NEC

NEC Electronics Inc.

μPD70236 (V53)
16-Bit Microprocessor: High-Speed, High-Integration, CMOS

T-49-17-15

Description

The V53™ is a high-speed, high-integration 16-bit CMOS microprocessor with a CPU that is object and source code compatible with the V20®/V30®. Integrated on the same die is a 4-channel DMA controller, a UART, three timer/counters, an interrupt controller, a refresh controller, a clock generator, and a bus controller.

(1) The DMA unit has four channels of high-bandwidth DMA (up to 8M bytes/sec). It has two sets of control registers, one compatible with the μPD71087/8237 and another with the μPD71071.

(2) The UART offers asynchronous serial I/O and is functionally compatible with the μPD71051 (8251).

(3) The three 16-bit general-purpose timer/counters are compatible with the μPD71054 (8254).

(4) The interrupt controller is identical to the μPD71059 (8259) and offers eight interrupt channels. External μPD71059s may be cascaded.

(5) The refresh controller generates a 16-bit refresh cycle for use with dynamic or pseudostatic RAMs.

(6) The clock generator uses a crystal at two times the desired frequency to produce the internal clock for the CPU and peripherals. A peripheral clock is also output.

(7) The bus controller generates μPD71088-style control signals for easy interface to external devices. The full V33 bus is also provided. Bus cycles are nominally two clock cycles long and can be extended using the internal wait state generator. Dynamic bus sizing can be used to set the data-path width for every bus cycle. Both 8- and 16-bit cycles are supported, allowing the V53 to be used on both 8- and 16-bit systems.

The V53 CPU is identical to the μPD70136 (V33™). Hardwired data-path control and a high-bandwidth bus give a performance level of 16 MHz, which is increased to four times that of the 10-MHz V30. The 1M-byte addressing range of the V30 is to 16M bytes using an on-chip address translation table.

The V53 instruction set is upward compatible with the native modes of the V20, V30, V40™, and V50™. It includes bit processing, bit field insertion and extraction, and BCD string arithmetic. Using a modified Booth's algorithm, the 16-MHz V53 executes 16-bit multiplies in 750 ns. The CPU performance is the highest currently available in a high-integration microprocessor.

The V53 has an undefined instruction trap that allows instructions not part of the V-series instruction set (such as commands for proprietary MMUs) to be emulated. High-speed numerics support is provided by the μPD72291 CMOS floating-point unit (530K FLOPs at 16 MHz).

The V53's combination of high-speed CPU and DMA makes it ideal for high-bandwidth data control applications such as disk or LAN controllers. The high integration and software compatibility of the CPU and peripherals with the V33 and V30 makes the V53 ideal for very compact personal computer applications such as diskless work stations and lap top computers, or embedded MS-DOS® compatible PCs for POS terminals or control applications.

Features

- ☐ High-speed, V30-compatible CPU
  - — 125-ns minimum instruction execution time at 16 MHz
  - — 750-ns 16-bit multiply at 16 MHz
  - — 1.19 μs 16-bit divide (16 MHz)
  - — Fastest high-integration MPU available
- ☐ Dual bus architecture
- ☐ 8-byte instruction queue
- ☐ Expanded LIM 4.0-compatible 24-bit addressing
- ☐ Four DMA channels (to 8M bytes/sec)
- ☐ On-chip serial I/O controller
- ☐ Three μPD71054-compatible 16-bit counter/timers
- ☐ Eight-channel μPD71059-compatible interrupt controller
- ☐ Refresh controller
- ☐ Bus controller with wait-state generator
- ☐ Clock generator with STOP mode control for low power
- ☐ 16-MHz (or 12.5-MHz) operation with 32-MHz (or 25-MHz) crystal

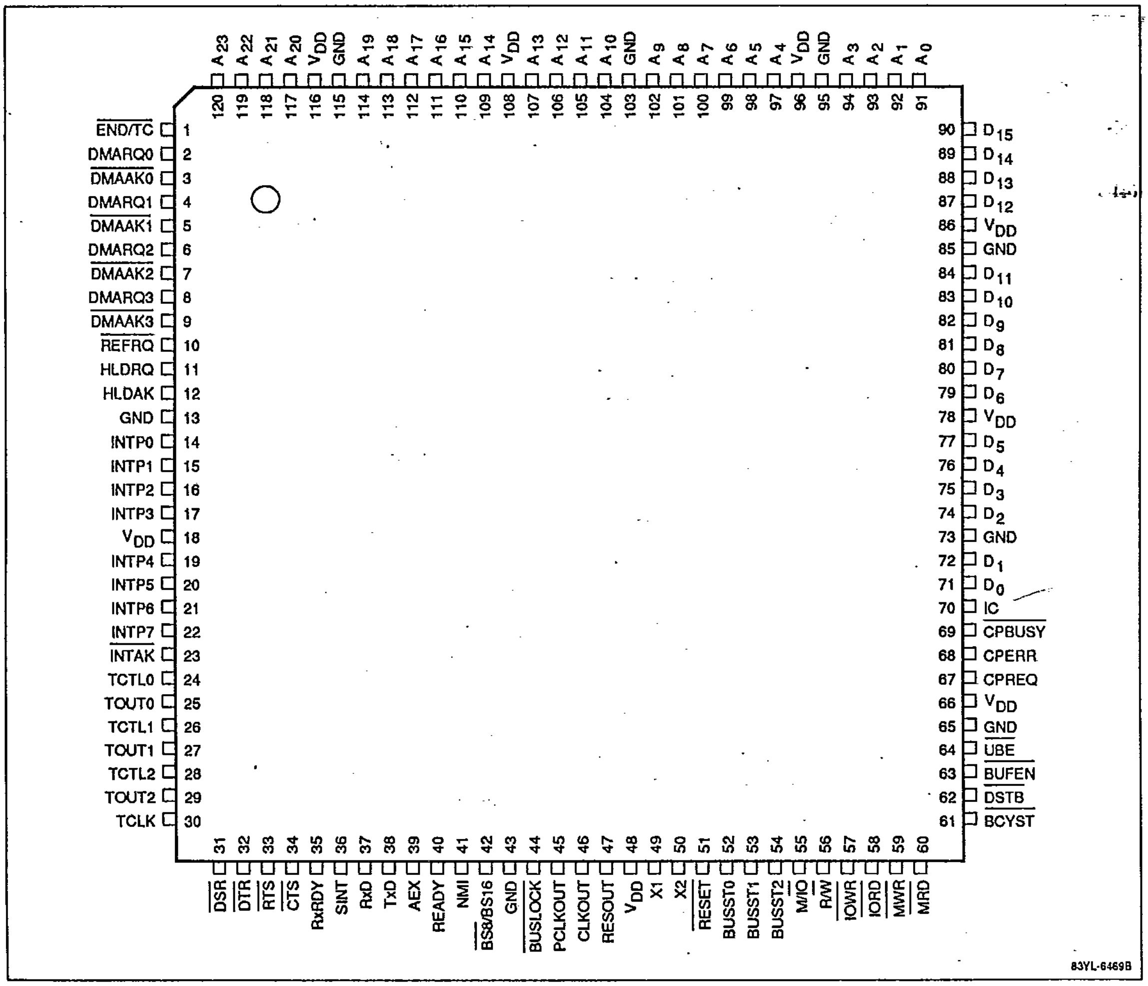
T-49-17-15

Ordering Information

Part Number	Clock (MHz)	Package
μPD70236GD-10	10	120-pin plastic QFP
GD-12	12	
GD-16	16	
R-10	10	132-pin ceramic PGA
R-12	12	
R-16	16	

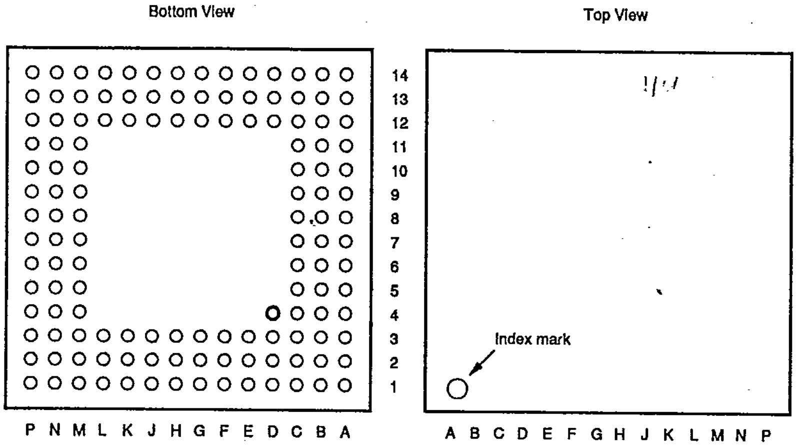
Pin Configurations

120-Pin Plastic QFP



T-49-17-15

132-Pin Ceramic PGA



Pin	Symbol	Pin	Symbol	Pin	Symbol	Pin	Symbol	Pin	Symbol	Pin	Symbol
A1	A_{22}	B9	A_9	D3	DMARQ0	H1	INTP2	L13	GND	N7	$\overline{\text{BUSLOCK}}$
A2	A_{20}	B10	A_5	D12	D_{14}	H2	INTP3	L14	CPERR	N8	RESOUT
A3	GND	B11	GND	D13	IC	H3	V_{DD}	M1	TOUT0	N9	X2
A4	A_{19}	B12	A_2	D14	D_{11}	H12	GND	M2	TCTL2	N10	BUSST0
A5	A_{16}	B13	IC	E1	HLDRQ	H13	D_2	M3	TCLK	N11	$\text{R/}\overline{\text{W}}$
A6	A_{14}	B14	D_{12}	E2	$\overline{\text{DMAAK3}}$	H14	D_3	M4	$\overline{\text{DTR}}$	N12	$\overline{\text{IORD}}$
A7	A_{12}	C1	$\overline{\text{DMAAK2}}$	E3	DMARQ2	J1	INTP4	M5	RxRDY	N13	$\overline{\text{BCYST}}$
A8	A_{11}	C2	$\overline{\text{DMAAK0}}$	E12	V_{DD}	J2	INTP5	M6	AEX	N14	$\overline{\text{UBE}}$
A9	NC	C3	IC	E13	D_{10}	J3	INTP7	M7	GND	P1	$\overline{\text{DSR}}$
A10	A_8	C4	A_{23}	E14	D_8	J12	IC	M8	V_{DD}	P2	$\overline{\text{CTS}}$
A11	A_6	C5	IC	F1	NC	J13	D_1	M9	BUSST1	P3	SINT
A12	A_4	C6	A_{18}	F2	HLDAK	J14	NC	M10	IC	P4	TxD
A13	A_3	C7	V_{DD}	F3	$\overline{\text{REFRQ}}$	K1	INTP6	M11	$\overline{\text{MRD}}$	P5	READY
A14	A_0	C8	GND	F12	D_9	K2	$\overline{\text{INTAK}}$	M12	IC	P6	$\overline{\text{BS8}}\text{/BS16}$
B1	DMARQ1	C9	A_7	F13	D_7	K3	TCTL1	M13	$\overline{\text{BUFEN}}$	P7	PCLKOUT
B2	$\overline{\text{END}}\text{/}\overline{\text{TC}}$	C10	V_{DD}	F14	D_6	K12	V_{DD}	M14	CPREQ	P8	CLKOUT
B3	A_{21}	C11	A_1	G1	INTP1	K13	$\overline{\text{CPBUSY}}$	N1	TOUT1	P9	X1
B4	V_{DD}	C12	D_{15}	G2	INTP0	K14	D_0	N2	IC	P10	$\overline{\text{RESET}}$
B5	A_{17}	C13	D_{13}	G3	GND	L1	TCTL0	N3	$\overline{\text{RTS}}$	P11	BUSST2
B6	A_{15}	C14	GND	G12	V_{DD}	L2	IC	N4	IC	P12	$\text{M/}\overline{\text{IO}}$
B7	A_{13}	D1	DMARQ3	G13	D_5	L3	TOUT2	N5	RxD	P13	$\overline{\text{IOWR}}$
B8	A_{10}	D2	$\overline{\text{DMAAK1}}$	G14	D_4	L12	$\overline{\text{DSTB}}$	N6	NMI	P14	$\overline{\text{MWR}}$

83YL-6470B

Pin Identification

Symbol	I/O	Function
A_0-A_{23}	Out	Address bus
AEX	Out	Address expansion mode flag
$\overline{BCYST}$	Out	Bus cycle start
$\overline{BS8}$/BS16	In	Data bus width specification
$\overline{BUFEN}$	Out	Buffer enable
$\overline{BUSLOCK}$	Out	Bus lock flag
BUSST0-BUSST2	Out	Bus status
CLKOUT	Out	System clock
$\overline{CPBUSY}$	In	Coprocessor busy
$\overline{CTS}$	Out	Clear to send
D_0-D_{15}	I/O	Data bus
$\overline{DMAAK0}$-$\overline{DMAAK3}$	Out	DMA acknowledge
DMARQ0-DMARQ3	In	DMA request
$\overline{DSR}$	In	Data set ready
$\overline{DSTB}$	Out	Data strobe
$\overline{DTR}$	Out	Data terminal ready
$\overline{END/TC}$	I/O	DMA service forced-end input; DMA service complete output
HLDAK	Out	Bus hold acknowledge
HLDRQ	In	Bus hold request
$\overline{INTAK}$	Out	Interrupt acknowledge
INTP0-INTP7	In	Maskable interrupt request
$\overline{IORD}$	Out	I/O read
$\overline{IOWR}$	Out	I/O write
M/$\overline{IO}$	Out	Memory I/O select
$\overline{MRD}$	Out	Memory read
$\overline{MWR}$	Out	Memory write
NMI	In	Nonmaskable interrupt request
PCLKOUT	Out	External I/O clock
$\overline{READY}$	In	Bus cycle end
$\overline{REFRQ}$	Out	Refresh request
$\overline{RESET}$	In	Reset
RESOUT	Out	System reset
$\overline{RTS}$	Out	Request to send
R/$\overline{W}$	Out	Read/write
RxD	In	Serial receive data
RxRDY	Out	Serial receive ready
SINT	Out	Serial interrupt request
TCLK	In	Timer clock
TCTL0-TCTL2	In	Timer control
TOUT0-TOUT2	Out	Timer output
TxD	Out	Serial transmit data
$\overline{UBE}$	Out	Data bus higher byte enable
X1, X2	In	Crystal/external clock
V_{DD}	In	+5-volt power source
GND		Ground
IC		Internal connection
NC		No connection

Table 1. Output Pin States

Symbol	Hold	Halt	Reset	DMA Cascade
A_0-A_{23}	Hi-Z	L	Hi-Z	Hi-Z
AEX	Note 6	Note 6	H/L	Note 6
$\overline{BCYST}$	Hi-Z	Note 4	Hi-Z	Hi-Z
$\overline{BUFEN}$	Hi-Z	H	Hi-Z	Hi-Z
$\overline{BUSLOCK}$	Note 5	Note 5	H	H
BUSST0-BUSST2	Hi-Z	H	Hi-Z	H
CLKOUT	O	O	O	O
D_0-D_{15}	Hi-Z	Note 3	Hi-Z	Hi-Z
$\overline{DMAAK0}$-$\overline{DMAAK3}$	H	O	H	O
$\overline{DSTB}$	Hi-Z	H	Hi-Z	Hi-Z
$\overline{DTR}$	O	O	H	O
$\overline{END/TC}$	Hi-Z	O	Hi-Z	O
HLDAK	H	H/L	L	L
$\overline{INTAK}$	H	H	H	H
$\overline{IORD}$	Hi-Z	H	Hi-Z	Hi-Z
$\overline{IOWR}$	Hi-Z	H	Hi-Z	Hi-Z
M/$\overline{IO}$	Hi-Z	L	Hi-Z	H
$\overline{MRD}$	Hi-Z	H	Hi-Z	Hi-Z
$\overline{MWR}$	Hi-Z	H	Hi-Z	Hi-Z
PCLKOUT	O	O	O	O
REFRQ	H	O	H	H
RESOUT	L	L	H	L
$\overline{RTS}$	O	O	H	O
R/$\overline{W}$	Hi-Z	L	Hi-Z	H
RxRDY	O	O	H	O
SINT	O	O	L	O

Table 1. Output Pin States (cont)

Symbol	Hold	Halt	Reset	DMA Cascade
TOUT0-TOUT2	O	O	O	O
TxD	O	O	H	O
$\overline{UBE}$	HI-Z	H	HI-Z	HI-Z

Notes:

(1) The pin states are interpreted as follows: H is high level; L is low level; H/L is high or low level; HI-Z is high impedance; O is indeterminate.

(2) Halt includes both the HALT and STOP modes.

(3) Undefined for the first two clocks of the halt acknowledge cycle and the HI-Z.

(4) L for the first clock of the halt acknowledge cycle and then H.

(5) L under either of the following conditions: an instruction is executed during hold with a BUSLOCK prefix, or the HALT instruction is executed with a BUSLOCK prefix. Otherwise, the value is H.

(6) H in address expansion mode; L in nonexpansion mode.

PIN FUNCTIONS

A_0-A_{23} (Address Bus)

These pins constitute an address bus that outputs real addresses when memory or an I/O device is accessed. Up to 64K bytes of I/O space and up to 16M bytes of memory space (including reserved areas) can be accessed through the address bus.

The address bus enters the high-impedance state if one of the following occurs.

- $\overline{RESET}$ signal is applied
- Microprocessor is in HOLD mode
- DMA requests are cascade connected

The status of the address bus is undefined during an interrupt acknowledge cycle. When interrupt requests are cascade connected, the slave ICU address is output on pins A_0-A_2.

When I/O is accessed, pins A_{16}-A_{23} go low. The address can be expanded even when the interrupt vector table is accessed.

AEX (Address Extension)

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

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

This signal indicates the start of a bus cycle by going low for one clock immediately after the bus cycle is started. When the bus is placed in the hold state, the $\overline{BCYST}$ pin enters the high-impedance state.

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

$\overline{BS8}$/BS16 is driven low by external logic when the μPD70236 addresses a device with an 8-bit data path. If the μPD70236 operand is 16 bits wide and $\overline{BS8}$/BS16 is low, then the μPD70236 will perform two 8-bit bus cycles. The current bus cycle will handle the low byte on D_0-D_7, and the next bus cycle will handle the upper byte also on D_0-D_7. This input is ignored during HLDAK, interrupt acknowledge, and coprocessor cycles.

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

$\overline{BUFEN}$ (Buffer Enable)

This signal is output to enable an external buffer, and becomes active during the read cycle, interrupt acknowledge cycle, and write cycle. It does not become active while the internal I/O is being accessed.

$\overline{BUSLOCK}$ (Bus Lock)

$\overline{BUSLOCK}$ should be used by external logic to exclude any other bus master (e.g., a DMA controller) from using a shared resource that the μPD70236 currently is using. When $\overline{BUSLOCK}$ is asserted high, HLDRQ will be ignored.

$\overline{BUSLOCK}$ is asserted when the $\overline{BUSLOCK}$ prefix is executed or when the μPD70236 is performing a bus operation that must not be interfered with, such as an interrupt acknowledge cycle. $\overline{BUSLOCK}$ has the same timing as the address bus A_0-A_{23} and is driven high during HLDAK and RESET.

BUSST0-BUSST2 (Bus Status)

These three pins encode and output information identifying the type of bus cycle currently being executed. They enter the high-impedance state in the bus hold mode. These pins are used with the M/$\overline{IO}$ and R/$\overline{W}$ signals, as shown in table 2.

Table 2. Bus Cycles

M/$\overline{\text{IO}}$	R/$\overline{\text{W}}$	BUSST2	BUSST1	BUSST0	Bus Cycle
0	1	0	0	0	Interrupt acknowledge cycle (from SLAVE)
0	1	1	0	0	Interrupt acknowledge cycle (from ICU)
0	1	0	0	1	External I/O read cycle
0	1	1	0	1	Internal I/O read cycle
0	0	0	0	1	External I/O write cycle
0	0	1	0	1	Internal I/O write cycle
0	1	0	1	0	Coprocessor read cycle
0	0	0	1	0	Coprocessor write cycle
0	0	0	1	1	Halt acknowledge cycle
1	1	0	0	0	Instruction fetch cycle
1	1	1	0	0	Refresh cycle
1	1	0	0	1	CPU memory read cycle
1	1	1	0	1	DMA read transfer cycle
1	0	0	0	1	CPU memory write cycle
1	0	1	0	1	DMA write transfer cycle
1	1	0	1	0	Coprocessor memory read cycle
1	0	0	1	0	Coprocessor memory write cycle
1	1	1	1	1	DMA cascade

Interrupt Acknowledge Cycle (from SLAVE). This cycle is the second interrupt acknowledge cycle during which an interrupt request from a slave interrupt control unit (ICU) is acknowledged. During this cycle, the data output by an external interrupt controller is processed as a vector. The bus sizing function cannot be effected in this cycle. The programmable wait function and $\overline{\text{READY}}$ signals are both valid, however.

Interrupt Acknowledge Cycle (from ICU). This cycle is output during the first interrupt acknowledge cycle, during which an interrupt request for a non-slave ICU is acknowledged. During this acknowledge cycle, the data output by the internal ICU is processed as a vector, and the bus sizing function cannot be effected. The programmable wait function and $\overline{\text{READY}}$ signal are both valid, however.

External I/O Read Cycle. This cycle is output when an external I/O area is read by executing the IN instruction. During this cycle, the bus sizing function can be effected. Also, the programmable wait function and $\overline{\text{READY}}$ signal are both valid.

Internal I/O Read Cycle. This cycle is output when the internal I/O area is read by executing the IN instruction. The bus sizing function cannot be effected. Both the programmable wait function and $\overline{\text{READY}}$ signal are invalid. However, two wait state clocks are automatically inserted into all internal I/O area cycles except those for the address expansion table and address expansion flag.

External I/O Write Cycle. This cycle is output when an external I/O area is written by executing the OUT instruction. The bus sizing function can be effected. Also, the programmable wait function and $\overline{\text{READY}}$ signal are both valid.

Internal I/O Write Cycle. This is output when the internal I/O area is written by executing the OUT instruction. The bus sizing function cannot be effected. Both the programmable wait function and $\overline{\text{READY}}$ signal are invalid. However, two wait state clocks are automatically inserted into all internal I/O area cycles except those for the address expansion table and address expansion flag.

Coprocessor Read Cycle. This cycle indicates that an external coprocessor is accessed for data read when a coprocessor instruction is executed. The bus timing and ac characteristics of this cycle are the same as those of the ordinary I/O read cycle.

Although the bus sizing function cannot be effected, coprocessor operations are not guaranteed if the bus sizing function is used. The programmable wait function is invalid, but the $\overline{\text{READY}}$ signal is valid.

Coprocessor Write Cycle. This cycle indicates that an external coprocessor instruction is executed. The bus timing and ac characteristics of this cycle are the same as those of the ordinary I/O write cycle.

Although the bus sizing function can be effected, coprocessor operations are not guaranteed if the bus sizing function is used. The programmable wait function is invalid, but the $\overline{\text{READY}}$ signal is valid.

Halt Acknowledge Cycle. This cycle is output when the HALT instruction is executed. During this bus cycle, the $\overline{\text{DSTB}}$ pin does not output a low level. The bus sizing function cannot be effected. Both the programmable wait function and $\overline{\text{READY}}$ signal are invalid.

Instruction Fetch Cycle. This cycle indicates that an instruction is being fetched. The bus sizing function can be effected. Also, the programmable wait function and $\overline{\text{READY}}$ signal are both valid.

Refresh Cycle. This cycle indicates that DRAM refreshing is in progress. The bus sizing function cannot be effected. (Note that $\overline{\text{BS8}}$/BS16 must be 16 bits.) The programmable wait function and $\overline{\text{READY}}$ signal are both valid.

CPU Memory Read Cycle. This cycle is output when the CPU reads data from memory. The bus sizing function can be effected. Also, the programmable wait function and $\overline{\text{READY}}$ signal are both valid.

DMA Read Transfer Cycle. This cycle is output when DMA transfer (that is, data transfer from memory to I/O) takes place. The bus sizing function cannot be effected. The programmable wait function and $\overline{\text{READY}}$ signal are both valid.

CPU Memory Write Cycle. This cycle is output when the CPU writes data to memory. The bus sizing function can be effected. Also, the programmable wait function and $\overline{\text{READY}}$ signal are both valid.

DMA Write Transfer Cycle. This cycle is output when write DMA transfer (that is, data transfer from I/O to memory) takes place. The bus sizing function cannot be effected. The programmable wait function and $\overline{\text{READY}}$ signal are both valid.

Coprocessor Memory Read Cycle. This cycle is output when data read from memory is sent to the coprocessor. Although the bus sizing function cannot be effected, coprocessor operations are not guaranteed if bus sizing is used. The programmable wait function and $\overline{\text{READY}}$ signal are both valid.

Coprocessor Memory Write Cycle. This cycle is output when data for a coprocessor is written to memory. The CPU does not drive the data bus. Instead, the coprocessor drives the data bus to write data to memory.

Although the bus sizing function cannot be effected, coprocessor operations are not guaranteed if the bus sizing function is used. The programmable wait function and $\overline{\text{READY}}$ signal are both valid.

DMA Cascade. This cycle indicates that the DMA is cascade connected to an external slave DMA controller. During this cycle, the buses are relinquished.

CLKOUT (Clock Output)

This pin outputs a square-wave clock pulse. The frequency of the output clock pulse is obtained by dividing the frequency of the clock signal input to the X1 and X2 pins by a specific value. The duty factor of the output clock pulse is 50%. The output frequency is the same as the operating frequency of the CPU (programmable to one-half, one-fourth, one-eighth, or one-sixteenth of the oscillation frequency).

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

$\overline{\text{CPBUSY}}$ is asserted low by a coprocessor (such as µPD72291) when it is busy with an internal operation. The µPD70236 uses this pin to check the status of the coprocessor. $\overline{\text{CPBUSY}}$ is sampled on the falling edge of each clock. This input is not internally synchronized. To ensure proper device operation, minimum setup and hold times must be met.

If a coprocessor is not connected to the µPD70236, $\overline{\text{CPBUSY}}$ should be grounded.

$\overline{\text{CTS}}$ (Clear to Send)

This is a serial transmission control input pin. The SCU is ready for data transmission when bit 0 of the SCM register is set to 1 and this pin is at low level. When this pin is made high while data transmission is in progress, transmission is stopped after the current data has been completely transmitted, and the TxD pin goes high.

D_0-D_{15} (Data Bus)

These pins constitute a data bus that inputs or outputs write data and read data when the external main memory or I/O device is accessed. The data bus is in the input mode during any bus cycle other than a write cycle. During the write bus cycle, the bus outputs data starting from the rising edge of the T1 clock until the cycle following the write bus end cycle.

$\overline{\text{DMAAK0}}$-$\overline{\text{DMAAK3}}$ (DMA Acknowledge)

These pins output active-low DMA acknowledge signals from channels 0 to 3 of the internal DMAU.

DMARQ0-DMARQ3 (DMA Request)

These pins input active-high DMA request signals from channels 0 to 3 of the internal DMA control unit (DMAU).

$\overline{\text{DSR}}$ (Data Set Ready)

This is a general-purpose input pin. The status of this pin can be determined by reading bit 7 of the serial status (SST) register.

$\overline{\text{DSTB}}$ (Data Strobe)

This is a strobe signal for read and write operations. The signal does not go low during the halt acknowledge cycle that indicates that the HALT instruction has been executed. When the buses are placed in the hold state, the $\overline{\text{DSTB}}$ pin enters the high-impedance state. The signal output timing of this pin differs depending on whether a read or write operation is performed. The $\overline{\text{DSTB}}$ signal does not go low when the internal I/O area is accessed.

$\overline{\text{DTR}}$ (Data Terminal Ready)

This is a general-purpose output pin. The status of this pin can be set by bit 1 of the SCM register.

END/$\overline{TC}$ (End/Terminal Count)

This pin inputs the END signal to or outputs the TC signal from the internal DMAU.

$\overline{END}$ Input. When a low-level pulse is input to this pin during DMA transfer, the DMA service under execution is terminated after the current bus cycle is over.

$\overline{TC}$ Output. When the count register of the DMAU channel currently performing DMA transfer becomes 0, and when the DMA transfer has been performed the specified number of times, the TC pin outputs a low-level pulse.

HLDAK (Hold Acknowledge)

This is an acknowledge signal that indicates that the V53 has accepted the HLDRQ signal, placed the address, data, and control buses in the high-impedance state, and relinquished the buses to an external device. The external devices that can acquire the buses are assigned the following priority.

REFU (highest priority)
DMAU
HLDRQ
CPU
REFU

If a bus hold request takes place while the buses are idle (TI state), during the CPU bus cycle, or during lowest-priority refresh cycle, the HLDRQ signal is accepted immediately after the bus cycle is over and the buses are relinquished.

If a DMA request or top-priority refresh request is generated while the buses are in the hold state, the HLDAK signal is forcibly made inactive. In this case, the external device must return control of the bus to the V53 (making the HLDRQ signal inactive). Therefore, the high-level width of the HLDAK signal when it is made inactive forcibly is 1 clock minimum.

HLDRQ (Hold Request)

HLDRQ is asserted high by external logic when an external bus master (e.g., a DMA controller) wants to take over the μPD70236 bus. When HLDRQ is detected high, the μPD70236 will release the bus after the current bus operation is completed. Note that this is not necessarily the current bus cycle. The μPD70236 releases its bus by floating the address, data, and control buses.

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

INTAK (Interrupt Acknowledge)

This is an active-low acknowledge signal for a maskable interrupt.

INTP0-INTP7 (Interrupt from Peripherals)

These are asynchronous interrupt request input pins for the internal interrupt control unit (ICU). The input signals can be triggered either at the rising edge or at high level. The priority of these signals can be fixed or rotated. These interrupt request inputs are also used to release the HALT and STOP modes.

$\overline{IORD}$ (I/O Read)

This active-low read signal goes low during the I/O read cycle. This signal is also output when write DMA transfer is performed. However, it is not output during the CPU's internal I/O read cycle.

$\overline{IOWR}$ (I/O Write)

This is an active-low write signal that goes low during the I/O write cycle. This signal is also output when read DMA transfer is performed in two output timing modes: the expansion write mode and the ordinary write mode. It is not output during the CPU's internal I/O write cycle.

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

This pin indicates whether a memory or other device (such as an I/O device or coprocessor) is currently accessed. The device to be accessed is determined by this pin and the BUSST0 and BUSST1 signals. The M/$\overline{IO}$ pin enters the high-impedance state in the bus hold mode. Its status changes at the falling edge of the T1 clock.

$\overline{MRD}$ (Memory Read)

This is an active-low read signal that goes low during a read cycle in which data is read from memory. This signal is output not only during the CPU's memory read, but also during the refresh cycle and when read DMA transfer is performed.

$\overline{MWR}$ (Memory Write)

This active-low write signal goes low when the memory write cycle is in progress. This signal is output not only during the CPU's memory write cycle, but also during the write DMA transfer and when write DMA transfer is performed in two output timing modes: the expansion write mode and the ordinary write mode.

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

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

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

Interrupt processing begins immediately after the end of the current instruction. Once NMI processing comences, no further NMI requests will be accepted until termination of the current NMI routine, which is indicated by the RETI instruction.

PCLKOUT (Peripheral Clock Output)

This pin outputs a square-wave clock pulse with a frequency one-fourth the frequency of the clock signal input to the X1 and X2 pins. The duty factor of the output clock pulse is 50%.

$\overline{\text{READY}}$ (System Ready)

The $\overline{\text{READY}}$ signal is asserted low when the external system is ready for the current bus cycle to terminate. While $\overline{\text{READY}}$ is not asserted, the μPD70236 will add TW (wait) states to the current bus cycle. The bus state in which $\overline{\text{READY}}$ is sampled low will be the last state of the cycle.

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

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

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

This signal is asserted during refresh cycles.

$\overline{\text{RESET}}$ (Reset)

This signal initializes the processor. The processor is reset when this signal is held low for six clocks or longer and then returned to the high level.

RESOUT (Reset Output)

This pin outputs an active-high signal which is an asynchronous $\overline{\text{RESET}}$ signal synchronized with the internal clock. This signal can be used to reset the system.

$\overline{\text{RTS}}$ (Request to Send)

This is a general-purpose output pin. The status of this pin can be set by bit 5 of the serial command (SCM) register.

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

This pin indicates whether the current bus cycle is a read cycle or a write cycle. This pin is valid only while a bus cycle is being executed, and goes high if the current bus cycle is a read cycle or during an interrupt acknowledge cycle; it goes low if the current bus cycle is a write cycle. The R/$\overline{\text{W}}$ pin enters the high-impedance state in the bus hold mode. The level of this pin changes at the falling edge of the T1 clock.

RxD (Receive Data)

When the serial control unit does not receive data, this pin is at high level (mark state). When the pin detects a start bit, the SCU starts receiving serial data from an external device.

RxRDY (Receive Ready)

When the serial control unit has received one character of data, and when that data is transferred to the receive data buffer (that is, when the receive data is ready to be read), this pin goes high.

SINT (Serial Interrupt)

This signal becomes active to output an interrupt request signal from the SCU when the transmit data buffer of the SCU is empty and when the interrupt of the transmitting side is not masked, or when it contains the SCU's receive buffer data to be read and the receive interrupt is not masked.

TCLK (Timer Clock)

This pin inputs a clock pulse from an external source to the internal timer/counter unit (TCU). When the system is initialized, either the external clock or the internal clock is selected to be supplied to the TCU.

TCTL0-TCTL2 (Timer Control)

These pins input control signals to the three TCU counters. The functions of the control signals input through these pins differ depending on the mode (six modes are available) set by the TCU.

TOUT0-TOUT2 (Timer Output)

These are output pins for the internal timer/counter unit. The TCU outputs signals through these pins in six different modes.

TxD (Transmit Data)

When the serial control unit (SCU) has no data to be transmitted to an external device, this pin is at high level (mark state). When transmit data is set in the SCU, the TxD pin automatically outputs a start bit, serial data that has been set in the SCU, a parity bit, and 1 or 2 stop bits.

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

When the microprocessor accesses external main memory or an I/O device that requires the upper 8 bits (D_8-D_{15}) of the data bus, this pin goes low at the falling edge of the T1 clock, enabling the upper byte on the bus. The lower 8 bits (D_0-D_7) of the data bus are controlled by the A_0 pin as shown in the following table.

$\overline{\text{UBE}}$	A_0	Operation
0	0	16 bits accessed
0	1	Upper 8 bits accessed
1	0	Lower 8 bits accessed
1	1	Second cycle (for use with bus sizing function)

When dynamic bus sizing is used to make a 16-bit access into an 8-bit, A_0 must be used as an address bit; $\overline{\text{UBE}}$ can be ignored.

X1, X2 (Crystal)

To use the internal clock generator, connect a crystal with a frequency twice the operating frequency across these pins. When using an external clock generator, input square waves with a frequency twice the operating frequency to the X1 pin. To the X2 pin, make the input signal 180° out of phase (an inverter output) with the signal input to the X1 pin.

UNIT OPERATION

Central Processing Unit (CPU)

The μPD70236 CPU is a high-performance engine whose performance surpasses most other 16-bit CPUs. To achieve this performance level, hardwired data path control was used (no microcode) so that instruction execution times are greatly reduced.

The μPD70236 CPU has functions equivalent to those of the μPD70136 (V33) and is therefore completely software compatible with the V33. The μPD70236 instruction set is upward compatible with the native modes of the V20, V30, V40, and V50.

Clock Generator (CG)

The clock generator divides the oscillation frequency of the crystal or external oscillator connected across pins X1 and X2 by 2, 4, 8, or 16 to generate a clock that is supplied to the CPU as an operation clock and to an external device through the CLKOUT pin. A clock having a frequency one-fourth the oscillation frequency is also output to the PCLKOUT pin.

Bus Interface Unit (BIU)

The bus interface unit controls the pins of the address bus, data bus, and control bus, which are used by the CPU, DMA unit (DMAU), and refresh control unit (REFU).

Bus Arbitration Unit (BAU)

The bus arbitration unit arbitrates the internal bus mastership. The priority of the bus mastership is:

CPU with $\overline{\text{BUSLOCK}}$ (highest priority)
REFU of top priority
DMAU
HLDRQ
Ordinary CPU
REFU of lowest priority

Wait Control Unit (WCU)

The function of the wait control unit is to insert wait states equivalent to 0 to 7 clocks automatically into the memory, I/O, DMA, and refresh cycles. The 16M-byte memory space can be divided into three blocks. In addition, any 1M-byte memory space can also be divided into three blocks.

Refresh Control Unit (REFU)

The REFU supports the DRAM refresh operation by generating 16-bit refresh addresses and a refresh signal ($\overline{\text{REFRQ}}$) indicating that the refresh cycle is currently taking place.

Timer/Counter Unit (TCU)

The timer/counter unit of the μPD70236 performs the same functions as the μPD71054. It provides a set of three independent 16-bit timer/counters.

Serial Control Unit (SCU)

The µPD70236 SCU has the same functions as the µPD71051 except the synchronous mode for supporting RS-232C protocol. This SCU is equipped with a dedicated baud rate generator.

The SCU provides serial communications functions of the start-stop synchronization type. Commands for the SCU in the V53 are similar to those of the µPD71051 except that the V53 uses two registers—SCM (serial command) register and SMD (serial mode) register—to implement the functions of the control word register of the µPD71051.

Interrupt Control Unit (ICU)

The ICU in the V53 has the same functions as those on the µPD71059 except the V53 does not have the CALL mode (8085 mode) or the slave mode of cascade connection. The µPD70236 ICU has eight external interrupt input pins and can arbitrate up to eight interrupt requests. The number of external interrupt inputs can be increased by cascade connecting the ICU to an external interrupt controller.

Unlike the µPD71059, µPD70208, and µPD70216, the INTP0 to INTP7 pins in the V53 do not have internal pullup resistors to reduce current dissipation.

