT_PS0: This word of information specifies the initial value of the program segment (PS) for the first user task.

INIT_SP0: This word of information specifies the initial value of the stack pointer (SP) for the first user task.

INIT_SS0: This word of information specifies the initial value of the stack segment (SS) for the first user task.

INIT_DS0: This word of information specifies the initial value of the data segment (DS) for the first user task.

The above set of register initial values is repeated for each user task.

Semaphore/Mailbox

SMA_CNT: This word of information specifies up to 256 semaphores to be used.

INIT_RSC0: This word of information supplies the initial number of resources for semaphore 0. After specification of semaphore 0, the initial number of resources of all other semaphores should be specified sequentially.

MBOX_CNT: This word of information specifies the number of mailboxes (up to 256) to be used.

Reserved Area

RESERVE is a one-word area. You must specify a value of 0 for RESERVE.

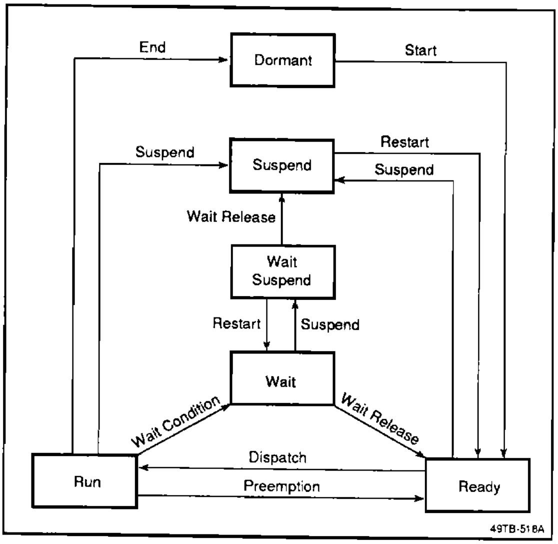
Task Status and Status Change

Table 3 shows the various task statuses. Figure 3 shows all task status changes.

Table 3. Task Status

Status	Meaning
RUN	One task, given priority to use the CPU, is currently being executed.
READY	A task is ready to execute. A READY task has a priority lower than the task currently under execution and is hence blocked by the priority handler.
WAIT	A task is waiting for an event to occur so it can go into the READY status. This status is caused by the following conditions: WAIT - a system call caused the status change and the task is either waiting for a resource with a semaphore, waiting for a message (through mail box or direct connection), or waiting for an interrupt.
SUSPEND	The system call SUS_TSK suspended execution forcibly when the task was in the RUN status. The task must wait for a system call to restart execution
WAIT SUSPEND	A task was forcibly moved into the WAIT status and has a double wait status. If the system call RSM_TSK is issued to a task in the WAIT SUSPEND status, the task is released from the SUSPEND status and goes into the WAIT status. If released from the WAIT status, the task goes into SUSPEND status.
DORMANT	When the system is initialized, only one task goes into the READY state. All other tasks go into the DORMANT status. If the system call EXT_TSK is issued to a task that is executing, this task becomes DORMANT.

Figure 3. Task Status Change



Idle Task

The μPD79011 operates an idle task when no user-set task needs to be executed. The user-specified maximum number plus 1 is used as the idle task number.

If the idle task begins execution, it executes the HALT instruction in the Interrupt Enable status, then waits for an interrupt to be issued.

FUNCTIONAL DESCRIPTION

The μPD79011 can handle up to 64 tasks numbered and assigned priorities from 0 to 63. Task numbers and priority levels correspond to each other. (For example, task 3 has a task priority of 3.) Level 0 is the highest priority; level 63 is the lowest priority.

Tasks are scheduled according to their priority levels. The μPD79011 selects and executes the READY task with the highest priority (RUN status).

Like the V25, the μPD79011 has 8 register banks (numbered 0 to 7). Task switching can be done at a high speed using these register banks. The operating system occupies bank 7. The remaining banks (0 to 6) are all assigned to tasks.

Of the 7 register banks, tasks numbered 0 to 5 are assigned to banks 0 to 5 and are resident in the banks. Because the bank-resident tasks do not require any processing to save/return the task status, task switching can be handled quickly.

The remaining tasks, numbered 6 to 63, are all assigned to bank 6. These tasks, unlike tasks resident in banks, require processing time to swap the task state to register bank 6.

Table 4 shows the register banks and tasks.

