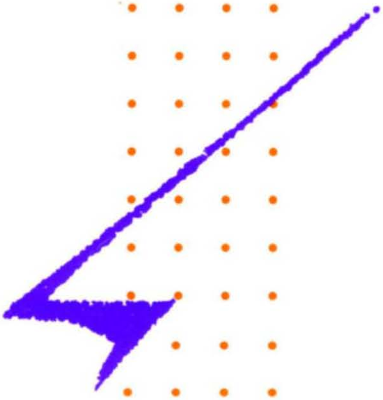
MC68340UM/AD



MC68340

INTEGRATED PROCESSOR USER'S MANUAL

MOTOROLA

MC68340

Integrated Processor User's Manual

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not authorized for use as components in life support devices or systems intended for surgical implant into the body or intended to support or sustain life. Buyer agrees to notify Motorola of any such intended end use whereupon Motorola shall determine availability and suitability of its product or products for the use intended. Motorola and Ⓜ are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Employment Opportunity/Affirmative Action Employer.

PREFACE

The complete documentation package for the MC68340 consists of the *MC68340 Integrated Processor Unit User's Manual* (MC68340UM/AD) and the *MC68340 Integrated Processor Unit Technical Summary* (MC68340/D).

The *MC68340 Integrated Processor Unit User's Manual* describes the programming, capabilities, registers, and operation of the MC68340. The *MC68340 Integrated Processor Unit Technical Summary* provides a description of the MC68340 capabilities and detailed electrical specifications.

This user's manual is organized as follows:

Section 1	Device Overview
Section 2	Signal Descriptions
Section 3	Bus Operation
Section 4	System Integration Module
Section 5	CPU32
Section 6	DMA Controller Module
Section 7	Serial Module
Section 8	Timer Modules
Section 9	IEEE 1149.1 Test Access Port
Section 10	Applications
Section 11	Electrical Characteristics
Section 12	Ordering Information and Mechanical Data

TABLE OF CONTENTS

Paragraph Number — Title — Page Number

Section 1
Device Overview

Section 2
Signal Descriptions

TABLE OF CONTENTS (Continued)

Paragraph Number | Title | Page Number

TABLE OF CONTENTS (Continued)

Paragraph Number | Title | Page Number

Section 3
Bus Operation

TABLE OF CONTENTS (Continued)

Section 4
System Integration Module

TABLE OF CONTENTS (Continued)

Section 5
CPU32

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Paragraph Number | Title | Page Number

TABLE OF CONTENTS (Continued)

Section 6
DMA Controller Module

TABLE OF CONTENTS (Continued)

Section 7
Serial Module

TABLE OF CONTENTS (Continued)

Paragraph Number — Title — Page Number

TABLE OF CONTENTS (Continued)

Section 8
Timer Modules

TABLE OF CONTENTS (Continued)

Paragraph Number | Title | Page Number

Section 9
IEEE 1149.1 Test Access Port

Section 10
Applications

TABLE OF CONTENTS (Concluded)

Section 11
Electrical Characteristics

Section 12
Ordering Information and Mechanical Data

Index

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Concluded)

LIST OF TABLES

Table Number — Title — Page Number

LIST OF TABLES (Continued)

LIST OF TABLES (Concluded)

SECTION 1 DEVICE OVERVIEW

The MC68340 is a 32-bit integrated processor unit, combining high-performance data manipulation capabilities with powerful peripheral subsystems. The MC68340 is a member of the M68300 Family of modular devices featuring fully static, high-speed complementary metal-oxide semiconductor (HCMOS) technology. Based on the powerful MC68000, the CPU32 central processing module of the MC68340 provides enhanced system performance and uses the extensive software base of the M68000 Family. Figure 1-1 shows the major components of the MC68340.

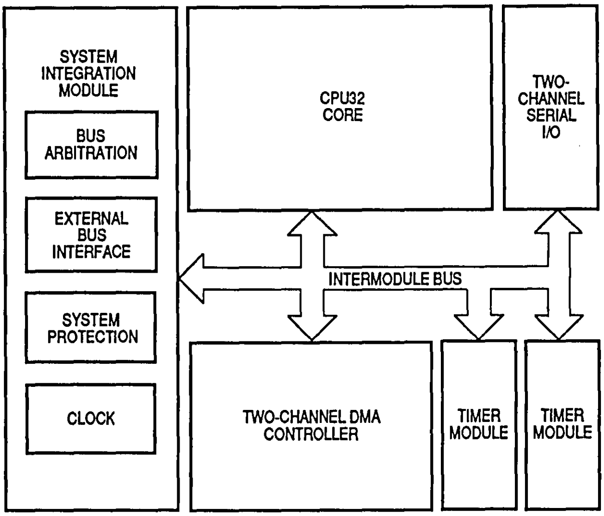


Figure 1-1. Block Diagram

The MC68340 also contains intelligent peripheral modules such as the direct memory access (DMA) controller, which provides two channels of single- or dual-address transfer capability. Two channels of high-speed serial communications are provided by the serial module with synchronous and asynchronous protocols available. The two timer modules are identical and can be externally cascaded. Four chip selects enhance system integration for easy external memory or peripheral access. These modules are connected on-chip via an intermodule bus (IMB).

The major features of the MC68340 are as follows:

- Integrated System Functions in a Single Chip
- 32-Bit M68000 Family Central Processor
  - — Upward Object-Code Compatible with the MC68000 and MC68010
  - — New Instructions for Controller Applications
  - — Higher Performance Execution
- Two-Channel DMA Capability for Low-Latency Memory Accesses
- Two-Channel Serial I/O
- Two Multiple-Mode 16-Bit Timers
- Four Programmable Chip-Select Signals
- System Failure Protection:
  - — Software Watchdog Timer
  - — Periodic Interrupt Timer
  - — Spurious Interrupt, Double Bus Fault, and Bus Timeout Monitors
  - — Automatic Programmable Bus Termination
- Up to 16 Discrete I/O Pins
- Low-Power Operation
  - HCMOS Technology Reduces Power in Normal Operation
  - LPSTOP Mode Provides Static State for Lower Standby Drain
- Frequency: 16.78-MHz Maximum Frequency at 5-V Supply, Software Programmable
- Packages: 144-Pin Ceramic Quad Flat Pack (CQFP)
  145-Pin Plastic Pin Grid Array (PGA)

1.1 CENTRAL PROCESSOR UNIT

The central processing unit of the MC68340 is the CPU32, an upward-compatible M68000 Family member that excels in processing calculation-intensive algorithms and supporting high-level languages. All MC68010 and most MC68020 enhancements, such as virtual memory support, loop mode operation, instruction pipeline, and 32-bit mathematical operations, are supported. Powerful addressing modes provide compatibility with existing software programs and increase the efficiency of high-level language compilers. New instructions, such as table lookup and interpolate and low-power stop, support the specific requirements of controller applications. Most instructions can execute in one-half the number of clocks required by an MC68000, yielding an overall 1.6 times performance of the same-speed MC68000.