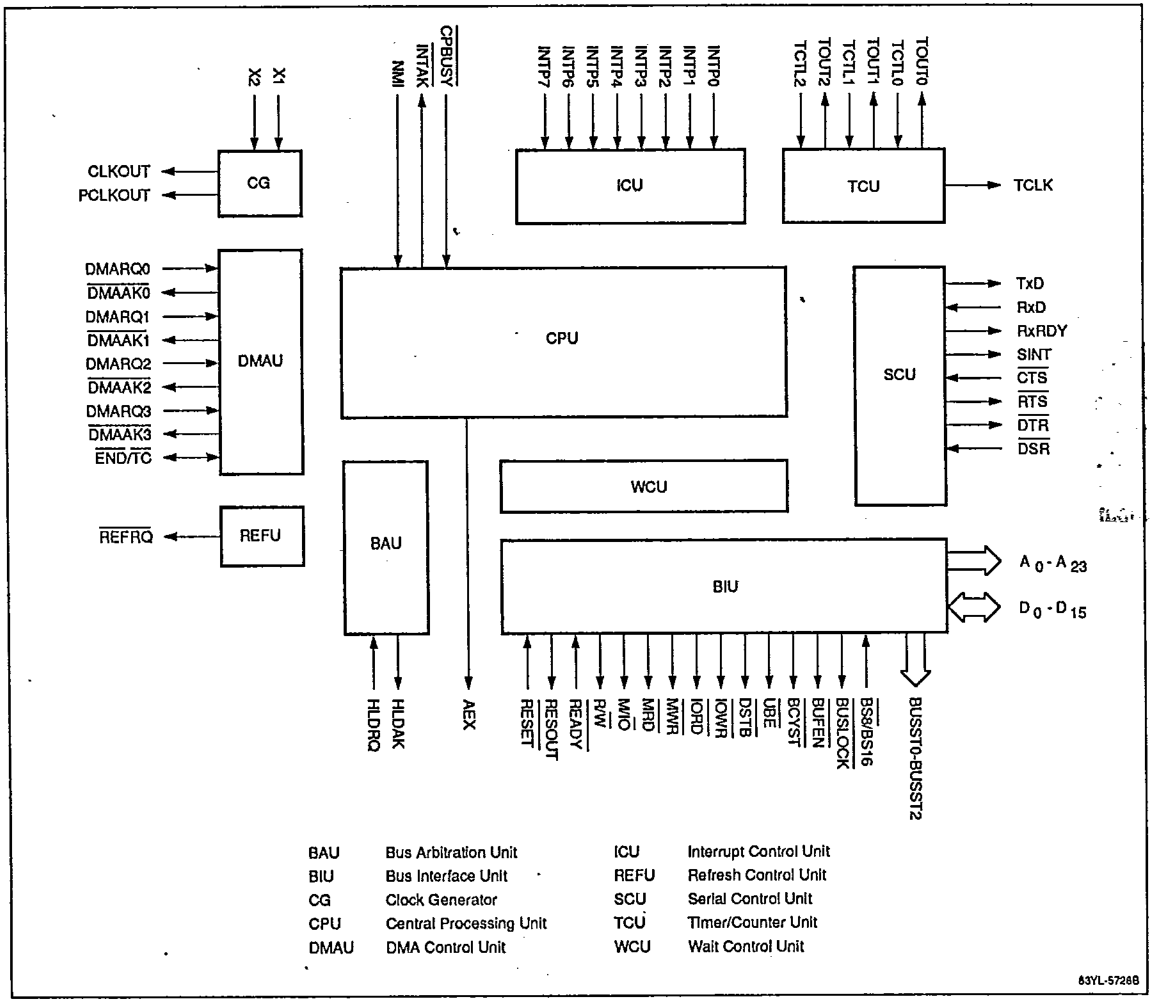
DMA Control Unit (DMAU)

The DMAU on the µPD70236 functions the same as the DMAUs on the µPD71071 and µPD71037 and, therefore, it can operate in two modes (µPD71071 mode and µPD71037 mode). You can set the operation modes using a register in the system I/O area.

In µPD71071 mode, source and destination addresses are 24 bits. In µPD71037 mode, source and destination addresses are 16 bits. To extend these addresses to 20 or 24 bits, four 8-bit bank registers are provided. These registers supply the upper address bits.

The DMA unit provides four channels of µPD71071-compatible or µPD71037-compatible DMA. External hardware requests DMA cycles via the DMA request inputs. DMA is always between an I/O device and memory (fly-by style DMA). External DMA controllers may be cascaded using the V53 DMAU.

µPD70236 Block Diagram



ELECTRICAL SPECIFICATIONS

Absolute Maximum Ratings

T_A = +25°C

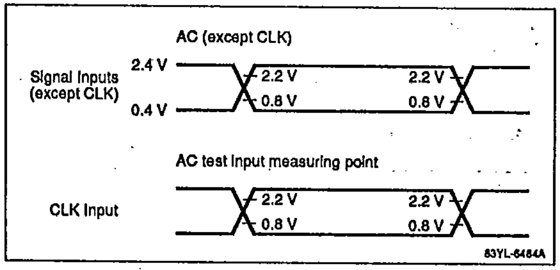
Power supply voltage, V_{DD}	−0.5 to +7.0 V
Input voltage, V_I	−0.5 to V_{DD} + 0.3 V
Clock input voltage, V_K	−0.5 to V_{DD} + 1.0 V
Output voltage, V_O	−0.5 to V_{DD} + 0.3 V
Output short circuit current, I_O	50 mA
Operating temperature, T_{OPT}	−10 to +70°C
Storage temperature, T_{STG}	−65 to +150°C

DC Characteristics

T_A = −10 to +70°C; V_{DD} = +5 V 10%

Parameter	Symbol	Min	Max	Unit	Conditions
Input voltage, high	V_{IH}	2.2	V_{DD} + 0.3	V	Except $\overline{\text{RESET}}$
		0.8 V_{DD}		V	$\overline{\text{RESET}}$
Input voltage, low	V_{IL}	−0.5	0.8	V	Except $\overline{\text{RESET}}$
			0.2 V_{DD}	V	$\overline{\text{RESET}}$
Clock input voltage, high	V_{KH}	0.8 V_{DD}	V_{DD} + 0.5	V	
Clock input voltage, low	V_{KL}	−0.5	0.6	V	
Output voltage,high	V_{OH}	0.7 V_{DD}		V	I_{OH} = −400 μA
Output voltage, low	V_{OL}		0.45	V	I_{OL} = 2.5 mA
Input leakage current, high	I_{LIH}		10	μA	V_I = V_{DD}
Input leakage current, low	I_{LIL}		−10	μA	V_I = 0 V
Output leakage current, high	I_{LOH}		10	μA	V_O = V_{DD}
Output leakage current, low	I_{LOL}		−10	μA	V_O = 0 V
Supply current	I_{DD}		10 f + 40	mA	Operating; f = 2 to 16 MHz
			40	mA	HALT mode
			200	μA	STOP mode

Voltage Thresholds for Timing Measurements



AC Characteristics

$T_A = -10$ to $+70°C$; $V_{DD} = 5\ V \pm 10\%$; C_L of output terminals = 100 pF max

Parameter	Symbol	Min	Max	Unit
Clocks (figure 1)				
CLKOUT period	t_{CYK}	62.5	500	ns
CLKOUT high-level width	t_{KKH}	$0.5\ t_{CYK} - 7$		ns
CLKOUT low-level width	t_{KKL}	$0.5\ t_{CYK} - 7$		ns
CLKOUT rise time	t_{KR}		7	ns
CLKOUT fall time	t_{KF}		7	ns
X1 input period	t_{CYX}	31.25	250	ns
X1 input high-level width	t_{XKH}	11		ns
X1 input low-level width	t_{XKL}	11		ns
X1 input rise time	t_{XKR}		5	ns
X1 input fall time	t_{XKF}		5	ns
X1 to CLKOUT delay	t_{DXK}		20	ns
PCLKOUT period	t_{CYPK}	125	1000	ns
PCLKOUT high-level width	t_{PKH}	$4\ t_{CYK} - 7$		ns
PCLKOUT low-level width	t_{PKL}	$4\ t_{CYK} - 7$		ns
PCLKOUT rise time	t_{PKR}		7	ns
PCLKOUT fall time	t_{PKF}		7	ns
Reset (figure 2)				
$\overline{\text{RESET}}$ setup time vs CLKOUT ↓	t_{SRSTK}	30		ns
$\overline{\text{RESET}}$ hold time vs CLKOUT ↓	t_{HKRST}	15		ns
$\overline{\text{RESET}}$ low-level width	t_{WRSTL}	6		t_{CYC}
RESOUT delay from CLKOUT ↓	t_{DKRO}	0	40	ns
Write, Read (figures 3-12, 16-19, 23-24, 28, 31) Note 2				
$\overline{\text{BCYST}}$ delay from CLKOUT ↓	t_{DKBC}	5	40	ns
$\overline{\text{BCYST}}$ low-level width	t_{BCBCL}	$t_{CYK} - 10$		ns
$\overline{\text{BCYST}}$ high-level width	t_{BCBCH}	$t_{CYK}\ (n+1) - 10$		ns
Address delay from CLKOUT ↓	t_{DKA}	5	40	ns
Control 2 delay from CLKOUT	t_{DKCT2}	0	40	ns
Status delay from CLKOUT ↓	t_{DKST}	5	40	ns
Data float delay from CLKOUT	t_{FK}	0	50	ns
DSTB ↓ delay from CLKOUT ↓	t_{DKDS}	5	40	ns
$\overline{\text{DSTB}}$ low-level width	t_{DSDSL}	$t_{CYC}\ (n+1) - 10$		ns
$\overline{\text{DSTB}}$ high-level width	t_{DSDSH}	$t_{KKL} + t_{KR} - 10$		ns
CLKOUT to $\overline{\text{IOWR}}$ delay	t_{DKIW}	0	40	ns
CLKOUT to $\overline{\text{IORD}}$ delay	t_{DKIR}	0	40	ns
CLKOUT to $\overline{\text{MRD}}$ delay	t_{DKMR}	0	40	ns
CLKOUT to $\overline{\text{MWR}}$ delay	t_{DKMW}	0	40	ns
CLKOUT ↑ to DSTB ↑	t_{DKDSH}	5	40	ns
Address/status output delay to DSTB ↓	t_{DADSL}	$t_{KKL} + t_{KR} - 15$		ns
Address/status hold time from $\overline{\text{DSTB}}$ ↑	t_{HDSHA}	$t_{KKL} + t_{KR} - 15$		ns
Data output delay from DSTB ↑	t_{DDSHD}	$t_{KKL} + t_{KR} - 15$		ns
Data output delay from address/status output	t_{DAD}	$t_{KKL} + t_{KR} - 15$		ns
Data output delay from CLKOUT ↑	t_{DKD}	5	40	ns
Data setup time to CLKOUT ↓	t_{SDK}	7		ns
Data hold time from CLKOUT ↓	t_{HKD}	10		ns
Data hold time from $\overline{\text{DSTB}}$ ↑	t_{HDSD}	0		ns
Data hold time from change point of address or status	t_{HASD}	0		ns
Dat hold time from R/W ↑	t_{HRWD}	0		ns
$\overline{\text{READY}}$ setup time to CLKOUT ↑	t_{SRYK}	7		ns
$\overline{\text{READY}}$ hold time from CLKOUT ↑	t_{HKRY}	15		ns

Notes:

(1) t_{CYC} = CPU clock period
n = number of wait states

(2) The clock-to-signal delays in the −10 (10 MHz) and −12 (12.5 MHz) parts are 45 ns compared to 40 ns in the −16 (16 MHz) part. For full electrical characteristics of the −10 and −12 parts, contact NEC.

AC Characteristics (cont)

Parameter	Symbol	Min	Max	Unit
Bus Sizing (figures 13, 14)				
$\overline{BS8}$/BS16 setup time to CLKOUT ↑	t_{SBSK}	7		ns
$\overline{BS8}$/BS16 hold time from CLKOUT ↑	t_{HKBS}	15		ns
Bus Hold (figure 17)				
HLDRQ setup time to CLKOUT ↑	t_{SHQK}	7		ns
HLDRQ hold time form CLKOUT ↑	t_{HKHQ}	15		ns
CLKOUT ↑ to HLDAK delay	t_{DKHA}	5	40	ns
Output floating to HLDAK delay	t_{DFHA}	$t_{KKL} + t_{KR} - 15$		ns
Input Setup and Hold (figure 15)				
$\overline{NMI}$, INTP0-INTP7, $\overline{CPBUSY}$ setup time to CLKOUT ↓	t_{SIK}	10		ns
$\overline{NMI}$, INTP0-INTP7, $\overline{CPBUSY}$ hold time from CLKOUT ↓	t_{HKT}	10		ns
Timer/Counter Unit TCU (figures 20-21)				
TCTL0-TCTL2 setup time to CLKOUT ↓	t_{SGK}	50		ns
TCTL0-TCTL2 hold time from CLKOUT ↓	t_{HKG}	100		
TCTL0-TCTL2 low-level width	t_{GGL}	50		ns
TCTL0-TCTL2 high-level width	t_{GGH}	50		ns
TOUT0-TOUT2 output delay from CLKOUT ↓	t_{DKTO}		100	ns
TCLK period	t_{CYTK}	100		ns
TCLK rise time	t_{TKR}		15	ns
TCLK fall time	t_{TKF}		15	ns
TCLK low-level width	t_{TKTKL}	45		ns
TCLK high-level width	t_{TKTKH}	30		ns
TCTL0-TCTL2 setup time to TCLK ↑	t_{SGTK}	50		ns
TCTL0-TCTL2 hold time from TCLK ↑	t_{HTKG}	100		ns
TOUT0-TOUT2 output delay from TCLK ↓	t_{DTKTO}		100	ns
TOUT0-TOUT2 output delay from TCTL ↓	t_{DGTO}		100	ns

Parameter	Symbol	Min	Max	Unit
Serial Control Unit, SCU (figure 22)				
RxD setup time vs SCU internal CLK ↓	t_{SRX}	1		ns
RxD hold time vs SCU internal CLK ↓	t_{HRX}	1		ns
TOUT1 ↑ to TxD delay	t_{DTX}		500	ns
Direct Memory Access, DMA (figures 24-26)				
CLKOUT ↓ to $\overline{MRD}$, IORD ↑ delay	t_{DKRH}	0	40	ns
CLKOUT ↓ to $\overline{MRD}$, IORD ↓ delay	t_{DKRL}	0	40	ns
CLKOUT ↑ to DMAAK0-DMAAK3 delay	t_{DKHDA}	0	40	ns
$\overline{IORD}$ ↓, $\overline{IOWR}$ ↓ delay from DMAAK0-DMAAK3 ↓	t_{DDARW}	$t_{KKH} - 30$		ns
DMAAK0-DMAAK3 ↑ delay from $\overline{IORD}$ ↑	t_{DRHDAH}	$t_{KKH} - 30$		ns
CLKOUT to control 1 delay	t_{DKCT1}	0	40	ns
$\overline{IORD}$ ↑ delay to $\overline{IOWR}$ ↑	t_{DWHRH}	5	40	
$\overline{TC}$ output delay from CLKOUT ↑	t_{DKTCL}	0	40	ns
$\overline{TC}$ off output delay from CLKOUT ↑	t_{DKTCF}	0	40	ns
$\overline{TC}$ pullup delay from CLKOUT ↑	t_{DKTCH}	0	40	ns
$\overline{TC}$ low-level width	t_{TCTCL}	$t_{CYC} - 15$		ns
END setup time to CLKOUT ↑	t_{SEDK}	35		ns
END low-level width	t_{EDEDL}	100		ns
$\overline{IORD}$, $\overline{MRD}$ low-level width	t_{RR}	$2t_{CYC} - 40$		
$\overline{IOWR}$, $\overline{MWR}$ low-level width (Expanded write)	t_{WW1}	$2t_{CYC} - 40$		
$\overline{IOWR}$, $\overline{MWR}$ low-level width (Normal write)	t_{WW2}	$t_{CYC} - 40$		
DMARQ0-DMARQ3 setup time to CLKOUT ↑	t_{SDQK}	15		ns
CLKOUT ↓ to $\overline{DMAAK0}$-$\overline{DMAAK3}$ delay	t_{DKLDA}	0	45	ns
Interrupt Control Unit, ICU (figure 27)				
INTP0-INTP7 low-level width	t_{IPIPL}	100		ns

T-49-17-15

Figure 1. Clock Timing

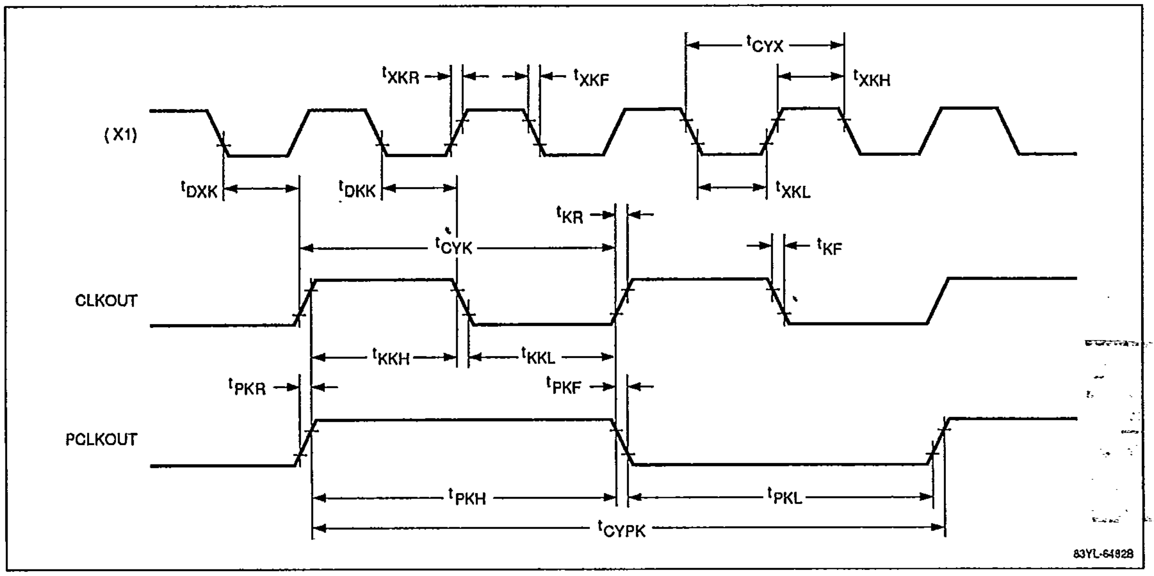


Figure 2. Reset Timing

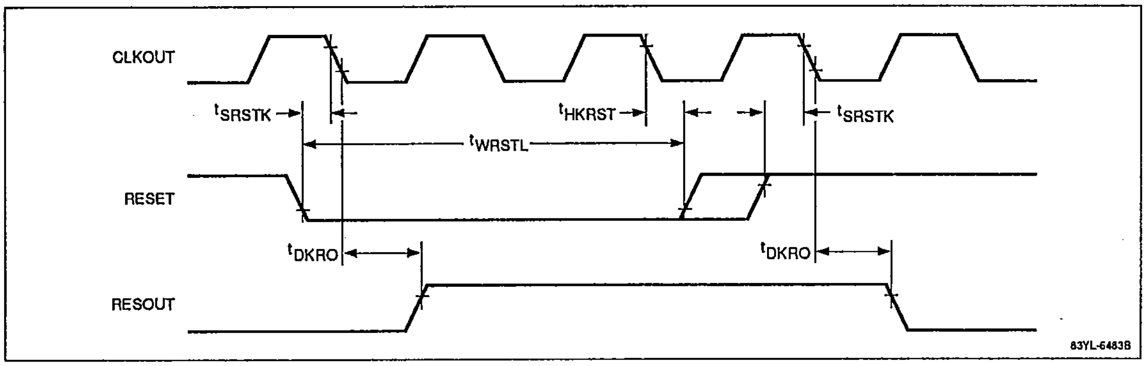


Figure 3. Basic Write (0 Wait)

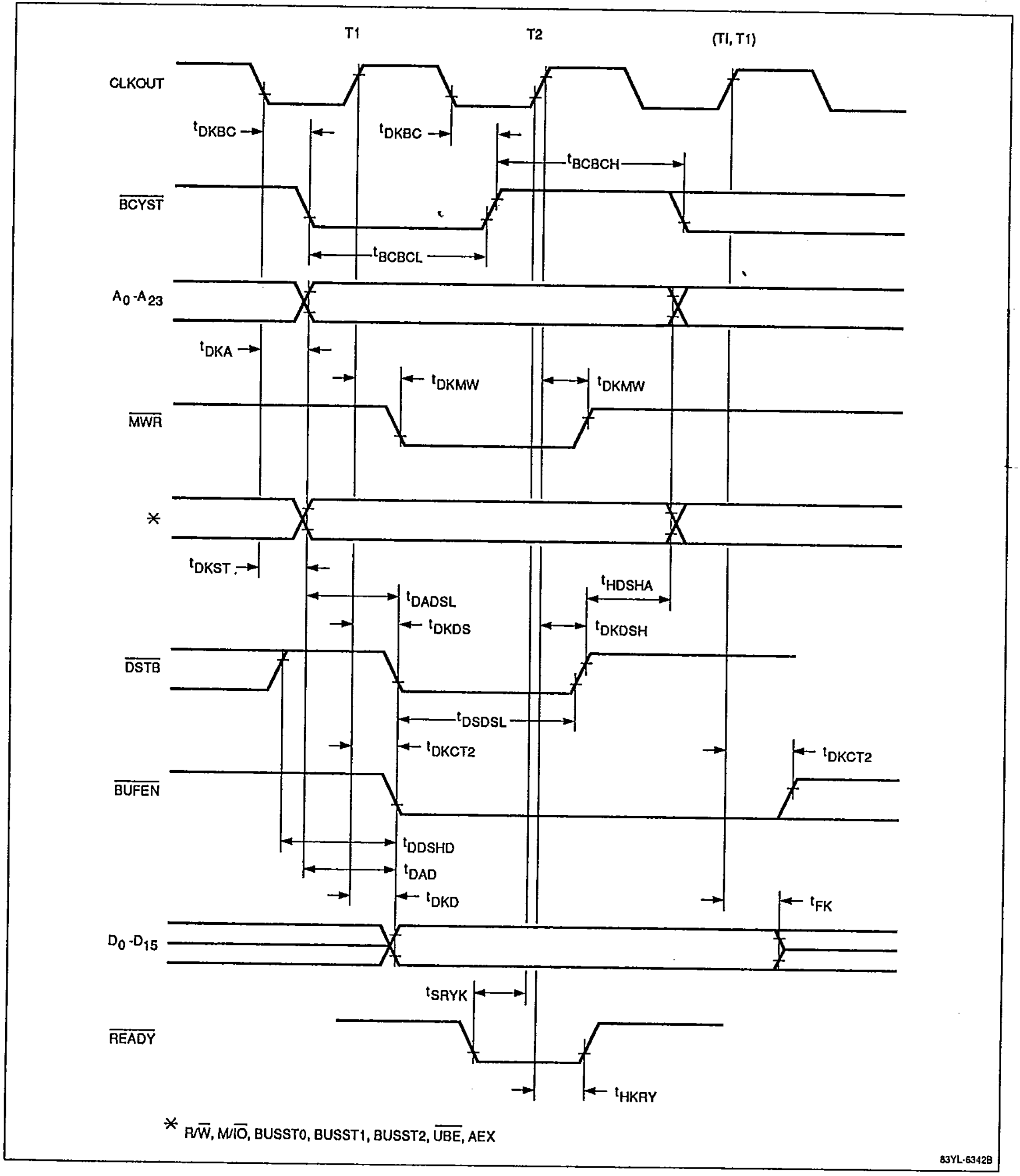
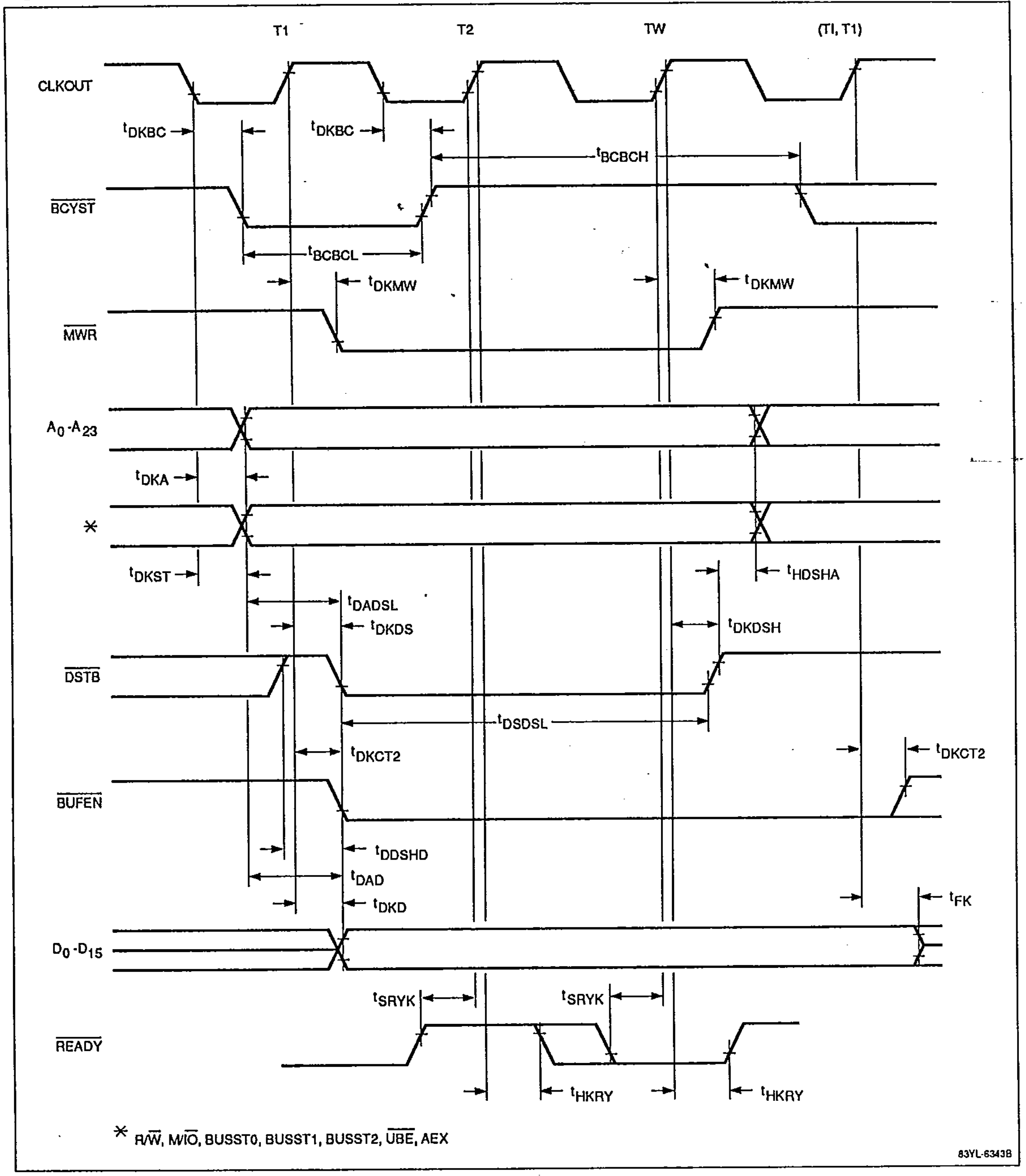
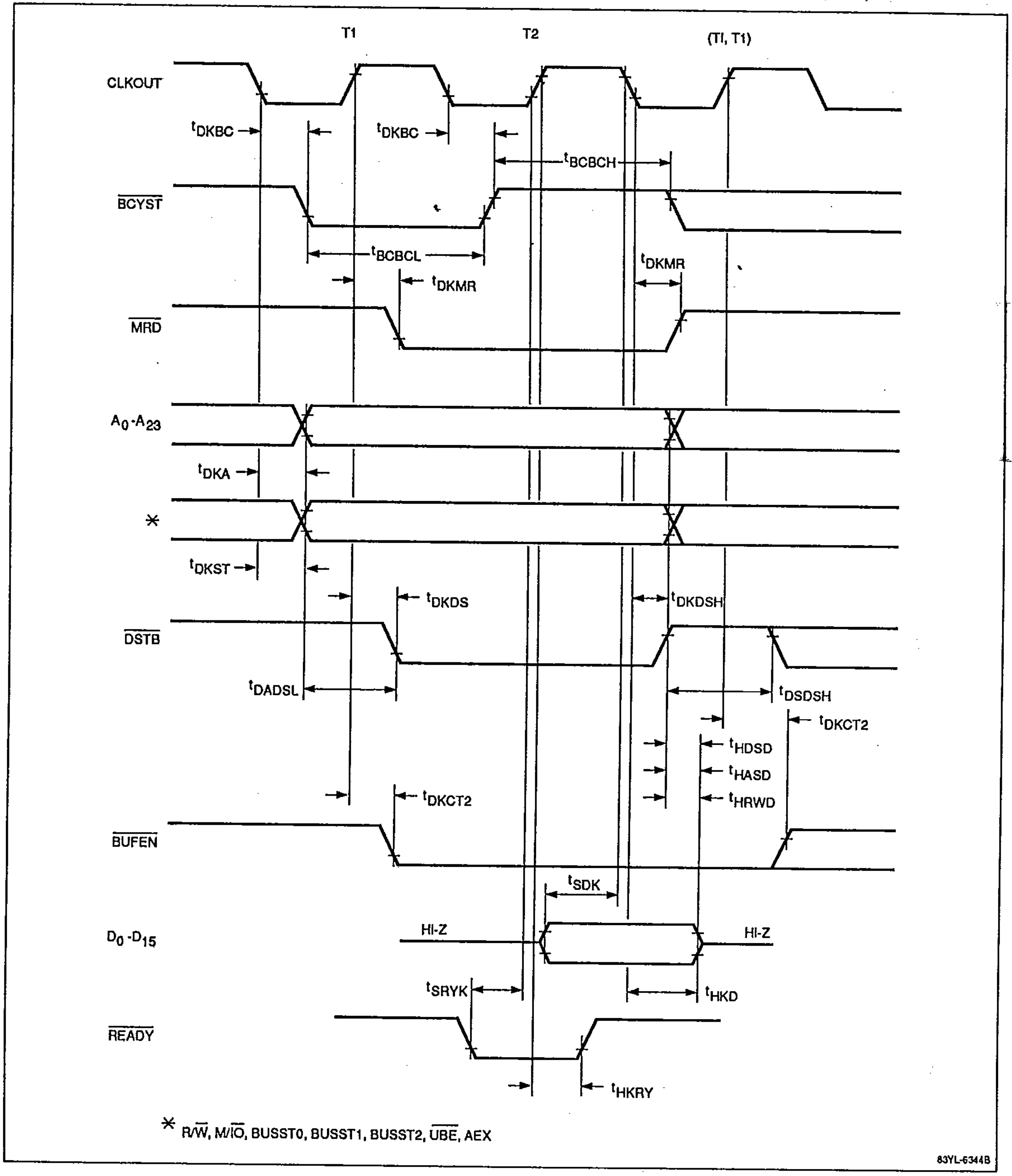


Figure 4. Basic Write (1 Wait)



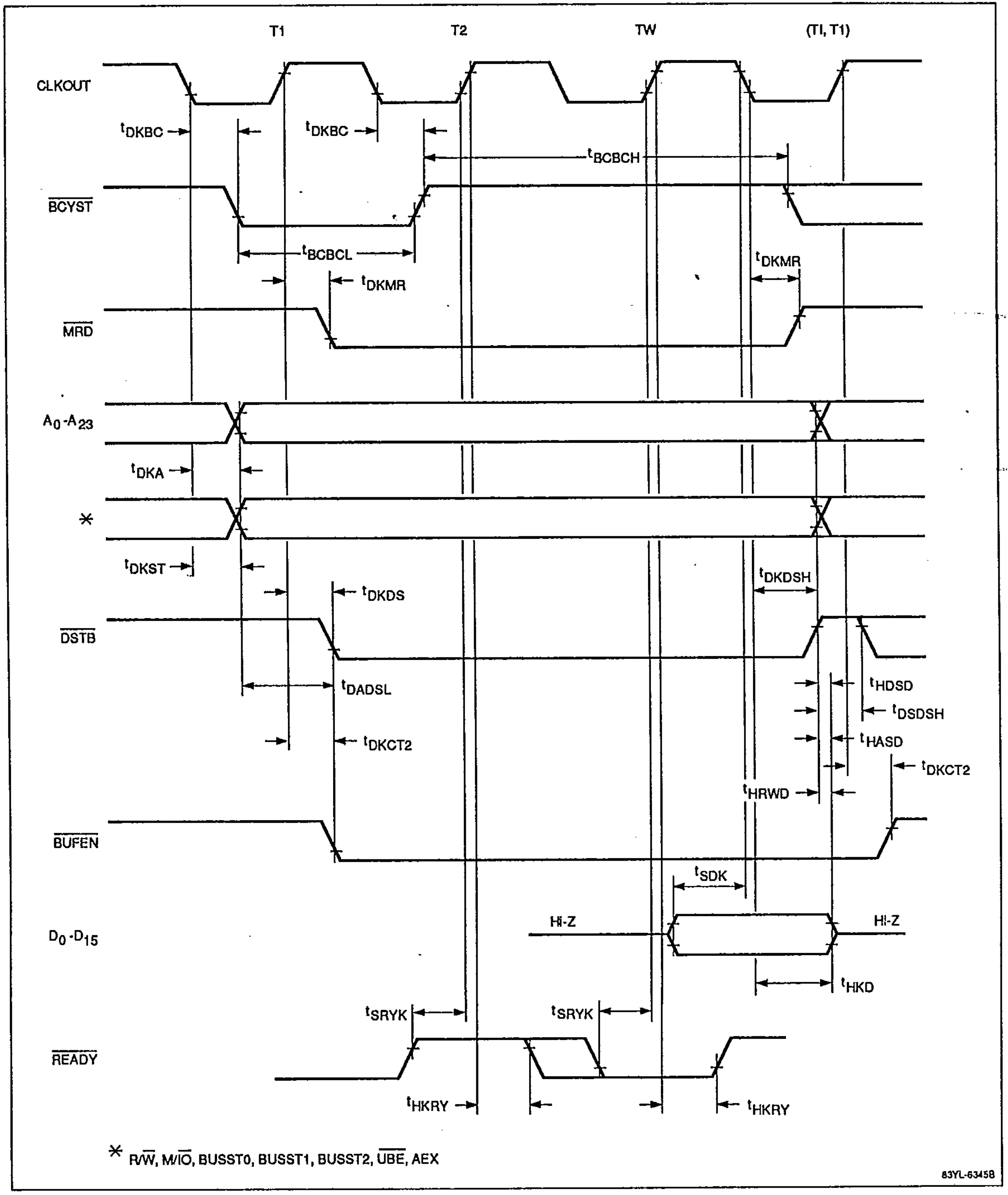
* R/$\overline{W}$, M/$\overline{IO}$, BUSST0, BUSST1, BUSST2, $\overline{UBE}$, AEX

Figure 5. Basic Read (0 Wait)



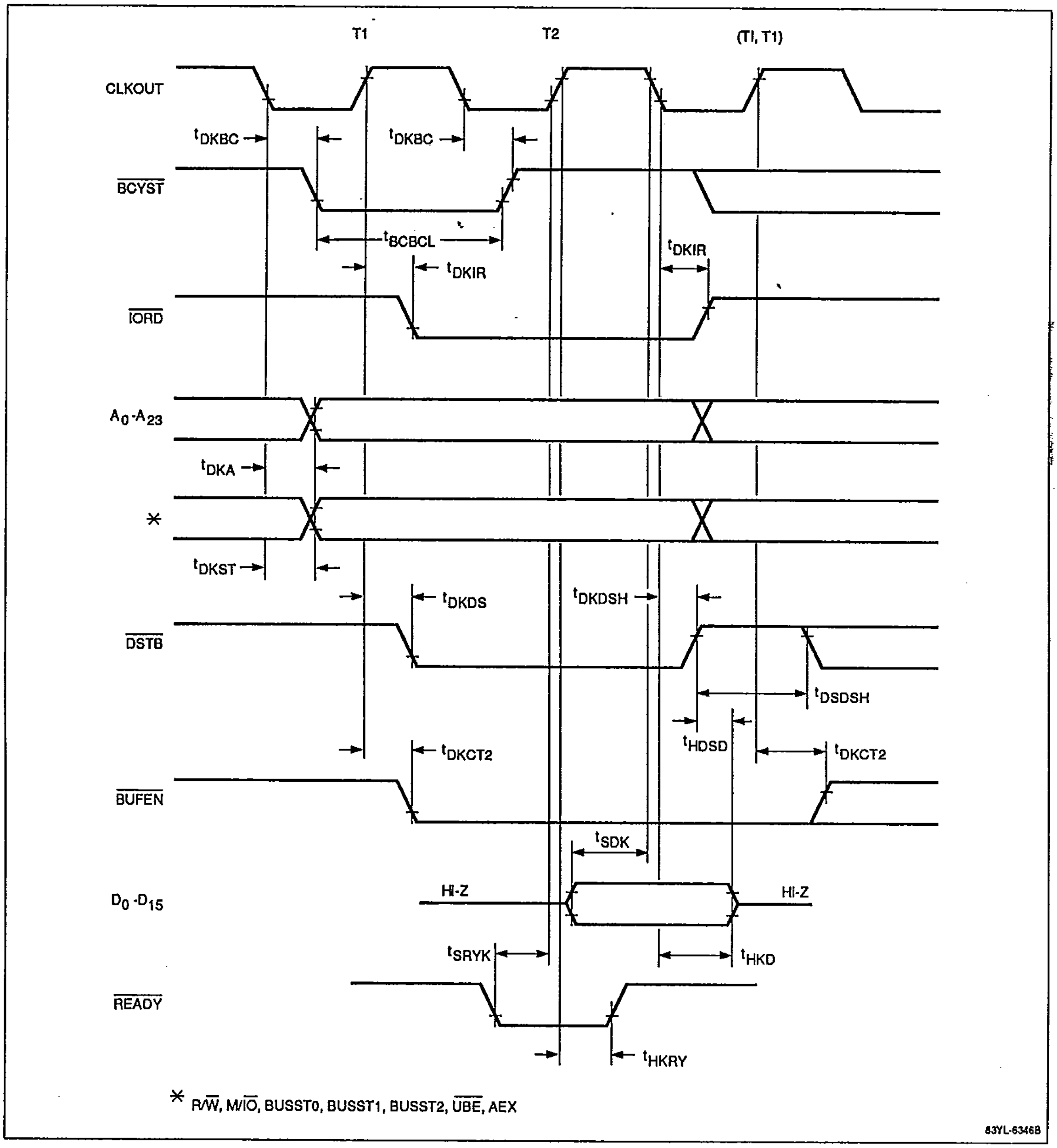
* R/$\overline{\text{W}}$, M/$\overline{\text{IO}}$, BUSST0, BUSST1, BUSST2, $\overline{\text{UBE}}$, AEX

Figure 6. Basic Read (1 Wait)