Table 4. Register Banks and Corresponding Tasks

Register Bank	Task	*Priority	Type
0	0	0	Resident
1	1	1	
2	2	2	
3	3	3	
4	4	4	
5	5	5	
6	6 to 63	6 to 63	Non-resident
7	—	—	Occupied by μPD79011 OS

No priority can be set for DMA or macroservice transfer.

Task Management

The task management function is used to terminate, start, suspend, restart tasks, and set the restart address.

If system call STA_TSK is issued to a task, the task exits the DORMANT status and goes into the READY status. If system call SUS_TSK is issued to a task, the specified task goes into the SUSPEND status. The task exits the SUSPEND status when system call RSM_TASK is issued, and its status becomes READY.

The restart address is set by issuing system call SET_ADR. The SET_ADR is always used with system call RES_INT to end the interrupt handler. (Refer to the section Interrupt Management for additional information.)

Synchronization/Communication Management

Tasks are synchronized by queuing or mutual exclusion. If tasks are queued, they are processed and executed one at a time.

Mutual exclusion is used in task processing to prohibit simultaneous access by more than one task to a shared resource (such as memory, an I/O device, etc.).

The μPD79011 uses semaphores for task synchronization and mailboxes for intertask communication.

Semaphores

The μPD79011 implements semaphores to manage resources and for queuing or mutually excluding tasks.

Both the P instruction (Obtain Resource) and the V instruction (Release Resource) manage only one resource at a time.

The P instruction can use the following system calls.

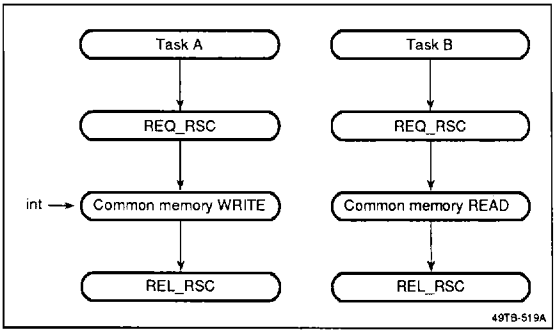
REQ_RSC: If the request to obtain resource is not accepted, the task goes into the WAIT status.

POL_RSC: If the request to obtain resource is not accepted, the system is notified that the request has been rejected.

The V instruction (system call REL_RSC) releases the occupied resource.

Figure 4 shows how to use system calls to avoid simultaneous read and write to shared memory. In figure 4, both tasks A and B share the same resource (memory). An interrupt is issued when task A is executed and control is passed between the two tasks. If the REQ_RSC request is not accepted because the resource is used by another task (task A), task B goes into the WAIT status.

Figure 4. Mutual Exclusion



Intertask Communication

Tasks communicate with each other in one of two ways, directly and nondirectly . Each task has a mailbox with a task queue for receiving messages and a message queue for sending messages. No mailbox is required for direct communication. Messages can be sent directly from one task to another.

If a task cannot receive a message for any reason (either directly or in a mailbox), one of the following system calls is issued.

RCV_MSG: Issued if a message was sent to a mailbox; the task goes into WAIT status.

RCV_DIR: Issued if a message was sent directly; the task goes into the WAIT status.

POL_MSG: Issued if a message was sent to a mailbox; notifies the system that no message can be received.

POL_DIR: Issued if a message was sent directly; notifies the system that no message can be received.

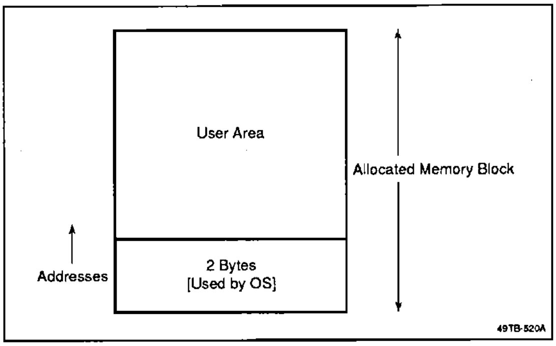
Memory Management

You can issue system calls to secure and return memory blocks dynamically on the μPD79011. The memory block size is specified at configuration time.

Task status remains the same and an the error code is returned when the GET_MEM system call is unable to secure a block of memory.

If a memory block is specified as the message area, the system uses the first two bytes of memory (figure 5). Consequently, available memory (specified in the configuration table) is reduced by 2 bytes.

Figure 5. Memory Block



Interrupt Management

For internally and externally generated interrupt requests, RTOS has the following functions to support the associated interrupt service routines.