1.2 INTELLIGENT PERIPHERALS

To improve total system throughput and reduce part count, size, and cost of system implementation, the MC68340 also features intelligent, on-chip, peripheral subsystems and typical glue logic. These subsystems include the system integration module (SIM), the DMA module, the serial module, and the timer modules.

1.2.1 System Integration Module

The SIM includes an external interface and various functions that reduce the need for external glue logic. The SIM contains the external bus interface, four chip selects, system protection, and clock generation.

1.2.1.1 EXTERNAL BUS INTERFACE. Based on the MC68020 bus, the external bus provides 32 address lines and 16 data lines. The data bus allows dynamic sizing between 8- and 16-bit data accesses. External bus arbitration is accomplished by a four-line handshaking interface. Transfers can be made in as little as two clock cycles.

1.2.1.2 CHIP SELECTS. Four independent chip selects can enable external circuits, providing all handshaking and timing signals with up to 265-ns access times. Block size is programmable in 256-byte increments up to the 4-Gbyte address capability. Accesses can be preselected for either 8- or 16-bit transfers.

1.2.1.3 SYSTEM PROTECTION SUBMODULE. The M68000 Family of processors is designed with the concept of providing maximum system safeguards. Additional system protection is provided on the MC68340 by various monitors and timers, including the bus monitor, double bus fault monitor, spurious interrupt monitor, software watchdog timer, and the periodic interrupt timer. These system functions are integrated on the MC68340 to reduce board size and the cost incurred with external components.

1.2.1.4 SYSTEM CLOCK. The system clock can be generated by an on-chip phase-locked loop (PLL) circuit to run the device up to 16.78 MHz from a 32.768-kHz watch crystal. An external clock can also be used. The system speed can be changed dynamically with the PLL, providing either high performance or low power consumption under software control. With its fully static HCMOS design, it is possible to completely stop the system clock in software while still preserving the contents of the registers.

1.2.2 Direct Memory Access Module

In dual-address mode, the DMA module supports 32 bits of address and 16 bits of data with each of its two independent channels. In single-address mode, the DMA module supports 32 bits of address and 32 bits of data as well as providing address and control signals during a single-ended transfer. The requesting device either sends or receives data to or from the specified address. In dual-address mode, two bus transfers occur, one from a source device and the other to a destination device. In dual-address mode, operands are packed or unpacked according to port sizes and addresses.

Each channel has an independent request, acknowledge, and done indication. The request mode can be internal, with four adjustable bus bandwidths, or external, with edge or level trigger. The DMA module can sustain a transfer rate of 33.3 Mbytes per second in single-address mode, and nearly 8.4 Mbytes per second in dual-address mode.

1.2.3 Serial Module

The serial module contains two fully independent serial ports that have a maximum transfer rate of 3 million bits per second (Mbps) in synchronous mode. Using the 1/16 clock in asynchronous mode, a channel can operate at 188 kbps. The baud rate generator can be programmed for different baud rates on transmit and receive.

Full modem support is provided with separate request to send (RTS) and clear to send (CTS) signals for each channel. Full duplex, local loopback, or remote loopback modes are available. The data format can be 5, 6, 7, or 8 bits with even, odd, or no parity and a programmable number of stop bits. A wide variety of maskable interrupt capability is provided on each channel. Channel 1 also provides service request signals.

1.2.4 Timer Modules

The two timer modules are identical, and the timers can be externally cascaded. Each timer has an 8-bit prescaler and a 16-bit counter. The prescaler input can be tied to the system clock or an external input frequency. The counter can be driven directly from the timer clock or tapped from any of the prescaler's eight bits.

Each timer has a variety of operational modes. Symmetrical or asymmetrical square-wave generation and pulse-width and period measurement are available. A variable-width single-shot pulse can be generated. Output compare and input capture can be performed concurrently, and a signal-generating capability based on output compare is available.

SECTION 2
SIGNAL DESCRIPTIONS

This section contains brief descriptions of the MC68340 input and output signals in their functional groups as shown in Figure 2-1.