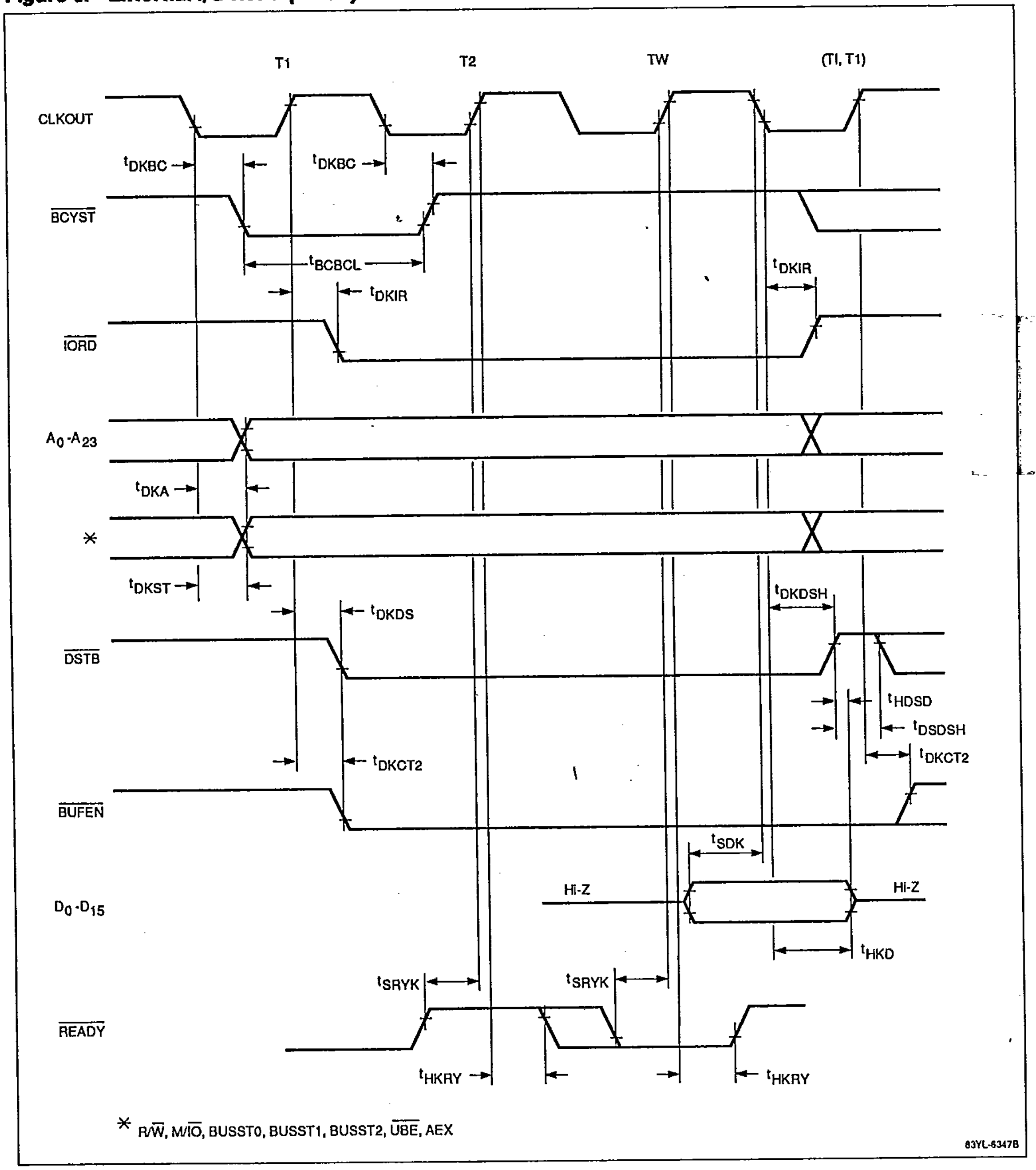
* R/W̅, M/I̅O̅, BUSST0, BUSST1, BUSST2, U̅B̅E̅, AEX

Figure 7. External I/O Read (0 Wait)



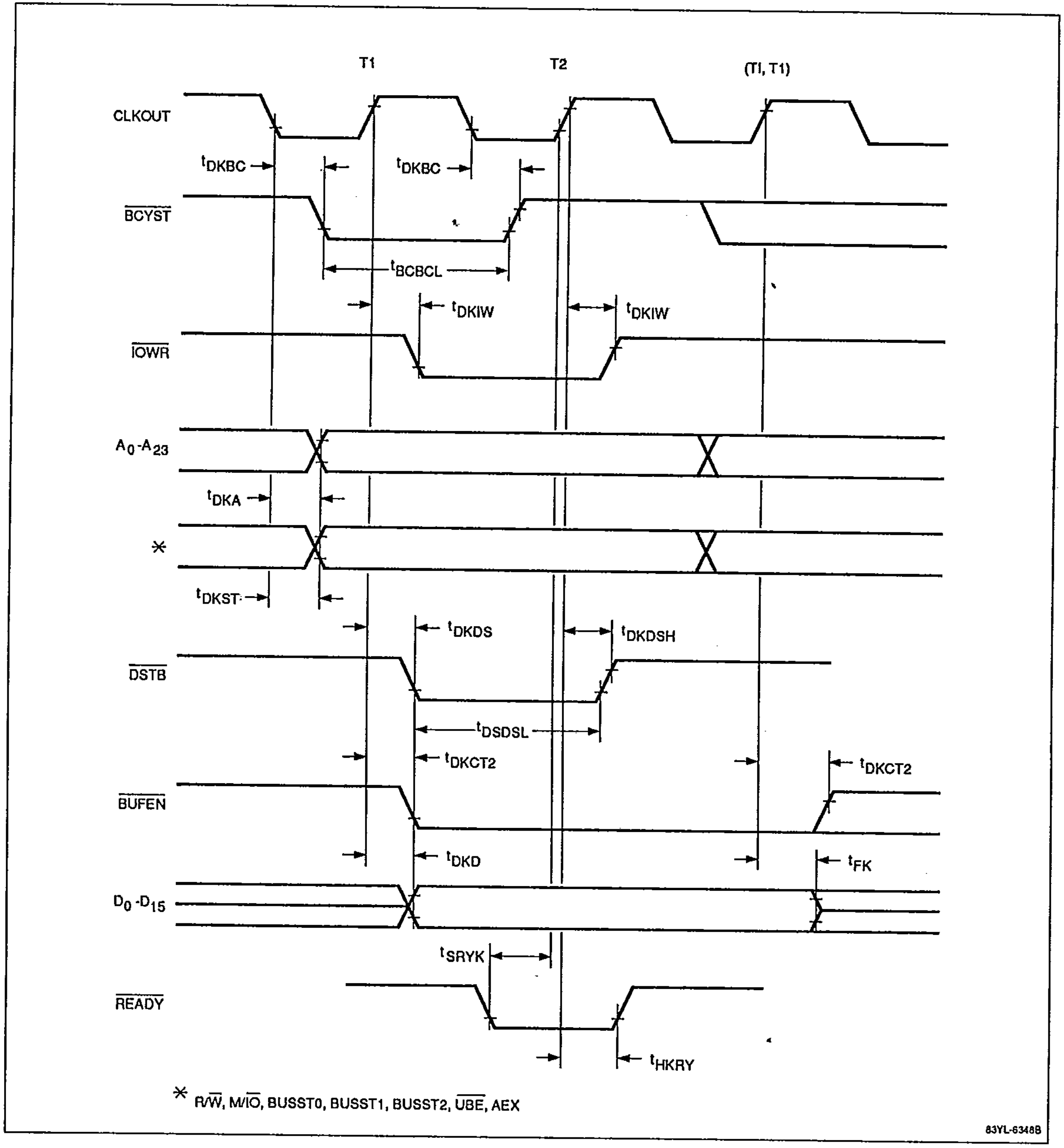
* R/$\overline{W}$, M/$\overline{IO}$, BUSST0, BUSST1, BUSST2, $\overline{UBE}$, AEX

Figure 8. External I/O Read (1 Wait)



* R/W, M/IO, BUSST0, BUSST1, BUSST2, UBE, AEX

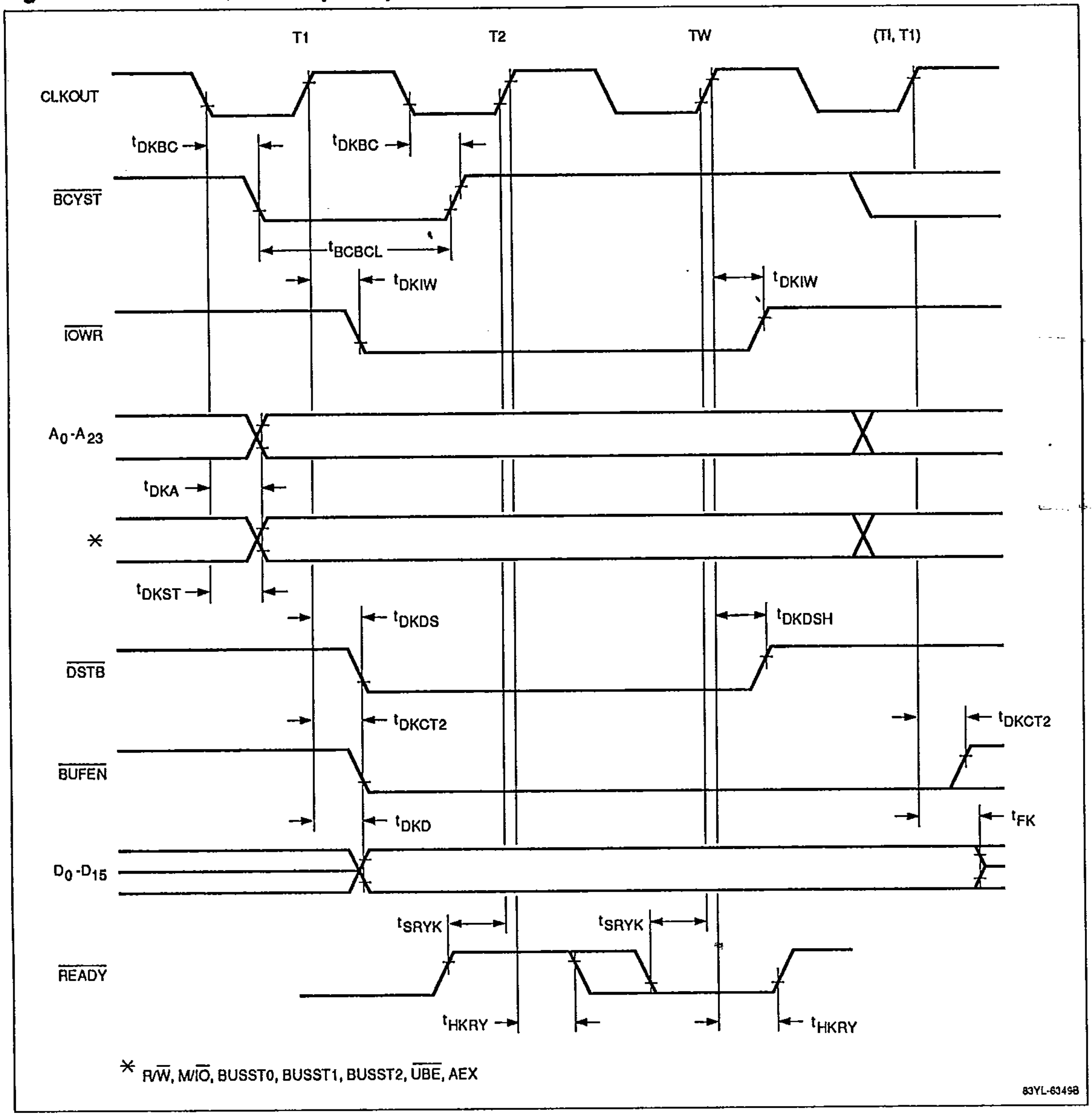
Figure 9. External I/O Write (0 Wait)



* R/$\overline{\text{W}}$, M/$\overline{\text{IO}}$, BUSST0, BUSST1, BUSST2, $\overline{\text{UBE}}$, AEX

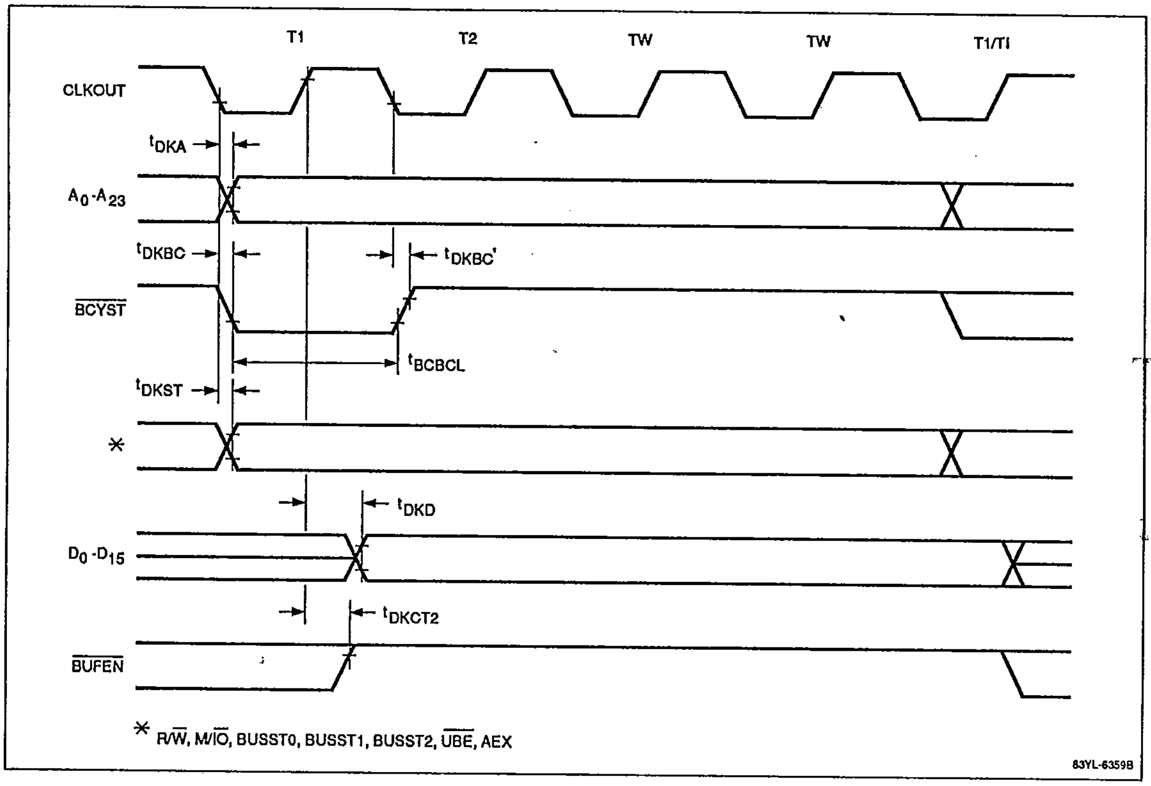
T-49-17-15

Figure 10. External I/O Write (1 Wait)



* R/W, M/IO, BUSST0, BUSST1, BUSST2, UBE, AEX

Figure 11. Internal I/O Read

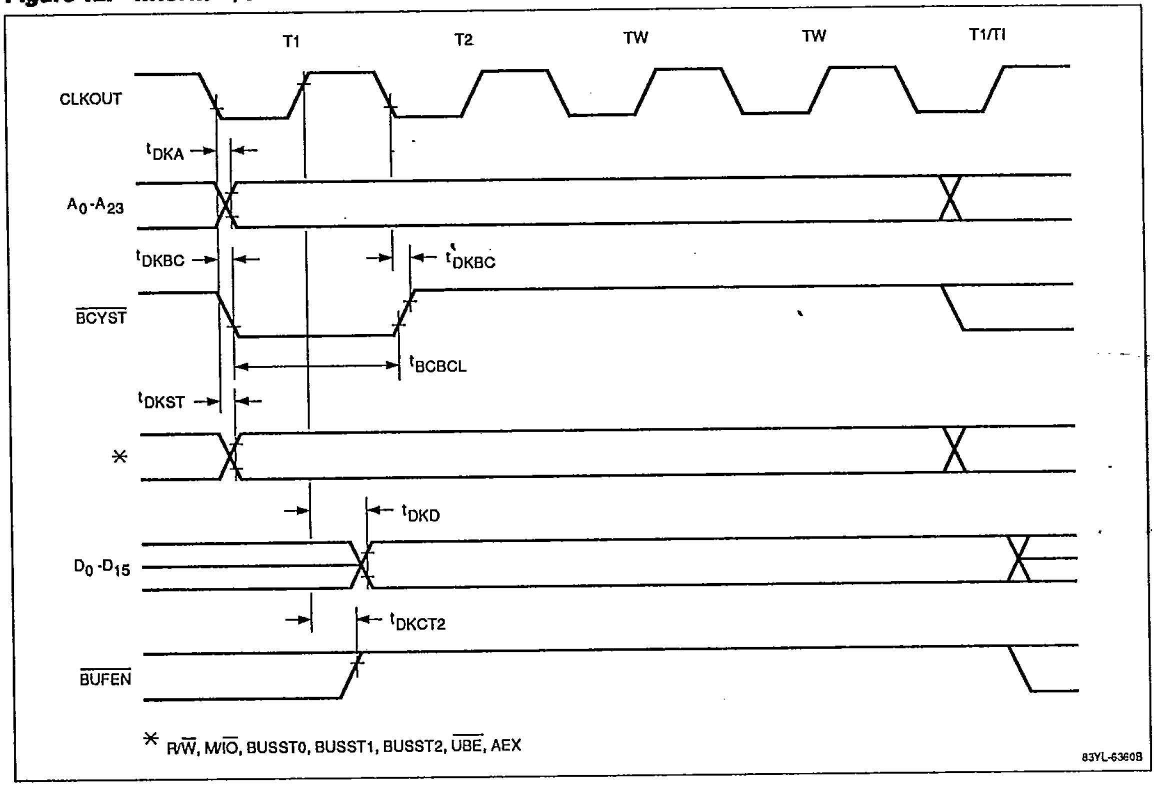


* R/$\overline{W}$, M/$\overline{IO}$, BUSST0, BUSST1, BUSST2, $\overline{UBE}$, AEX

83YL-6359B

T-49-17-15

Figure 12. Internal I/O Write



* R/W, M/IO, BUSST0, BUSST1, BUSST2, UBE, AEX

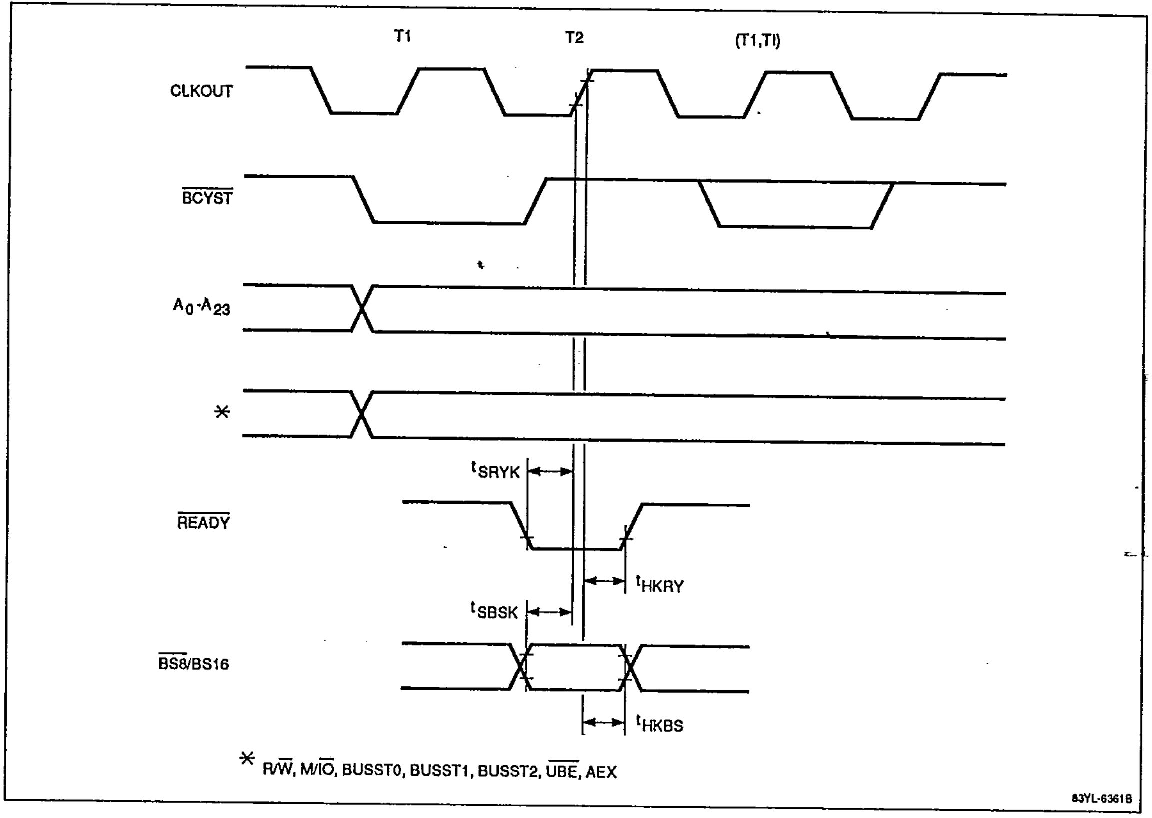


Figure 13. Bus Sizing (0 Wait)

Figure 14. Bus Sizing (1 Wait)

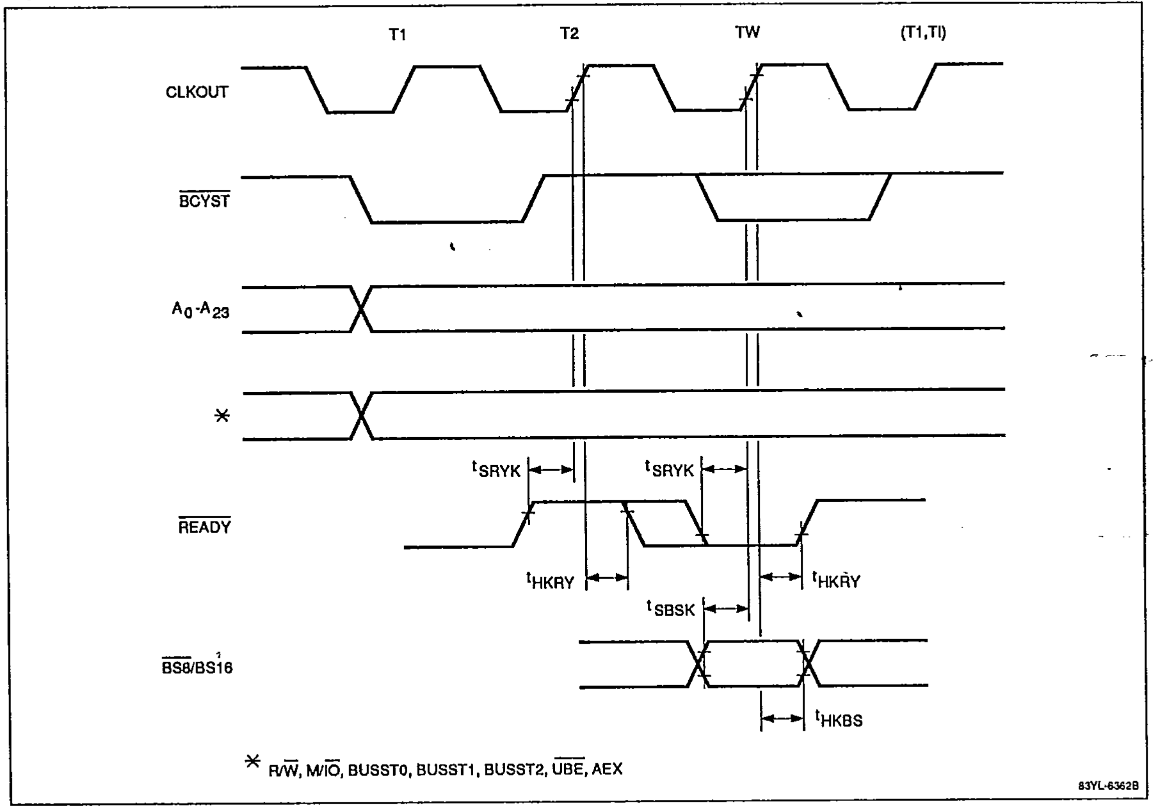


Figure 15. Input Setup/Hold

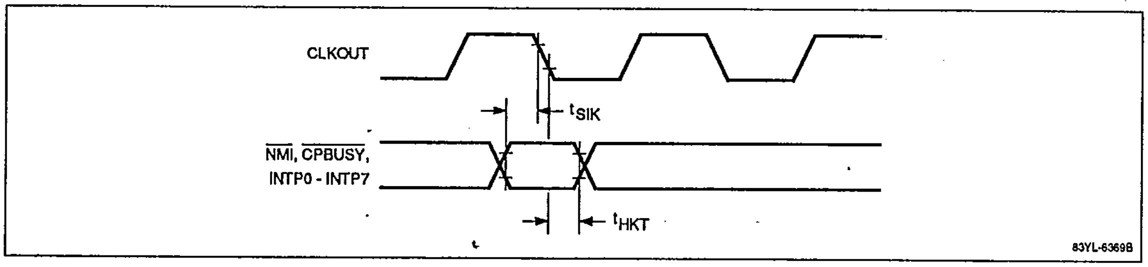
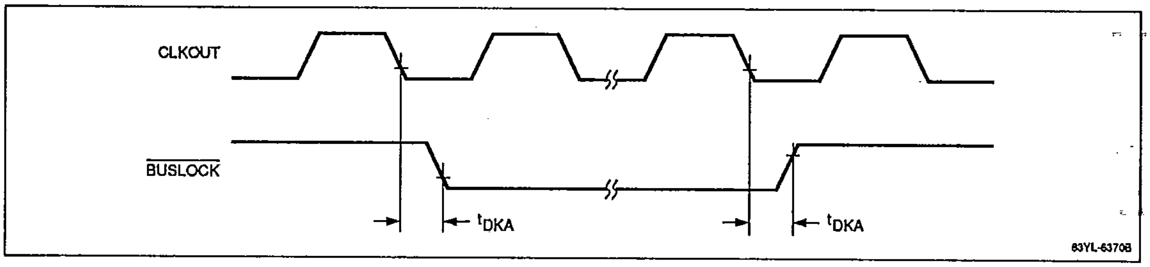
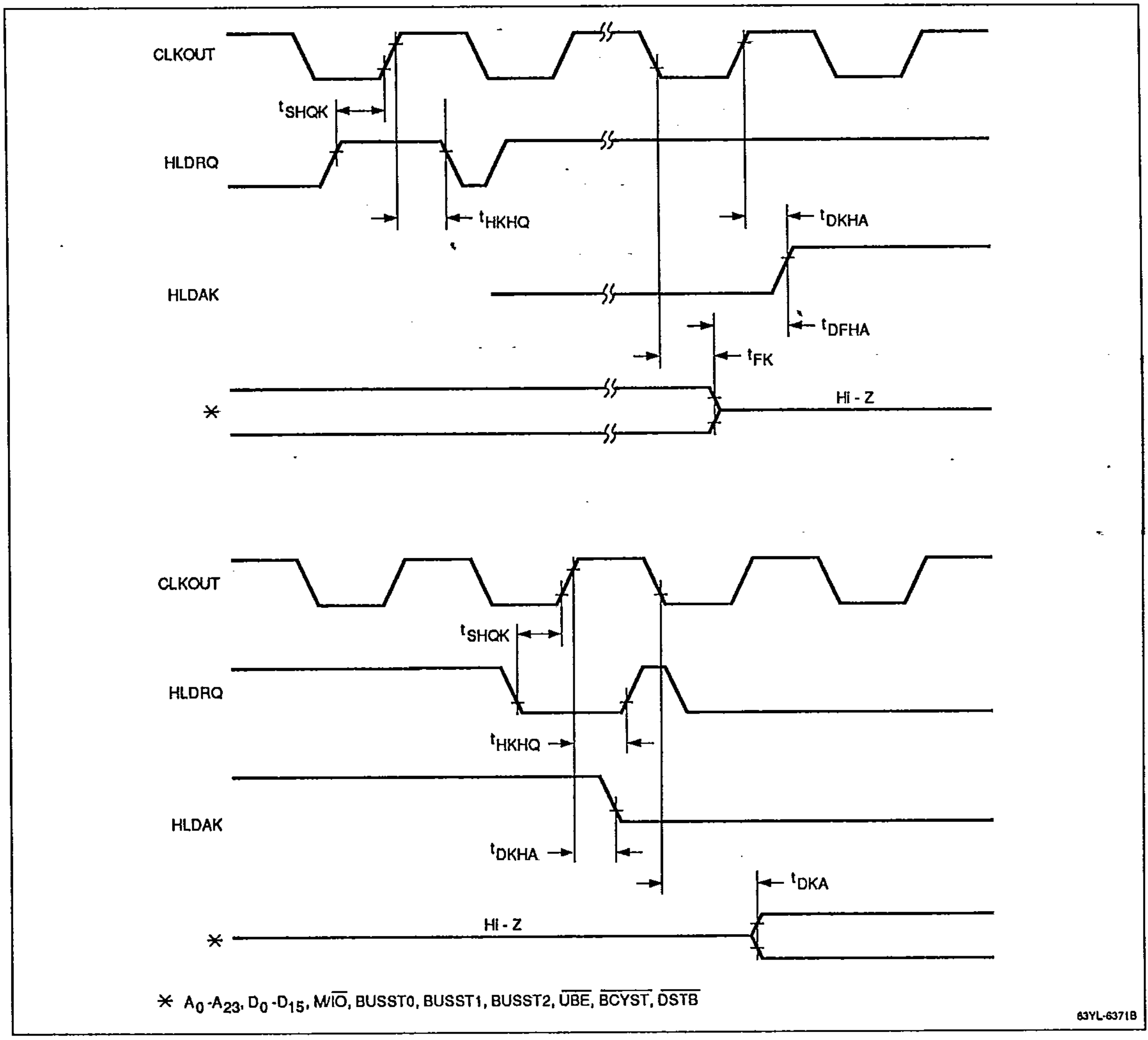


Figure 16. Bus Lock



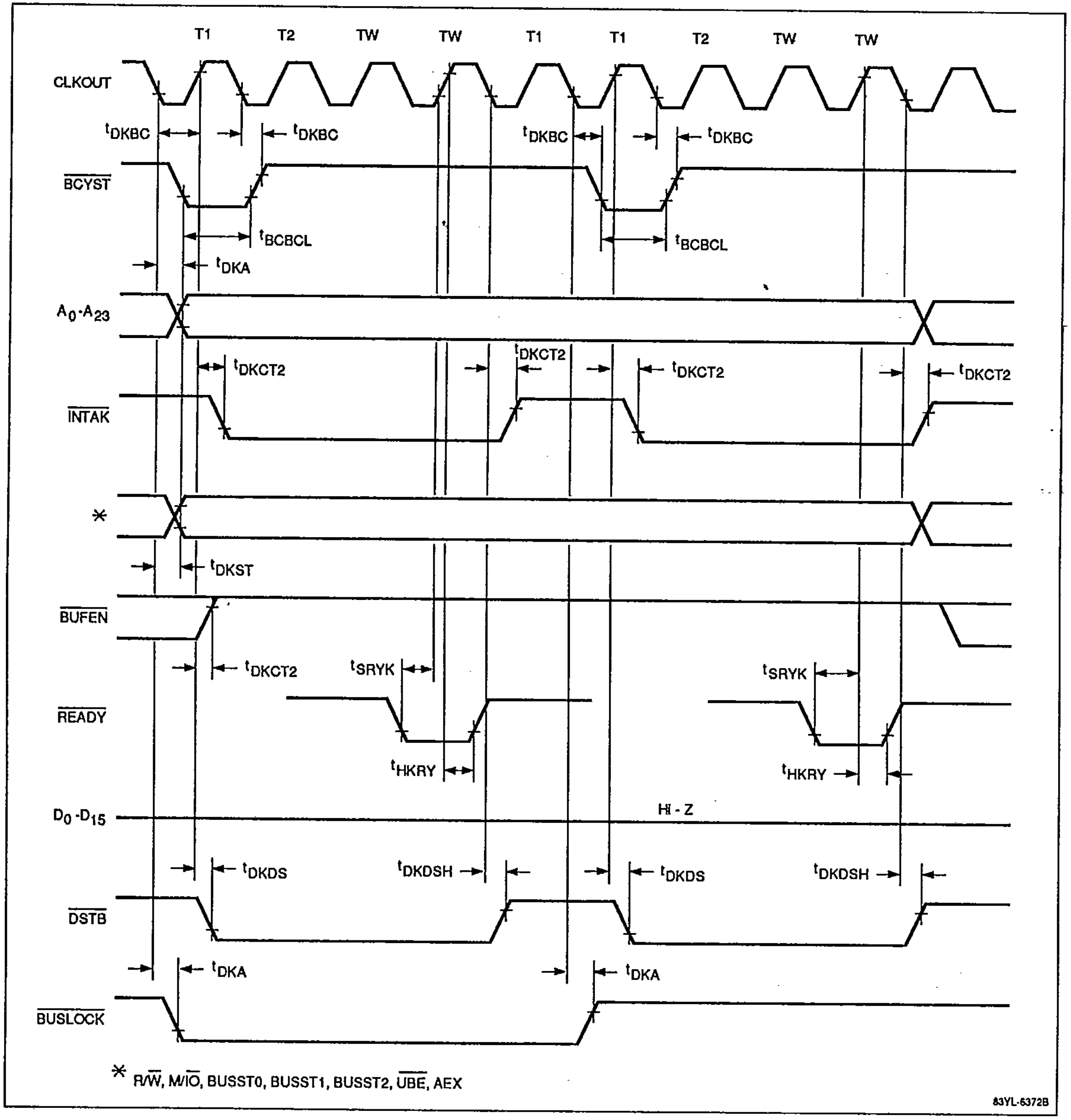
T-49-17-15

Figure 17. Bus Hold



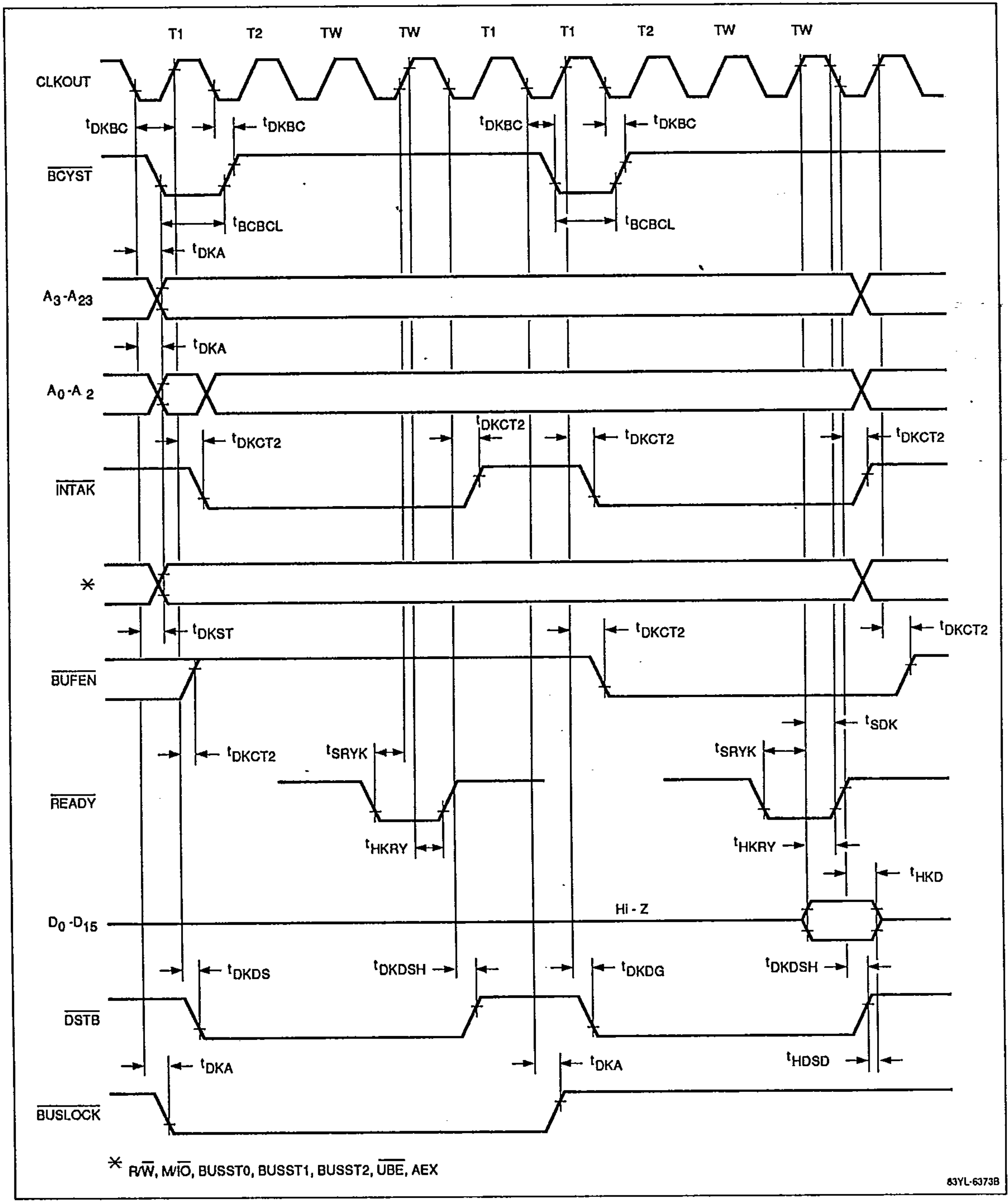
* A_0-A_{23}, D_0-D_{15}, M/$\overline{\text{IO}}$, BUSST0, BUSST1, BUSST2, $\overline{\text{UBE}}$, $\overline{\text{BCYST}}$, $\overline{\text{DSTB}}$

Figure 18. Interrupt Acknowledge (Single Mode)



* R/W̄, M/IŌ, BUSST0, BUSST1, BUSST2, UB̄Ē, AEX

Figure 19. Interrupt Acknowledge (Cascade Mode)



* R/W, M/IO, BUSST0, BUSST1, BUSST2, UBE, AEX

Figure 20. Timer Control Unit (TCU)

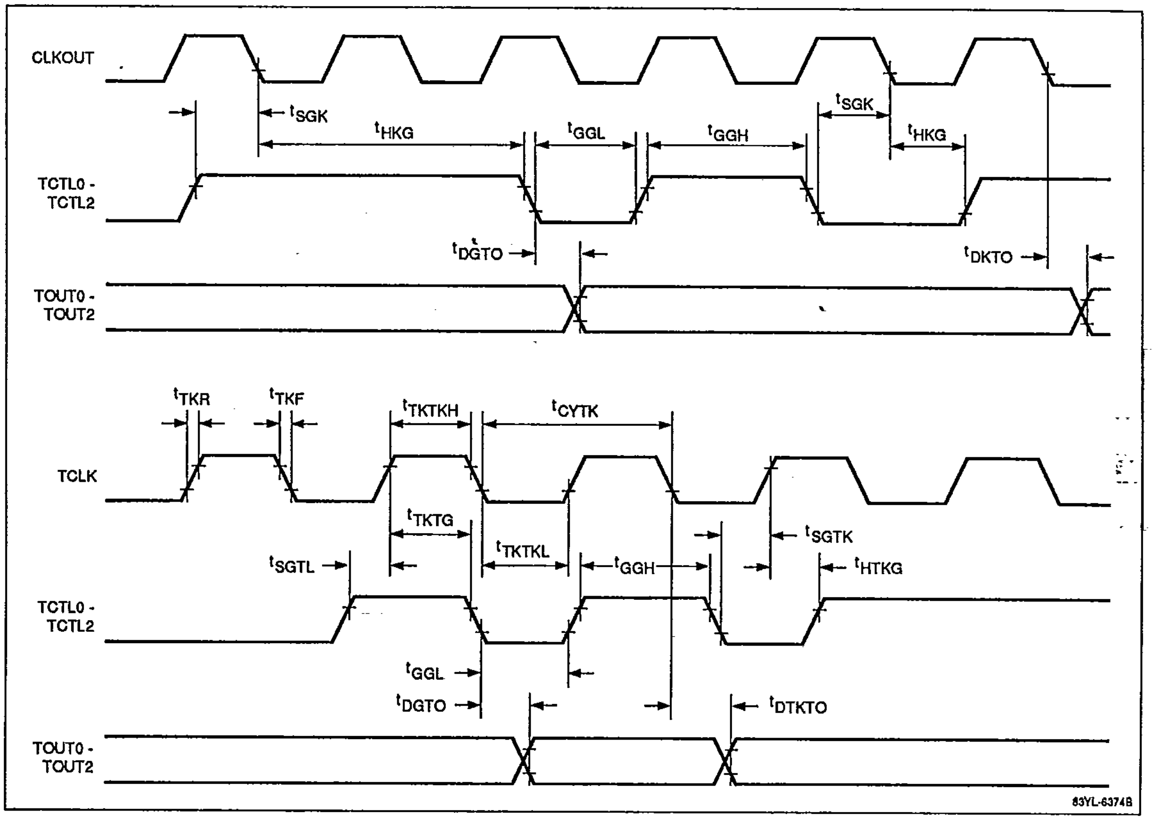


Figure 21. Serial Control Unit (SCU)