- Interrupt handler assignment
- Interrupt handler return
- Interrupt enable/disable
- Interrupt wait status

When DEF_INT is issued, a correspondence is set between the request level (or vector type) of an external μPD71059 interrupt controller and the starting address of its service routine.

The ENA_INT and DIS_INT calls allow interrupts to be enabled or disabled.

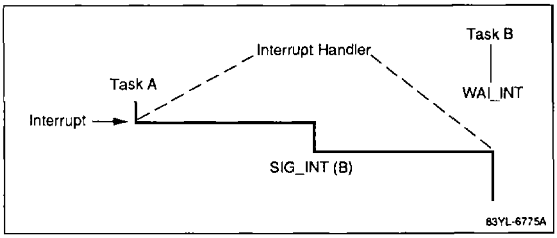
The SIG_INT and RES_INT system calls terminate the interrupt handler and pass control to the top-queued task (queued by the WAI_INT call).

Figure 6 shows how SIG_INT passes control to a task. The following events occur in the figure.

- Due to WAI_INT, task B waits for an interrupt.
- An interrupt is issued while task A is running.
- SIG_INT is issued to task B at the end of interrupt handling.

If the priority of task A is higher than that of task B, control is passed to task A when SIG_INT is executed (the interrupted task). If the priority of task B is higher, control is passed to task B.

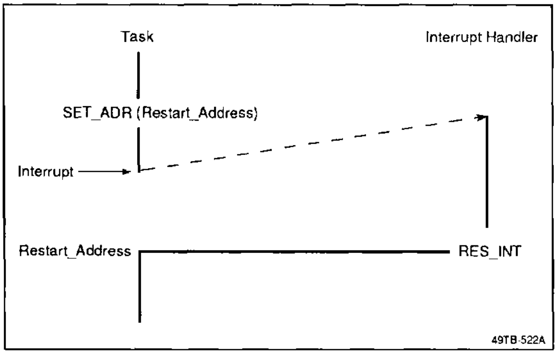
Figure 6. SIG_INT Examples



The RES_INT system call is always used with the SET_ADR system call to set the restart address. If SET_ADR has already been issued in an interrupted task handler that issues RES_INT, RES_INT passes control to the restart address specified by SET_ADR, not to the address where the interrupt was issued.

Figure 7 shows how to use the RES_INT system call to pass control to a task.

Figure 7. RES_INT Example



SYSTEM CALLS

The μPD79011 provides the following types of system calls.

- Task management
- Synchronization/communication management
- Memory management
- Time management
- Interrupt management

The system calls all have ID numbers assigned to them. Descriptions of system calls include their syntax and any error codes that may be returned to the task when the call is issued.

You can use the C language or assembly language to develop programs for the μPD79011. If using the C language, an error code is returned as a function value of the system call. If using assembly language, an error code is returned to the AW register of the μPD79011 as a return parameter.

C Language Interface

The μPD79011 supports the C language, a high-level language for developing large or small programs. To issue system calls in the C language, an assembler routine is required as an interface between the μPD79011 operating system and the C language. Refer to the Assembly Language Interface section for details on writing the interface.

Following is the syntax use for issuing calls in the C language.

```
err = <name> ([<parameter>]);
```

Argument	Description
err	Function value returned by RTOS
<name>	7-letter System Call Name
<parameter>	Input parameter

Assembly Language Interface

The μPD79011 has a C language-oriented architecture. Therefore, when issuing system calls using assembly language, the μPD79011 always sends and receives parameters via a stack. (If the system call requires no parameters, no stacking is needed.)

The syntax for issuing system calls using assembler and loading the stack for operation are shown below. If the parameter is a pointer, the offset value is stacked in the lower address area of the stack, and the segment value is stacked in the upper address area.

err = <name> (arg1, arg2, arg3);

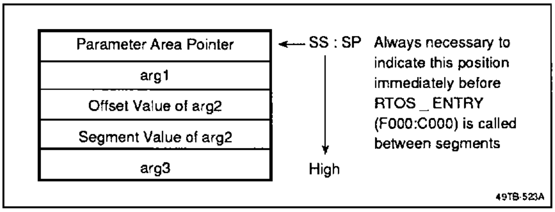
Argument	Description
<name>	7-letter System Call Name
arg1	unsigned int
arg2	int
arg3	unsigned int

The system call is issued in the following sequence.