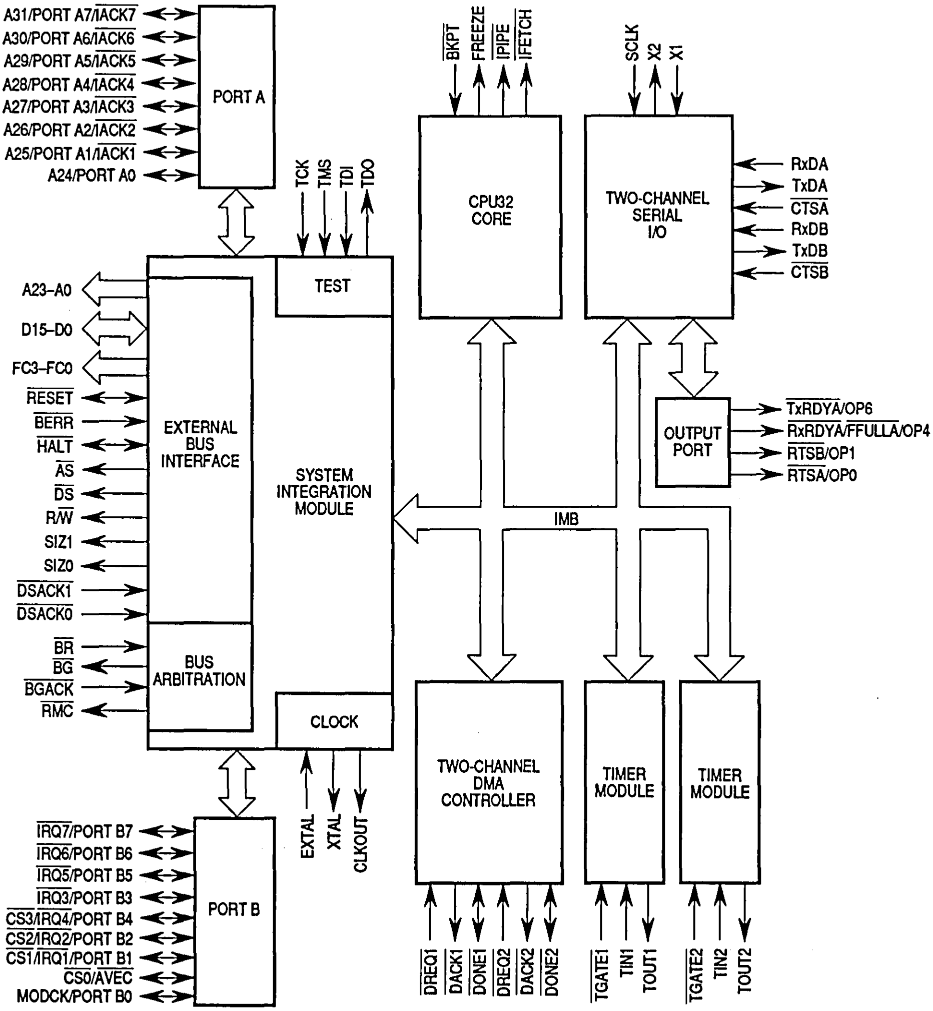


Figure 2-1. Functional Signal Groups

2.1 SIGNAL INDEX

The input and output signals for the MC68340 are listed in Table 2-1. The name, mnemonic, and brief functional description are presented. For more detail on each signal, refer to the paragraph named for the signal. Guaranteed timing specifications for the signals listed in Table 2-1 can be found in MC68340/D, *MC68340 Technical Summary*.

Table 2-1. Signal Index

Signal Name	Mnemonic	Function
Address Bus	A23–A0	Lower 24 bits of address bus
Address Bus/Port A7–A0/$\overline{\text{IACK7}}$–$\overline{\text{IACK1}}$	A31–A24	Upper eight bits of address bus, parallel I/O port, or interrupt acknowledge lines
Data Bus	D15–D0	16-bit data bus used to transfer byte or word data
Function Codes	FC3–FC0	Identifies the processor state and the address space of the current bus cycle
Chip Select/$\overline{\text{IRQ4}}$, $\overline{\text{IRQ2}}$, $\overline{\text{IRQ1}}$/Port B4, B2, B1, $\overline{\text{AVEC}}$	$\overline{\text{CS3}}$–$\overline{\text{CS0}}$	Enables peripherals at programmed addresses or provides parallel I/O and automatic vector request during an interrupt acknowledge cycle
Bus Request	$\overline{\text{BR}}$	Indicates that an external device requires bus mastership
Bus Grant	$\overline{\text{BG}}$	Indicates that the current bus cycle is complete and the MC68340 has relinquished the bus
Bus Grant Acknowledge	$\overline{\text{BGACK}}$	Indicates that an external device has assumed bus mastership
Data and Size Acknowledge	$\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$	Provides asynchronous data transfers and dynamic bus sizing
Read-Modify-Write Cycle	$\overline{\text{RMC}}$	Identifies the bus cycle as part of an indivisible read-modify-write operation
Address Strobe	$\overline{\text{AS}}$	Indicates that a valid address is on the address bus
Data Strobe	$\overline{\text{DS}}$	During a read cycle, DS indicates that an external device should place valid data on the data bus. During a write cycle, DS indicates that valid data is on the data bus.
Size	SIZ1, SIZ0	Indicates the number of bytes remaining to be transferred for this cycle
Read/Write	R/$\overline{\text{W}}$	Indicates the direction of data transfer on the bus
Interrupt Request Level/Port B7, B6, B5, B3	$\overline{\text{IRQ7}}$, $\overline{\text{IRQ6}}$, $\overline{\text{IRQ5}}$, $\overline{\text{IRQ3}}$	Provides an interrupt priority level to the CPU32 or provides parallel I/O
Reset	$\overline{\text{RESET}}$	System reset
Halt	$\overline{\text{HALT}}$	Suspends external bus activity
Bus Error	$\overline{\text{BERR}}$	Indicates an erroneous bus operation is being attempted
System Clock Out	CLKOUT	Internal system clock

Table 2-1. Signal Index (Continued)

Signal Name	Mnemonic	Function
Crystal Oscillator	EXTAL, XTAL	Connections for an external crystal to the internal oscillator circuit
External Filter Capacitor	XFC	Connection pin for an external capacitor to filter the circuit of the phase-locked loop
Clock Mode Select/Port B0	MODCK	Selects the source of the internal system clock or furnishes a parallel I/O bit
Instruction Fetch	$\overline{\text{IFETCH}}$	Indicates when the CPU32 is performing an instruction word prefetch and when the instruction pipeline has been flushed
Instruction Pipe	$\overline{\text{IPIPE}}$	Used to track movement of words through the instruction pipeline
Breakpoint	$\overline{\text{BKPT}}$	Signals a hardware breakpoint to the CPU32
Freeze	FREEZE	Indicates that the CPU32 has acknowledged a breakpoint
Receive Data	RxDA, RxDB	Serial input to the serial module
Transmit Data	TxDA, TxDB	Serial output from the serial module
Clear to Send	$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$	Serial module clear to send inputs
Request to Send/OP1,OP0	$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$	Serial module request to send outputs or can be parallel outputs
Serial Crystal Oscillator	X1, X2	Connections for an external crystal to the serial module internal oscillator circuit
Serial Clock	SCLK	External serial module clock input
Transmitter Ready/OP6	$\overline{\text{TxRDYA}}$	Indicates transmit buffer has a character or can be a parallel output
Receiver Ready/FIFO Full/OP4	$\overline{\text{RxRDYA}}$	Indicates receive buffer has a character, the receiver FIFO buffer is full, or can be a parallel output
DMA Request	$\overline{\text{DREQ2}}$, $\overline{\text{DREQ1}}$	Input that starts DMA process
DMA Acknowledge	$\overline{\text{DACK2}}$, $\overline{\text{DACK1}}$	Output that signals an access during DMA
DMA Done	$\overline{\text{DONE2}}$, $\overline{\text{DONE1}}$	Bidirectional signal that indicates last transfer
Timer Gate	$\overline{\text{TGATE2}}$, $\overline{\text{TGATE1}}$	Counter enable input to timer
Timer Input	TIN2, TIN1	Time reference input to timer
Timer Output	TOUT2, TOUT1	Output waveform from timer
Test Clock	TCK	Provides a clock for IEEE 1149.1 test logic
Test Mode Select	TMS	Controls test mode operations
Test Data In	TDI	Serial test instructions and test data signal
Test Data Out	TDO	Serial test instructions and test data signal
Synchronizer Power	V_{CCSYN}	Power supply to VCO
System Power Supply and Return	V_{CC}, GND	Power supply and return to the MC68340

2.2 ADDRESS BUS

The address bus consists of the following two groups. Refer to **SECTION 3 BUS OPERATION** for information on the address bus and its relationship to bus operation.