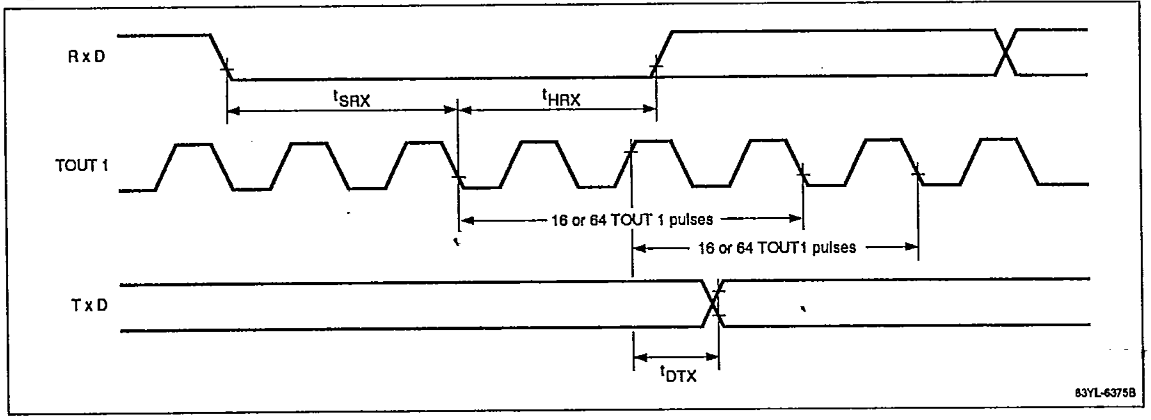


Figure 22. Refresh Timing

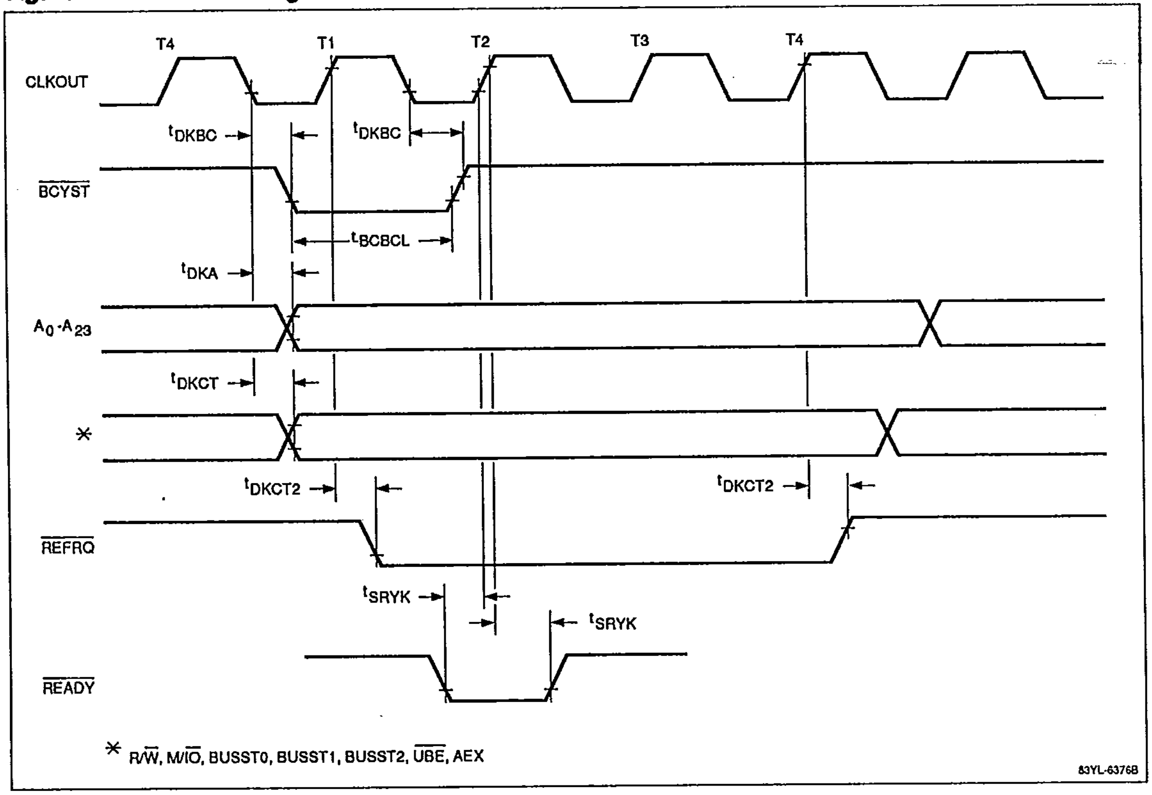


Figure 23. DMA Timing 1

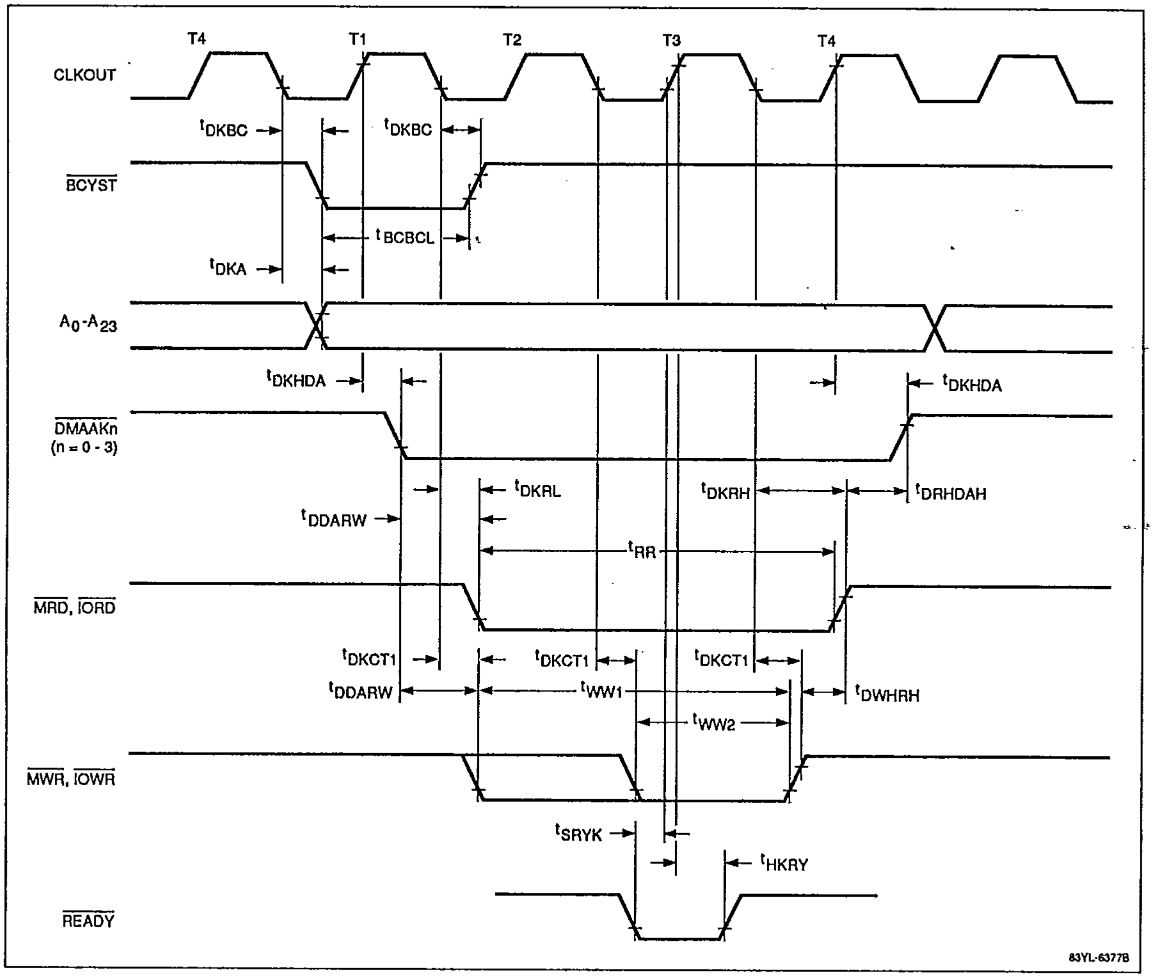
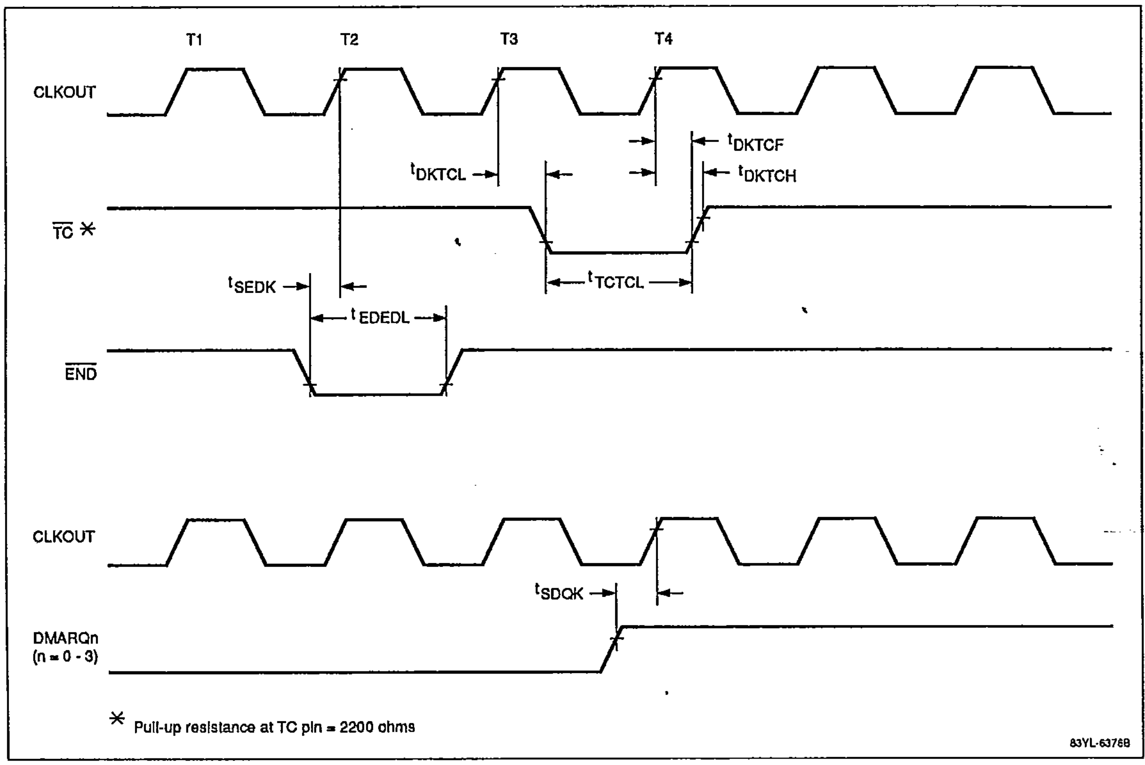


Figure 24. DMA Timing 2



* Pull-up resistance at TC pin = 2200 ohms

Figure 25. DMA Timing 3; Cascade Mode (Normal Operation)

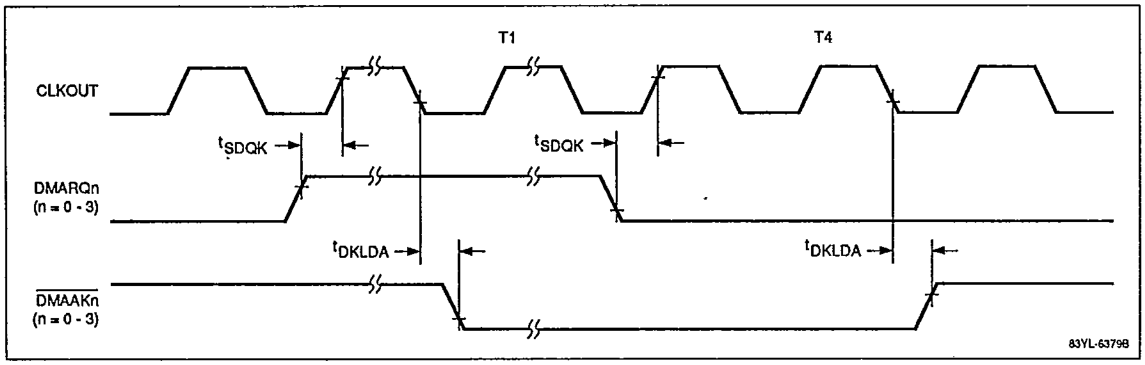


Figure 26. DMA Timing 4; Refresh Cycles To Be Inserted

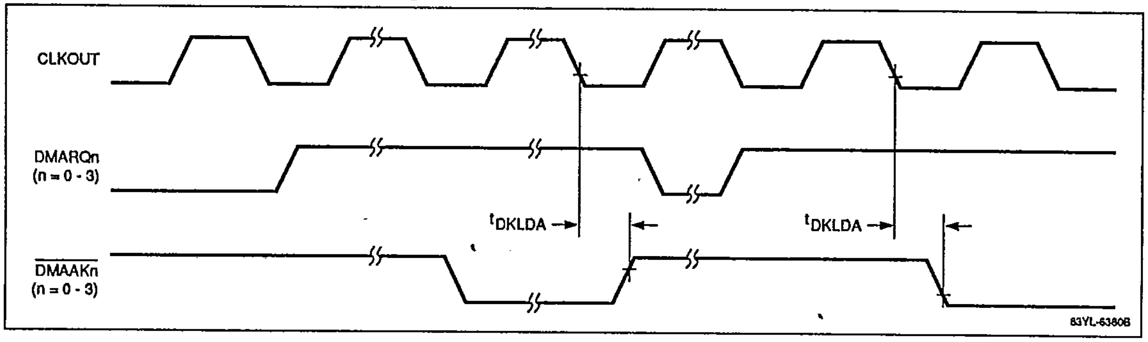


Figure 27. ICU Timing

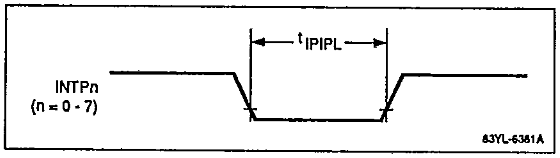


Figure 28. Memory Write for Coprocessor (0 Wait)

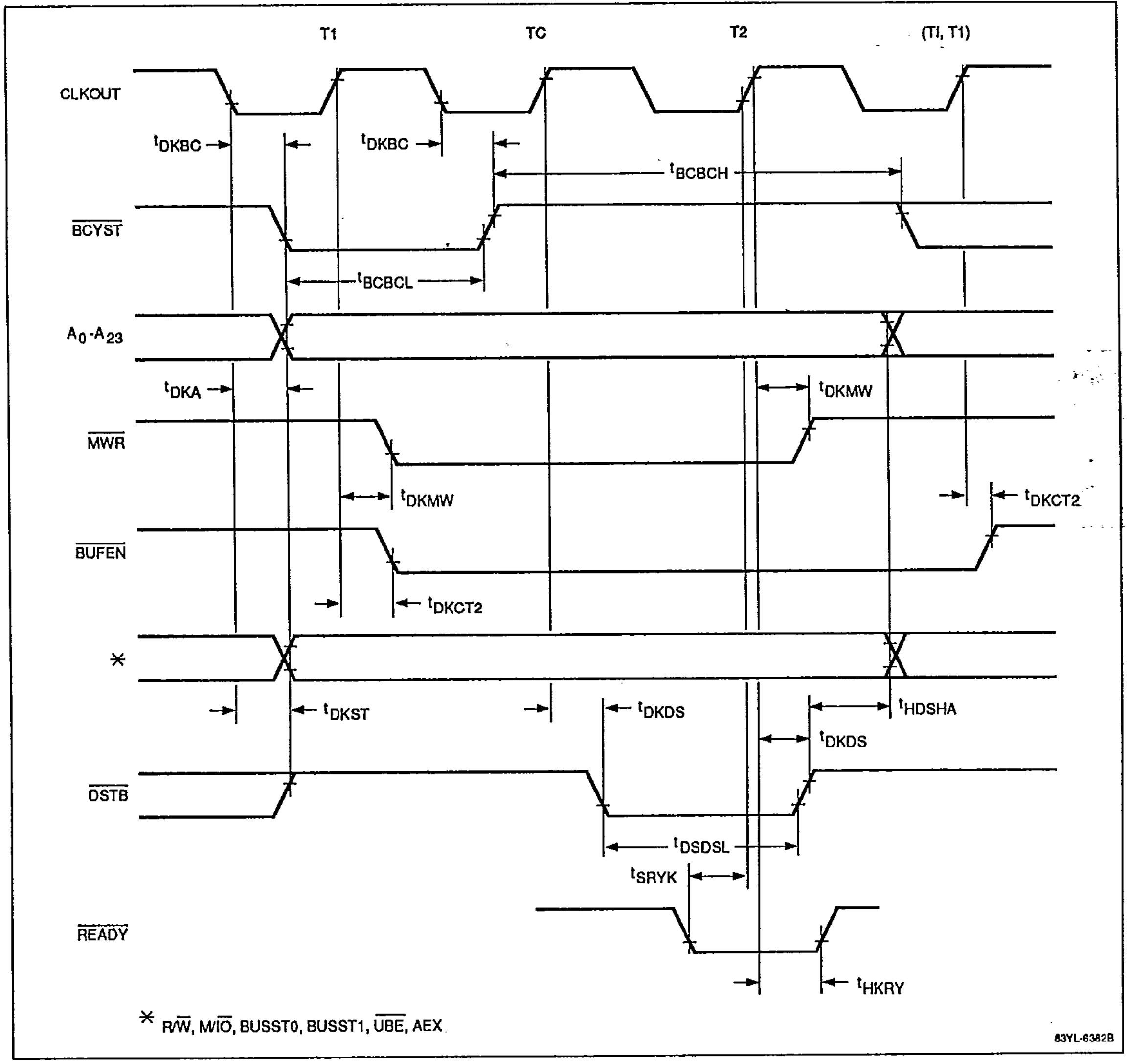
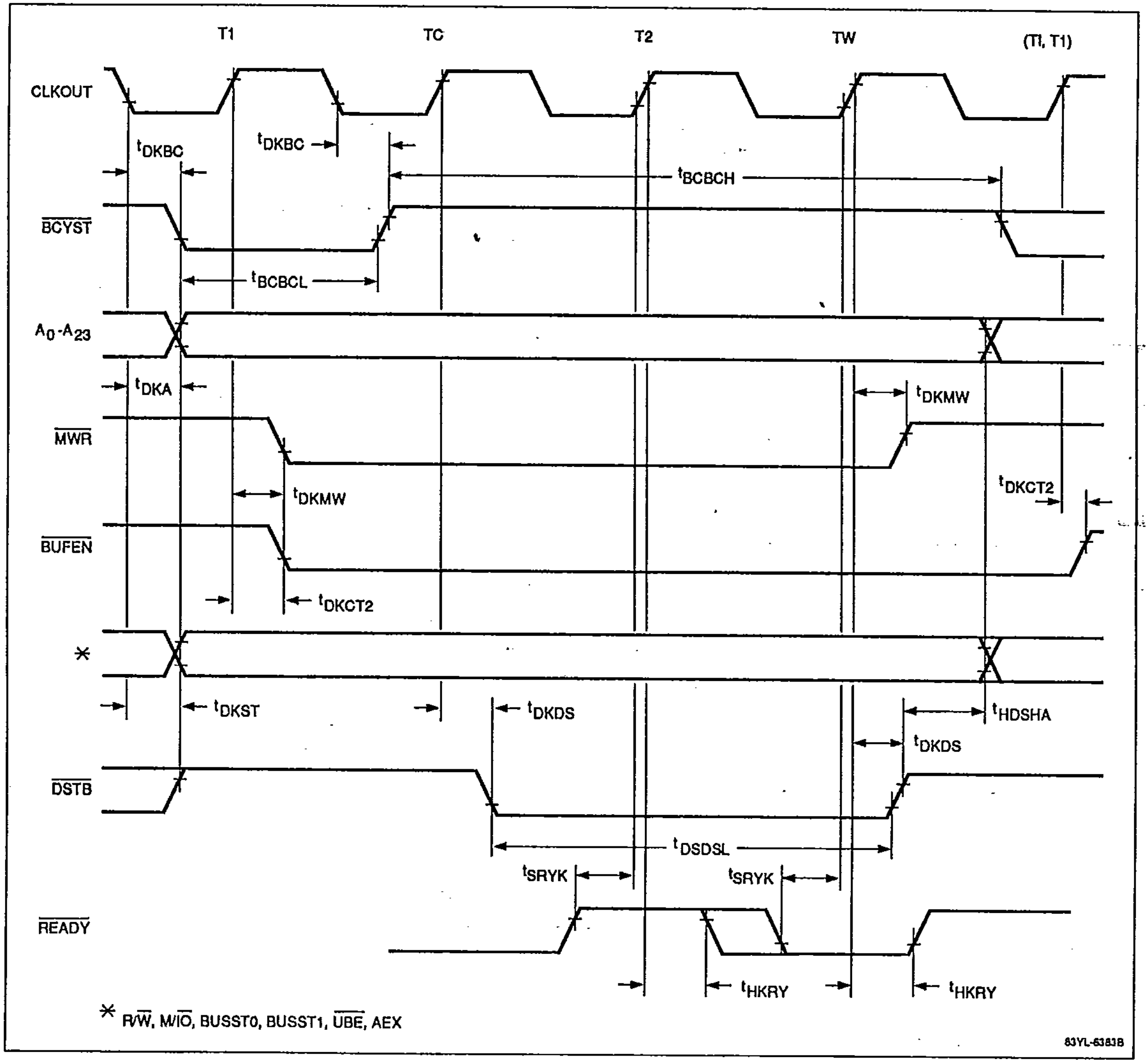


Figure 29. Memory Write for Coprocessor (1 Wait)



T-49-17-15

Figure 30. Memory Read for Coprocessor (0 Wait)

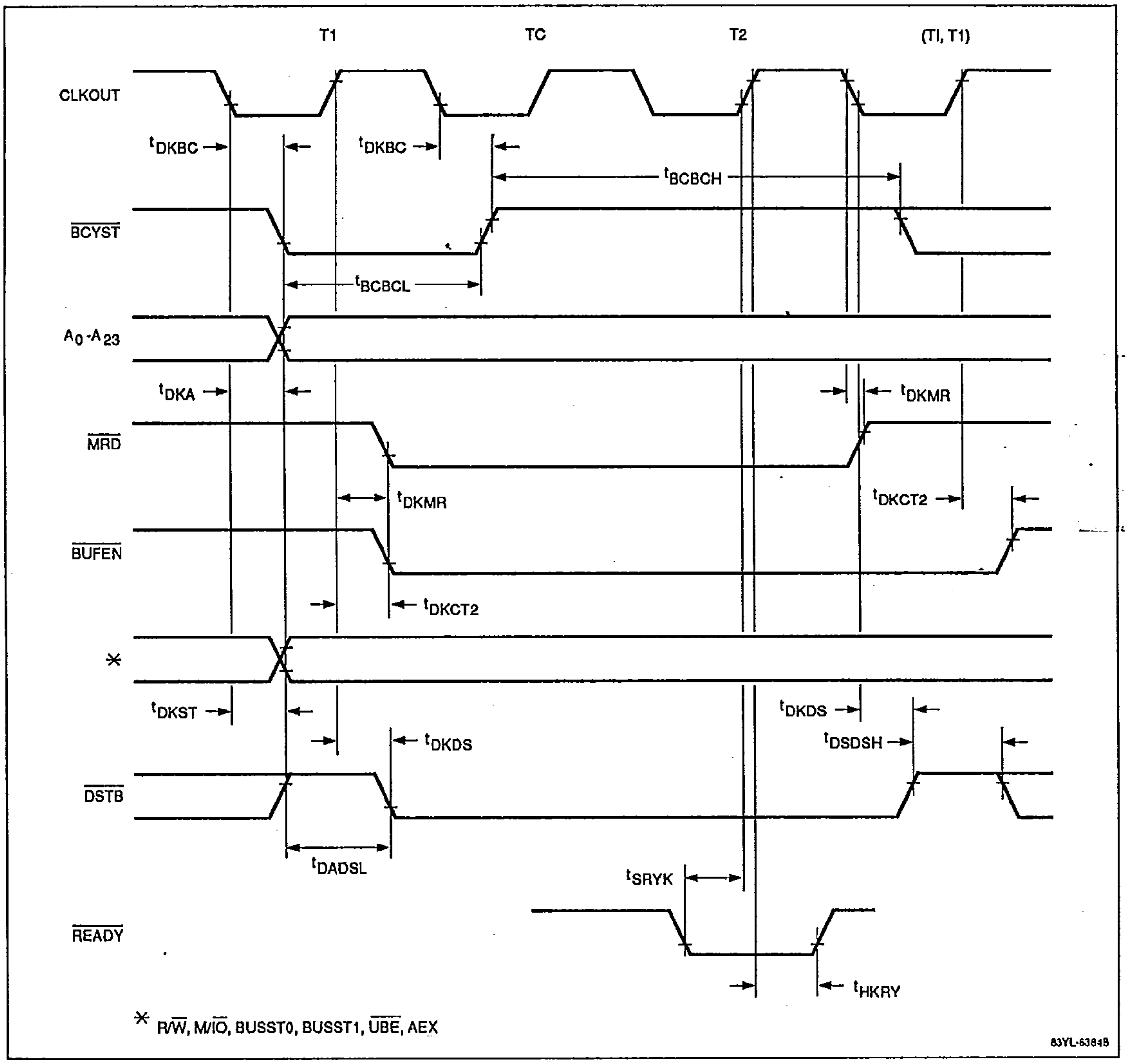
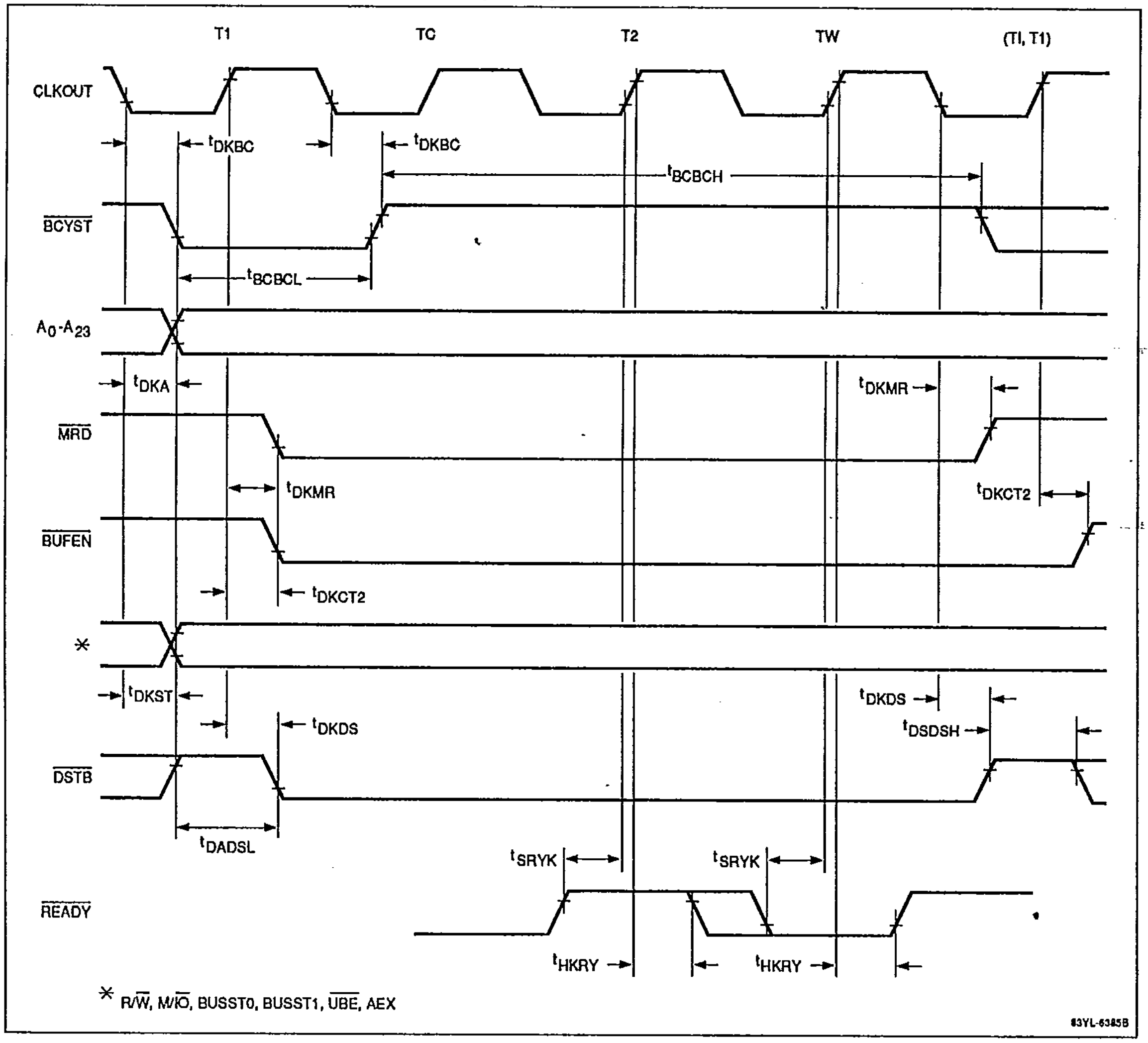


Figure 31. Memory Read for Coprocessor (1 Wait)



* R/$\overline{W}$, M/$\overline{IO}$, BUSST0, BUSST1, $\overline{UBE}$, AEX

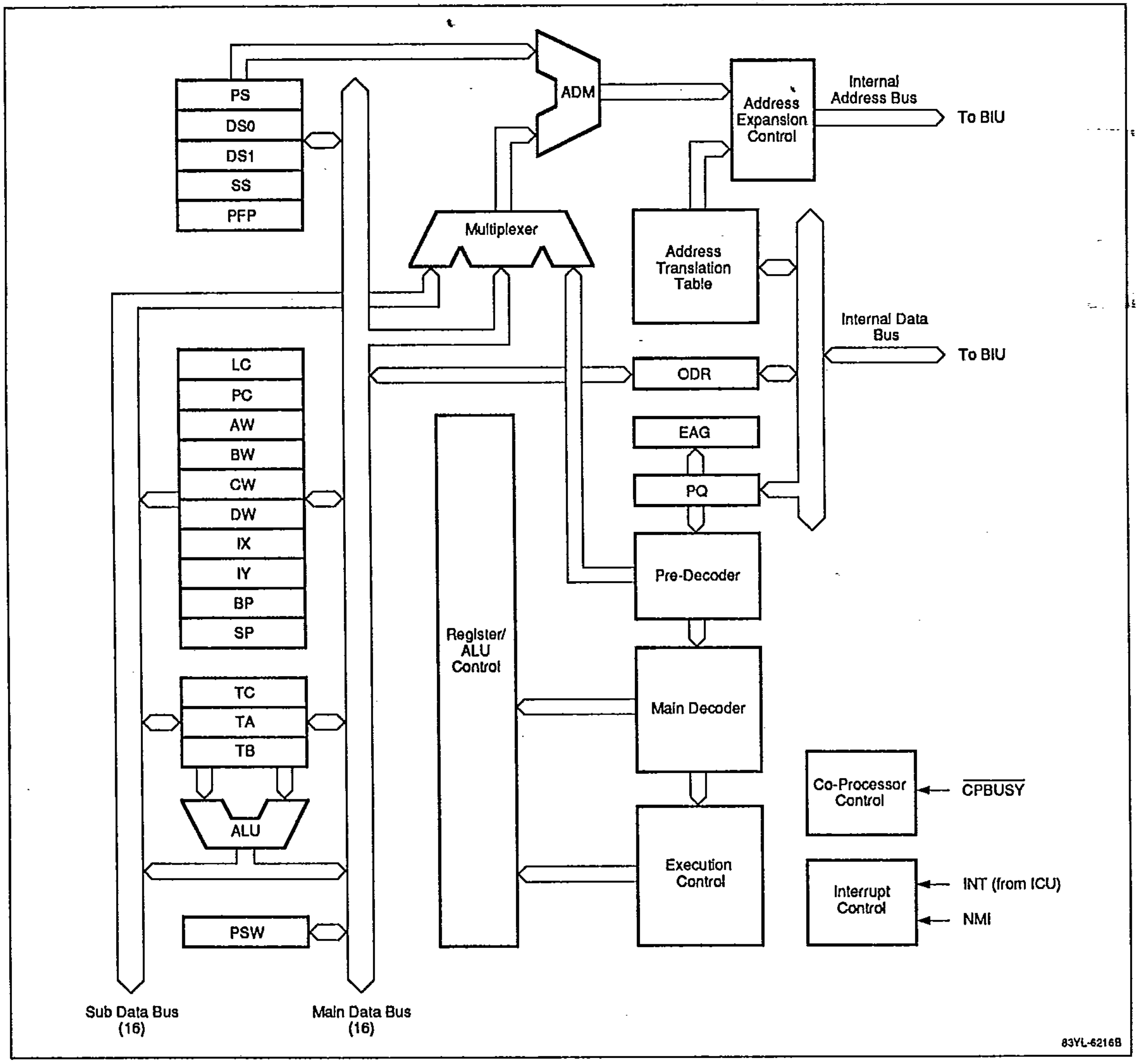
T-49-17-15

FUNCTIONAL OPERATION

The μPD70236 is described under these major headings.

- Central Processing Unit
- Clock Generator
- Bus Operation
- System Control I/O
- Wait Control Unit
- Refresh Control Unit
- Timer/Counter Unit
- Serial Control Unit
- Interrupt Control Unit
- DMA Control Unit
- Power Conservation

Figure 32. CPU Block Diagram



CENTRAL PROCESSING UNIT (CPU)

Architecture

A unique hardware architecture feature of the CPU is that it contains no microcode. Instruction decode and data path control are implemented using logic and small independent state machines. This greatly enhances instruction execution speed. The V53 is four times faster than the V30.

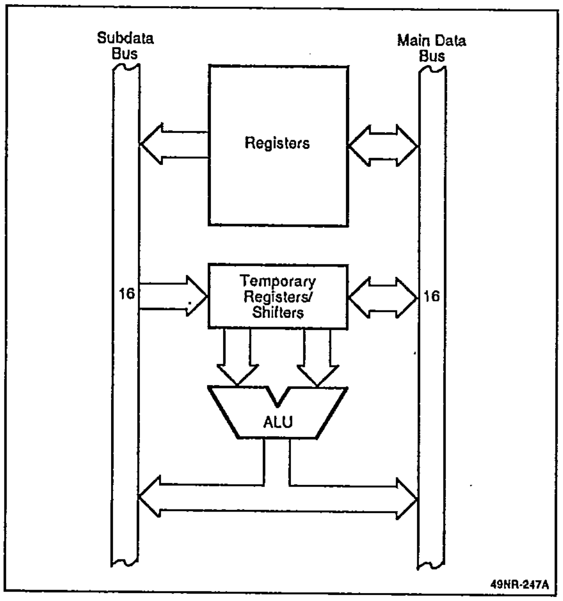
The CPU comprises the execution unit and the address generator. Figure 32 is the CPU block diagram.

CPU Execution Unit

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

In addition to the hardware control logic, the most significant feature of the execution unit is a dual-bus internal data path (figure 33). The ALU and many registers are dual ported with a data bus on each port. This allows two operands to be transferred in one clock cycle instead of two. Performance is improved as much as 30% by the dual data bus concept.

Figure 33. Dual Data Buses



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

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

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

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

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

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

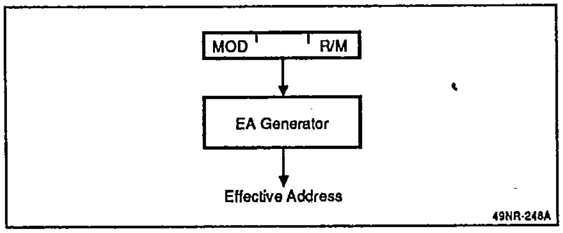
Instruction Prefetching. The V53 is a pipelined machine. To keep the pipeline running efficiently, it should be kept full of instructions in various stages of execution. Instructions are fetched before they are needed and placed in the instruction processing queue (IPQ).

Data in the IPQ is broken out by the decoder logic to determine what addressing modes will be used and what CPU resources are required to execute the prefetched instruction. To keep the 8-byte IPQ full, the bus control logic schedules an instruction prefetch cycle whenever there are at least 2 unused bytes in the IPQ.

The IPQ is cleared whenever a control transfer instruction (any branch, call, return, or break is executed). This is done because a different instruction stream will be used following a control transfer, and the IPQ will then contain instruction data that will never be used. When this happens, the V53's pipeline is emptied and performance is reduced. To maximize performance, the number of control transfers should be minimized.

Effective Address Generator. The effective address generator (EAG) logic computes a 16-bit effective address for each operand. This address is an offset into one of the four segments. Refer to figure 34. This effective address is passed on to the address modifier adder. The EAG decodes the first byte(s) of each instruction to determine the addressing mode and initiates any bus cycles required to fetch pointers/offsets from memory. Effective addresses are calculated in a maximum of 1 clock period as compared with 5 to 12 clocks for a microprogrammed machine.

Figure 34. Effective Address Generator



Address Generator

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

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

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

CPU Addressing Mechanism

The V53 is completely compatible with the µPD70108/116 in its addressing modes and in the way that addresses are computed. It offers a method of expanding the memory address space to 16M bytes.

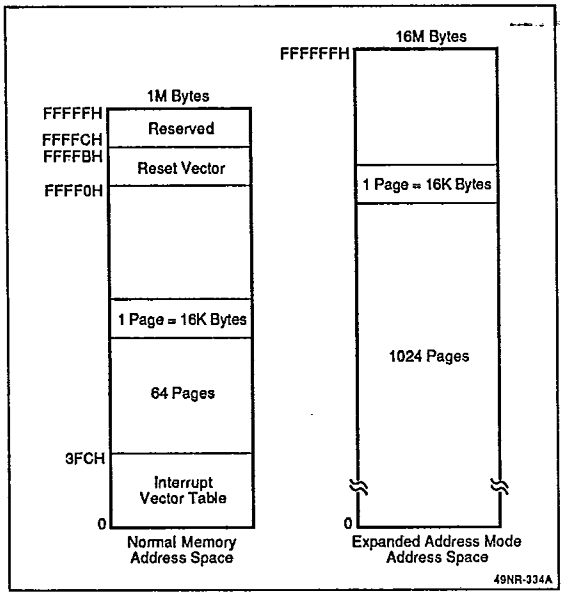
The I/O space is 64K bytes (16-bit address). The normal memory address space is 1M byte (20-bit address), and the expanded address space is 16M bytes (24-bit address). See figure 35. Expanded addressing is enabled or disabled using the BRKXA and RETXA instructions.