- Parameter 3 (arg3) is stacked.
- Parameter 2 (arg2) is stacked.
- Parameter 1 (arg1) is stacked.
- A pointer to the parameter area is stacked.
- The system call number is set in the AW register.
- RTOS_ENTRY (FC000H) is called between segments.

An intersegment system call is needed even when the RTOS_ENTRY address is within the same segment.

Figure 8. Stacking Conditions



Always necessary to indicate this position immediately before RTOS _ ENTRY (F000:C000) is called between segments

The procedures for issuing the SIG_INT and RES_INT system calls are different. They are explained later in this data sheet.

TASK MANAGEMENT SYSTEM CALLS

The following system calls are used for task management.

System Call	Description
STA_TSK	Starts task processing
EXT_TSK	Terminates task processing
SUS_TSK	Suspends task processing
RSM_TSK	Restarts task processing
SET_ADR	Sets restart address

Start Task (STA_TSK)

System Call 0. STA_TSK starts task processing during which the task goes into the READY status from the DORMANT status. It has the following syntax.

int STA_TSK (task_no)

(1) Parameter.

I/O	Name	Description
In	int task_no;	Task number (0 to 62)

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_DMT	1	Task is not DORMANT

(3) C format.

```
short task_no;
ercode = STA_TSK(task_no);
```

STA_TSK can only be issued to a task that is in the DORMANT status.

The started task processing is done in one of the following ways.

- Executed for the first time.
- After it is terminated once, it is restarted.

If a task is executed for the first time, the task processing starts from the initial address. Initial values from the configuration table are also used for the stack pointer, stack segment, and data segment values. Other register values are not defined.

If the task processing is ended once and then restarted, the task also resumes at the initial address. In this case, the stack pointer, stack segment, data segment values, and other register values assume the values they had just before the EXT_TSK system call was issued.

Exit Task (EXT_TSK)

System Call 1. EXT_TSK terminates task processing and moves the task into the DORMANT status from the RUN status. It has the following syntax.

int EXT_TSK ( )

(1) Return value.

Error Code	Number	Description
E_OK	0	Normal end

(2) C format.

```
ercode = EXT_TSK( );
```

If STA_TSK restarts a task in the DORMANT status (due to EXT_TSK), the start address returns to the initial value. Other register values retain the values they had when EXT_TSK was issued. Thus, the stack pointer, stack segment, data segment values may not match the values assumed at configuration time.

Suspend Task (SUS_TSK)

System Call 2. SUS_TSK suspends a task and puts it into the SUSPEND status. It has the following syntax.

int SUS_TSK (task_no)

(1) Parameter.

I/O	Name	Description
In	int task_no;	Task number (0 to 62)

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_DMT	1	Task is DORMANT
E_SUS	2	Task is in SUSPEND status

(3) C format.

```
short task_no;
ercode = SUS_TSK(task_no);
```

SUS_TSK cannot be issued to tasks that are in the DORMANT status or in the SUSPEND status.

If SUS_TSK is issued to a task in the WAIT status, the task goes into the WAIT SUSPEND status.

Resume Task (RSM_TSK)

System Call 3. RSM_TSK restarts a task that is in the SUSPEND status. It has the following syntax.

int RSM_TSK (task_no)

(1) Parameter.

I/O	Name	Description
In	int task_no;	Task number (0 to 62)

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_DMT	1	Task is DORMANT
E_SUS	2	Task is not in SUSPEND status

(3) C format.

```
short task_no;
ercode = RSM_TSK(task_no);
```

RSM_TSK cannot be issued to tasks that are in the DORMANT status or in the SUSPEND status.

If it is issued to a task in the WAIT SUSPEND status, the task is released from the SUSPEND status and goes into the WAIT status.

Set Restart Address (SET_ADR)

System Call 4. SET_ADR sets the restart address of a task. It has the following syntax.

int SET_ADR (restart_adr)

(1) Parameter.

I/O	Name	Description
In	int (restart_adr);	Task restart address

(2) Return value

Error Code	Number	Description
E_OK	0	Normal end

(3) C format.

```
ercode = STA_TSK(restart_adr);
pointer restart_adr;
```

SET_ADR is always used in conjunction with the RES_INT system call. If RES_INT is issued on return from the interrupt handler, control is passed to the restart address set previously by SET_ADR.

SET_ADR can be issued more than once, but the system only validates the last restart address that was issued. Setting the restart address to 0 clears current restart address.