2.2.1 Address Bus (A23–A0)

These three-state outputs (along with A31–A24) provide the address for the current bus cycle, except in the CPU address space. Refer to **SECTION 3 BUS OPERATION** for more information on the CPU address space. A23 is the most significant address signal in this group.

2.2.2 Address Bus (A31–A24)

These pins can be programmed as the most significant eight address bits, port A parallel I/O, or interrupt acknowledge strobes. These pins can be used for more than one of their multiplexed functions as long as the external demultiplexing circuit properly resolves collisions between the different functions.

A31–A24. These pins can function as the most significant eight address bits. A31 is the most significant address signal in this group.

Port A7–A0. These eight pins can serve as a dedicated parallel I/O port. See **SECTION 4 SYSTEM INTEGRATION MODULE** for more information on programming these pins.

$\overline{\text{IACK7}}$–$\overline{\text{IACK1}}$. The MC68340 asserts one of these pins to indicate the level of an external interrupt during an interrupt acknowledge (IACK) cycle. Peripherals can use the IACK strobes instead of monitoring the address bus and function codes to determine that an IACK cycle is in progress and to obtain the current interrupt level. See **SECTION 3 BUS OPERATION** for more information. Only seven of these eight pins are used as IACK strobe outputs since there is no IACK0 strobe.

2.3 DATA BUS (D15–D0)

These three-state bidirectional signals provide the general-purpose data path between the MC68340 and all other devices. Although the data path is a maximum of 16 bits wide, it can be dynamically sized to support 8- or 16-bit transfers. D15 is the most significant bit of the data bus. Refer to **SECTION 3 BUS OPERATION** for information on the data bus and its relationship to bus operation.

2.4 FUNCTION CODES (FC3–FC0)

These three-state outputs identify the processor state and the address space of the current bus cycle as noted in Table 2-2. Refer to **SECTION 3 BUS OPERATION** for more information.

Table 2-2. Address Space Encoding

Function Code Bits				Address Spaces
3	2	1	0	
0	0	0	0	Reserved (Motorola)
0	0	0	1	User Data Space
0	0	1	0	User Program Space
0	0	1	1	Reserved (User)
0	1	0	0	Reserved (Motorola)
0	1	0	1	Supervisor Data Space
0	1	1	0	Supervisor Program Space
0	1	1	1	CPU Space
1	x	x	x	DMA Space

2.5 CHIP SELECTS (CS3–CS0)

These pins can be programmed to be chip-select output signals, port B parallel I/O and autovector input, or additional interrupt request lines.

CS3–CS0. The chip-select output signals enable peripherals at programmed addresses. CS0 is the chip select for a ROM containing the user's reset vector and initialization program; therefore, it functions as the boot chip select immediately after reset. Refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for more information on chip selects.

Port B4, B2, B1, $\overline{\textbf{AVEC}}$. This signal group functions as three bits of parallel I/O and the autovector input. Refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for information on parallel I/O signals. $\overline{\text{AVEC}}$ requests an automatic vector during an interrupt acknowledge cycle. Refer to **SECTION 3 BUS OPERATION** and **SECTION 4 SYSTEM INTEGRATION MODULE** for more information on the autovector function.

$\overline{\textbf{IRQ4}}$, $\overline{\textbf{IRQ2}}$, $\overline{\textbf{IRQ1}}$. Interrupt request lines are prioritized external lines to the CPU32. These additional interrupt request lines are selected by the FIRQ bit in the MCR (see **SECTION 4 SYSTEM INTEGRATION MODULE**). Only three functions are available since there is no $\overline{\text{IRQ0}}$ function.

2.6 INTERRUPT REQUEST LEVEL ($\overline{\text{IRQ7}}$, $\overline{\text{IRQ6}}$, $\overline{\text{IRQ5}}$, $\overline{\text{IRQ3}}$)

These pins can be programmed to be either prioritized interrupt request lines or port B parallel I/O.

$\overline{\text{IRQ7}}$, $\overline{\text{IRQ6}}$, $\overline{\text{IRQ5}}$, $\overline{\text{IRQ3}}$. $\overline{\text{IRQ7}}$, the highest priority, is nonmaskable. $\overline{\text{IRQ6}}$–$\overline{\text{IRQ1}}$ are internally maskable interrupts. Refer to **SECTION 5 CPU32** for more information on interrupt request lines.

Port B7, B6, B5, B3. These pins can be used as port B parallel I/O. Refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for more information on parallel I/O signals.

2.7 BUS CONTROL SIGNALS

These signals control the bus transfer operations of the MC68340.

2.7.1 Data and Size Acknowledge ($\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$)

These two active-low input signals allow asynchronous data transfers and dynamic data bus sizing between the MC68340 and external devices as listed in the following table. Refer to **SECTION 3 BUS OPERATION** for more information on these signals and their relationship to dynamic bus sizing.

Table 2-3. $\overline{\text{DSACKx}}$ Codes and Results

$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	Result
1 (Negated)	1 (Negated)	Insert Wait States in Current Bus Cycle
1 (Negated)	0 (Asserted)	Complete Cycle — Data Bus Port Size Is 8 Bits
0 (Asserted)	1 (Negated)	Complete Cycle — Data Bus Port Size Is 16 Bits
0 (Asserted)	0 (Asserted)	Reserved — Defaults to 16-Bit Port Size, Can Be Used for 32-Bit DMA cycles

2.7.2 Autovector ($\overline{\text{AVEC}}$)

See **2.5 CHIP SELECTS (CS3–CS0).**

2.7.3 Address Strobe ($\overline{\text{AS}}$)

This output signal is driven by the bus master to indicate a valid address on the address bus. The function code, size, and read/write signals are also valid when $\overline{\text{AS}}$ is asserted. Refer to **SECTION 3 BUS OPERATION** for information about the relationship of $\overline{\text{AS}}$ to bus operation.

2.7.4 Data Strobe ($\overline{\text{DS}}$)

During a read cycle, this output signal is driven by the bus master to indicate that an external device should place valid data on the data bus. During a write cycle, the data strobe indicates that valid data is on the data bus. Refer to **SECTION 3 BUS OPERATION** for information about the relationship of $\overline{\text{DS}}$ to bus operation.