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

Certain areas of the V53 address spaces (physical for normal mode and logical for expanded addressing mode) are reserved. . Memory addresses 0-3FCH are used for the interrupt vector table (figure 35) located in the interrupt operation section. Memory addresses FFFF0H-FFFFFH must contain a branch to boot code; PC, PFP, and PS are initialized at RESET to point to this area.

I/O addresses FF00H-FFFFH are reserved for the address translation registers and system control registers. The DMAU, TCU, ICU, and SCU sections each contain a block of registers with programmable base addresses. They may be located inside any 256-byte block in the I/O space. See figure 36.

Figure 35. Memory Address Space



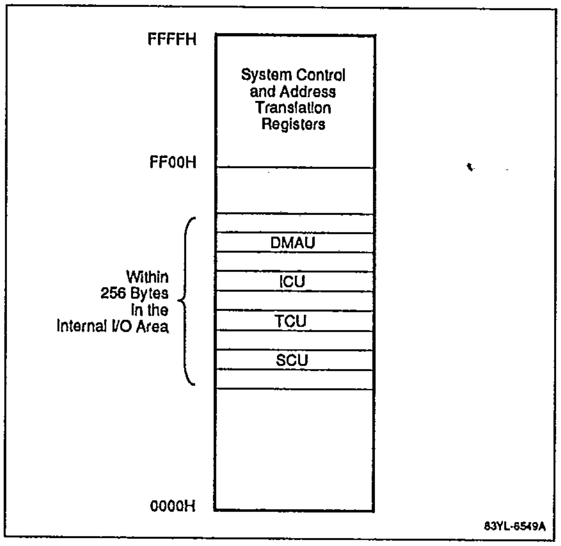
I/O Addresses

I/O devices can be referenced by 8-bit immediate addresses or by 16-bit addresses via the DW register. If I/O operations require other more complex addressing modes, the I/O devices must be placed in the memory address space (using memory-mapped I/O techniques). For memory-mapped I/O devices, there are no restrictions on instruction or addressing mode usage. However, the V53 will not automatically insert 6 clock cycles after

T-49-17-15

memory-mapped I/O operations; external logic must provide the necessary I/O device recovery time.

Figure 36. I/O Address Space

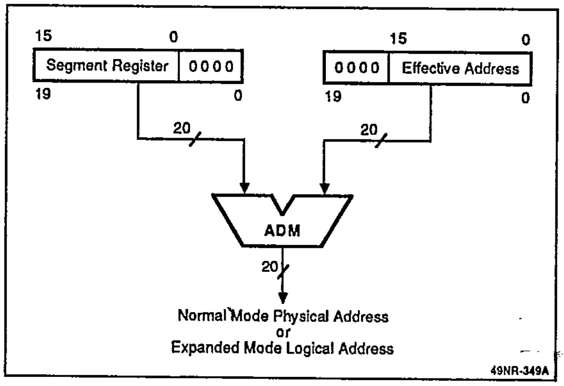


Normal Memory Addresses

The V53 is a 16-bit device with 16-bit registers. To allow a memory address space larger than 64K bytes, memory segmentation is used. The 1M-byte memory address space is divided into 64K-byte segments. Up to four segments can be in use at any given time. The base addresses of the four active segments (program segment, stack segment, data segment 0, and data segment 1) are contained in four 16-bit segment registers (PS, SS, DS0, and DS1, respectively). The 16-bit value in each register is the upper 16 bits of the 20-bit memory address. Thus, segments must start on 16-byte boundaries.

As described above, the V53 hardware generates a 16-bit effective address for each memory operation. This effective address is an offset into one of the four active segments. The actual 20-bit memory address is computed by adding the EA to the segment register value expanded with zeros to 20 bits. Figure 37 shows this process.

Figure 37. 20-Bit Address



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

Expanded Addresses

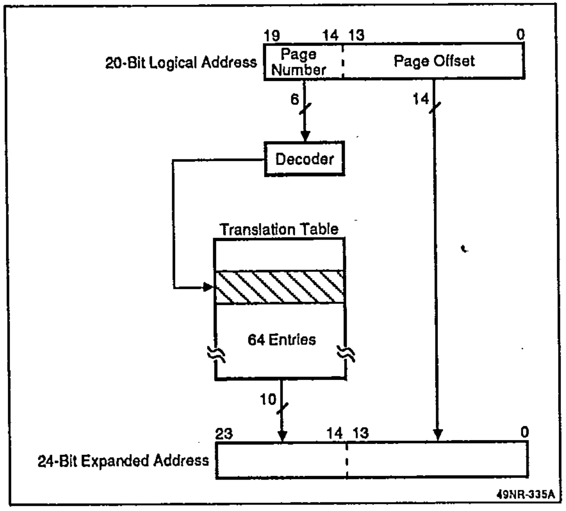
In the expanded addressing mode, the memory space is divided into 1024 pages (figure 35). Each page is 16K bytes. Each page of the normal 20-bit address space is mapped to a page in the expanded address space using a 64-entry address translation table. The table is made up of 64 page registers that reside in the I/O space.

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

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

The address translation mechanism is shown in figure 38. The upper 6 bits of the logical 20-bit address select one of the entries in the address translation table, which supplies a 10-bit value. This value is substituted for the original 6 bits in the normal address to create a 24-bit expanded address.

Figure 38. Address Translation Mechanism



Address Expansion Registers

These are the page and XAM registers, accessed by the word IN and OUT instructions. Figure 39 shows page register usage and I/O addresses. The page registers contain the 10-bit physical page base address. The XAM register is a read-only status flag that indicates whether expanded addressing is enabled.

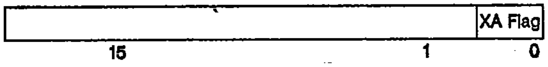
Unused data bits in the XAM register are read as 0. Expanded addressing must be disabled before accessing any of the page registers. That is, if expanded mode is enabled, the page registers cannot be accessed. This prevents an expanded mode task from accidentally modifying its memory map.

Figure 39. Address Expansion Registers

Page Registers

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

XAM Register



Operand Addressing Modes

For operand addressing, the V53 offers nine modes.

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

Register. The operand is in a V53 register pointed to by the instruction.

Immediate. The operand is in the instruction stream following the opcode of the instruction. This data will have been prefetched. Immediate data uses the V53 pipeline efficiently.

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

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

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

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

Based Indexed. One or two bytes of immediate data are treated as a signed displacement that is added to two

16-bit registers (BP or BW and IX or IY) to form the effective address. This mode is useful for array addressing.

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

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

Instruction Addressing Modes

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

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

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

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

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

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

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

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

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

CPU Register Configuration

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

Segment Registers (PS, SS, DS0, DS1). There are four segment registers, each containing the upper 16 bits of the base address of a 64K logical segment. Since logical segments reside on 16-byte boundaries, the lower 4 bits of the base address are always zero. Normal 20-bit memory addresses are formed by adding the 16-bit effective address to the base address of one of the segments. During this operation, certain types of effective addresses will be paired with specific segment registers.

Segment Register	Default Offset
PS (program segment)	PFP
SS (stack segment)	SP, effective address
DS0 (data segment 0)	IX, effective address
DS1 (data segment 1)	IY

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

General-Purpose Registers (AW, BW, CW, DW). The four 16-bit general-purpose registers can be accessed as 16-bit or 8-bit quantities. When the AW, BW, CW, or DW destination is used, the register will be 16 bits. When AL, AH, BL, BH, CL, CH, DL, or DH is used, the register will be 8 bits. AL will be the low byte of AW and AH will be the high byte, etc.

Some operations require the use of specific registers.

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

Pointer (SP, BP) and Index Registers (IX, IY). These registers are used as base pointers and index registers when based, indexed, or based indexed addressing modes are used.

They may also be used as general-purpose registers for data transfer, arithmetic, and logical instructions. They can only be accessed as 16-bit registers.

Some operations use these registers in specific ways.

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

Program Status Word (PSW). The program status word reflects the status of the CPU by six status flags and affects the operation of the CPU by three control flags.

15 PSW 8

1	1	1	1	V	DIR	IE	BRK

7 0

S	Z	0	AC	0	P	1	CY

Status Flags	
V	Overflow
S	Sign
Z	Zero
AC	Auxiliary carry
P	Parity
CY	Carry

Control Flags	
DIR	Direction
IE	Interrupt enable
BRK	Break

The DIR control flag determines whether address pointers are incremented (1) or decremented (0) for block (string) operations. IE enables interrupts (1) or disables interrupts (0). BRK enables (1) or disables (0) the single stepping trap (vector 1).

The PSW cannot be accessed directly as a 16-bit register. Specific instructions set/reset the control flags. When the PSW is pushed on the stack (as during interrupt processing), the PSW is set as shown in the diagram above.

Interrupt Operation

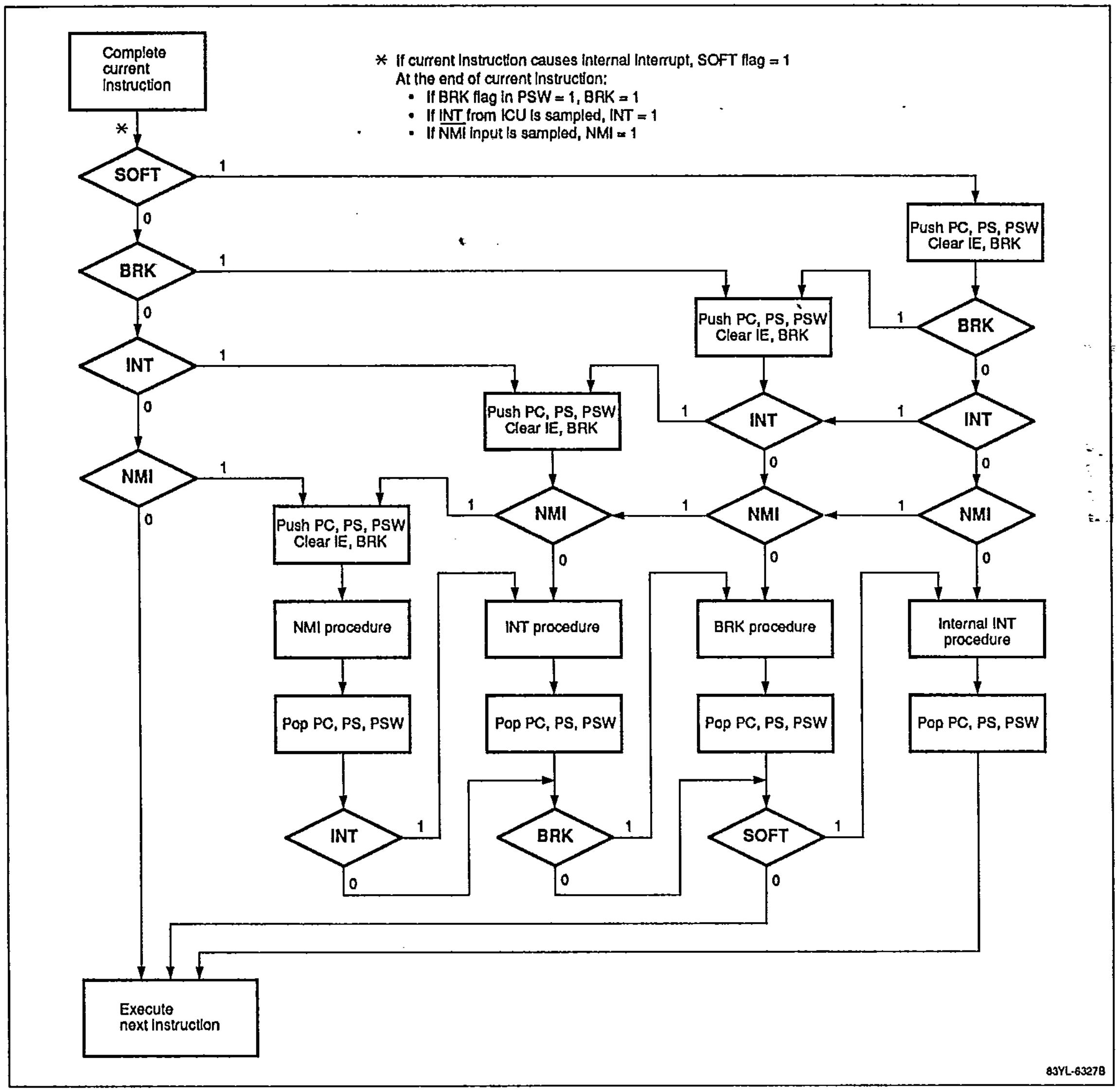
The interrupts supported by the V53 can be divided into two types: those generated by external interrupt requests and traps generated by software processing. Interrupts of each type are listed below.

- External Interrupts
  - — NMI input (nonmaskable)
  - — INTP0-INTP7 (maskable)
- Software Traps
  - — Divide error during DIV or DIVU instruction
  - — Array bound error during CHKIND
  - — Single-step (PSW BRK flag = 1)
  - — Undefined instruction
  - — Coprocessor error
  - — Coprocessor not connected
  - — Break instructions (BRKV, BRK3, BRK imm8, BRKXA)

The eight INTP interrupts are handled by the interrupt control unit (ICU). The ICU prioritizes the INTPs and produces a single INT output, an internal signal that goes to the CPU interrupt logic. There the interrupt prioritization flow diagram (figure 40) is implemented. Interrupts are prioritized by the CPU as follows.

NMI (highest priority
INT
BRK flag
Other software interrupts and exceptions

Figure 40. Interrupt Prioritization Flow Diagram



Interrupts are not accepted by the CPU at certain times. NMI, INT, and BRK flags are not accepted under the following conditions.

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

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

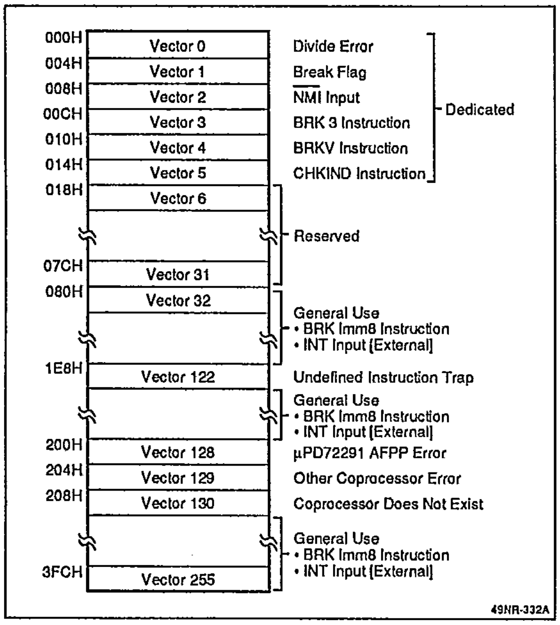
(3) Between a repeat or BUSLOCK prefix and the next instruction.

INT is not accepted when the PSW IE flag is 0, or between an RETI or POP PSW and the next instruction.

Once an interrupt has been accepted by the CPU, an interrupt service routine will be entered. The address of this routine is specified by an interrupt vector stored in the interrupt vector table (figure 41). For most interrupts, the vector used depends on what interrupt is being processed (e.g., NMI always uses vector 2). For INT and BRK imm8 interrupts, any vector may be used; the vector number is supplied by the ICU or an external device (such as a µPD71059) in the case of INT, or by immediate data in the case of BRK.

The interrupt vector table uses 1K bytes of memory at addresses 000H to 3FFH and stores up to 256 vectors.

Figure 41. Interrupt Vector Table

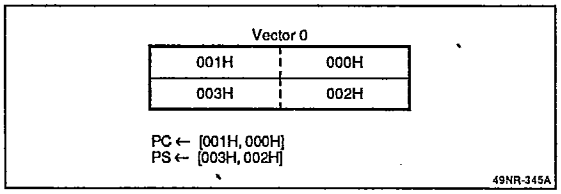


000H	Vector 0	Divide Error	Dedicated
004H	Vector 1	Break Flag	
008H	Vector 2	NMI Input	
00CH	Vector 3	BRK 3 Instruction	
010H	Vector 4	BRKV Instruction	
014H	Vector 5	CHKIND Instruction	
018H	Vector 6		Reserved
07CH	Vector 31		
080H	Vector 32		General Use • BRK Imm8 Instruction • INT Input (External)
1E8H	Vector 122	Undefined Instruction Trap	
			General Use • BRK Imm8 Instruction • INT Input (External)
200H	Vector 128	µPD72291 AFPP Error	
204H	Vector 129	Other Coprocessor Error	
208H	Vector 130	Coprocessor Does Not Exist	
3FCH	Vector 255		General Use • BRK Imm8 Instruction • INT Input (External)

49NR-332A

Each interrupt vector consists of four bytes. The two low bytes are loaded into the PC as the offset, and the two high bytes are loaded into the PS as the base address. See figure 42.

Figure 42. Interrupt Vector 0



PC ← [001H, 000H]
PS ← [003H, 002H]

49NR-345A

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

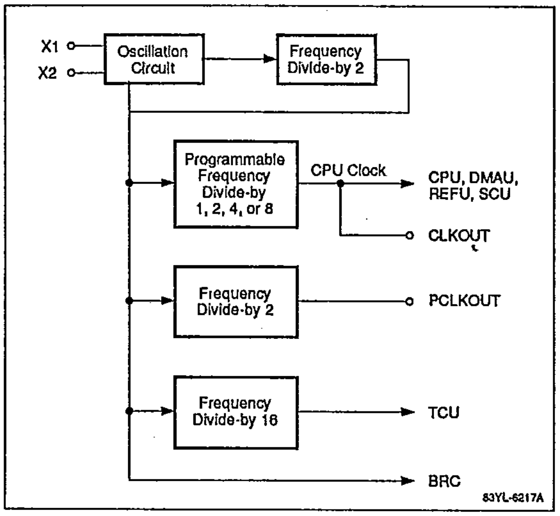
(SP – 1, SP – 2) ← PSW
(SP – 3, SP – 4) ← PS
(SP – 5, SP – 6) ← PC
SP ← SP – 6
IE ← 0, BRK ← 0
PS ← vector high bytes
PC ← vector low bytes

When an interrupt is accepted, two possible PC values could be saved. For some interrupts, the offset of the current instruction is saved. These interrupts are divide error, CHKIND, illegal opcode, µPD72291 FPP error, other coprocessor error, and CP not present. For the other interrupts (NMI, BRK flag, BRK instruction, or ICU interrupt), the offset of the next instruction is saved.

CLOCK GENERATOR (CG)

The clock generator (figure 43) is driven by a crystal connected to pins X1 and X2 or an oscillator connected to pin X1 with no connection at pin X2. The source frequency is divided to supply various clocks to internal units (CPU, DMAU, etc.) and to external devices at pins CLKOUT and PCLKOUT.

Figure 43. Clock Generator Diagram



BUS OPERATION

The V53 uses a synchronous bus interface. The X1 and X2 inputs provide a reference oscillator frequency for the internal clock generator, which supplies the main system clock to the other internal devices and to external devices via the CLKOUT pin. All V53 bus timings and instruction execution clock counts are specified relative to the CLKOUT signal. Bus cycles start on the falling edge of CLKOUT.

The V53's internal bus is a multimaster, shared bus. The CPU, DMAU, or REFU can all be bus masters. Each requests bus mastership from the bus arbitration unit (BAU). External devices can also request mastership of the bus using the HLDRQ input.

Bus Interface Unit (BIU)

The BIU contains the interface logic that allows the three internal bus masters (CPU, DMAU, and REFU) to control the external address, data, and control buses. The BIU also synchronizes the $\overline{\text{BS8}}$/BS16, $\overline{\text{RESET}}$, and $\overline{\text{READY}}$ inputs to the system clock. When a reset signal is accepted, the BIU asserts the RESOUT output.

Bus Arbitration Unit (BAU)

The BAU accepts and grants five different requests for bus mastership in the following priority order.

REFU demand (highest)
DMAU request
HLDRQ
CPU request
REFU request

The refresh unit is assigned both the highest and the lowest priorities. Normally, REFU requests are made, and if the bus is not granted, they are placed in a queue. Once the queue depth reaches seven requests, a refresh demand is made, and the BAU gives this the highest priority.

Bus Wait Function

When the bus is active and the BAU receives a higher priority request, the BAU will take away its grant to the current bus master. But the current master may not release the bus immediately. The BAU will wait until the current master takes away its request before granting the bus to the higher priority requester. This is called bus waiting.

For example, if an external device has been granted the bus via the HLDAK output, and the DMAU requests the bus (DMA is higher priority than HLDRQ), the V53 will deassert HLDAK but will not take the bus back until the external master deasserts HLDRQ. Note that the external master is not required to immediately release the bus back to the V53; the BAU will wait until HLDRQ is removed.

Usually a higher priority request will be granted quickly; for example, if a DMA request is accepted during T2 of a CPU bus cycle, the next bus cycle will usually be a DMA cycle. However, each internal bus master will hold onto the bus under certain circumstances.

The CPU will not let go of the bus as long as the $\overline{\text{BUSLOCK}}$ prefix is used, or until the current bus operation is completely finished (an unaligned or bus-sizing operation may take more than one bus cycle). Likewise, when it is in bus hold mode, the DMAU will not release the bus until all active DMA requests have been processed.

This mode should be used with care as it can result in DRAM refresh errors if the DMA takes a long time to complete. Note that bus hold mode is only available when DMAU is in μPD71071 compatibility mode; μPD71037 mode is always in bus release mode.

External Bus Masters

At times, external bus masters will need to use the V53 bus. There are two methods provided for that purpose: hold request and DMA cascade. Up to five external bus masters can be connected to the V53.

Hold Request. The external bus master can request the bus using a hold request. Hold request is implemented using the HLDRQ and HLDAK signals. The V53 grants the bus by floating many of its outputs and asserting HLDAK to notify the external device that the bus is now free.

DMA Cascade. DMA cascade is very similar to hold request; the difference is that a DMARQ/$\overline{\text{DMAAK}}$ signal pair requests and grants the bus. While DMA cascade is meant to be used to connect additional DMA controllers, it can be used by any type of external bus master. Since there are four DMA channels, each of which can be in cascade mode, up to four external masters can be connected by DMA cascade.

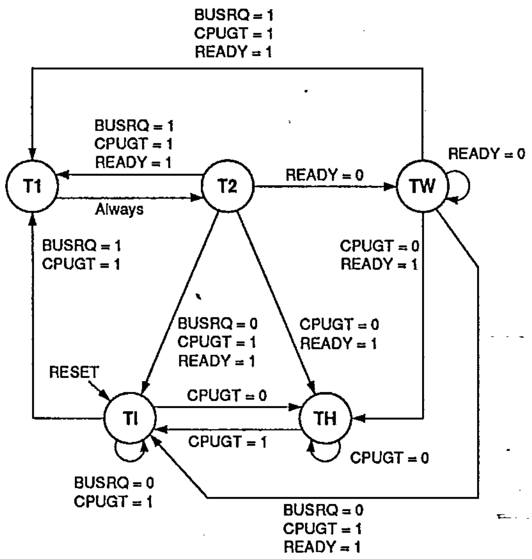
Bus Cycle Descriptions

Each of the internal bus masters uses the V53 bus interface in a different way: DMA bus cycles have a different structure than CPU bus cycles or REFU cycles. There are 18 different V53 bus cycles summarized previously in table 1.

CPU Bus Cycles

The bus state diagram for CPU cycles is shown in figure 44. CPU bus cycles are nominally two clock periods long, and may be extended by adding wait states using either the internal wait state generator or the external $\overline{\text{READY}}$ input.

Figure 44. CPU Bus State Diagram



TI	Idle state
T1	Start bus cycle
T2	Sample READY, DATA
TW	Wait for READY = 1
TH	Bus hold state; release bus to external BAU

BUSRQ = 1 when a read or write bus cycle is requested.
CPUGT = 1 when BAU grants internal bus to CPU.

83YL-6548A

The first state of every bus cycle is T1, and it is followed immediately by T2. $\overline{\text{READY}}$ is sampled on the rising (middle) edge of T2. If $\overline{\text{READY}}$ is not asserted, the next bus state will be the TW wait state. TWs will be inserted until $\overline{\text{READY}}$ is sampled low, after which the bus cycle will finish. TWs also will be inserted by the wait state generator, and the $\overline{\text{READY}}$ input is ignored until all TWs programmed in the wait state have been inserted. The dynamic bus sizing input, $\overline{\text{BS8}}$/BS16, is sampled at the same time as $\overline{\text{READY}}$.

Note that dynamic bus sizing is only implemented for CPU cycles; DMAU or REFU cycles do not use this input.

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

Write data is driven on D_0-D_{15} following the rising (middle) edge of T1, and maintained until after the rising edge of T2 or the last TW. Read data is sampled on the trailing edge of T2 or the last TW state. A strobe, $\overline{\text{DSTB}}$, gives the status of the V53 data bus. $\overline{\text{DSTB}}$ is asserted after the rising (middle) edge of T1. $\overline{\text{DSTB}}$ is deasserted after the rising edge of T2 or the last TW for a write cycle, and after the trailing edge of T2 or the last TW for a read cycle.

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

Dynamic Bus Sizing for CPU Cycles

The V53 supports dynamic bus sizing for CPU cycles. On a cycle-by-cycle basis, the width of the data bus can be changed from 16 to 8 bits. This simplifies connection with 8-bit I/O devices that may have internal registers at consecutive byte addresses. Other 16-bit CPUs require two ROMs for startup code, but the V53 dynamic bus sizing makes it possible to use a single 8-bit wide ROM.

External logic requests an 8-bit data path by driving $\overline{\text{BS8}}$/BS16 low in time for the V53 to sample it on the rising edge of T2 (or TW). The V53 will perform an additional cycle if needed to finish the operation in byte-wide pieces.

If the bus operation is already 8 bits wide, no further bus cycles will occur (refer to tables 3 and 4). For a read cycle, the data will be sampled on D_7-D_0. For a write cycle to an even address, data will be driven on D_7-D_0. On all byte writes to an odd address, the V53 will put the byte data on both the upper and lower data buses so that the write data will be on D_7-D_0 as well as D_{15}-D_8.

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

Table 3. Write Cycle Bus Sizing

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

Note: Lower = low-order byte; Upper = high-order byte

Table 4. Read Cycle Bus Sizing

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

Note: Lower = low-order byte; Upper = high-order byte

CPU Bus Cycle Types. There are many types of CPU bus cycles (shown previously in table 2). They comprise read, write, and acknowledge cycles.

CPU Read Cycles. There are six CPU read cycles: memory, external I/O, internal I/O coprocessor, coprocessor data reads, and instruction fetch. All have the general timing described previously. Coprocessor reads access the internal registers of an external coprocessor. Coprocessor data reads transfer data from memory to an internal coprocessor register. Instruction fetches fill the V53's 8-byte instruction queue from the memory space. I/O and memory reads transfer data to the V53 from an internal or external I/O device or a memory location. During internal I/O reads, the $\overline{\text{IORD}}$ and $\overline{\text{BUFEN}}$ outputs are not asserted.

Dynamic bus sizing is ignored during internal I/O read cycles, and is not recommended for coprocessor data read cycles. The wait state generator does not affect internal I/O reads or coprocessor reads. $\overline{\text{READY}}$ is used for all CPU read cycles.

CPU Write Cycles. There are five types of CPU writes. Memory writes transfer data from the V53 to a memory location. External and internal I/O writes transfer data from the V53 to external or internal I/O devices. During internal I/O writes, the $\overline{\text{IOWR}}$ and $\overline{\text{BUFEN}}$ outputs are not asserted. Coprocessor data writes transfer data from an external coprocessor to a memory location. Coprocessor writes transfer data from the V53 directly to a coprocessor internal register.

Dynamic bus sizing is ignored during internal I/O read/writes, and is not recommended for coprocessor data write cycles. The wait state generator does not affect internal I/O writes or coprocessor writes. $\overline{\text{READY}}$ is used for all CPU write cycles.

Interrupt Acknowledge Cycles. The CPU interrupt acknowledge operation takes two consecutive bus cycles. The first cycle freezes the state of the internal interrupt control unit (ICU) and any external slave μPD71059 interrupt controllers. The second bus cycle reads an 8-bit vector number on D_7-D_0, supplied by either the ICU or an external slave. This vector number is then used by the CPU as an index into the interrupt vector table to select an interrupt handler. The $\overline{\text{BUSLOCK}}$ output is asserted for the first cycle, and remains asserted until after the second to guarantee that no other bus master will take control of the bus until the interrupt has been accepted.

There are two types of interrupt acknowledge cycles produced by the V53: a master and a slave. The $\overline{\text{INTAK}}$ output is asserted for both types, and should be connected to the interrupt acknowledge inputs of all slave devices. The master cycle is used for the first $\overline{\text{INTAK}}$ to both internal and external ICUs, and the second $\overline{\text{INTAK}}$ to the internal ICU. The slave cycle is used only for the second $\overline{\text{INTAK}}$ to an external slave device.

During the slave cycle, the address of the slave device to be used is presented on A_2-A_0. These address lines should be buffered and then connected to the slave address inputs of the external ICUs. Buffering is necessary because the slave address pins of external devices might be in an output state on power-up, producing a bus conflict on A_2-A_0 if they are connected directly.

Dynamic bus sizing is ignored during the interrupt acknowledge cycles. Wait states can be inserted by the internal wait state generator or by the $\overline{\text{READY}}$ input.

Halt Acknowledge Cycle. When the CPU executes a HALT instruction, a halt acknowledge bus cycle is issued to notify external logic that the V53 is entering a standby mode. This cycle is always two clocks long; $\overline{\text{READY}}$ is ignored and $\overline{\text{DSTB}}$ is not asserted. The V53 has several standby modes.

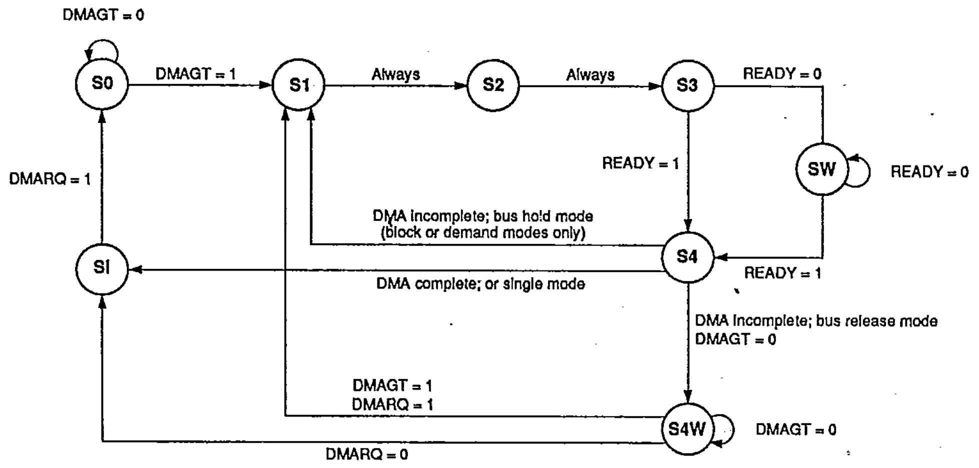
DMA Unit Bus Cycles

Figure 45 shows the bus state diagram for DMA bus cycles. There are eight different states. When the DMAU is idle, it is in state SI. In this state, it is continually sampling the four DMARQ inputs. When a request is detected, the DMAU requests use of the V53 bus from the BAU, and enters state S0. It remains in S0 until the BAU grants the bus to the DMAU, at which point the actual DMA bus cycle starts with state S1. Addresses and control status are output along with BCYST and $\overline{\text{DMAAK}}$.

DMA bus cycles are nominally four clocks long, but they can be stretched by the internal wait state generator or $\overline{\text{READY}}$. S1 always changes to S2 and then to S3. Memory and I/O strobes are asserted during S2, S3, and SW. $\overline{\text{READY}}$ is sampled during S2 for use during S3. If waits are inserted, the SW state is entered. Control stays in that state until no more waits are desired. If no waits are inserted, S3 moves to S4 and the current cycle is over.

Depending on the DMA mode, another DMA cycle might be ready to start immediately (e.g., in burst mode), or another DMA request input may now be asserted. During S4, a decision is made whether to begin another DMA cycle at S1, to return to SI, or to enter the bus wait state S4W. The latter transition will be made if another DMA cycle is ready to start but the BAU has taken the bus away from the DMAU. In S4W, the DMAU releases the bus, but is ready to begin as soon as the bus is granted again and the DMA request is still pending.

Figure 45. DMAU Bus State Diagram



SI	Idle state; sample DMA requests.
S0	Wait for BAU to grant bus to DMAU.
S1	Output DMA memory address.
S2	Assert strobes; sample READY, END.
S3	Check READY.
S4	End of DMA cycle; check DMA completion.
S4W	Bus wait state; release bus to BAU.
SW	Wait for READY = 1.

83YL-6330B

DMA Read Cycle. The DMAU performs "fly-by" DMA. During one DMA read bus cycle, data moves from the source address in memory to the destination I/O device. The V53 puts the memory address on A_{23}-A_0, and asserts $\overline{MRD}$. At the same time, $\overline{IOWR}$ is asserted. Memory will drive the DMA data onto the bus, and the $\overline{IOWR}$ signal will latch the data into the I/O device. $\overline{DMAAK}$ should be used to control chip select at the I/O device. Since the V53 does not use the data, $\overline{BUFEN}$ is not asserted during DMA bus cycles.

DMA Write Cycle. The DMAU performs "fly-by" DMA. During one DMA write bus cycle, data moves from the source I/O device to the destination address in memory. The V53 puts the memory address on A_{23}-A_0, and asserts $\overline{MWR}$. At the same time, $\overline{IORD}$ is asserted. The I/O device will drive the DMA data onto the bus, and the $\overline{MWR}$ signal will latch the data into memory. $\overline{DMAAK}$ should be used to control chip select at the I/O device. Since the V53 does not use the data, $\overline{BUFEN}$ is not asserted during DMA bus cycles

Note that when DMA writes are made to DRAM, it may be necessary to generate a delayed CAS strobe because the data is being supplied by an I/O device that may have long access time. The write data may not be valid when the normal CAS signal is asserted.

Dynamic bus sizing cannot be used for DMA operations. The internal wait state generator and $\overline{READY}$ can be used to stretch the cycle.

DMA Cascade. During DMA cascade, the DMA state machine releases the V53 bus to an external bus master such as a μPD71071 or μPD71037 DMA controller. $\overline{DMAAK}$ is connected to the HLDAK input of the external device. $\overline{DMAAK}$ will stay asserted until the external master deasserts DMARQ. If the V53 BAU needs to give the bus to a higher priority bus master, $\overline{DMAAK}$ will be deasserted. The external bus master is expected to then deassert the DMARQ input, at which point the bus will be given to the higher priority bus master.

Refresh Unit Bus Cycles

The refresh unit performs memory read cycles from consecutive memory addresses. These bus cycles are the same as CPU memory read cycles, except that the REFRQ output is asserted. External logic should use the REFRQ logic to enable RAS for all memory banks, regardless of the address decoding scheme, so that all banks are refreshed.

Dynamic bus sizing cannot be used during refresh operations. The internal wait state generator and $\overline{\text{READY}}$ can be used to stretch the cycle.

SYSTEM INTERFACE

System Memory Access Time

Table 5 shows the system memory access time required for 12.5-MHz and 16-MHz V53 systems to run with zero, one, two, and three wait states. This is the time from when the address bus is valid to when the external system must present the read data on the data bus. These numbers are based on the preliminary ac timing given in this document and are subject to change.

Table 5. Performance vs. Wait States

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

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

Wait States

Table 5 also illustrates the effect of wait states on performance. The V53 CPU overlaps bus interface operations in time with instruction execution. This greatly reduces the effect of wait states on performance. Each bus cycle is nominally two clocks long, while the minimum instruction is two clocks with many instructions taking longer.

There is some idle bus time when the CPU is processing a long instruction and the prefetch queue is full. Wait states can often fill these idle states. However, adding wait states to bus cycles reduces the bus bandwidth available for other bus masters, such as DMA controllers. This is because some of the idle time that would have been available to them is used for CPU cycles.

Note that in all cases, a 16-MHz V53 with N+1 wait states is faster than a 12.5-MHz device with N wait states but slower memory.

Note also that the numbers are for comparison only. Different results will be obtained for other program mixes.

Interfacing the µPD72291 AFPP

The AFPP is a very-high-performance floating-point coprocessor able to process more than 530K floating-point operations per second at 16 MHz.

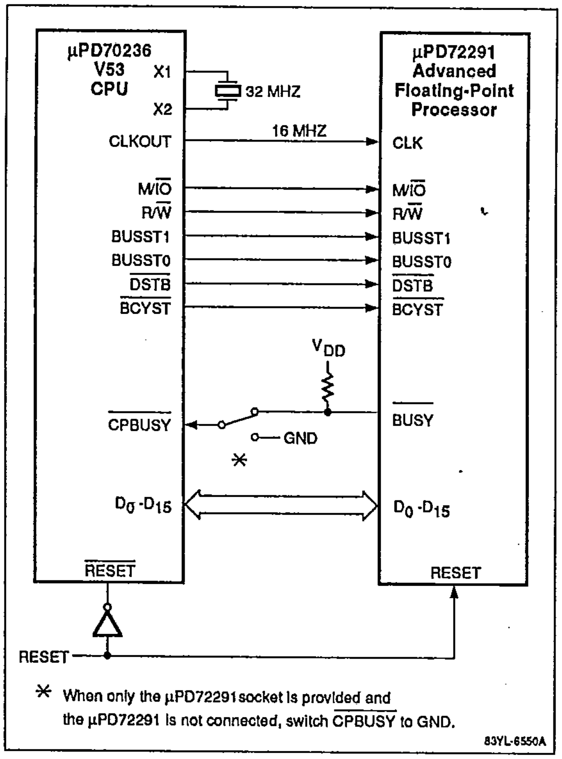
The AFPP is programmed as an extension of the V53 instruction set. The AFPP executes floating-point operations, computes transcendental functions, and performs vector multiplications.

AFPP instructions use the FP01 and FP02 formats. When one of these opcodes is encountered and an AFPP is connected, a coprocessor protocol routine is entered. The V53 computes any effective addresses required, reads or writes the operands for the AFPP, and tells the AFPP which operation should be performed.

The AFPP responds by asserting its BUSY output when it starts the operation. The V53 will not start another AFPP operation until BUSY is deasserted, but may execute CPU instructions. When BUSY is deasserted, the V53 will transfer the AFPP status to the AW register.

Figure 46 shows how to connect a V53 CPU to a µPD72291 AFPP. The CPU reads and writes status and commands to the AFPP using coprocessor read and write cycles, which always take two clocks. AFPP operands are written using coprocessor memory write/read cycles, which always require one wait state. The V53 automatically inserts one wait state into these cycles so no external wait generation logic is required.

Figure 46. Connections Between the V53 and µPD72291



* When only the µPD72291 socket is provided and the µPD72291 is not connected, switch $\overline{CPBUSY}$ to GND.

On reset, $\overline{CPBUSY}$ is sampled. If it is low, the V53 assumes that a coprocessor is connected. CPERR is also sampled to determine what kind of coprocessor is connected as follows.