SYNCHRONIZATION/COMMUNICATION MANAGEMENT SYSTEM CALLS

The following system calls are used for synchronization/communication management:

System Call	Description
REQ_RSC	Requests resource from a semaphore
POL_RSC	Requests resource from a semaphore (no wait)
REL_RSC	Releases resource for a semaphore
RCV_MSG	Receives messages from a mailbox
POL_MSG	Receives messages from a mailbox (no wait)
SND_MSG	Sends messages to a mailbox
RCV_DIR	Receives messages sent to this task
POL_DIR	Receives messages sent to this task (no wait)
SND_DIR	Sends messages to the specified task

Request Resource (REQ_RSC)

System Call 5. REQ_RSC requests a resource from the specified semaphore. It has the following syntax.

int REQ_RSC (semaphore_no)

(1) Parameter.

I/O	Name	Description
In	int semaphore_no;	Semaphore number (0 to specified number)

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end

(3) C format.

```
ercode = REQ_RSC(semaphore_no);
short semaphore_no;
```

If REQ_RSC is issued when the resource count is 0, the task goes into the WAIT status. If the resource count is more than 1, the resource count is decremented by one.

Each semaphore has a task queue. But, if REQ_RSC causes a task to go into the WAIT status, the task is placed in the last position in the queue regardless of its priority.

Poll Resource (POL_RSC)

System Call 6. POL_RSC is used to request resources from the specified semaphore. It has the following syntax.

```
int POL_RSC (semaphore_no)
```

(1) Parameter.

I/O	Name	Description
In	int semaphore_no;	Semaphore number (0 to specified number)

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_RSC	6	Resource count is 0

(3) C format.

```
ercode = POL_RSC(semaphore_no);
short semaphore_no;
```

POL_RSC is used to determine whether any resources are left in the specified semaphore. Unlike the REQ_RSC, POL_RSC never causes a task to go into the WAIT status. Instead, it returns the E_RSC error code when the resource count is 0. If the resource count is more than 1, the count is decremented by 1.

Release Resource (REL_RSC)

System Call 7. REL_RSC releases resource for the specified semaphore. It has the following syntax.

```
int REL_RSC (semaphore_no)
```

(1) Parameter.

I/O	Name	Description
In	int semaphore_no;	Semaphore number (0 to specified number)

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end

(3) C format.

```
ercode = REL_RSC(semaphore_no);
short semaphore_no;
```

When REL_RSC is issued, the semaphore resource count is increased by 1. If WAIT tasks exist, the earliest-wait task is selected and released from the WAIT status.

The initial value of the semaphore resource count is set when the system is started. No error occurs even when the resource count exceeds the initial value as a result of issuing REL_RSC. If the resource count exceeds 65,535, the resource count is cleared to 0 automatically and no error is generated.

Receive Message (RCV_MSG)

System Call 8. RCV_MSG receives messages from mailboxes. It has the following syntax.

```
int RCV_MSG (mailbox_no)
```

(1) Parameter.

I/O	Name	Description
In	int mailbox_no;	Mailbox number (0 to specified number)

(2) C format.

```
seg = RCV_MSG(mailbox_no);
short mailbox_no;
```

If RCV_MSG is issued when messages are present in mailboxes, the earliest message is selected and the segment value of the message area is returned as the function value.

If there is no message, the task goes into the WAIT status and it is placed in the last position in the mailbox queue.

Poll Message (POL_MSG)

System Call 9. POL_MSG receives messages from mailboxes. It has the following syntax.

```
int POL_MSG (mailbox_no)
```

(1) Parameter.

I/O	Name	Description
In	int mailbox_no;	Mailbox number (0 to specified number)

(2) Return value. If there are any messages in the specified mailbox, the message area segment value is returned. If there is no message, the following error code is returned.

Error Code	Number	Description
E_MSG	7	No message found

(3) C format.

```
seg = POL_MSG(mailbox_no);
short mailbox_no;
```

If POL_MSG is issued when messages are present in mailboxes, the earliest message is selected and the segment value of the message area is returned as the function value.

If no message is found, unlike the RCV_MSG system call, the task never goes into the WAIT status. Instead, the E_MSG error code is returned.

Send Message (SND_MSG)

System Call 10. SND_MSG sends messages to mailboxes. It has the following syntax.

```
int SND_MSG (mailbox_no, msg_seg)
```

(1) Parameter.