2.7.5 Transfer Size (SIZ1, SIZ0)

These output signals are driven by the bus master to indicate the number of operand bytes remaining to be transferred in the current bus cycle as noted in Table 2-4. Refer to **SECTION 3 BUS OPERATION** for more information.

Table 2-4. SIZx Signal Encoding

SIZ1	SIZ0	Transfer Size
0	1	Byte
1	0	Word
1	1	3 Byte
0	0	Long Word

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

This active-high output signal is driven by the bus master to indicate the direction of data transfer on the bus. A logic one indicates a read from a slave device; a logic zero indicates a write to a slave device. Refer to **SECTION 3 BUS OPERATION** for more information.

2.8 BUS ARBITRATION SIGNALS

The following signals are the four bus arbitration control signals used to determine the bus master. Refer to **SECTION 3 BUS OPERATION** for more information.

2.8.1 Bus Request ($\overline{\mathrm{BR}}$)

This active-low input signal indicates that an external device needs to become the bus master. This input is typically wire-ORed. Refer to **SECTION 3 BUS OPERATION** for more information.

2.8.2 Bus Grant ($\overline{\mathrm{BG}}$)

Assertion of this active-low output signal indicates that the bus master has relinquished the bus. Refer to **SECTION 3 BUS OPERATION** for more information.

2.8.3 Bus Grant Acknowledge ($\overline{\mathrm{BGACK}}$)

Assertion of this active-low input indicates that an external device has become the bus master. Refer to **SECTION 3 BUS OPERATION** for more information.

2.8.4 Read-Modify-Write Cycle ($\overline{\mathrm{RMC}}$)

This output signal identifies the bus cycle as part of an indivisible read-modify-write operation; it remains asserted during all bus cycles of the read-modify-write operation to indicate that bus ownership cannot be transferred. Refer to **SECTION 3 BUS OPERATION** for additional information.

2.9 EXCEPTION CONTROL SIGNALS

These signals are used by the integrated processor unit to recover from an exception.

2.9.1 Reset ($\overline{\mathrm{RESET}}$)

This active-low, open-drain, bidirectional signal is used to initiate a system reset. An external reset signal (as well as a reset from the SIM) resets the MC68340 as well as all external devices. A reset signal from the CPU32 (asserted as part of the RESET instruction) resets external devices only — the internal state of the CPU32 is not affected; other on-chip modules are reset, but the configuration is not altered. When asserted by the MC68340, this signal is guaranteed to be asserted for a minimum of 512 clock cycles. Refer to **SECTION 3 BUS OPERATION** for a description of bus reset operation and **SECTION 5 CPU32** for information about the reset exception.

2.9.2 Halt ($\overline{\text{HALT}}$)

This active-low, open-drain, bidirectional signal is asserted to suspend external bus activity, to request a retry when used with $\overline{\text{BERR}}$, or to perform a single-step operation. As an output, $\overline{\text{HALT}}$ indicates a double bus fault by the CPU32. Refer to **SECTION 3 BUS OPERATION** for a description of the effects of $\overline{\text{HALT}}$ on bus operation.

2.9.3 Bus Error ($\overline{\text{BERR}}$)

This active-low input signal indicates that an invalid bus operation is being attempted or, when used with $\overline{\text{HALT}}$, that the processor should retry the current cycle. Refer to **SECTION 3 BUS OPERATION** for a description of the effects of $\overline{\text{BERR}}$ on bus operation.

2.10 CLOCK SIGNALS

These signals are used by the MC68340 for controlling or generating the system clocks. Refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for more information on the various clock signals.

2.10.1 System Clock (CLKOUT)

This output signal is the system clock and is used as the bus timing reference by external devices. CLKOUT can be slowed in low-power stop mode. See **SECTION 4 SYSTEM INTEGRATION MODULE** for more information.

2.10.2 Crystal Oscillator (EXTAL, XTAL)

These two pins are the connections for an external crystal to the internal oscillator circuit. If an external oscillator is used, it should be connected to EXTAL, with XTAL left open. See **SECTION 4 SYSTEM INTEGRATION MODULE** for more information.

2.10.3 External Filter Capacitor (XFC)

This pin is used to add an external capacitor to the filter circuit of the phase-locked loop. The capacitor should be connected between XFC and V_{CCSYN}.

2.10.4 Clock Mode Select (MODCK)

This pin selects the source of the internal system clock during reset. After reset, it can be programmed to be port B parallel I/O.

MODCK. The state of this active-high input signal during reset selects the source of the internal system clock. If MODCK is high during reset, the internal voltage-controlled oscillator (VCO) furnishes the system clock. If MODCK is low during reset, an external frequency appearing at the EXTAL pin furnishes the system clock.

Port B0. This pin can be used as port B parallel I/O. Refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for more information on parallel I/O signals.

2.11 INSTRUMENTATION AND EMULATION SIGNALS

These signals are used for test or software debugging.

2.11.1 Instruction Fetch ($\overline{\text{IFETCH}}$)

This active-low output signal indicates when the CPU32 is performing an instruction word prefetch and when the instruction pipeline has been flushed. Refer to **SECTION 5 CPU32** for information about $\overline{\text{IFETCH}}$.

2.11.2 Instruction Pipe ($\overline{\text{IPIPE}}$)

This active-low output signal is used to track movement of words through the instruction pipeline. Refer to **SECTION 5 CPU32** for information about $\overline{\text{IPIPE}}$.

2.11.3 Breakpoint ($\overline{\text{BKPT}}$)

This active-low input signal is used to signal a hardware breakpoint to the CPU32. Refer to **SECTION 5 CPU32** for information about $\overline{\text{BKPT}}$.

2.11.4 Freeze (FREEZE)

Assertion of this active-high output signal indicates the CPU32 has acknowledged a breakpoint and has initiated background mode operation. See **SECTION 5 CPU32** for more information about FREEZE and background mode.

2.12 TEST SIGNALS

The following signals are used with the onboard test logic defined by the IEEE 1149.1 standard. See **SECTION 9 IEEE 1149.1 TEST ACCESS PORT** for more information on the use of these signals.

2.12.1 Test Clock (TCK)

This input provides a clock for onboard test logic defined by the IEEE 1149.1 standard.

2.12.2 Test Mode Select (TMS)

This input controls test mode operations for onboard test logic defined by the IEEE 1149.1 standard.