$\overline{CPBUSY}$	CPERR	Coprocessor Connected
1	x	None
0	0	µPD72291
0	1	Another kind

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

SYSTEM CONTROL I/O

On-Chip Control Registers

The V53 provides many on-chip control registers. Some of these reside in the 256-byte system I/O area (I/O space addresses FF00 to FFFF). These are shown in table 6. Other registers reside in small blocks associated with an on-chip peripheral (addresses are programmable). There are register blocks for DMAU, TCU, ICU, and SCU. The base addresses for these register blocks are programmable using the OPHA, DULA, TULA, and SULA registers in the system I/O area. See figure 47.

Figure 47. Peripheral Relocation

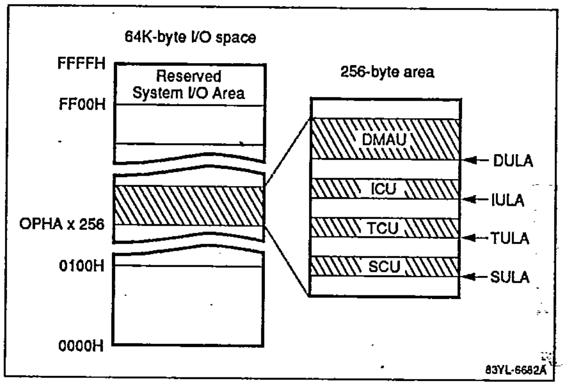


Table 6. System I/O Area

I/O Address	Register Name	Figure
FFFFH	Reserved	–
FFFEH	SCTL	48
FFFDH	OPSEL	49
FFFCH	OPHA	50
FFFBH	DULA	50
FFFAH	IULA	50
FFF9H	TULA	50
FFF8H	SULA	50
FFF7H	Reserved	–
FFF6H	WCY4	61
FFF5H	WCY3	60
FFF4H	WCY2	59
FFF3H	WMB1	55
FFF2H	RFC	62
FFF1H	SBCR	110
FFF0H	TCKS	51
FFEFH-FFEEF	Reserved	–
FFEDH	WAC	56
FFECH	WCY0	57
FFEBH	WCY1	58
FFEAH	WMB0	54
FFE9H	BRC	52
FFE8H	Reserved	–
FFE7H-FFE2H	Reserved	–

Table 6. System I/O Area (cont)

I/O Address	Register Name	Figure
FFE1H	BADR	102
FFE0H	BSEL	103
FFDFH-FF81H	Reserved	—
FF80H	XAM (Read Only)	39
FF7FH-FF00H	PGR64-PGR1	39

Note: All registers are Read/Write except XAM.

System Control Register (SCTL)

The SCTL register (figure 48) selects the 8-bit or 16-bit boundary of an internal peripheral relocation address. It also sets the internal DMAU in the µPD71071 or µPD71037 modes. In µPD71037 mode, SCTL controls propagation of carry from A_{15} to A_{16} or from A_{19} to A_{20}. SCTL selects the baud rate generator or TOUT as the SCU clock.

Figure 48. System Control Register (SCTL)

7							0
—	—	—	SC	CE1	CE0	DMAM	IOAG

Address FFFEH

SC	SCU Input Clock
0	TOUT1
1	From baud rate generator
CE1	**Carry to A_{20} in µPD71037**
0	Does not propagate
1	Propagates
CE0	**Carry to A_{16} in µPD71037**
0	Does not propagate
1	Propagates
DMAM	**DMAU Mode**
0	µPD71071
1	µPD71037
IOAG	**Internal I/O Address**
0	Even or odd (16-bit boundary)
1	Contiguous (8-bit boundary)

On-Chip Peripheral Selection Register (OPSEL)

The OPSEL registers (figure 49) controls the V53 internal peripherals. Any of the four peripherals (DMAU, TCU, ICU, or SCU) can be independently enabled or disabled by setting the appropriate OPSEL bit.

Figure 49. On-Chip Peripheral Selection Register (OPSEL)

7							0
—	—	—	—	SS	TS	IS	DS

SS	SCU Operation
0	Disabled
1	Enabled
TS	**TCU Operation**
0	Disabled
1	Enabled
IS	**ICU Operation**
0	Disabled
1	Enabled
DS	**DMAU Operation**
0	Disabled
1	Enabled

Internal Peripheral Relocation Registers

The five internal peripheral registers fix the I/O addresses of the DMAU, ICU, TCU, and SCU. Register OPHA fixes the high-order byte of the 16-bit I/O addresses. Registers DULA, IULA, TULA, and SULA select the low-order byte of the I/O addresses for the DMAU, ICU, TCU, and SCU peripherals, respectively.

The formats of the individual internal peripheral registers are shown in figure 50. Since address checking is not performed, two peripheral I/O address spaces should not be overlapped.

Figure 50. *Internal Peripheral Relocation Registers*

OPHA Register

	7							0
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8

I/O Address FFFCH

DULA Register

	7							0
IOAG = x*	A_7	A_6	A_5	A_4	—	—	—	—
IOAG = 0 #	A_7	A_6	A_5	A_4	—	—	—	—
IOAG = 1 #	A_7	A_6	A_5	—	—	—	—	A_0

I/O Address FFFBH

* µPD71071 mode
\# µPD71037 mode

IULA Register

	7							0
IOAG = 0	A_7	A_6	A_5	A_4	A_3	A_2	—	—
IOAG = 1	A_7	A_6	A_5	A_4	A_3	—	—	A_0

I/O Address FFFAH

TULA Register

	7							0
IOAG = 0	A_7	A_6	A_5	A_4	A_3	A_2	—	—
IOAG = 1	A_7	A_6	A_5	A_4	A_3	—	—	A_0

I/O Address FFF9H

SULA Register

	7							0
IOAG = 0	A_7	A_6	A_5	A_4	A_3	A_2	—	—
IOAG = 1	A_7	A_6	A_5	A_4	A_3	—	—	A_0

I/O Address FFF8H

The IOAG bit of the SCTL register changes how the DULA, IULA, TULA, and SULA registers are used. When IOAG = 1, the DAMU, ICU, TCU, and SCU registers are on contiguous bytes. When IOAG = 0, each of these byte-wide registers is put on a word boundary. Bit A_0 selects the low or high byte of the word. This allows code written for a 16-bit system to be ported to a V53 design with no modifications. Because the DMAU registers in µPD71071 mode are 16-bit, the IOAG bit in figure 50 is noted as "x" (don't care).

Timer Clock Selection Register (TCKS)

The TCKS register (figure 51) selects the clock source for the timer/counters as well as the divisor for the internal clock prescaler. The clock source for each timer/counter is independently selected from an internal clock (figure 43) or an external clock source (TCLK).

The frequency of the internal clock selected by bits 2, 3, and 4 is programmable. The PS bits allow the clock to be set to the external oscillator frequency divided by 4, 8, 16, or 32.

Figure 51. *Timer Clock Selection Register (TCKS)*

7							0
—	—	—	CS2	CS1	CS0	PS	

I/O Address FFF0H

CS2	Clock Input to TCT2
0	Internal clock
1	TCLK pin
CS1	**Clock Input to TCT1**
0	Internal clock
1	TCLK pin
CS0	**Clock Input to TCT0**
0	Internal
1	TCLK pin
PS	**Prescale Divisor of External Oscillator**
0 0	4
0 1	8
1 0	16
1 1	32

Baud Rate Counter (BRC)

The BRC (figure 52) is an 8-bit, frequency-division counter for the dedicated baud rate generator. It sets the value by which an internal frequency is to be divided to provide the SCU with its baud rate clock.

Figure 52. Baud Rate Counter (BRC)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

7 I/O Address FFE9H 0

Table 7 illustrates the relationship between the baud rate and the value set in the BRC.

Table 7. Baud Rate Setting by BRC

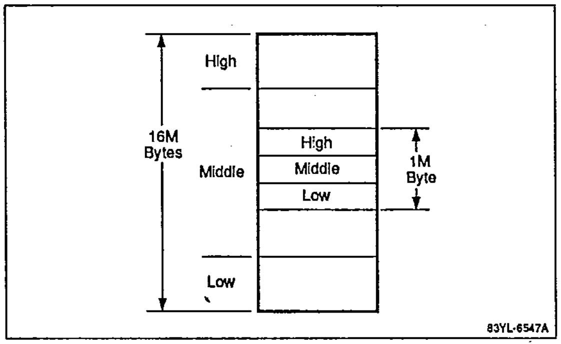
Oscillation frequency	24.576 MHz		29.4912 MHz	
Oscillation frequency ÷ 2	12.288 MHz		14.7456 MHz	
Baud rate factor (÷)	16	64	16	64
Internal frequency	0.768	0.192	0.9216	0.2304
Baud Rate	**Number of Counts Set in BRC**			
1200	—	160	—	192
2400	—	80	—	96
4800	160	40	192	48
9600	80	20	96	24
19,200	40	10	48	12
38,400	20	5	24	6

WAIT CONTROL UNIT

The wait control unit (WCU) inserts from 0 to 7 wait states (TW) into a bus cycle to compensate for the varying access times of different memory and I/O devices. Each wait state is equivalent to one CPU clock cycle. The number of wait states can be individually programmed for CPU, DMAU, REFU, INTAK, and external I/O cycles. The INTAK cycles can be programmed for 2-7 wait states.

For memory accesses, the address space is divided into a total of six sections (labeled High, Middle, and Low in figure 53). A different number of wait states can be programmed for each section, allowing much flexibility in the system design. The WCU works with the external $\overline{\text{READY}}$ input. After the proper number of TWs have been inserted into the bus cycle, $\overline{\text{READY}}$ will be sampled, and wait states will be inserted until it is asserted.

Figure 53. Memory Space Division



The WCU can insert waits into memory or external I/O cycles, but not into coprocessor, internal I/O, or halt acknowledge cycles.

Eight system I/O registers (figures 54-61) control the WCU. They are the wait state memory boundary registers (WMB0 and WMB1), the WCU address control register (WAC), and the wait state cycle count registers (WCY0-WCY4).

Memory Boundary Registers (WMB0, WMB1)

The WMB0 register divides the entire 16M-byte address space into three sections. The ELMB and EUMB fields specify the size of the upper and lower memory blocks. The middle block is the area left in between. The WCY0 and WCY1 registers specify the wait states of each expanded memory block.

In addition to dividing expanded memory, a specific 1M-byte memory area can also be partitioned into three blocks for wait state generation. The WAC register determines which 1M-byte area is referenced. The WMB1 register divides this area into three blocks in the same manner as described above for WMB0. Registers WCY2 and WCY3 specify the wait states for each block and also I/O.

T-49-17-15

Figure 54. Memory Boundary Register 0 (WMB0)

–	ELMB	–	EUMB

7 I/O Address FFEAH 0

ELMB/EUMB	Memory Block Size (Bytes)
000	1M
001	2M
010	3M
011	4M
100	5M
101	6M
110	7M
111	8M

Figure 55. Memory Boundary Register 1 (WMB1)

–	LMB	–	UMB

7 I/O Address FFF3H 0

LMB/UMB	Memory Block Size (Bytes)
000	32K
001	64K
010	96K
011	128K
100	192K
101	256K
110	384K
111	512K

Figure 56. WCU Address Control Register (WAC)

				UWA

7 I/O Address FFEDH 0

UWA = Upper 4 bits of expanded address specifying a 1M-byte memory space

Wait State Cycle Count Registers (WCY0-WCY4)

Each WCY register has one or two 3-bit fields that set the number of waits for a particular kind of cycle or the number of waits to be inserted into cycles during which certain memory blocks are accessed.

(1) WCY0 and WCY1 (figures 57 and 58) pertain to the 16M-byte memory space set by the WMB0 register.

(2) WCY2 and WCY3 (figures 59 and 60) pertain to the 1M-byte memory space set by the WMB1 register.

(3) Also, the IOW field of WCY3 sets the number of waits for external I/O cycles and interrupt acknowledge cycles.

(4) The waits set by WCY3 cannot be inserted into the internal I/O area read/write cycle.

(5) WCY4 (figure 61) sets the number of waits for DMA cycles and refresh cycles.

After RESET, the WCY registers are set to all 1s, thereby inserting seven waits into all cycles. This allows the use of slow ROMs. Initialization code must set the WCY registers to their values.

Figure 57. WCY0 Register

–	–	–	–	–	EUMW

7 I/O Address FFECH 0

EUMW	*Wait States
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

* Upper section of 16M-byte memory space

Figure 58. WCY1 Register

–	EMMW	–	ELMW

7 I/O Address FFEBH 0

EMMW/ELMW	*Wait States
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

* Middle and lower sections of 16M-byte memory space

Figure 59. WCY2 Register

–	MMW	–	LMW

7 I/O Address FFF4H 0

MMW/LMW	*Wait States
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

* Middle and lower sections of 1M-byte memory space

Figure 60. WCY3 Register

7			0
—	IOW	—	UMW
	I/O Address FFF5H		

	Wait States	
IOW	**Ext I/O Cycles**	**Int Ack Cycles**
000	0	2
001	1	3
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7

UMW	***Wait States**
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

* Upper section of 1M-byte memory space

Figure 61. WCY4 Register

7			0
—	DMAW	—	RFW
	I/O Address FFF6H		

DMAW/RFW	***Wait States**
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

* DMA cycle or refresh cycle.

REFRESH CONTROL UNIT

The refresh control unit (REFU) refreshes external dynamic devices by periodically performing a memory read cycle from consecutive, incrementing addresses. A 16-bit counter provides the refresh address. The upper bits (A_{23}-A_{16}) are low during refreshes. Each refresh bus cycle has two wait states inserted, so that it will be a minimum of 4 clocks long. Refresh cycles can be distinguished from other memory reads by the assertion of the REFRQ output or by the bus status code.

If the V53 is busy when it is time to perform a refresh, the refresh request is placed in a refresh queue until the bus is no longer busy. Normally, the REFU has the lowest bus priority. However, after seven refreshes are queued, the REFU is given the highest bus priority. The REFU gets control of the bus, performs a burst of four refreshes, and then falls back to the lowest priority. This refresh queue ensures that refresh cycles are not lost even when the V53 is busy for long periods of time.

Refresh Control Register (RFC)

The RFC (figure 62) controls the refresh control unit. The RE bit enables or disables the REFU. The refresh interval is set by the RTM field by choosing refresh interval factor N, which determines how many CPU clock cycles elapse between refreshes.

$$\text{Refresh interval} = 16 \times N \times t_{CYC}$$

With a 16-MHz CPU clock, this allows a range of intervals from 1 to 32 µs. After $\overline{\text{RESET}}$, N will be 9, which gives an interval of 9 µs.

Since the V53 may operate with either 8- or 16-bit memory devices, the refresh address can be incremented by 1 (for 8-bit memory) or by 2 (for 16-bit memory). The RDB8 bit in the RFC makes the selection. In the word mode, $\overline{\text{UBE}}$ is always low (active) for refresh cycles. In the byte mode, $\overline{\text{UBE}}$ is asserted only for refreshes to an odd address.

Figure 62. Refresh Control Register (RFC)

7			0
RE	RDB8	—	RTM
	I/O Address FFF2H		

RE	**Refresh**
0	Disable
1	Enable

RDB8	**TCU Clock for Channel 2**
0	Increment by 2 ($\overline{\text{UBE}}$ = low level)
1	Increment by 1 ($\overline{\text{UBE}}$ = high level for even addresses and low level for odd-addresses)

RTM	**Refresh Interval Factor N**
00000	1
00001	2
00010	3
00011	4
00100	5
:	:
:	:
:	:
11110	31
11111	32

TIMER/COUNTER UNIT

The timer/counter unit (TCU) provides a set of three independent 16-bit timer/counters. Each timer has an individual output and gate control input. The clock source for each channel is set individually to either the prescaled CPU clock or the external TCLK. TOUT1 is also internally connected to supply the baud rate clock to the SCU. Figure 63 is the TCU block diagram.

The TCU has the following features.

- Three 16-bit timer/counters
- Six programmable count modes
- Binary/BCD counting
- Multiple latch command
- Count latch command
- Choice of two clock sources
- 16-MHz operation
- Functionally compatible with μPD71054 (8254)

Because $\overline{\text{RESET}}$ leaves the TCU in an uninitialized state, each timer/counter must be initialized by specifying an operating mode and a count. Once programmed, a timer/counter will continue to operate in that mode until another mode is selected. When the count has been written to the counter and transferred to the down counter, a new count operation starts. Both the current count and the counter status can be read while count operations are in progress. Figure 64 is a flow diagram for TCU operations.

Figure 63. TCU Block Diagram

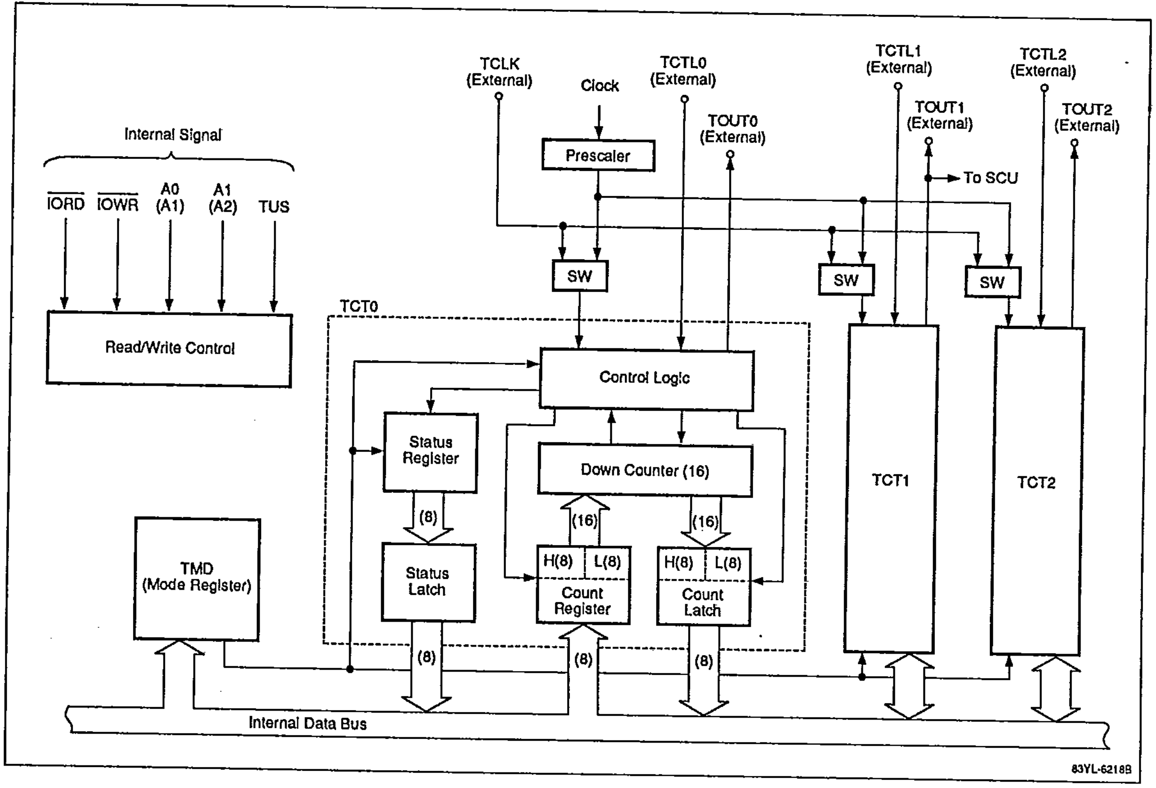
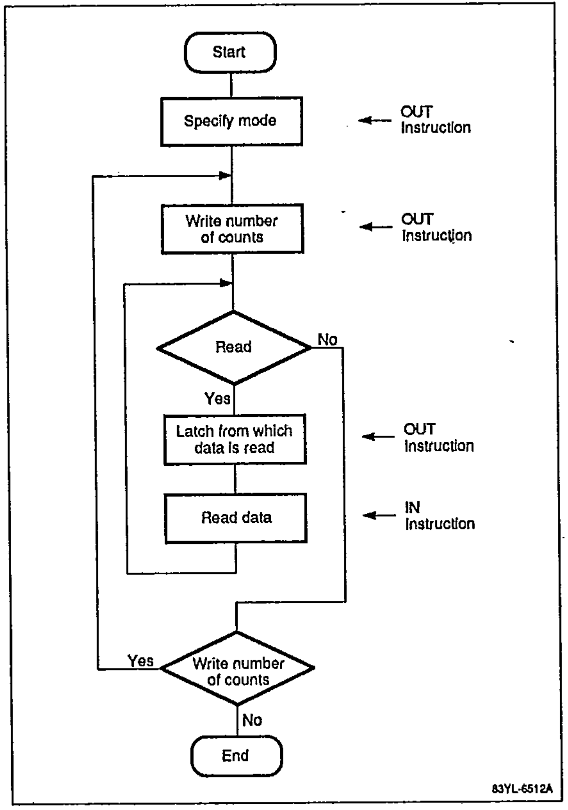


Figure 64. TCU Operating Procedure



TCU Commands

The TCU is programmed by issuing I/O instructions to the I/O port addresses programmed in the OPHA and TULA registers. The individual TCU registers are selected by address bits A_2 and A_1 or (A_1) and (A_0) as follows.

A_2 (A_1)	A_1 (A_0)	Register	Operation
0	0	TCT0 TST0	Read/write Read
0	1	TCT1 TST1	Read/write Read
1	0	TCT2 TST2	Read/write Read
1	1	TMD	Write

Timer Mode Register (TMD)

The TMD register selects the operating mode for each timer/counter and issues the latch command for one or more timer/counters. Figures 65, 66, and 67 show three configurations of the TMD register.

Figure 65. TMD Register; Mode Word

SC	RWM	CMODE	BD

7 0

SC	**Counter**
00	TCT0
01	TCT1
10	TCT2
11	Multiple latch command

RWM	**Read/Write Mode**
00	Counter latch command
01	Lower byte only
10	Upper byte only
11	Lower byte followed by upper byte

CMODE	**Count Mode**
000	Mode 0
001	Mode 1
x10	Mode 2
x11	Mode 3
100	Mode 4
101	Mode 5

BD	**Count**
0	Binary count
1	BCD count

x = Don't care.

Figure 66. TMD Register; Count Latch Command

SC	0	0	0	0	0	0

7 0

SC	**Counter To Be Latched**
00	TCT0
01	TCT1
10	TCT2

Figure 67. TMD Register; Multiple Latch Command

1	1	CL	SL	CT2	CT1	CT0	0

7 0

CL	**Latches Count Data**
0	Yes
1	No

SL	**Latches Status**
0	Yes
1	No

CTn	**Selects Counter TCTn**
0	No
1	Yes

T-49-17-15

Timer/Counter Registers (TCT)

Writes to the timer/counter registers (TCT0-TCT2) stores the new count in the appropriate timer/counter. The count latch command is used before reading count data to latch the current count and prevent inaccuracies.

Timer Status Registers (TST)

The timer status registers (TST0-TST2) contain status information for the specified counter. See figure 68. The latch command is used to latch the appropriate counter status before reading status information. If both status and counter data are latched for a counter, the first read operation returns the status data and subsequent read operations obtain the count data.

Figure 68. Timer Status Registers (TSTn)

OL	NC	RWM	CMODE	BD
7				0

OL	**TOUTn Level**
0	Low
1	High
NC	**Null Count**
0	Valid
1	Invalid
RWM	**Read/Write Mode**
	Same as TMD register.
CMODE	**Count Mode**
	Same as TMD register.
BD	**Count**
	Same as TMD register.

Count Modes

There are six programmable timer/counter modes. The timing waveforms for these modes are shown in figure 69.

Mode 0 (Interrupt on End of Count). In mode 0, TOUT changes from low to high level when the specified count is reached. This mode is available on all timer/counters.

Mode 1 (Retriggerable One-Shot). In mode 1, a low-level, one-shot pulse triggered by TCTL is output from the TOUT pin.

Mode 2 (Rate Generator). In mode 2, TOUT cyclically goes low for one clock period when the counter reaches the 0001H count. A counter in this mode operates as a frequency divider.

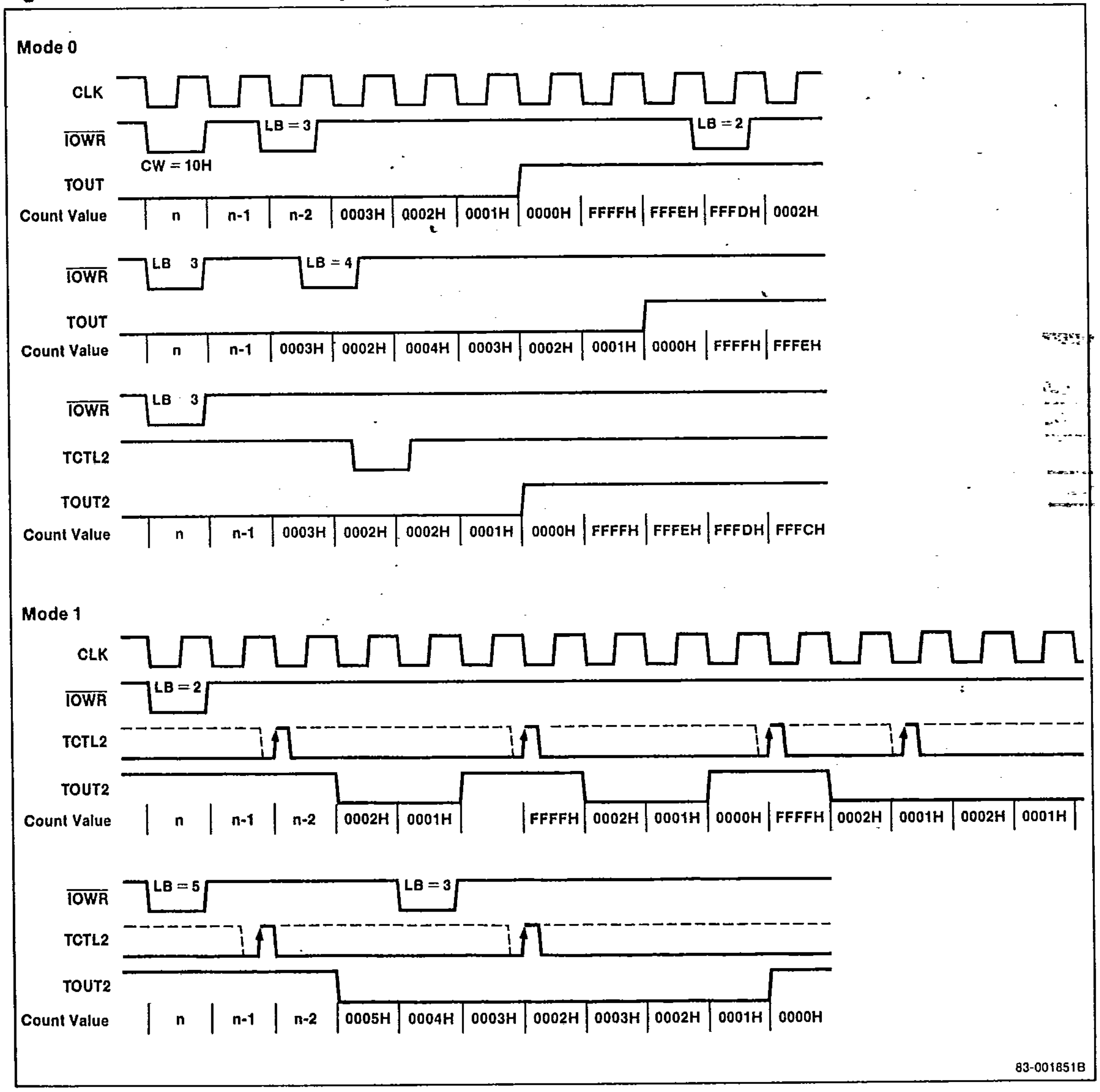
Mode 3 (Square-Wave Generator). Mode 3 is a frequency divider similar to mode 2, but the output has a symmetrical duty cycle.

Mode 4 (Software-Triggered Strobe). In mode 4, when the specified count is reached, TOUT goes low for the duration of one clock pulse.

Mode 5 (Hardware-Triggered Strobe). Mode 5 is similar to mode 4 except that operation is triggered by the TCTL input and can be retriggered.

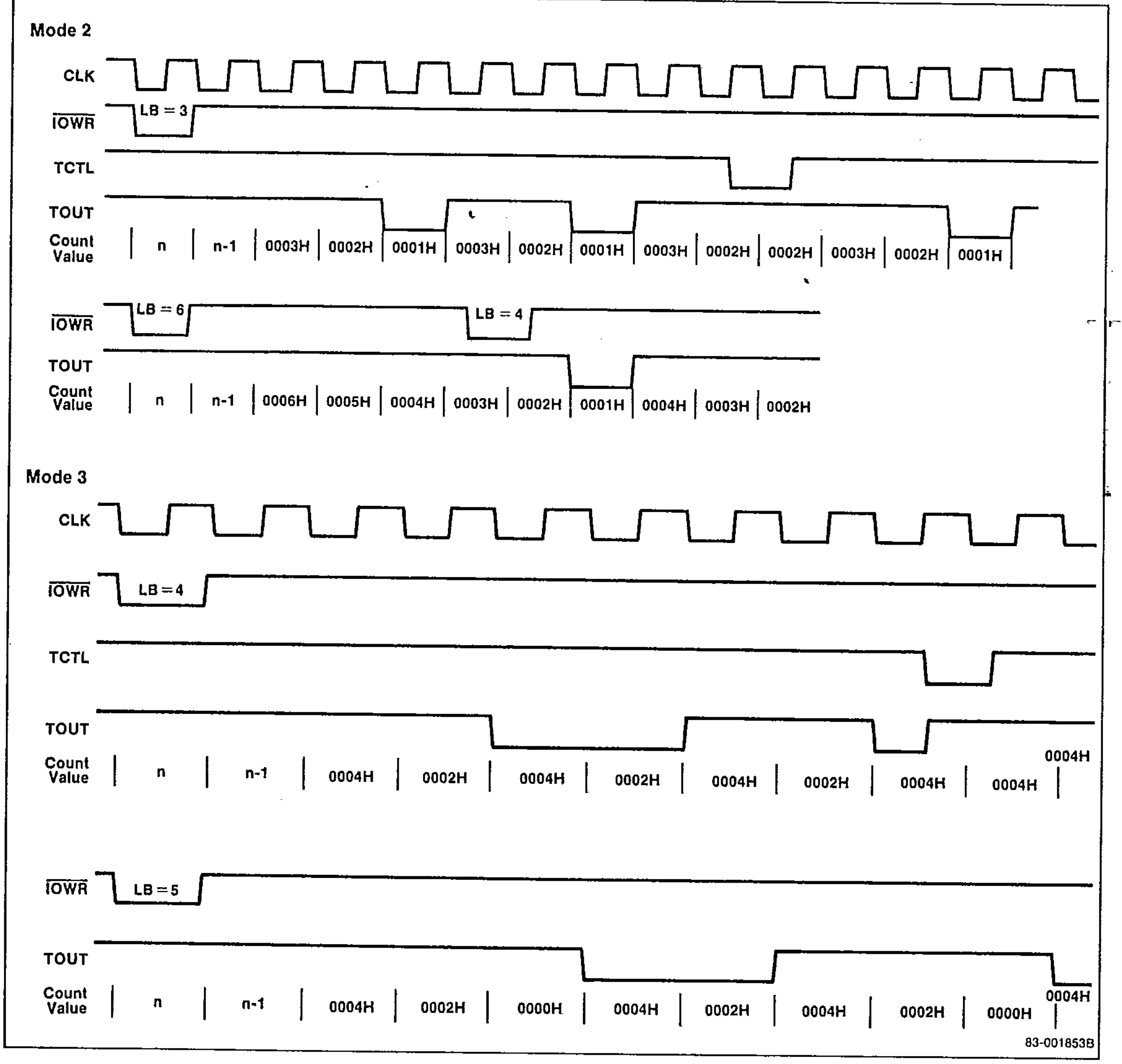
T-49-17-15

Figure 69. Timer Counter Unit (TCU) Waveforms (Sheet 1 of 3)



T-49-17-15

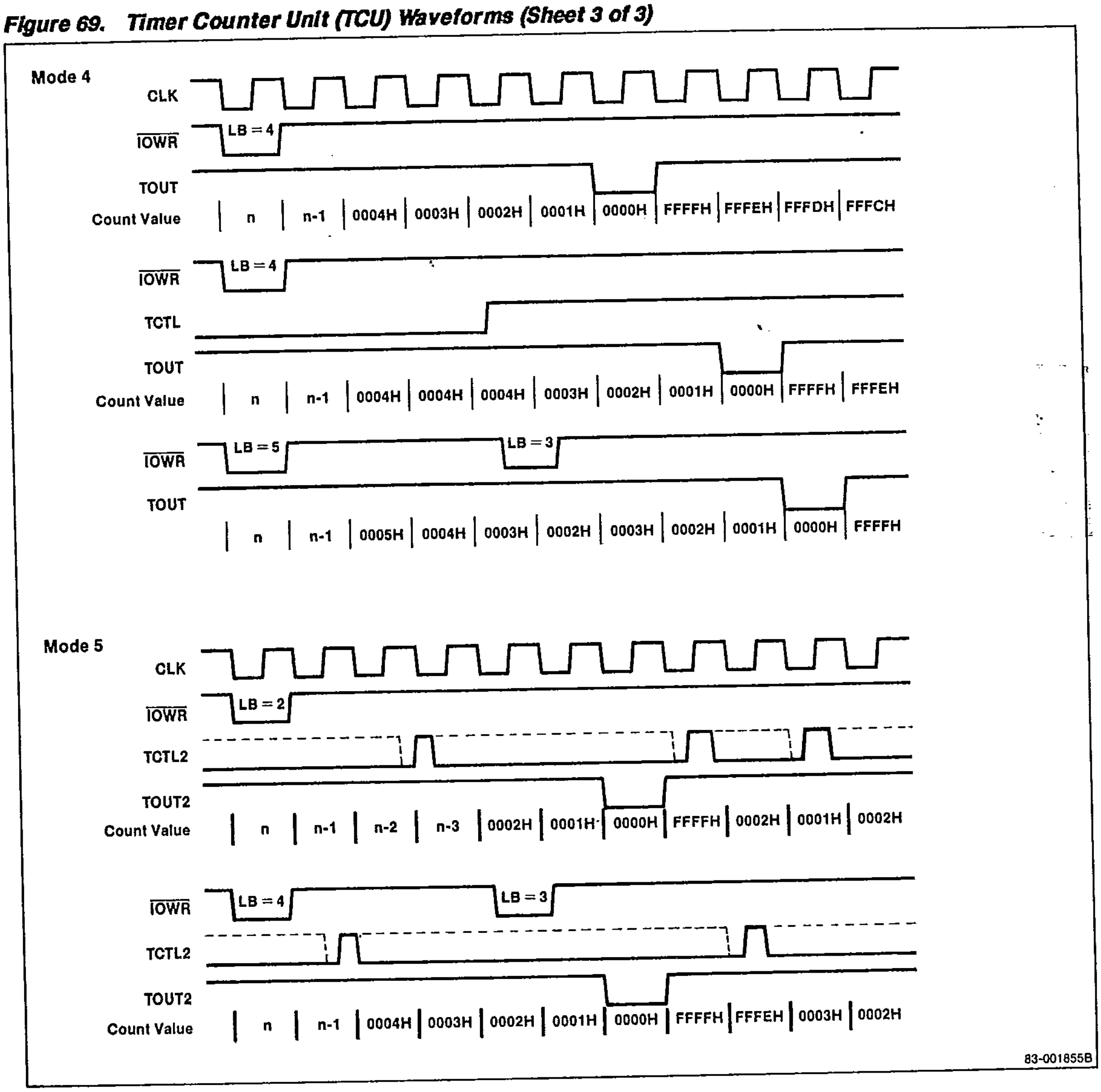
Figure 69. Timer Counter Unit (TCU) Waveforms (Sheet 2 of 3)



3

T-49-17-15

Figure 69. Timer Counter Unit (TCU) Waveforms (Sheet 3 of 3)

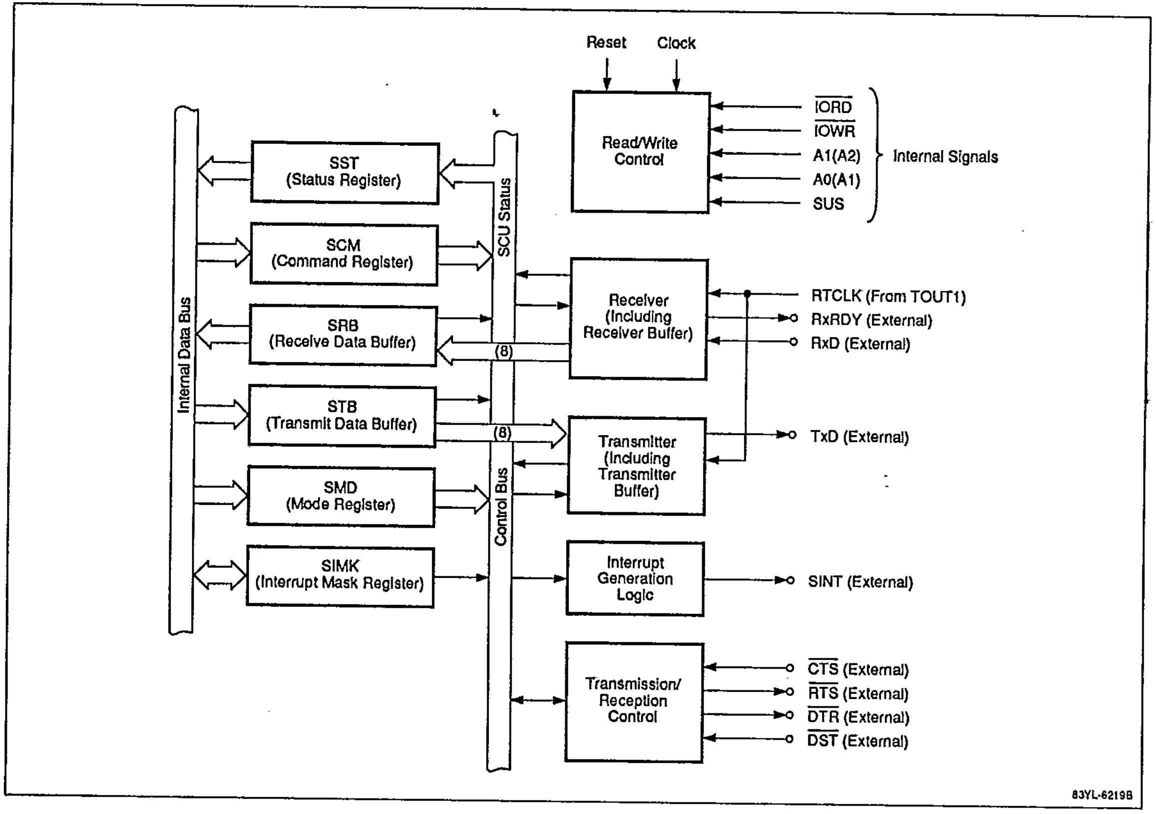


T-49-17-15

SERIAL CONTROL UNIT

The serial control unit (SCU) is a single asynchronous channel that performs serial communication between the V53 and an external device. The SCU is similar to the µPD71051 Serial Control Unit except for the lack of synchronous communication protocols. Figure 70 is a block diagram of the SCU.

Figure 70. SCU Block Diagram



The SCU has the following features.