I/O	Name	Description
In	int mailbox_no;	Mailbox number (0 to specified number)
In	int msg_seg;	Send message area segment

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end

(3) C format.

```
ercode = SND_MSG(mailbox_no, msg_seg);
short mailbox_no;
short msg_seg;
```

If SND_MSG is issued when a task is waiting to be processed, the task is released from the WAIT status, and the send message area segment value is returned.

If no tasks are in the WAIT status, the message is queued in the mailbox. Like tasks, messages are queued using the first-in, first-out (FIFO) method.

Receive Direct Message (RCV_DIR)

System Call 11. RCV_DIR receives messages sent directly to a task. It has the following syntax.

```
int RCV_DIR ( )
```

(1) C format.

```
ercode = RCV_DIR( );
```

If RCV_DIR is issued when there is no message, the task goes into the WAIT status. If a message is present, the message area segment value is returned.

Poll Direct Message (POL_DIR)

System Call 12. POL_DIR receives messages sent by a task to itself. It has the following syntax.

```
int POL_DIR ( )
```

(1) Return value. If POL_DIR is issued when a message is present, the message area segment value is returned. If no message is present, the following error code is returned and the task does not enter WAIT status.

Error Code	Number	Description
E_MSG	7	No message is present

(2) C format.

```
ercode = POL_DIR(n);
```

Send Direct Message (SND_DIR)

System Call 13. SND_DIR specifies a task and sends a message to the specified task. It has the following syntax.

```
int SND_DIR (task_no, msg_seg)
```

(1) Parameter.

I/O	Name	Description
In	int task_no;	Task number (0 to 62)
In	int msg_seg;	Send message area segment

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end

(3) C format.

```
ercode = SND_DIR(task_no, msg_seg);
short task_no;
short msg_seg;
```

If SND_DIR is issued when the specified task is waiting for a message directly, the task is released from the WAIT status. The message area segment value is returned to the task.

If the specified task is not waiting for any message directly, the message is placed in the task message queue using the FIFO method.

MEMORY MANAGEMENT SYSTEM CALLS

The following system calls are used for memory management.

System	Call Description
GET_MEM	Gets a memory block
REL_MEM	Releases the memory block

Get Memory (GET_MEM)

System Call 14. GET_MEM allocates a memory block. It has the following syntax.

int GET_MEM ( )

(1) Return value. If a memory block is available when GET_MEM is issued, the memory block segment value is returned. If no memory block is present, the following error code is returned.

Error Code	Number	Description
E_BLK	3	No memory block found

(2) C format.

ercode = GET_MEM( );

GET_MEM can use the memory block as a message area for intertask communications. The memory block size is specified when the system is started, and the value is fixed.

If the error code is returned, the task never goes into the WAIT status.

Release Memory (REL_MEM)

System Call 15. REL_MEM releases the specified memory block. It has the following syntax.

int REL_MEM (mem_blk)

(1) Parameter.

I/O	Name	Description
In	int mem_blk;	Segment value of the released memory block

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end

(3) C format.

ercode = REL_MEM(mem_blk);
short mem_blk;

REL_MEM cannot release memory blocks containing messages in a mailbox or task queue. The memory block can only be released after a message is received.

TIME MANAGEMENT SYSTEM CALLS

The following system calls are used for time management.

System Call	Description
GET_TIM	Reads the system time
SET_TIM	Sets the system time

Get Time (GET_TIM)

System Call 16. GET_TIM reads the system time. It has the following syntax.

int GET TIM (time_ptr)

(1) Parameter.

I/O	Name	Description
In	struct t_time * time_ptr;	Pointer to location of system time

(2) Time structure.

```
struct t_time{
 int l_time;
 int m_time;
 int h_time;};
```

(3) Return value.

Error Code	Number	Description
E_OK	0	Normal end

(4) C format.

ercode = GET_TIM(time_ptr);
pointer time_ptr;

The system time is 3-word data. The lower order word is stored in the lowest order address; the intermediate data is in the intermediate address; and the upper order data is in the highest address.

The minimum resolution of the system time is determined by the value set in the time base counter in the μPD79011. However, since interrupts to the μPD79011 are inhibited during system call processing, choose the minimum resolution of the system time with system call overhead time in mind.

Set Time (SET_TIM)

System Call 17. SET_TIM sets the system time. It has the following syntax.

```
int SET_TIM (time_ptr)
```

(1) Parameter.

I/O	Name	Description
In	struct t_time * time_ptr;	Time pointer

(2) Time structure.

```
struct t_time{
 int l_time;
 int m_time;
 int h_time;};
```

(3) Return value.