2.12.3 Test Data In (TDI)

This input is used for serial test instructions and test data for onboard test logic defined by the IEEE 1149.1 standard.

2.12.4 Test Data Out (TDO)

This output is used for serial test instructions and test data for onboard test logic defined by the IEEE 1149.1 standard.

2.13 SERIAL MODULE SIGNALS

The following signals are used by the serial module for data and clock signals. See **SECTION 7 SERIAL MODULE** for more information on the serial module signals.

2.13.1 Serial Crystal Oscillator (X2,X1)

These pins furnish the connection to a crystal or external clock, which must be supplied when using the baud rate generator. An external clock is connected to the X1 pin only. See **SECTION 7 SERIAL MODULE** for more information.

2.13.2 Serial External Clock Input (SCLK)

This input can be used as the external clock input for channel A or channel B, bypassing the baud rate generator. The clock inputs are controlled by the clock select registers (CSR). See **SECTION 7 SERIAL MODULE** for more information.

2.13.3 Receive Data (RxDA, RxDB)

These input signals furnish serial data input to the serial module. Data is sampled on the rising edge of the selected clock source, with the least significant bit received first. See **SECTION 7 SERIAL MODULE** for more information.

2.13.4 Transmit Data (TxDA, TxDB)

These signals are the transmitter serial data outputs from the serial module. The output is held high (mark condition) when the transmitter is disabled, idle, or in local loopback mode. Data is shifted out on the falling edge of the selected clock source, with the least significant bit transmitted first. See **SECTION 7 SERIAL MODULE** for more information.

2.13.5 Clear to Send ($\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$)

These active-low inputs can be programmed as clear to send inputs. See **SECTION 7 SERIAL MODULE** for more information.

2.13.6 Request to Send ($\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$)

These active-low outputs can be programmed as request to send outputs or used as discrete outputs. See **SECTION 7 SERIAL MODULE** for more information.

$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$. These signals function as the channel request to send outputs.

OP1, OP0. These signals reflect the complement of the value of bit 1 and bit 0, respectively, in the output port data register.

2.13.7 Transmitter Ready ($\overline{\text{TxRDYA}}$)

This active-low output can be programmed as the channel A transmitter ready status indicator or used as a discrete output. See **SECTION 7 SERIAL MODULE** for information on controlling the function of this bit.

TxRDYA. This signal reflects the complement of the value of bit 2 in the channel A status register and can control parallel data flow by acting as an interrupt when the transmitter contains a character.

OP6. This signal reflects the complement of the value of bit 6 in the output port data register.

2.13.8 Receiver Ready (RxRDYA)

This active-low output can be programmed as the channel A receiver ready status indicator, the channel A FIFO full indicator, or used as a discrete output. See **SECTION 7 SERIAL MODULE** for information on controlling the function of this bit.

RxRDYA. This signal reflects the complement of the value of bit 1 in the interrupt status register and can control parallel data flow by acting as an interrupt when the receiver contains a character.

FFULLA. This signal reflects the complement of the value of bit 1 in the interrupt status register and can control parallel data flow by acting as an interrupt when the receiver FIFO is full.

OP4. This signal reflects the complement of the value of bit 4 in the output port data register.

2.14 DMA MODULE SIGNALS

The following signals are used by the direct memory access (DMA) module to provide external handshake for either a source or destination. See **SECTION 6 DMA MODULE** for additional information.

2.14.1 DMA Request (DREQ2, DREQ1)

These inputs from a peripheral start the DMA transfer process. The assertion level can be either active-low or falling edge.

2.14.2 DMA Acknowledge (DACK2, DACK1)

These outputs to a peripheral are asserted during accesses, after a DMA transfer is in progress.

2.14.3 DMA Done ($\overline{\text{DONE2}}$, $\overline{\text{DONE1}}$)

These active-low bidirectional signals indicate that the last transfer is being performed.

2.15 TIMER SIGNALS

The following external signals are used by the timer modules.

2.15.1 Timer Gate ($\overline{\text{TGATE2}}$, $\overline{\text{TGATE1}}$)

The low state of these input signals furnish counter enable inputs to the timer modules in most modes of operation. See **SECTION 8 TIMER MODULES** for additional information.

2.15.2 Timer Input (TIN2, TIN1)

The falling edge of these input signals can be programmed to furnish time references to the timer modules. See **SECTION 8 TIMER MODULES** for additional information on programming this function.

2.15.3 Timer Output (TOUT2, TOUT1)

These output signals provide the various output waveforms from the timer modules. See **SECTION 8 TIMER MODULES** for additional information.

2.16 SYNTHESIZER POWER (V_{CCSYN})

This pin supplies a quiet power source to the VCO to provide greater frequency stability.

2.17 SYSTEM POWER AND GROUND (V_{CC} AND GND)

These pins provide system power and return to the MC68340. Multiple pins are provided for adequate current capability. All power supply pins must have adequate bypass capacitance for high-frequency noise suppression.

2.18 SIGNAL SUMMARY

Table 2-5 presents a summary of all the signals discussed in the preceding paragraphs.

Table 2-5. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Three-State
Address Bus	A23–A0	Output	—	Yes
Address Bus/Port A7–A0/ $\overline{\text{IACK7}}$–$\overline{\text{IACK1}}$	A31–A24	Output/I/O/ Output	—/—/Low	Yes
Data Bus	D15–D0	I/O	—	Yes
Function Codes	FC3–FC0	Output	—	Yes
Chip Select/$\overline{\text{IRQ4}}$, $\overline{\text{IRQ2}}$, $\overline{\text{IRQ1}}$/Port B4, B2, B1, $\overline{\text{AVEC}}$	$\overline{\text{CS3}}$–$\overline{\text{CS0}}$	Output/Input/ I/O	Low/Low/—	No
Bus Request	$\overline{\text{BR}}$	Input	Low	—
Bus Grant	$\overline{\text{BG}}$	Output	Low	No
Bus Grant Acknowledge	$\overline{\text{BGACK}}$	Input	Low	
Data and Size Acknowledge	$\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$	Input	Low	—
Read-Modify-Write Cycle	$\overline{\text{RMC}}$	Output	Low	Yes
Address Strobe	$\overline{\text{AS}}$	Output	Low	Yes
Data Strobe	$\overline{\text{DS}}$	Output	Low	Yes
Size	SIZ1, SIZ0	Output	—	Yes
Read/Write	R/$\overline{\text{W}}$	Output	High/Low	Yes
Interrupt Request Level/Port B7, B6, B5, B3	$\overline{\text{IRQ7}}$, $\overline{\text{IRQ6}}$, $\overline{\text{IRQ5}}$, $\overline{\text{IRQ3}}$	Input/I/O	Low/—	—
Reset	$\overline{\text{RESET}}$	I/O	Low	No
Halt	$\overline{\text{HALT}}$	I/O	Low	No
Bus Error	$\overline{\text{BERR}}$	Input	Low	—
System Clock Out	CLKOUT	Output	—	No
Crystal Oscillator	EXTAL	Input	—	—
Crystal Oscillator	XTAL	Output	—	—
External Filter Capacitor	XFC	Input	—	—
Clock Mode Select/Port B0	MODCK	Input/I/O	—/—	—
Instruction Fetch	$\overline{\text{IFETCH}}$	Output	Low	No
Instruction Pipe	$\overline{\text{IPIPE}}$	Output	Low	No
Breakpoint	$\overline{\text{BKPT}}$	Input	Low	—
Freeze	FREEZE	Output	High	No
Receive Data	RxDA, RxDB	Input	—	—
Transmit Data	TxDA, RxDB	Output	—	No
Clear to Send	$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$	Input	Low	—