- Full-duplex, asynchronous serial controller
- Clock rate divisor: 16 or 64
- Baud rates to 640 kb/s (external clock), 500 kb/s (internal clock)
- Dedicated baud-rate generator or can use timer 1
- Full modem signaling support (ATS, CTS, DSR, DTR)
- Character length: 7 or 8 bits
- Stop bit length: 1 or 2 bits
- Break transmission and detection
- Full-duplex, double-buffered transmitter/receiver
- Even, odd, or no parity
- Parity, overrun, and framing error detection
- Receiver-full/transmitter-empty interrupt

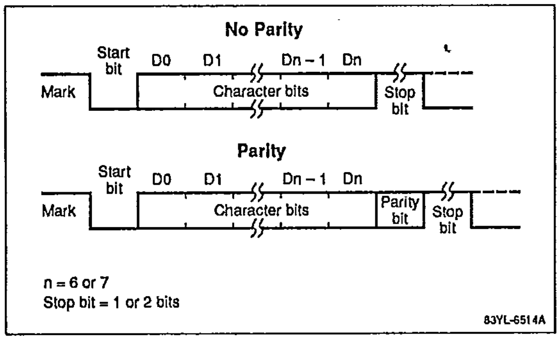
The SCU contains four separately addressable registers for reading/writing data, reading status, and controlling operation of the SCU. The serial receive buffer (SRB) and the serial transmit buffer (STB) store the incoming and outgoing character data. The serial status register (SST) allows software to determine the current state of both the transmitter and the receiver.

The serial command (SCM) and serial mode registers (SMD) determine the operating mode of the SCU while the serial interrupt mask register (SIMK) allows software control of the SCU receive and transmit interrupts.

Serial Data Format

Figure 71 shows the format of the serial data processed by the SCU. In this serial data, the character bits are transferred between the CPU and SCU. The start bit, parity bit, and stop bit(s) sandwiching the character bits are control information necessary for serial data communications. They are automatically appended when data is transmitted or deleted when data is received by the SCU.

Figure 71. Serial Data Format



Receiver Operation

While the RxD pin is high, the receiver is in an idle state. A transition on RxD from high to low indicates the start of new serial data. When a complete character has been received, it is transferred to the SRB register. The receive buffer ready (RBRDY) bit in the SST register is set and (if unmasked) an interrupt is generated. The SST also latches any parity, overrun, or framing errors at this time.

The receiver detects a break condition when a null character with zero parity is received. The BRK bit is set for as long as the subsequent receive data is low and resets when RxD returns to a high level.

Transmitter Operation

TxD is kept high while the STB register is empty. When the transmitter is enabled and a character is written to the STB register, the data is converted to serial format and output on the TxD pin. The start bit indicates the start of the transmission and is followed by the character stream (LSB to MSB) and an optional parity bit. One or two stop bits are then appended, depending on the programmed mode. When the character has been transferred from the STB, the TBRDY bit in the SST is set and if unmasked, a transmit buffer empty interrupt is generated.

Serial data can be transmitted and received by polling the SST register and checking the TBRDY or RBRDY flags. Data can also be transmitted and received by SCU-generated interrupts. The SCU generates an interrupt in either of these conditions:

(1) The receiver is enabled, the SRB is full, and receive interrupts are unmasked.

(2) The transmitter is enabled, the STB is empty, and transmit interrupts are unmasked.

SCU Registers and Commands

I/O instructions to the I/O addresses selected by the OPHA and SULA registers are used to read/write the SCU registers. Address bits A_2 and A_1 (or A_1 and A_0) and the read/write lines select one of the six internal registers as shown below.

A_2 (A_1)	A_1 (A_0)	Register	Operation
0	0	SRB	Read
0	0	STB	Write
0	1	SST	Read
0	1	SCM	Write
1	0	SMD	Write
1	1	SIMK	Read/write

The baud rate counter (BRC) register is fixed at address FFE9H in the system I/O area.

The SRB and STB are 8-bit registers. When the character length is 7 bits, the lower 7 bits of the SRB register are valid and bit 7 is cleared to 0. If programmed for 7-bit characters, bit 7 of the STB is ignored.

The SST register (figure 72) contains the status of the transmit and receive data buffers and the error flags. Error flags are persistent. Once an error flag is set, it remains set until a clear error flags command is issued.

SCU Initialization

After a hardware reset, the SCU is set to the following condition.

Baud rate factor	x64
Character length	7 bits
Stop bit	1 bit
Transmit/receive	Disabled
Break detection	No
Errors	No
$\overline{RTS}$, $\overline{DTR}$ pins	High level

Figure 72. Serial Status Register (SST)

7							0
DSR	BKD	FE	OVE	PE	1	RBRDY	TBRDY

DSR	**$\overline{\text{DSR}}$ Input Pin**
0	High level
1	Low level
BKD	**Break Detection**
0	Normal reception
1	Break status detected
FE	**Framing Error**
0	No error
1	Error
OVE	**Overrun Error**
0	No error
1	Error
PE	**Parity Error**
0	No error
1	Error
RBRDY	**Receive Data Buffer**
0	SRB empty
1	SRB full
TBRDY	**Transmit Data Buffer**
0	STB full
1	STB empty

The SCM register (figure 73) stores the command word that controls transmission, reception, error flag reset and break transmission.

The SMD register (figure 74) stores the mode word that determines serial characteristics such as baud rate divisor, parity, character length, and stop bit length.

Initialization software should first program the SMD register followed by the SCM register. Unlike the µPD71051, the SMD register can be modified anytime without resetting the SCU.

Figure 73. Serial Command Register (SCM)

7							0
—	—	RTS	ECL	SBRK	RE	DTR	TE

RTS	**Controls $\overline{\text{RTS}}$ Output Pin**
0	High level
1	Low level
ECL	**Clears Error Flags**
0	No operation
1	Clears error flags
SBRK	**Break Transmission**
0	TxD pin operates normally
1	TxD pin outputs low level
RE	**Enables/Disables Reception**
0	Disables
1	Enables
DTR	**Controls $\overline{\text{DTR}}$ Pin**
0	High level
1	Low level
TE	**Enables/Disables Transmission**
0	Disables
1	Enables

Figure 74. Serial Mode Register (SMD)

7			0
STL	PS	CL	BF

STL	**Number of Stop Bits**
x0	Illegal
01	1 stop bit
11	2 stop bits
PS	**Parity Selection**
x0	Parity disabled
01	Odd parity
11	Even parity
CL	**Character Length**
x0	Illegal
10	7 bits
11	8 bits
BF	**Baud Rate**
0x	Illegal
10	RTCLK frequency/16
11	RTCLK frequency 64

The SIMK register (figure 75) controls the occurrence of RBRDY and TBRDY interrupts. When an interrupt is masked, it is prevented from propagating to the interrupt control unit.

Figure 75. Serial Interrupt Mask Register (SIMK)

7							0
—	—	—	—	—	—	TM	RM

TM	TBRDY Interrupt Mask
0	Unmasked
1	Masked
RM	**RBRDY Interrupt Mask**
0	Unmasked
1	Masked

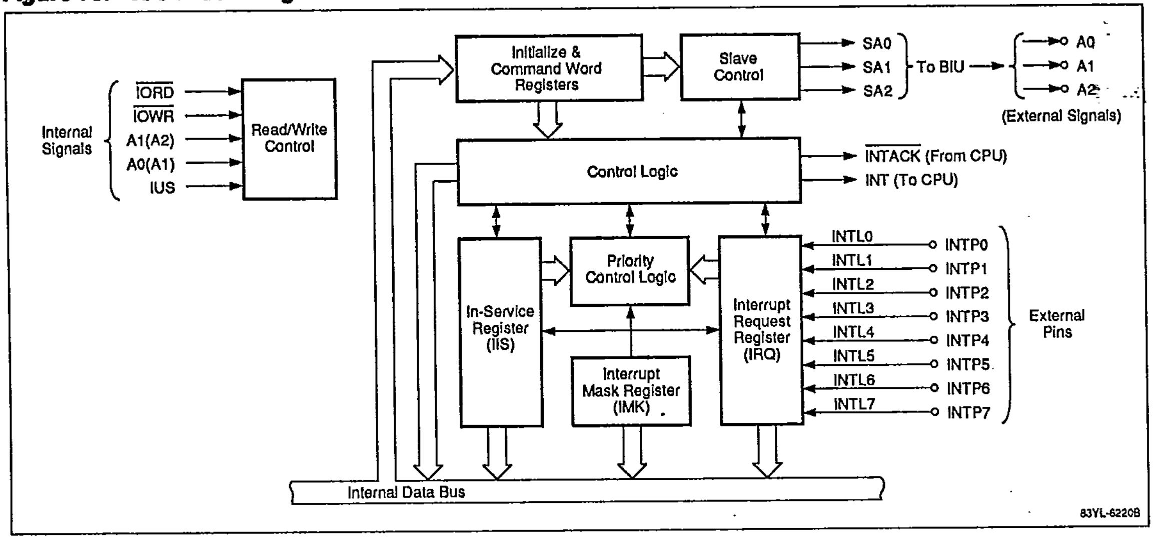
Baud Rate Clock

The baud rate clock may come from either of two sources: the internal baud rate generator or timer 1. The internal baud rate generator is discussed in the System I/O section, and timer 1 is described in the TCU section. The SCTL system I/O register controls the selection of the baud rate clock.

INTERRUPT CONTROL UNIT

The interrupt control unit (ICU) is a programmable interrupt controller equivalent to the μPD71059. The ICU arbitrates up to eight interrupt inputs, generates a CPU interrupt request, and outputs the interrupt vector number on the internal data bus during an interrupt acknowledge cycle. Cascading up to seven external slave μPD71059 interrupt controllers permits the V53 to support up to 56 interrupt sources. Figure 76 is the block diagram for the ICU.

Figure 76. ICU Block Diagram



To reduce current drain in the standby modes, the V53 does not have internal pullup resistors on the INTP0-INTP7 pins. This is different from the μPD71059 and V40/V50.

The ICU has the following features.

- Eight external interrupt request inputs
- Cascadable with μPD71059 interrupt controllers
- Programmable edge- or level-triggered interrupts (TCU, edge-triggered only)
- Individually maskable interrupt requests
- Programmable interrupt request priority
- Polling mode

ICU Registers

Use I/O instructions to the I/O addresses selected by the OPHA and IULA registers to read from and write to the ICU registers. Address bit A_1 and the command word select an ICU internal register. See table 7.

Table 7. ICU Register Selection

	A_1 (A_0)	Other Condition	Operation
Read	0	IMD selects IRQ	CPU ← IRQ data
	0	IMD selects IIS	CPU ← IIS data
	0	*Polling phase	CPU ← Polling data
	1	—	CPU ← IMKW
Write	0	D4 = 1	CPU → IIW1
	0	D4 = 0 and D3 = 0	CPU → IPFW
	0	D4 = 0 and D3 = 1	CPU → IMDW
	1	During initialization	CPU → IIW2
	1		CPU → IIW3
	1		CPU → IIW4
	1	After initialization	CPU → IMKW

* In the polling phase, polling data has priority over the contents of the IRQ or IIS register when read.

Initializing the ICU

The ICU is always used to service maskable interrupts in a V53 system. Prior to accepting maskable interrupts, the ICU must first be initialized. See figure 77. Note that RESET does not initialize the ICU.

Interrupt Initialization Words 1-4. Words IIW1-IIW4 (figures 78-81) indicate whether external µPD71059s are connected as slaves, select the base interrupt vector, and select edge- or level-triggered inputs for INT1-INT7. Interrupt sources from the TCU are fixed as edge-triggering. INT0 is internally connected to TOUT0, and INT2 may be connected to TOUT1 by the IRSW field in the OPCN.

The initialization words are written in consecutive order starting with IIW1. IIW2 sets the interrupt vector. IIW3 specifies which interrupts are connected to slaves. IIW3 is only required in extended systems. The ICU will only expect to receive IIW3 if SNGL = 0 (bit D_1 of IIW1). IIW4 is only written if II4 = 1 (bit D_0 of IIW1).

Figure 77. Initialization Sequence

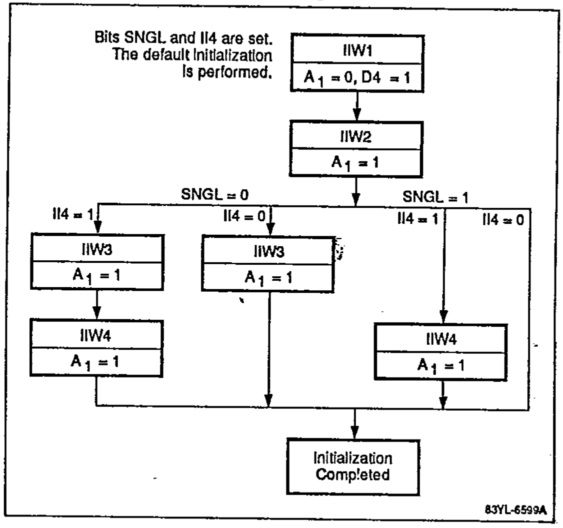


Figure 78. ICU Initialize, Word 1 (IIW1)

—	—	—	1	LEV	—	SNGL	II4

D7 D0

LEV	Input Trigger Mode
0	Rising-edge trigger
1	High-level trigger

SNGL	Mode
0	Expanded mode (slave controllers)
1	Single mode (no slave controllers)

II4	Write to W4
0	IIW4 not required
1	IIW4 required

Figure 79. ICU Initialize, Word 2 (IIW2)

V_7	V_6	V_5	V_4	V_3	—	—	—

D7 D0

V_7-V_3 = Higher 5 bits of interrupt vector number

Figure 80. ICU Initialize, Word 3 (IIW3)

S_7	S_6	S_5	S_4	S_3	S_2	S_1	0
D7							D0

S_n	Slave Connection
0	INTn is not a slave input
1	INTn is a slave input

Figure 81. ICU Initialize, Word 4 (IIW4)

0	0	0	EXTN	—	—	SFI	1
D7							D0

EXTN	External Nesting Mode
0	Normal
1	Expanded
SFI	**Self-Finish Interrupt**
0	FI command mode
1	Self-finish mode

Command Words. The interrupt mask word (MKW) contains programmable mask bits for each of the eight interrupt inputs. The interrupt priority and finish word (IPFW) is used by the interrupt handler to terminate processing of an interrupt or change interrupt priorities. The interrupt mode word (IMDW) selects the polling register, interrupt request (IRQ) or interrupt in-service (IIS) register, and the nesting mode. See figures 82-84.

Figure 82. Command Word IMKW

M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0
D7							D0

M_n	Interrupt Request Mask
0	INTn not masked
1	INTn masked

Figure 83. Command Word IPFW

RP	SIL	FI	0	0	IL2	IL1	IL0
D7							D0

RP	Rotate Priority
0	No rotation
1	Rotation
SIL	**Level**
0	Not specified
1	Specified
FI	**Finish Interrupt**
0	Non-FI command
1	FI command
IL2-IL0	**Interrupt Level**
000	INT0
001	INT1
010	INT2
011	INT3
100	INT4
101	INT5
110	INT6
111	INT7

Figure 84. Command Word IMDW

—	SNM	EXCN	0	1	POL	SR	IS/IR
D7							D0

SNM	EXCN	Nesting Mode 2
0	—	No operation
1	0	Release exceptional nesting mode
1	1	Set exceptional nesting mode
POL		**Polling Mode**
0		No operation
1		Polling command
SR	**IS/IR**	**Register to Be Read**
0	—	No operation
1	0	Interrupt request register (IRQ)
1	1	Interrupt in-service register (IIS)

µPD71059 Cascade Connection

To increase the number of maskable interrupts, up to seven slave µPD71059 interrupt controllers can be cascaded. During cascade operation, each slave µPD71059 INT output is routed to one of the V53 INTP inputs.

During the second interrupt acknowledge bus cycle, the ICU places the slave address on the address lines AD_{10}-AD_8. Each slave compares this address with the slave address programmed using interrupt initialization word 3 (IIW3). If the same, the slave will place the interrupt vector on pins AD_7-AD_0 during the second interrupt acknowledge bus cycle.

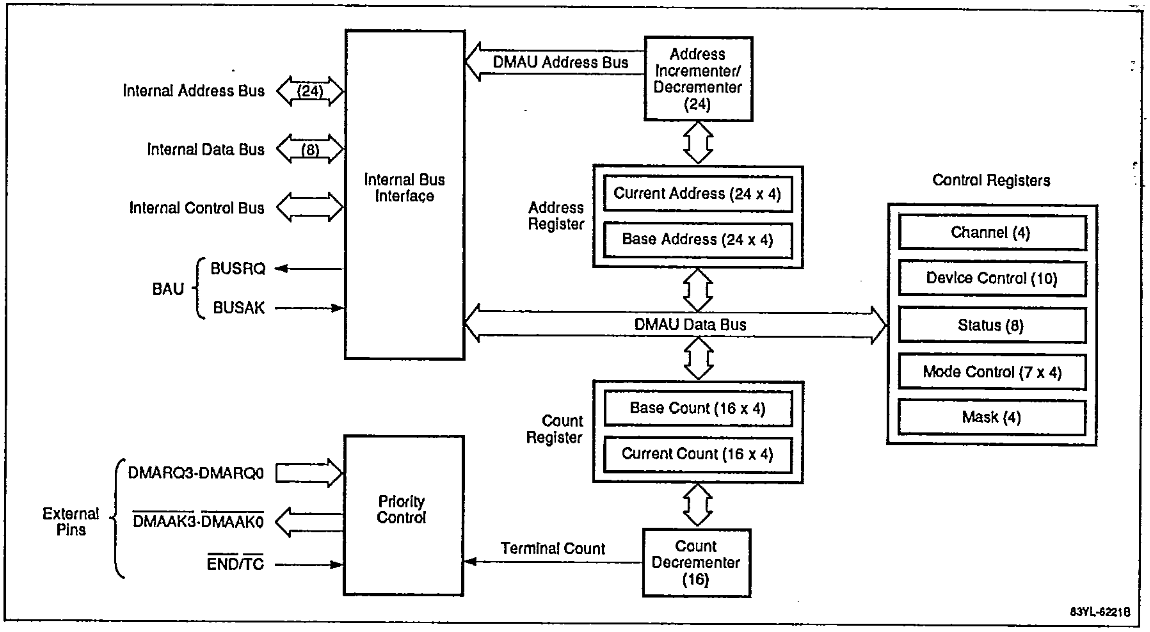
DMA CONTROL UNIT

The DMA control unit (DMAU) is a high-speed DMA controller compatible with the µPD71071 and µPD71037 DMA controllers. The DMAU has four independent DMA channels and performs high-speed data transfers between memory and external peripheral devices at speeds as high as 4M words/second in a 16-MHz system. Figure 85 is the block diagram for the DMAU.

The DMAU has the following features.

- Four independent DMA channels
- µPD71037 or µPD71071 compatibility modes
- Cascade mode for slave DMA controllers
- 24-bit address registers
- 16-bit transfer count registers
- Single, demand, and block transfer modes
- Autoinitialization
- Address increment/decrement
- Fixed/rotating channel priorities
- TC output at transfer end
- Forced termination of service by END input

Figure 85. DMAU Block Diagram



µPD71071 and µPD71037 Mode Comparison

The DMAU has two operating modes selected by the SCTL system control register. Respectively, the µPD71071 and µPD71037 modes offer hardware and software compatibility with existing systems based on the µPD71071 DMA controller (also the V40/V50 microprocessor) and the µPD8237 DMA controller.

In applications where DMA software compatibility is not an issue, programming flexibility is greater in the µPD71071 mode. However, the software DMA request capability of the µPD71037 mode is often useful.

The following compares the major functional differences between the two modes.

Function	µPD71037 Mode	µPD71071 Mode
DMA channel selection	Mode control register by write data (operand); other registers have a unique address	Referenced by channel register (DCH)
Base and current register access	Consecutive 8-bit quantities	16-bit quantities
Base registers	Write only	Read and write
DMA termination	Bus release mode	Bus release and bus hold modes
Software DMA requests	Yes	No
DMA transfers	Byte	Byte or word

The DMAU is intended for high-speed data transfers between memory and peripherals with minimum latency. Neither mode provides memory-to-memory DMA transfers because the powerful string moves of the CPU can accomplish block memory transfers as fast as dedicated DMA hardware could. The DMAU does not provide compressed timing as do the µPD71071 and µPD71037.

Master/Slave Mode

The DMAU operates in either master or slave mode. In slave mode, the DMAU samples the four DMARQ input pins every clock. If one or more inputs are active, the corresponding DMA request bits are set and the DMAU sends a bus request to the BAU while continuing to sample the DMA request inputs.

After the BAU returns the DMA bus acknowledge signal, the DMAU stops DMA request sampling, selects the DMA channel with the highest priority, and enters the bus master mode to perform the DMA transfer. While in the bus master mode, the DMAU controls the external bus and performs DMA transfers based on the preprogrammed channel information.

See figure 45 and the associated text for a detailed description of DMA bus cycles.

Terminal Count

The DMAU ends DMA service when the terminal count condition is generated or when the END input is asserted. A terminal count (TC) is produced when the contents of the current count register underflows from zero. If autoinitialization is not enabled when DMA service terminates, the mask bit of the channel is set and the DMARQ input of that channel is masked. Otherwise, the current count and address registers are reloaded from the base registers, and new DMA transfers are again enabled.

DMA Transfer Type

The type of transfer the DMAU performs depends on the following conditions.

- Transfer direction (each channel)
- Bus mode
- Transfer mode (each channel)

Transfer Direction

All DMA transfers use memory as a reference point. Therefore, a DMA read operation (figure 86) transfers data from memory to I/O port and writes the data into memory. During memory-to-I/O transfer, the DMA mode register (DMD) is used to select the transfer directions for each channel and activate the appropriate control signals.

Operation	Transfer	Signals Activated
DMA read	Memory to I/O	$\overline{\text{IOWR}}$, $\overline{\text{MRD}}$
DMA write	I/O to memory	$\overline{\text{IORD}}$, $\overline{\text{MWR}}$
DMA verify	No transfer	Addresses only

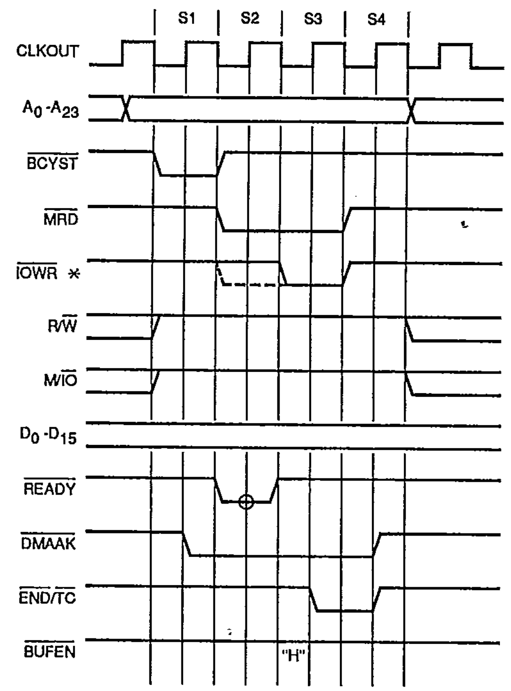
Bus Mode

The two available modes for determining how the DMAU releases the CPU bus are bus release and bus hold. In µPD71037 mode, the DMAU always functions in bus release mode. In µPD71071 mode, the DMAU is programmable for bus release or bus hold mode via the DMA device control (DDC) register.

In bus release mode, bus control is always relinquished each time the service has completed. Therefore, if multiple DMA requests are generated simultaneously, a bus cycle other than that for the DMAU is inserted between consecutive DMA services (see figure 87). Consequently, in certain applications DMA response may be delayed. However, bus release mode gives better assurance that the CPU will continue to execute programs in DMA intensive environments.

In bus hold mode, if another DMA request is generated before the end of one service, that request can be serviced without the DMAU relinquishing the bus. However, the same channel cannot be serviced consecutively. This mode provides better DMA response but may prevent CPU bus activity for extended periods of time.

Figure 86. Typical Memory-to-I/O DMA Cycle



* Dotted line shows effect of selecting "Early Write" timing in DDC register or command register.

83YL-6515A

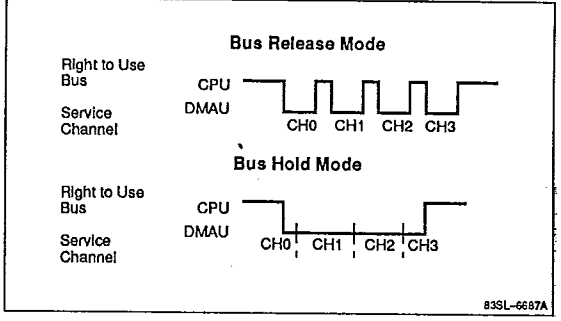
Transfer Modes

The DMAU has three transfer modes as listed below. In µPD71071 mode, bits 6 and 7 (TMODE) of the mode control register (DMD) select the transfer mode. In µPD71037 mode, bits 6 and 7 of the channel mode register specify the mode. Transfer mode operation is the same in both µPD71071 and µPD71037 modes.

Transfer Mode	Termination Conditions
Single	After each byte/word transfer END input Terminal count
Demand	END input Terminal count Service channel DMARQ dropped Generation of a higher priority DMARQ (bus hold mode)
Block	END input Terminal count

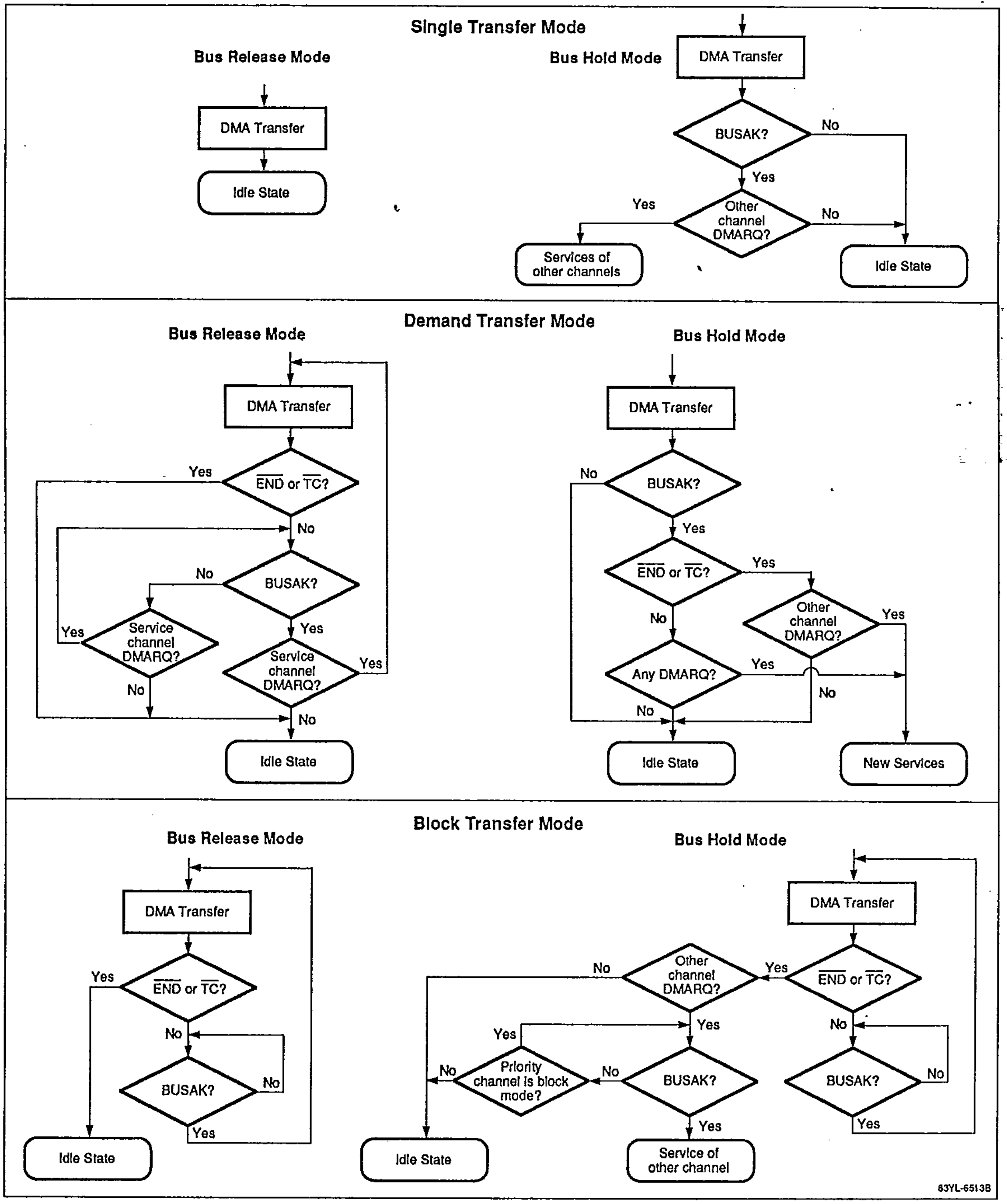
The operation of single, demand, and block transfers depends on whether the DMAU is in bus release or bus hold mode. Figure 88 shows the operations flow for the six possible transfer and bus mode operations in DMA transfer.

Figure 87. Bus Modes



T-49-17-15

Figure 88. Transfer Modes



Single Transfer Mode. In bus release mode, when a channel completes transfer of a single byte or word, the DMAU enters the slave mode regardless of the state of DMA request inputs. In this manner, other lower priority bus masters can access the bus.

In bus hold mode (µPD71071 mode only), when a channel completes transfer of a single byte or word, the DMAU terminates the channel's service even if the DMARQ request signal is asserted. The DMAU will then service any other requesting channel. If there are no requests from any other DMA channels, the DMAU releases the bus and enters the idle state.

Demand Transfer Mode. In bus release mode, the currently active channel continues to transfer data as long as the DMA request of that channel is active, even though other DMA channels are issuing higher priority requests. When the DMA request of the serviced channel becomes inactive, the DMAU releases the bus and enters the idle state.

In bus hold mode (not available in µPD71037 mode), when the active channel completes a single transfer, the DMAU checks the other DMA request lines without ending the current service. If there is a higher priority DMA request, the DMAU stops the service of the current channel and starts servicing the highest priority channel requesting service. If there is no higher request than the current one, the DMAU continues to service the currently active channel. Lower priority DMA requests are honored without releasing the bus after the current channel service is complete.

Block Transfer Mode. In bus release mode, the current channel continues DMA transfers until a terminal count or the external $\overline{\text{END}}$ input becomes active. During this time, the DMAU ignores all other DMA requests. After completion of the block transfer, the DMAU releases the bus and enters the idle state, even if DMA requests from other channels are active.

In bus hold mode (µPD71071 mode only), the current channel transfers data until an internal or external $\overline{\text{END}}$ signal becomes active. When the service is complete, the DMAU checks all DMA requests without releasing the bus. If there is an active request, the DMAU immediately begins servicing the request. The DMAU releases the bus after it honors all DMA requests or a higher priority bus master requests the bus.

Autoinitialize

This function is enabled by programming the mode register (µPD71071 and µPD71037 modes).

When a mode register enables autoinitialize for a channel, the DMAU automatically reinitializes the address and count registers when $\overline{\text{END}}$ is asserted or the terminal count condition is reached. The contents of the base address and base count registers are transferred to the current address and current count registers, and the applicable bit of the mask register remains cleared.

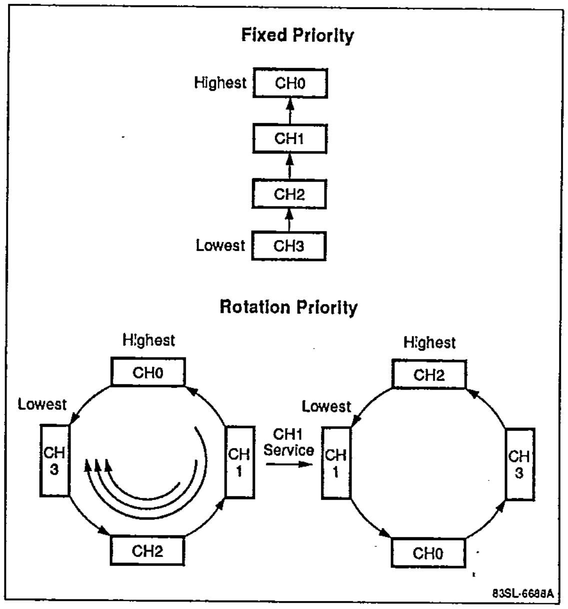
Channel Priority

Each of the four DMAU channels is assigned a priority. When multiple DMA requests from several channels occur simultaneously, the channel with the highest priority will be serviced first.

The device control register selects one of two priority schemes: fixed or rotating (figure 89). In fixed priority, channel 0 is assigned the highest priority, and channel 3, the lowest. In rotating priority, priority order is rotated after each service so that the channel last serviced receives the lowest priority. This method prevents the exclusive servicing of higher priority channels and the lockout of lower priority DMA channels.

The rotating priority feature is selected by programming the DMA device control (DDC) register in µPD71071 mode or by a write to the command register in µPD71037 mode.

Figure 89. ***Priority Order***

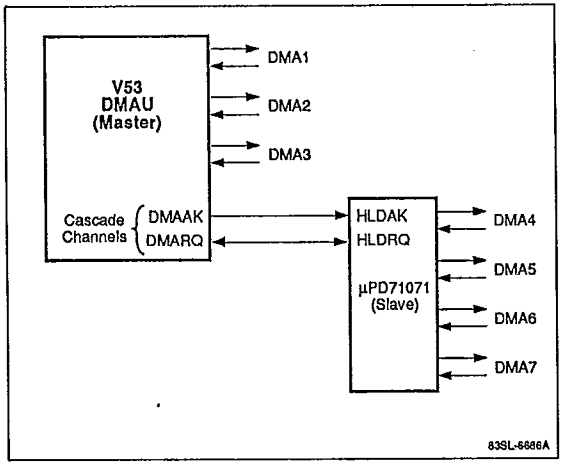


Cascade Connection

Slave DMA controllers can be cascaded to easily expand the system DMA channel capacity to 16 DMA channels. Figure 90 shows an example of cascade connection. During cascade operation, the DMAU acts as a mediator between the BAU and the slave DMA controller. During DMA cascade mode operation, it is the responsibility of external logic to isolate the cascade bus master from the V53 control outputs. These outputs are listed near the beginning of this document.

The DMAU always operates in the bus release mode while a cascade channel is in service, even when the bus hold mode is programmed. Other DMA requests are held pending while a slave DMA controller channel is in service. When the cascaded device ends service and moves into the idle state, the DMAU also moves to the idle state and releases the bus. The DMAU continues to operate normally with the other noncascaded channels.

Figure 90. μPD71071 Cascade Example

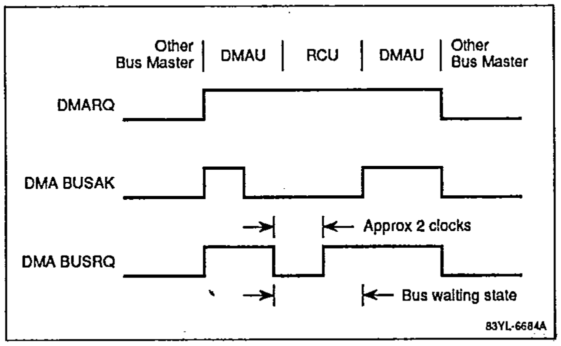


Bus Waiting Operation

The DMAU automatically performs a bus waiting operation (figure 91) whenever the REFU refresh request queue fills. When the DMA bus acknowledge goes inactive, the DMAU enters the bus waiting mode and inactivates the DMA bus request signal. Control of the bus is then transferred to the higher priority REFU by the BAU.

Two clocks later, the DMAU reasserts its internal DMA bus request. The bus waiting mode is continued until the DMA bus acknowledge signal again becomes active and the interrupted DMA service is immediately restarted.

Figure 91. Bus Waiting Operation



Address and Count Registers

Each DMA channel has a 24-bit base address register and a 24-bit current address register. In addition, each channel also has its own 16-bit current count register and base count register. The base registers hold a value determined by the CPU and transfer this value to the current registers during autoinitialization. These registers are available in both μPD71071 mode and μPD71037 mode, but the method of accessing these registers changes with compatibility mode.

The BNKR registers extend the μPD71037 mode addresses from 16 to 24 bits. In μPD71071 mode, the count register and lower word of the address registers can be accessed in 16-bit quantities. In μPD71037 mode, these registers must be accessed in 8-bit quantities.

Programming the DMAU

To prepare a channel for DMA transfer, the following characteristics must be programmed.

- Starting address for the transfer
- Transfer count
- DMA operating mode
- Transfer size (byte/word in μPD71071 mode)

The contents of the OPHA and DULA registers determine the base I/O port address of DMAU. Addresses A_3-A_0 are used to select a particular register. There are two register sets, one for μPD71071 mode and the other for μPD71037 mode.

μPD71071 Mode

The μPD71071 mode is selected by programming the DMAU bit of the SCTL register to zero. The register set for this mode (table 7) is mapped into A_3-A_0 regardless of the IOAG value in the SCTL register.

Table 7. Register Selection (μPD71071 Mode)

A_3-A_0	Address	Register	Operation	Notes
0000	0H	DICM	Write	1
0001	1H	DCH	Read/Write	1
0010	2H	DBC/DCC (low)	Read/Write	2
0011	3H	DBC/DCC (high)	Read/Write	2
0100	4H	DBA/DCA (low)	Read/Write	2
0101	5H	DBA/DCA (high)	Read/Write	2
0110	6H	DBA/DCA (upper)	Read/Write	1, 2
0111	7H	Reserved	—	
1000	8H	DDC (low)	Read/Write	
1001	9H	DDC (high)	Read/Write	
1010	AH	DMD	Read/Write	1, 2
1011	BH	DST	Read	1
1100	CH	Reserved	—	
1101	DH	Reserved	—	
1110	EH	Reserved	—	
1111	FH	DMK	Read/Write	1

Notes:

(1) Register can be accessed only with byte In/Out instructions. All others can be accessed with 16-bit In/Out instructions.

(2) There are four such registers, one for each DMA channel. The particular register accessed is determined by the DCH register.

DMAU Registers in μPD71071 Mode

Initialize. The DMA initialize command register (DICM) performs a software reset of the DMAU. The DICM is accessed using the byte OUT instruction. See figure 92.

The DMAU initializes the registers as follows.

Register	Name	Operation
DICM	Initialize	Clear
DCH	Channel	Select channel 0
DBC, DCC	Count	No change
DBA, DCA	Address	No change
DDC	Device control	Clear
DMD	Mode control	Clear
DST	Status	Clear
DMK	Mask	Set (mask all channels)

Figure 92. DMA Initialize Command Register (DICM); μPD71071 Mode

—	—	—	—	—	—	—	RES

7 Address 0H 0
Byte OUT Instruction

RES	Reset
0	No operation
1	Reset DMAU

Channel Register. Writes to the DMA channel register (DCH) select one of the four DMA channels for programming and also the base/current registers. Reads of the DCH register return the currently selected channel and the register access mode. See figure 93.