Error Code	Number	Description
E_OK	0	Normal end

(4) C format.

```
pointer time_ptr;
ercode = SET_TIM(time_ptr);
```

The μPD79011 uses the on-chip timer base counter output as the system real-time clock source.. The on-chip timer therefore starts its counting operation when the system is started.The interval from the SET_TIM call to the next real-time clock interrupt is an error term associated with the initial call to SET_TIM, and all subsequent calls produce additional pseudorandom error times. The real-time clock interval is set at configuration time.

INTERRUPT MANAGEMENT SYSTEM CALLS

The following system calls are used for interrupt management:

System Call	Description
DEF_INT	Sets the start address of the interrupt handler
SIG_INT	Starts a task waiting for an interrupt and terminates the interrupt handler operation
WAI_INT	Waits for an interrupt
CAN_INT	Releases a task waiting for an interrupt from WAIT status
DIS_INT	Disables interrupts by device number
ENA_INT	Enables interrupts by device number
RES_INT	Terminates interrupt handler operation and calls the restart address

Define Interrupt Handler (DEF_INT)

System Call 18. DEF_INT sets the start address of the interrupt handler. It has the following syntax.

```
int DEF_INT (device_no, start_adr)
```

(1) Parameter.

I/O	Name	Description
In	int device_no;	Device number (interrupt level or vector type)
In	int (start_adr) ( );	Pointer to interrupt handler start address

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_DVN	4	Device number error
E_SYS	5	System error

(3) C format.

```
ercode = DEF_INT(device_no, start_adr);
short device_no;
pointer start_adr;
```

If DEF_INT is issued, correspondence between interrupt request level of external μPD71059 interrupt controller (or interrupt request vector type) and start address of the interrupt handler is established. When an interrupt request vector type is specified, the interrupt request control register can also be set at the same time.

If 0 is specified for the interrupt handler start address, the existing start address is cleared. If the start address of the interrupt handler is cleared after an interrupt request level of the interrupt controller is specified, the mask bit (IMK) equivalent to the specified interrupt request level is set and the interrupt is masked. Then the existing start address is cleared.

If the start address of the interrupt handler is cleared after an interrupt request vector type is specified, the existing start address is cleared and the interrupt request control register is set. At this time, the interrupt mask can be set at the same time by explicitly setting bit 6 of the interrupt request control register.

If the start address of the interrupt handler is not 0, the address is set with no other changes. The IMK (mask bit) is never altered.

If the start address of the interrupt handler is set after the vector type of interrupt request is specified, the interrupt request control register is also set. Therefore, interrupt mask operation can be specified using bit 6 of the interrupt request control register.

External Interrupt Controller Definition

The interrupt request level of the external interrupt controller can be specified by setting 0 in bit 7. It is specified as follows.

Bit(s)	Description
0-2	Slave level interrupt request level
3	If 0, master/slave configuration; if 1, master only
4-6	Master interrupt request level
7	Fixed to 0
8-15	Upper-order byte is fixed to 0

The low-order byte is used to specify the interrupt level. Bits 0 to 2 specify the slave interrupt request level when in master-slave configuration; bit 3, whether to use any slave device; bits 4 to 6, the master interrupt request level.

The interrupt request level of the interrupt controller and each interrupt request vector type are in one-to-one corrrespondence The interrupt request is divided into 72 levels, and they correspond to interrupt request vector types 56 to 127.

For example, if, the device consists of only the master, 0 is specified for the master interrupt request level; this interrupt request vector type becomes 56. Slave interrupt request level 7 must be connected to master interrupt request level 7 when in master-slave configuration and becomes vector type 127.

The interrupt request vector type can be specified by setting 1 in bit 7. It is specified as follows.

Bit(s)	Description
0-6	Vector type
7	Fixed to 1
8-15	Interrupt request control register value

The μPD79011 operating system uses the on-chip time base counter as the system timer. As a result, other tasks cannot specify vector type 31 (equivalent to the time base counter) when the system timer function is used.

Signal Interrupt (SIG_INT)

System Call 19. SIG_INT activates a task waiting for an interrupt and terminates the currently executing interrupt handler. It has the following syntax.

```
void SIG_INT (task_no)
```

(1) Parameter.

I/O	Name	Description
In	int task_no;	Target task number (0 to 62)

(2) C format.

```
ercode = SIG_INT(task_no);
short task_no;
```

SIG_INT can be issued only from inside an interrupt handler.