Table 2-5. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Three-State
Request to Send/OP1,OP0	$\overline{RTSA}$, $\overline{RTSB}$	Output/Output	Low/—	No
Serial Crystal Oscillator	X1	Input	—	—
Serial Crystal Oscillator	X2	Output	—	—
Serial Clock	SCLK	Input	—	—
Transmitter Ready/OP6	$\overline{TxRDYA}$	Output/Output	Low/—	No
Receiver Ready/FIFO Full/OP4	$\overline{RxRDYA}$	Output/Output/ Output	Low/Low/—	No
DMA Request	$\overline{DREQ2}$, $\overline{DREQ1}$	Input	Low	—
DMA Acknowledge	$\overline{DACK2}$, $\overline{DACK1}$	Output	Low	No
DMA Done	$\overline{DONE2}$, $\overline{DONE1}$	I/O	Low	No
Timer Gate	$\overline{TGATE2}$, $\overline{TGATE1}$	Input	Low	—
Timer Input	TIN2, TIN1	Input	—	—
Timer Output	TOUT2, TOUT1	Output	—	Yes
Test Clock	TCK	Input	—	—
Test Mode Select	TMS	Input	High	—
Test Data In	TDI	Input	High	—
Test Data Out	TDO	Output	High	—
Synchronizer Power	V_{CCSYN}	—	—	—
System Power Supply and Return	V_{CC}, GND	—	—	—

SECTION 3 BUS OPERATION

This section provides a functional description of the bus, the signals that control it, and the bus cycles provided for data transfer operations. It also describes the error and halt conditions, bus arbitration, and reset operation. Operation of the external bus is the same whether the MC68340 or an external device is the bus master; the names and descriptions of bus cycles are from the viewpoint of the bus master. For exact timing specifications, refer to MC68340/D, *MC68340 Technical Summary*.

The MC68340 architecture supports byte, word, and long-word operands allowing access to 8- and 16-bit data ports through the use of asynchronous cycles controlled by the size outputs (SIZ1, SIZ0) and data size acknowledge inputs ($\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$). The MC68340 requires word and long-word operands to be located in memory on word boundaries. The only type of transfer that can be performed to an odd address is a single-byte transfer, referred to as an odd-byte transfer. For an 8-bit port, multiple bus cycles may be required for an operand transfer due to a word or long-word operand.

3.1 BUS TRANSFER SIGNALS

The bus transfers information between the MC68340 and external memory or a peripheral device. External devices can accept or provide 8 bits or 16 bits in parallel and must follow the handshake protocol described in this section. The maximum number of bits accepted or provided during a bus transfer is defined as the port width. The MC68340 contains an address bus that specifies the address for the transfer and a data bus that transfers the data. Control signals indicate the beginning and type of the cycle as well as the address space and size of the transfer. The selected device then controls the length of the cycle with the signal(s) used to terminate the cycle. Strobe signals, one for the address bus and another for the data bus, indicate the validity of the address and provide timing information for the data. Both asynchronous and synchronous operation is possible for any port width. In asynchronous operation, the bus and control input signals are internally synchronized to the MC68340 clock, introducing a delay. This delay is the time required for the MC68340 to sample an input signal, synchronize the input to the internal clocks, and determine whether it is high or low. In synchronous operation, the bus and control input signals must be timed to setup and hold times. Since no synchronization is needed, bus cycles can be completed in three clock cycles in this mode. Additionally, using the fast-termination option of the chip-select signals, two-clock operation for 16-bit ports is possible.

Furthermore, for all inputs, the MC68340 latches the level of the input during a sample window around the falling edge of the clock signal. This window is illustrated in Figure 3-1. To ensure that an input signal is recognized on a specific falling edge of the clock, that input must be stable during the sample window. If an input makes a transition during the window time period, the level recognized by the MC68340 is not predictable; however, the MC68340 always resolves the latched level to either a logic high or low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

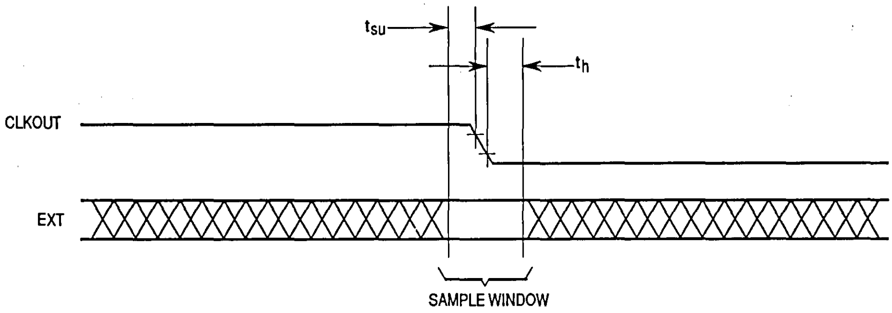


Figure 3-1. Input Sample Window

3.1.1 Bus Control Signals

The MC68340 initiates a bus cycle by driving the address, size, function code, and read/write outputs. At the beginning of a bus cycle, SIZ1 and SIZ0 are driven with the function code signals. SIZ1 and SIZ0 indicate the number of bytes remaining to be transferred during an operand cycle (consisting of one or more bus cycles). Table 3-1 lists the encoding of SIZ1 and SIZ0. These signals are valid while address strobe ($\overline{AS}$) is asserted. The read/write (R/$\overline{W}$) signal determines the direction of the transfer during a bus cycle. Driven at the beginning of a bus cycle, R/$\overline{W}$ is valid while $\overline{AS}$ is asserted. R/$\overline{W}$ only transitions when a write cycle is preceded by a read cycle or vice versa. The signal may remain low for consecutive write cycles. The read-modify-write cycle ($\overline{RMC}$)

signal is asserted at the beginning of the first bus cycle of a read-modify-write operation and remains asserted until completion of the final bus cycle of the operation.