Figure 93. DMA Channel Register (DCH); μPD71071 Mode

Channel Register Read

—	—	—	BASE	SEL3	SEL2	SEL1	SEL0

7 Address 1H 0
Byte IN Instruction

BASE	Access Conditions
0	Read: current only Write: base and current
1	Read/write: base only

SEL3-SEL0	Selected Channel
0001	0
0010	1
0100	2
1000	3

Channel Register Write

—	—	—	—	—	BASE	SELCH

7 Address 1H 0
Byte OUT Instruction

BASE	Access Conditions
0	Read: current only Write: base and current
1	Read/write: base only

SELCH	Selected Channel
00	Channel 0
01	Channel 1
10	Channel 2
11	Channel 3

Count Registers. When bit 2 of the DCH register is cleared, a write to the DMA count register (figure 94) updates both the DMA base count (DBC) and the DMA current count (DCC) registers with a new count. If bit 2 of the DCH register is set, a write to the DMA count register affects only the DBC register.

The DBC register holds the initial count value until a new count is specified. If autoinitialization is enabled, this value is transferred to the DCC register when a terminal count or END condition occurs. For each DMA transfer, the current count register is decremented by 1. The count value loaded into the DBC/DCC register is 1 less than the desired transfer count.

Figure 94. DMA Count Registers (DBC, DCC); µPD71071 Mode

C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0

7 Address 2H 0
IN/OUT Instruction

C_{15}	C_{14}	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8

7 Address 3H 0
IN/OUT Instruction

Address Register. Use either byte or word I/O instructions with the lower 2 bytes (4H and 5H) of the DMA address register (figure 95). However, byte I/O instructions must be used to access the high-order byte (6H) of this register. When bit 2 of the channel register is cleared, a write to the DMA address register updates both the DMA base address (DBA) and the DMA current address (DCA) registers with the new address. If bit 2 of the DCH register is set, a write to the DMA address register affects only the DBA register.

The DBA register holds the starting address value until a new address is specified. This value is transferred to the DCA register automatically if autoinitialization is selected. For each DMA transfer, the current address register is updated by 2 during word transfers and by 1 during byte transfers.

Figure 95. DMA Address Registers (DBA, DCA); µPD71071 Mode

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

7 Address 4H 0
IN/OUT Instruction

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8

7 Address 5H 0
IN/OUT Instruction

A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	A_{17}	A_{16}

7 Address 6H 0
IN/OUT Instruction

Device Control Register. The DMA device control register (DDC) (figure 96) is used to program the DMA transfer characteristics common to all DMA channels. It controls the bus mode, write timing, priority logic, and enable/disable of the DMAU See figure 97.

Figure 96. DMA Device Control Register (DDC); µPD71071 Mode

—	—	EXW	ROT	—	DDMA	—	—

7 Address 8H 0
IN/OUT Instruction

EXW	**Writing** (Note 1)	
0		Normal
1		Extended
ROT	**Priority**	
0		Fixed
1		Rotational
DDMA	**DMA Operation** (Note 2)	
0		Enable
1		Disable

—	—	—	—	—	—	WEV	BHLD

7 Address 9H 0
IN/OUT Instruction

WEV	**Wait During Verify** (Note 3)	
0		Disable
1		Enable
BHLD	**Bus Mode**	
0		Bus release
1		Bus hold

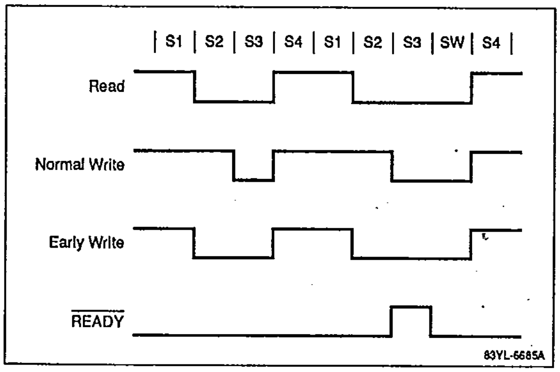
Notes:

(1) Disables BUSRQ to the BAU to prevent incorrect DMA operation while the DMAU registers are being initialized or modified.

(2) When EXW = 0, the write signal becomes active (normal write) during S3 and SW. When EXW = 1, the write signal becomes active during S2, S3, and SW (like the read signal).

(3) Wait states are generated by the $\overline{\text{READY}}$ signal during a verify transfer.

Figure 97. Early Write Cycle Timing



Mode Control Register. The DMA mode control register (DMD) selects the operating mode for each DMA channel. The DCH register selects which DMD register will be accessed. A byte IN/OUT instruction must be used to access this register. See figure 98.

Figure 98. DMA Mode Control Register (DMD); µPD71071 Mode

TMODE	ADIR	AUTI	TDIR	—	W/$\overline{B}$
7		Address 0AH			0

TMODE	Transfer Mode
00	Demand
01	Single
10	Block
11	Cascade
ADIR	**Address Direction**
0	Increment
1	Decrement
AUTI	**Autoinitialize**
0	Disable
1	Enable
TDIR	**Transfer Direction**
00	Verify
01	I/O-to-memory
10	Memory-to-I/O
11	Not allowed
W/$\overline{B}$	**Word/Byte Transfer**
0	Byte
1	Word

Addresses and count registers are updated as follows during byte/word transfers.

Register	Byte Transfer	Word Transfer
Address register	±1	±2
Count register	−1	−1

During word transfers, two bytes starting at an even address are handled as a single word. If the starting address is odd, a DMA transfer is started after first decrementing the address by 1. For this reason, always select even addresses. The A_0 and $\overline{UBE}$ outputs control byte and word DMA transfers. The following shows the relationship between the data bus width, A_0, and $\overline{UBE}$ signals, and data bus status.

A_0	$\overline{UBE}$	Data Bus Status
0	1	D_0-D_7 valid
1	0	D_8-D_{15} valid
0	0	D_0-D_{15} valid

Status Register. The DMA status register (DST) contains information about the current state of each DMA channel. Software can determine if a termination condition has been reached (TC0-TC3) or if a DMA service request is present (RQ0-RQ3). The byte IN instruction must be used to read this register. See figure 99.

Figure 99. DMA Status Register (DST); µPD71071 Mode

RQ3	RQ2	RQ1	RQ0	TC3	TC2	TC1	TC0
7			Address 0BH Byte IN instruction				0

RQn	DMA Request, Channel n
0	No DMA request active
1	DMA request active
TCn	**Terminal Count, Channel n**
0	Not ended (for each read)
1	$\overline{END}$ or terminal count

Mask Register. The DMA mask register (DMK) allows software to individually enable and disable DMA channels. The DMK register can only be accessed via byte I/O instructions. See figure 100.

Figure 100. DMA Mask Register (DMK); µPD71071 Mode

—	—	—	—	M3	M2	M1	M0
7			Address 0FH Byte IN/OUT instruction				0

Mn	DMARQ Mask, Channel n
0	Not masked
1	Masked

µPD71037 Mode

The µPD71037 mode is selected by programming the DMAM bit of the SCTL register to 1. See figure 48. Note that on RESET, the DMAU is put into µPD71071 mode. The register set for the µPD71037 mode (table 8) is mapped into A_3-A_0 (IOAG = 0) or A_4-A_1 (IOAG = 1). For the case where IOAG = 1, the DULA system I/O register determines whether the DMAU responds to A_0 = 0 or 1.

Table 8. Register Set for µPD71037 Mode

Channel	Register	Read/Write	Address
0	DCA	R	0000
	DCA, DCB	W	
	DCC	R	0001
	DCC, DBC	W	
1	DCA	R	0010
	DCA, DCB	W	
	DCC	R	0011
	DCC, DBC	W	
2	DCA	R	0100
	DCA, DCB	W	
	DCC	R	0101
	DCC, DBC	W	
3	DCA	R	0110
	DCA, DCB	W	
	DCC	R	0111
	DCC, DBC	W	
	DST	R	1000
	DDC	W	
	DSRQ	W	1001
	DSCM	W	1010
	DMD	W	1011
	DMK	W	1111

The registers in table 8 can be accessed only by byte I/O operations. The IOAG bit of the SCTL register determines whether these registers reside in contiguous bytes, or whether they each occupy one-half word (i.e., whether the registers are byte or word aligned). If word aligned (IOAG = 1), the low bit of the DULA register determines whether the DMAU will use the upper or lower byte of the word. In µPD71071 mode, the setting of the IOAG bit makes no difference; the register addresses do not change.

µPD71037 Commands

In addition to the registers explained above, three I/O addresses cause commands to be executed when they are written to. The value of the data written is not important; it is the action of performing an I/O write to one of these addresses that initiates the desired action. The commands and their corresponding addresses (A_4-A_0) are shown here.

Command	IOAG = 0	IOAG = 1
Clear byte select flag	x1100	1100x
Initialize	x1101	1101x
Clear mask register	x1110	1110x

DMAU Registers in µPD71037 Mode

Most of the DMAU registers in this mode are the same as those in the µPD71071 mode, but with a different I/O address or method of access.

Count and Address Registers. The DCA, DBA, DCC, and DBC registers are 16 bits wide, but can only be accessed in byte-wide chunks. The byte select flag (BSF) determines which byte is accessed. When the BSF is low, the low byte is used; when the BSF is high, the high byte is used. The BSF cannot be read; to set it to a known state, a byte select flag clear command must be issued by performing an 8-bit I/O write to address x1100b. To read or write one of these registers, first clear the BSF, and then perform two consecutive 8-bit I/O operations. The low byte will be accessed first and the high byte second.

Bank Registers. The DMA memory addresses in the µPD71037 mode are 16 bits, compared with 24-bit addresses in the µPD71071 mode. To expand the 16-bit addresses into the full 24-bit address space of the V53, a set of bank registers is provided, BNKR0-BNKR3, one per DMA channel.

Each 8-bit register contains the upper address bits, A_{23}-A_{16}, to be used when a DMA channel is active. DMA addresses are modified after each transfer to point to the next address in the DMA buffer. The SCTL system I/O register, CE1-CE0 bits, control whether a carry is propagated into the upper address bits when the DMA address is incremented or decremented. CE0 controls the carry propagation to A_{16} and CE1 controls the carry to A_{20}.

The BNKR registers are read or written using byte I/O operations. See figure 101. As with other V53 internal registers, the I/O address to which the BNKR registers respond is programmable. The BADR system I/O register (address FFE1H) sets the base address of the BNKR registers in the 256-byte block of I/O space selected by the OPHA register. See figure 102.

Also, to allow maximum flexibility, the low two address bits of each BNKR register are programmable. The BSEL system I/O register (address FFE0H) sets the low two address bits for each BNKR register. See figure 103. As with other programmable addresses, the IOAG bit of the

SCTL register has the effect of shifting the settable address one bit position to the left.

The bank registers are only enabled in μPD71037 mode. In μPD71071 mode, they cannot be read or written.

Figure 101. DMA Bank Registers (BNKR); μPD71037 Mode

A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	A_{17}	A_{16}

7 BNKR0 0
IN/OUT

A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	A_{17}	A_{16}

7 BNKR1 0
IN/OUT

A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	A_{17}	A_{16}

7 BNKR2 0
IN/OUT

A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	A_{17}	A_{16}

7 BNKR3 0
IN/OUT

Figure 102. Bank Address Register (BADR); μPD71037 Mode

A_7	A_6	A_5	A_4	A_3	A_2	$*A_1$	$*A_0$

7 Address FFE1H 0
IOAG = 0

A_7	A_6	A_5	A_4	A_3	$*A_2$	$*A_1$	A_0

7 Address FFE1H 0
IOAG = 1

*Address bits are set by the BSEL register.

Figure 103. Bank Select Register (BSEL); μPD71037 Mode

BNK3	BNK2	BNK1	BNK0

7 Address FFE0H 0

BNKn	*Address Bits in BADR Register
00	00
01	01
10	10
11	11

* Address bits are A_1, A_0 if IOAG = 0 or A_2, A_1 if IOAG = 1. (IOAG is a bit in the SCTL register.)

Device Control Register. In μPD71037 mode, there are fewer device options. The wait during verify and bus hold control bits are not offered. The DMA device control register (DDC) has only one byte to control early write cycles, channel priority, and global DMA enable. See figure 104.

Figure 104. DMA Device Control Register (DDC); μPD71037 Mode

—	—	EXW	ROT	—	DDMA	—	—

7 Byte OUT Instruction 0

EXW	**Write Timing** (Note 1)
0	Normal
1	Early
ROT	**Channel Priority**
0	Fixed
1	Rotational
DDMA	**DMA Operation**
0	Enable
1	Disable

Notes:

(1) When EXW = 0, the write signal becomes active during S3 and SW. When EXW = 1, the write strobe is asserted earlier during S2, S3, and SW (same as read strobe).

Channel Mode Registers. Each channel has a mode register allocated to it. All four registers are accessed using the same I/O address. The low two bits of the data written to the DMD register select the channel. Note that byte transfers are supported but 16-bit transfers are not. Figure 105 shows the format of the channel mode register.

3f

Figure 105. DMA Channel Mode Registers (DMD); μPD71037 Mode

TMODE	ADIR	AUTI	TDIR	SELCH

7 Byte OUT Instruction 0

TMODE	Transfer Mode
00	Demand
01	Single
10	Block
11	Cascade
ADIR	**Address Direction**
0	Increment
1	Decrement
AUTI	**Autoinitialize**
0	Disable
1	Enable
TDIR	**Transfer Direction**
00	Verify
01	I/O-to-memory
10	Memory-to-I/O
11	Not allowed
SELCH	**Channel Selection for Mode Change**
00	Channel 0
01	Channel 1
10	Channel 2
11	Channel 3

Status Register. This DST register (figure 74) is identical to the μPD71071 mode DST register, but is at I/O address x1000b.

Figure 106. DMA Status Registers (DST); μPD71037 Mode

RQ3	RQ2	RQ1	RQ0	TC3	TC2	TC1	TC0

7 Address x1000b 0
Byte IN Instruction

RQn	DMA Request, Channel n
0	No DMA request active
1	DMA request active
TCn	**Terminal Count, Channel n**
0	Not ended (for each read)
1	$\overline{\text{END}}$ or terminal count

Mask Register and Single-Channel Mask Control Register. The format and I/O address of this DMK register (figure 107) is the same as in μPD71071 mode except that it cannot be read; it is a write-only register. The DMK register can be put into a known state by writing to it directly, by using the clear mask register command, or by using the single-channel mask control register (DSCM) at I/O address x1010b to set or clear the enable bit for an individual channel (figure 108).

Figure 107. DMA Mask Register (DMK); μPD71037 Mode

—	—	—	—	M3	M2	M1	M0

7 Address 0FH 0
Byte OUT Instruction

Mn	DMARQ Mask, Channel n
0	Not masked
1	Masked

Figure 108. DMA Single-Channel Mask Control Register (DSCM); μPD71037 Mode

—	—	—	—	—	—	SMQ	SELCH

7 Byte OUT Instruction 0

SMQ	Mask Setting
0	Clear mask bit
1	Set mask bit
SELCH	**DMARQ Mask Channel Selection**
00	Channel 0
01	Channel 1
10	Channel 2
11	Channel 3

Software DMA Request Register. The DSRQ register is used by software to trigger a DMA operation. One application is to simulate the assertion of a hardware DMA request for diagnostic purposes. This register is written with the number of the targeted channel and a bit that sets or clears an internal request flag associated with that channel. Figure 109 shows the format of this register.

Figure 109. Software DMA Request Register (DSRQ); μPD71037 Mode

—	—	—	—	—	—	SRQ	SELCH

7 Byte OUT Instruction 0

SRQ	Request
0	Clear request bit
1	Set request bit
SELCH	**Software DMARQ Channel Selection**
00	Channel 0
01	Channel 1
10	Channel 2
11	Channel 3

Initialization. In μPD71037 mode, there is no DICM initialize register. Instead, the DMAU is initialized by performing an I/O write to address x1100b.

POWER CONSERVATION

The V53 has three power conservation features.

- Scalable system clock
- Low-power HALT standby mode
- Very-low-power STOP mode

These features give three levels of power reduction, making the V53 ideal for use in portable or other low-power applications. The standby control register (SBCR) at address 0FFF1H in the system I/O area controls all three functions. See figure 110.

Scalable System Clock

The V53 is a CMOS device and power consumption is directly proportional to clock frequency. By reducing the frequency, power use can be significantly decreased. The system clock is used by the CPU and internal peripherals. The CLKC field in the SBCR selects a scale factor that divides the oscillation frequency by 2, 4, 8, or 16 to produce the system clock. This value can be changed dynamically to adjust the clock rate to the most efficient performance level for the task at hand.

Caution: The system clock must not be set to less than the minimum frequency specified in the AC Characteristics table.

Figure 110. Standby Control Register (SBCR)

—	—	—	CLKC	WT	STOP

7 Address FFF1H 0

CLKC	System Clock Frequency f_{CLK}
00	f_{CLK} = Osc freq ÷ 2
01	f_{CLK} = Osc freq ÷ 4
10	f_{CLK} = Osc freq ÷ 8
11	f_{CLK} = Osc freq ÷ 16

WT	* Oscillation Stabilization Time
00	$2^{19} \div f_{CLK}$
01	$2^{18} \div f_{CLK}$
10	$2^{17} \div f_{CLK}$
11	$2^{16} \div f_{CLK}$

STOP	When HALT Instruction Is Executed
0	Sets HALT mode
1	Sets STOP mode

* For example, if WT = 11 and f_{CLK} = 16 MHz, time = 4.096 ms

HALT Standby Mode

Power can be further reduced by putting the CPU in HALT standby mode. In this mode, the CPU is not operating, but all the internal peripherals are still enabled and may be drawing power. HALT mode is entered by setting the STOP bit in the SBCR to 0 and executing a HALT instruction.

The V53 will come out of HALT standby mode in response to RESET, NMI, or an interrupt from the internal interrupt control unit. If interrupts were enabled (IE = 1) before HALT mode was entered, an ICU interrupt wakeup will result in the interrupt handler being entered; if interrupts were not enabled (IE = 0), then execution will resume at the instruction following the HALT that put the CPU in the standby mode. If NMI wakes up the CPU, the NMI handler is always entered.

The bus hold (HLDRQ/HLDAK) function still operates during standby mode. External bus masters can take the bus from V53. Also, refresh and DMA cycles can still occur. The SCU and TCU can both be active, and can supply the wakeup interrupt if desired.

Refer to table 1 to find out what state the V53 outputs will be in HALT standby mode.

STOP Mode

This mode provides the maximum power reduction. The clock generator is disabled; the oscillator circuit is turned off. Power usage is minimal. STOP mode is entered by setting the STOP bit in the SBCR to 1 and executing a HALT instruction. Since the system clock is not active, none of the on-chip peripherals can be used in this mode.

If the timer unit's TCLK input is used and driven by an external oscillator, the timer will continue to function and consume power.

The output pins in STOP mode are in the same state as in the HALT mode. Refer to table 1 for details. The V53 will wake up from STOP mode in response to a RESET or NMI.

Oscillator Stabilization Time

When the V53 is reset or when it wakes up from STOP mode, the oscillator circuit is started up. This circuit can take a relatively long time to come up to speed and to stabilize. The oscillator stabilization time field (WT) in the SBCR does not affect the physical startup time; it determines how long the V53 will wait for the clock generator oscillator circuit to stabilize. The user should determine the worst case stabilization time and select a longer value of WT.

RESET FUNCTION

The V53 is reset when a falling edge is input to the $\overline{\text{RESET}}$ pin and is subsequently held low for six clocks or longer than the oscillator stabilization time and then made high.

CPU Operations

When the V53 is reset, the CPU is initialized as shown in figure 111 and starts prefetching instructions from address FFFF0H.

Figure 111. CPU Reset Status

Prefetch Pointer	PFP	0000H
Program Counter	PC	0000H
Program Segment Register	PS	FFFFH
Stack Segment Register	SS	0000H
Data Segment 0 Register	DS0	0000H
Data Segment 1 Register	DS1	0000H
Queue		Cleared
Program Status Word	PSW	

1	1	1	1	V 0	DIR 0	IE 0	BRK 0

15 0

S 0	Z 0	0	AC 0	0	P 0	1	CY 0

7 0

Internal Register Operations

Some internal registers are also initialized by the $\overline{\text{RESET}}$ input signal. See figure 112. The rest of the registers retain the status they had immediately before the $\overline{\text{RESET}}$ signal was applied, but their contents are undefined at power up.

Figure 112. Register Reset Status

	Initial Value, Bits 7-0							
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
System I/O Area								
SCTL	—	—	—	0	0	0	0	0
OPSEL	—	—	—	—	0	0	0	0
WCY0	—	—	—	—	—	1	1	1
WCY1	—	1	1	1	—	1	1	1
WCY2	—	1	1	1	—	1	1	1
WCY3	—	1	1	1	—	1	1	1
WCY4	—	1	1	1	—	1	1	1
WMB0	—	1	1	1	—	1	1	1
WMB1	—	1	1	1	—	1	1	1
WAC	—	—	—	—	0	0	0	0
TCKS	—	—	—	0	0	0	0	0
RFC	—	0	—	0	1	0	0	0
SBCR	—	—	—	0	0	0	0	0

Figure 112. Register Reset Status (cont)

	Initial Value, Bits 7-0							
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Serial Control Unit								
SMD	0	1	0	0	1	0	1	1
SCM	—	—	0	0	0	0	0	0
SIMK	—	—	—	—	—	—	1	1
SST	—	0	0	0	0	1	0	0
DMA Control Unit								
DCH	—	—	—	0	0	0	0	1
DMD	0	0	0	0	0	0	—	0
DDC (8H)	—	—	0	0	—	0	—	—
DDC (9H)	—	—	—	—	—	—	0	0
DST	0	0	0	0	0	0	0	0
DMK	—	—	—	—	1	1	1	1

INSTRUCTION SET HIGHLIGHTS

Enhanced Instructions

In addition to the µPD8088/86 instructions, the µPD70236 has enhanced instructions listed in table 8.

Table 8. Enhanced Instruction

Instruction	Function
PUSH imm	Pushes immediate data onto stack
PUSH R	Pushes 8 general registers onto stack
POP R	Pops 8 general registers onto stack
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

Enhanced Stack Operation Instructions

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

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

Enhanced Multiplication Instructions

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

Enhanced Shift and Rotate Instructions

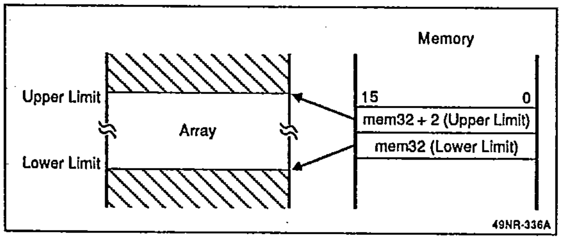
SHL reg, imm8; SHR reg, imm8; SHRA reg, imm8. These instructions allow the contents of a register to be shifted by the number of bits defined by the immediate data.

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

Check Array Boundary Instruction

CHKIND reg16, mem32. This instruction is used to verify that index values pointing to the elements of an array data structure are within the defined range. See figure 113. The lower limit of the array should be in memory location mem32, the upper limit in mem32 + 2. If the index value in reg16 is not between these limits when CHKIND is executed, a BRK 5 will occur. This causes a jump to the location in interrupt vector 5.

Figure 113. Check Array Boundary



Block I/O Instruction

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

Stack Frame Instruction

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

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

Unique Instructions

In addition to the μPD8088/86 instructions and the enhanced instructions, the μPD70236 has the unique instructions listed in table 9.

Table 9. Unique Instructions

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

Variable Length Bit Field Operation Instructions

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

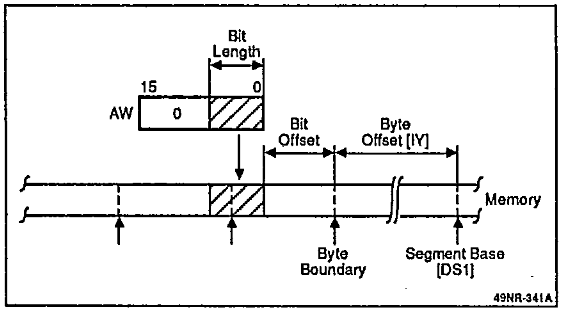
INS reg8, reg8; INS reg8, imm4. This instruction transfers low bits from the 16-bit AW register (the number of bits is specified by the second operand) to the memory location specified by the segment base (DS1 register) plus the byte offset (IY register). The starting bit position within this byte is specified as an offset by the lower 4 bits of the first operand. See figure 114.

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

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

Bit field data may overlap the byte boundary of memory.

Figure 114. Bit Field Insertion



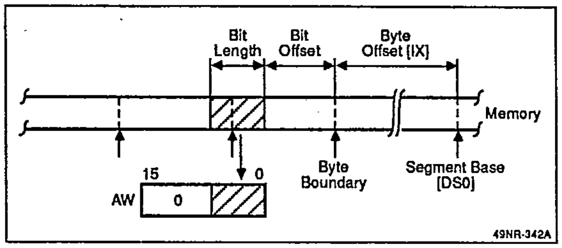
EXT reg8, reg8; EXT reg8, imm4. This instruction loads to the AW registers the bit field data whose bit length is specified by the second operand of the instruction from the memory location that is specified by the DS0 segment register (segment base), the IX index register (byte offset), and the lower 4 bits of the first operand (bit offset). See figure 115.

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

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

Bit field data may overlap the byte boundary of memory.

Figure 115. Bit Field Extraction



Packed BCD Operation Instructions

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

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

ADD4S. This instruction adds the packed BCD string addressed by the IX index register to the packed BCD string addressed by the IY index register, and stores the result in the string addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register, and the result of the operation will affect the V (overflow), CY, and Z flags.

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

SUB4S. This instruction subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY register, and stores the result in the string addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register, and the result of the operation will affect the V, CY, and Z flags.

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

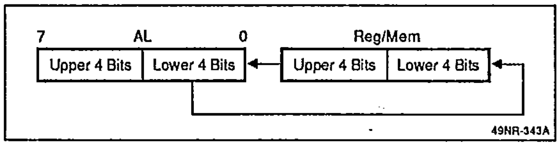
CMP4S. This instruction performs the same operation as SUB4S except that the result is not stored and only the V, CY, and Z flags are affected.

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

ROL4. This instruction treats the byte data of the register or memory operand specified by the instruction as

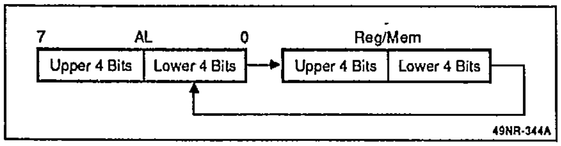
BCD data and uses the lower 4 bits of the AL register (AL_L) to rotate that data one BCD digit to the left. See figure 116.

Figure 116. BCD Rotate Left



ROR4. This instruction treats the byte data of the register or memory specified by the instruction as BCD data and uses the lower 4 bits of the AL register (AL_L) to rotate that data one BCD digit to the right. See figure 117.

Figure 117. BCD Rotate Right



Bit Manipulation Instructions

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

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

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

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

Repeat Prefix Instructions

REPC. This instruction causes the µPD70236 to repeat the following primitive block transfer instruction until the CY flag becomes cleared or the CW register becomes zero.

REPNC. This instruction causes the µPD70236 to repeat the following primitive block transfer instruction until the CY flag becomes set or the CW register becomes zero.

Address Expansion Control Instructions

BRKXA imm8. This instruction is used to turn on expanded addressing. The 8-bit immediate data specifies an interrupt vector. The PC field of this vector is loaded into the PC (and PFP). The XA flag in the XAM register is set to 1, thereby enabling the expanded addressing mode. The µPD70236 will begin fetching from the new PFP through the address translation table. That is, the new PC is treated as a logical address and is translated to the new, larger physical address space.

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

RETXA imm8. This instruction is used to turn off expanded addressing. It is identical in operation to BRKXA, except that the expanded addressing mode is turned off before fetching from the new address. That is, the XA flag in the XAM register is set to 0, and the PC is loaded with the value of the PC field in the interrupt vector selected by the immediate data..

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

Porting µPD70116/70108 Code to µPD70236

The µPD70236 is completely software compatible with the µPD70116/70108. However, the µPD70236 offers some improvements that may affect the porting of µPD70116 code to the µPD70236. These improvements are:

(1) The µPD70116 does not trap on undefined opcodes. The µPD70236 will trap, and also will trap when a register addressing mode is used for any of these instructions:

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

(2) During signed division (DIV), if the quotient is 80H (byte operation) or 8000H (word), the µPD70116 will take a Divide By 0 trap. The µPD70236 will perform the calculation.

(3) When the µPD70116 executes the POLL instruction, it will wait for the POLL input signal to be asserted. The µPD70236 has no POLL input; instead, when this instruction is executed, if a coprocessor is not connected, then a Coprocessor Not Present trap will be taken. If a coprocessor is attached, then no operation takes place.

The µPD70116 accepts FP01 and FP02 as opcodes for the iAPX8087 coprocessor. The µPD70236 accepts these as opcodes for the µPD72291 coprocessor, which is not compatible with the iAPX8087.

(4) During the POP R instruction, the µPD70116 does not restore the SP register. The µPD70236 does restore the SP.

(5) When processing a divide error, the μPD70116 saves the address of the next instruction. The μPD70236 saves the address of the current instruction (the divide instruction).

(6) The μPD70116 allows up to three prefix instructions in any combination. The μPD70236 also allows three prefixes, but only one of each type can be used. The μPD70236 could operate incorrectly if there are two prefixes of the same type. For example, consider:

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

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

(7) The μPD70116 accepts $\overline{\text{NMI}}$ requests even while processing an NMI. The μPD70236 does not allow nesting of NMIs; the $\overline{\text{NMI}}$ input will be ignored until the NMI interrupt handler is exited.

INSTRUCTION SET

Symbols

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

Clocks

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

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

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

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

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

Symbols

Symbol	Meaning
acc	Accumulator(AW or AL)
duso	Displacement (8 or 16 bits)
dmem	Direct memory address
dst	Destination operand or address
ext-disp8	16-bit displacement (sign-extension byte + 8-bit displacement)
far_label	Label within a different program segment
far_proc	Procedure within a different program segment
fp_op	Floating-point instruction operation
imm	8- or 16-bit immediate operand
imm3/4	3- or 4-bit immediate bit offset
imm8	8-bit immediate operand
imm16	16-bit immediate operand
mem	Memory field (000 to 111); 8- or 16-bit memory location
mem8	8-bit memory location
mem16	16-bit memory location
mem32	32-bit memory location
memptr16	Word containing the destination address within the current segment
memptr32	Double word containing a destination address in another segment
mod	Mode field (00 to 10)
near_label	Label within the current segment
near_proc	Procedure within the current segment
offset	Immediate offset data (16 bits)
pop_value	Number of bytes to discard from the stack
reg	Register field (000 to 111); 8- or 16-bit general-purpose register
reg8	8-bit general-purpose register
reg16	16-bit general-purpose register
regptr	16-bit register containing a destination address within the current segment
regptr16	Register containing a destination address within the current segment
seg	Immediate segment data (16 bits)
short_label	Label between −128 and +127 bytes from the end of the current instruction
sr	Segment register
src	Source operand or address
temp	Temporary register (8/16/32 bits)
AC	Auxiliary carry flag
AH	Accumulator (high byte)
AL	Accumulator (low byte)
AW	Accumulator (16 bits)
BH	BW register (high byte)
BL	BW register (low byte)
BP	BP register
BRK	Break flag
BW	BW register (16 bits)
CH	CW register (high byte)
CL	CW register (low byte)
CW	CW register (16 bits)
CY	Carry flag
DH	DW register (high byte)
DIR	Direction flag
DL	DW register (low byte)
DS0	Data segment 0 register (16 bits)
DS1	Data segment 1 register (16 bits)
DW	DW register (16 bits)
IE	Interrupt enable flag
IX	Index register (source) (16 bits)
IY	Index register (destination) (16 bits)
MD	Mode flag
P	Parity flag
PC	Program counter (16 bits)
PS	Program segment register (16 bits)
PSW	Program status word (16 bits)
R	Register set
S	Sign extend operand field S = No sign extension S = Sign extend immediate byte operand
S	Sign flag
SP	Stack pointer (16 bits)
SS	Stack segment register (16 bits)
V	Overflow flag
W	Word/byte field (0 to 1)
X, XXX, YYY, ZZZ	Data to identify the instruction code of the external floating-point arithmetic chip
XXH	Two-digit hexadecimal value
XXXXH	Four-digit hexadecimal value
Z	Zero flag

31

Flag Operations

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

Memory Addressing Modes

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

Register Selection (mod = 11)

reg	W = 0	W = 1
000	AL	AW
001	CL	CW
010	DL	DW
011	BL	BW
100	AH	SP
101	CH	BP
110	DH	IX
111	BH	IY

Segment Register Selection

sr	Segment Register
00	DS1
01	PS
10	SS
11	DS0

Instruction Set

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

Instruction Set (cont)

Mnemonic	Operand	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Clocks	Bytes	AC	CY	V	P	S	Z
Data Transfer Instructions (cont)																									
XCH	reg, reg	1	0	0	0	0	1	1	W	1	1	reg			reg			3	2						
	mem, reg	1	0	0	0	0	1	1	W	mod		reg			mem			8/12	2-4						
	AW, reg16	1	0	0	1	0	reg											3	1						
Repeat Prefixes																									
REPC		0	1	1	0	0	1	0	1									2	1						
REPNC		0	1	1	0	0	1	0	0									2	1						
REP REPE REPZ		1	1	1	1	0	0	1	1									2	1						
REPNE REPNZ		1	1	1	1	0	0	1	0									2	1						
Block Transfer Instructions																									
MOVBK	dst, src	1	0	1	0	0	1	0	W									3 + 4n (W = 0) 3 + 4n (W = 1, even addresses) 3 + 8n (W = 1, odd addresses) 3 + 6n (W = 1, odd/even addresses)	1						
CMPBK	dst, src	1	0	1	0	0	1	1	W									3 + 7n (W = 0) 3 + 7n (W = 1, even addresses) 3 + 11n (W = 1, odd addresses) 3 + 9n (W = 1, odd/even addresses)	1	x	x	x	x	x	x
CMPM	dst	1	0	1	0	1	1	1	W									3 + 5n (W = 0) 3 + 5n (W = 1, even addresses) 3 + 7n (W = 1, odd addresses)	1	x	x	x	x	x	x
LDM	src	1	0	1	0	1	1	0	W									5 + 2n (W = 0) 5 + 2n (W = 1, even addresses) 5 + 4n (W = 1, odd addresses)	1						
STM	dst	1	0	1	0	1	0	1	W									3 + 2n (W = 0) 3 + 2n (W = 1, even addresses) 3 + 4n (W = 1, odd addresses)	1						

n = number of returns
String instruction execution clocks for a single-instruction execution are in parentheses.

Instruction Set (cont)

		Opcode																		Flags					
Mnemonic	**Operand**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Clocks**	**Bytes**	**AC**	**CY**	**V**	**P**	**S**	**Z**
I/O Instructions (cont)																									
IN	acc, imm8	1	1	1	0	0	1	0	W									5/7	2						
	acc, DW	1	1	1	0	1	1	0	W									3/5	1						
OUT	imm8, acc	1	1	1	0	0	1	1	W									3/5	2						
	DW, acc	1	1	1	0	1	1	1	W									3/5	1						
INM	dst, DW	0	1	1	0	1	1	0	W									3 + 11n (W = 0) 3 + 8n (W = 1, even addresses) 3 + 22n (W = 1, odd addresses) 3 +20n (W = 1, odd/even addresses; odd for I/O) 3 + 13n (W = 1, odd/even addresses; odd for memory)	1						
OUTM	DW, src	0	1	1	0	1	1	1	W									3 + 11n (W = 0) 3 + 8n (W = 1, even addresses) 3 + 22n (W = 1, odd addresses) 3 + 20n (W = 1, odd addresses; odd for I/O) 3 + 13n (W = 1, odd addresses; odd for memory)	1						

n = number of transfers
String instruction execution clocks for a single-instruction execution are in parentheses.
Use the right side of the slash (/) for DMA I/O accesses.

Mnemonic	**Operand**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Clocks**	**Bytes**	**AC**	**CY**	**V**	**P**	**S**	**Z**
BCD Instructions																									
ADJBA		0	0	1	1	0	1	1	1									4	1	x	x	u	u	u	u
ADJ4A		0	0	1	0	0	1	1	1									2	1	x	x	u	x	x	x
ADJBS		0	0	1	1	1	1	1	1									4	1	x	x	u	u	u	u
ADJ4S		0	0	1	0	1	1	1	1									2	1	x	x	u	x	x	x
ADD4S	dst, src	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0	2 + 18n	2	u	x	u	u	u	x
SUB4S	dst, src	0	0	0	0	1	1	1	1	0	0	1	0	0	0	1	0	2 + 18n	2	u	x	u	u	u	x
CMP4S	dst, src	0	0	0	0	1	1	1	1	0	0	1	0	0	1	1	0	7 + 14n	2	u	x	u	u	u	x
ROL4	reg8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0	9	3						
		1	1	0	0	0	reg																		
	mem8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0	15	3-5						
		mod		0	0	0	mem																		
ROR4	reg8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0	13	3						
		1	1	0	0	0	reg																		
	mem8	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0	19	3-5						
		mod		0	0	0	mem																		

n = number of BCD digits divided by 2

Instruction Set (cont)

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

T-49-17-15

Instruction Set (cont)

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

T-49-17-15

Instruction Set (cont)

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

3

Instruction Set (cont)

		Opcode																		Flags					
Mnemonic	Operand	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Clocks	Bytes	AC	CY	V	P	S	Z
Bit Manipulation Instructions (cont)																									
SET1 (cont)	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	9	4-6						
		mod		0	0	0	mem																		
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	1	9/13	4-6						
		mod		0	0	0	mem																		
	CY	1	1	1	1	1	0	0	1									2	1		1				
	DIR	1	1	1	1	1	1	0	1									2	1						
CLR1	reg, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	W	4	3						
		1	1	0	0	0	reg																		
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	0	9	3-5						
		mod		0	0	0	mem																		
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	9/13	3-5						
		mod		0	0	0	mem																		
	reg, imm3/4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	W	4	4						
		1	1	0	0	0	reg																		
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	0	9	4-6						
		mod		0	0	0	mem																		
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	1	9/13	4-6						
		mod		0	0	0	mem																		
	CY	1	1	1	1	1	0	0	0									2	1		0				
	DIR	1	1	1	1	1	1	0	0									2	1						
NOT1	reg, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	W	4	3						
		1	1	0	0	0	reg																		
	mem8, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	0	9	3-5						
		mod		0	0	0	mem																		
	mem16, CL	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1	9/13	3-5						
		mod		0	0	0	mem																		
	reg, imm3/4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	W	4	4						
		1	1	0	0	0	reg																		
	mem8, imm3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	W	9	4-6						
		mod		0	0	0	mem																		
	mem16, imm4	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1	9/13	4-6						
		mod		0	0	0	mem																		
	CY	1	1	1	1	0	1	0	1									2	1		x				

T-49-17-15

Instruction Set (cont)

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

n = number of shifts

31

T-49-17-15

Instruction Set (cont)

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

Instruction Set (cont)

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

T-49-17-15

Instruction Set (cont)

		Opcode																		Flags					
Mnemonic	Operand	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Clocks	Bytes	AC	CY	V	P	S	Z
CPU Control Instructions (cont)																									
DS0:, DS1:, PS:, SS: (segment override prefixes)		0	0	1	seg		1	1	0									2	1						
Address Expansion Control Instructions																									
BRKXA	imm8	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	12	3						
					imm8																				
RETXA	imm8	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	12	3						
					imm8																				