If SIG_INT is issued, the interrupt handler operation ends and control is passed to the target task. Therefore, when SIG_INT is used, control is never passed to the address following SIG_INT.

If an error occurs, no error code is returned and the specified task is not started. In this case, control is returned immediately to the point where the interrupt was issued.

SIG_INT is not used to control multiprocessing of external or internal interrupt requests. Nesting management related to the interrupt handler and execution of the EOI (End Of Interrupt) and FINT (Finish Interrupt) instructions must be done in each interrupt handler.

The procedures used to issue SIG_INT (and system call RES_INT) differ from those to issue other system calls. When using assembly language, SIG_INT is issued as follows.

Procedure	Description
PUSH task_no	The target task number is set in stack
BR SIG_INT_ ENTRY	Far jump to absolute address 0FC00EH

Wait for Interrupt (WAI_INT)

System Call 20. WAI_INT moves a task into the WAIT status. It has the following syntax.

```
int WAI_INT ( )
```

(1) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_INT	8	Release from interrupt wait status

(2) C format.

```
ercode = WAI_INT;
```

When issuing this system call, the current task goes into the interrupt wait status. If the SIG_INT system call is issued to a waiting task (which was invoked by WAI_INT), the specified task is released from the WAIT status.

A task can release another task's WAIT (for interrupt) status by means of the CAN_INT system call. Otherwise an interrupt handler will release the WAIT status after an interrupt is presented.

If SIG_INT is used to release a task from the WAIT status, the error code E_OK is returned. If CAN_INT is used to release the WAIT status, the error code E_INT is returned.

Cancel Interrupt (CAN_INT)

System Call 21. CAN_INT releases the specified task from the WAIT status. It has the following syntax.

int CAN_INT (task_no)

(1) Parameter.

I/O	Name	Description
In	int task_no;	Task number (0 to 62)

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_INT	8	Task is not waiting for interrupt

(3) C format.

```
ercode = CAN_INT(task_no);
short task_no;
```

If CAN_INT is issued to a task that is waiting for an interrupt (due to system call WAI_INT), the specified task exits the WAIT status. If CAN_INT is issued when the specified task is not waiting for any interrupt, the E_INT error code is returned.

Disable Interrupt (DIS_INT)

System Call 22. DIS_INT disables interrupts in units of device number (interrupt request level or interrupt request vector type). It has the following syntax.

int DIS_INT (device_no)

(1) Parameter.

I/O	Name	Description
In	int device_no;	Device number

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_DVN	4	Device number error

(3) C format.

```
ercode = DIS_INT(device_no);
short device_no;
```

DIS_INT can be issued from either a task or an interrupt handler.

To specify the interrupt request level of the interrupt controller, set the corresponding IMR (mask bit) of the external 71059. To specify the interrupt request vector type, set bit 6 of the interrupt request control register.

Enable Interrupt (ENA_INT)

System Call 23. ENA_INT enables interrupts in units of device number (interrupt request level or interrupt request vector type). It has the following syntax.

int ENA_INT (device_no)

(1) Parameter.

I/O	Name	Description
In	int device_no;	Device number

(2) Return value.

Error Code	Number	Description
E_OK	0	Normal end
E_DVN	4	Device number error

(3) C format.

```
ercode = ENA_INT(device_no);
short device_no;
```

ENA_INT can be issued from either a task or an interrupt handler.

To specify the interrupt request level of the interrupt controller, reset the corresponding IMR (mask bit) of external 71059. To specify the interrupt request vector type, reset bit 6 of the interrupt request register.

Reset Interrupt (RES_INT)

System Call 24. RES_INT terminates interrupt handler operation and passes control to the restart address. It has the following syntax.

void RES_INT ( )

(1) C format.

```
ercode = RES_INT( );
```

RES_INT is always used in conjunction with system call SET_ADR.

If SET_ADR has been already issued in a task that was interrupted by a handler that issues RES_INT, control is passed to the specified restart address. If SET_ADR has not been issued to that task, control is returned to the point where the interrupt was issued.

RES_INT cannot be used to control multiple interrupt processing, neither for internal nor for external 71059 sources. Management of interrupt handler nesting and the execution of EOI (End Of Interrupt) and FINT (Finish Interrupt) instructions must be done in each interrupt handler.

The procedure for issuing RES_INT (and system call SIG_INT) differs from the procedure for issuing other system calls. Use the following syntax to issue RES_INT using assembly language.

```
    BR RES_INT_ ENTRY;  Far jump to absolute
                        address FC020H
```