Table 3-1. SIZx Signal Encoding

SIZ1	SIZ0	Transfer Size
0	1	Byte
1	0	Word
1	1	3 Byte
0	0	Long Word

3.1.2 Function Codes

The function code signals (FC3–FC0) are outputs that indicate one of 16 address spaces to which the address applies. Fifteen of these spaces are designated as either normal or direct memory access (DMA) cycle, user or supervisor, and program or data spaces. One other address space is designated as CPU space to allow the CPU32 to acquire specific control information not normally associated with read or write bus cycles. The function code signals are valid while $\overline{\text{AS}}$ is asserted.

Function codes (see Table 3-2) can be considered as extensions of the 32-bit linear address that can provide up to 16 different 4-Gbyte address spaces. Function codes are automatically generated by the CPU32 to select address spaces for data and program at both user and supervisor privilege levels, a CPU address space for processor functions, and an alternate master address space. User programs access only their own program and data areas to increase protection of system integrity and can be restricted from accessing other information. The S-bit in the CPU32 status register is set for supervisor accesses and cleared for user accesses to provide differentiation. Refer to **3.4 CPU SPACE CYCLES** for more information.

3.1.3 Address Bus (A31–A0)

The address bus signals are outputs that define the address of the byte (or the most significant byte) to be transferred during a bus cycle. The MC68340 places the address on the bus at the beginning of a bus cycle. The address is valid while $\overline{\text{AS}}$ is asserted.

Table 3-2. Address Space Encoding

Function Code Bits				Address Spaces
3	2	1	0	
0	0	0	0	Reserved (Motorola)
0	0	0	1	User Data Space
0	0	1	0	User Program Space
0	0	1	1	Reserved (User)
0	1	0	0	Reserved (Motorola)
0	1	0	1	Supervisor Data Space
0	1	1	0	Supervisor Program Space
0	1	1	1	CPU Space
1	x	x	x	DMA Space

3.1.4 Address Strobe ($\overline{\text{AS}}$)

$\overline{\text{AS}}$ is an output timing signal that indicates the validity of an address on the address bus and of many control signals. $\overline{\text{AS}}$ is asserted approximately one-half clock after the beginning of a bus cycle.

3.1.5 Data Bus (D15–D0)

The data bus signals comprise a bidirectional, nonmultiplexed, parallel bus that contains the data being transferred to or from the MC68340. A read or write operation may transfer 8 or 16 bits of data (one or two bytes) in one bus cycle. During a read cycle, the data is latched by the MC68340 on the last falling edge of the clock for that bus cycle. For a write cycle, all 16 bits of the data bus are driven, regardless of the port width or operand size. The MC68340 places the data on the data bus approximately one-half clock cycle after $\overline{\text{AS}}$ is asserted in a write cycle.

3.1.6 Data Strobe ($\overline{\text{DS}}$)

$\overline{\text{DS}}$ is an output timing signal that applies to the data bus. For a read cycle, the MC68340 asserts $\overline{\text{DS}}$ and $\overline{\text{AS}}$ simultaneously to signal the external device to place data on the bus. For a write cycle, $\overline{\text{DS}}$ signals to the external device that the data to be written is valid on the bus. The MC68340 asserts $\overline{\text{DS}}$ approximately one clock cycle after the assertion of $\overline{\text{AS}}$ during a write cycle.

3.1.7 Bus Cycle Termination Signals

The following signals can terminate a bus cycle.

3.1.7.1 DATA TRANSFER AND SIZE ACKNOWLEDGE SIGNALS (DSACK1 AND DSACK0). During bus cycles, external devices assert DSACK1 and/or DSACK0 as part of the bus protocol. During a read cycle, this signals the MC68340 to terminate the bus cycle and to latch the data. During a write cycle, this indicates that the external device has successfully stored the data and that the cycle may terminate. These signals also indicate to the MC68340 the size of the port for the bus cycle just completed (see Table 3-3). Refer to **3.3.1 Read Cycle** for timing relationships of DSACK1 and DSACK0.

Additionally, the system integration module (SIM) can be programmed to internally generate DSACK1 and DSACK0 for external accesses, eliminating logic required to generate these signals. The SIM can alternatively be programmed to generate a fast termination for 16-bit, synchronous accesses. Refer to **3.2.6 Fast Termination Cycles** for additional information on these cycles.

3.1.7.2 BUS ERROR (BERR). This signal is also a bus cycle termination indicator and can be used in the absence of DSACKx to indicate a bus error condition. BERR can also be asserted in conjunction with DSACKx to indicate a bus error condition, provided it meets the appropriate timing described in this section and in MC68340/D, *MC68340 Technical Summary*. Additionally, BERR and HALT can be asserted together to indicate a retry termination. Refer to **3.5 BUS EXCEPTION CONTROL CYCLES** for additional information on the use of these signals.

The internal bus monitor can be used to generate the BERR signal for internal and internal-to-external transfers in all the following descriptions. If the bus cycles of an external bus master are to be monitored, external BERR generation must be provided since the internal BERR monitor has no information about transfers initiated by an external bus master.

3.1.7.3 AUTOVECTOR (AVEC). This signal can be used to terminate interrupt acknowledge cycles, indicating that the MC68340 should internally generate a vector number to locate an interrupt handler routine. AVEC can be generated either externally or internally by the SIM (refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for additional information). AVEC is ignored during all other bus cycles.

3.2 DATA TRANSFER MECHANISM

The MC68340 supports byte, word, and long-word operands, allowing access to 8- and 16-bit data ports through the use of asynchronous cycles controlled by DSACK1 and DSACK0. The MC68340 also supports byte, word, and long-word operands, allowing access to 16-bit data ports through the use of synchronous cycles controlled by the fast termination capability of the SIM. The MC68340 supports 32-bit single address mode DMA transfers on a 32-bit external bus, in a single bus cycle, if a 32-bit DSACKx is provided.

3.2.1 Dynamic Bus Sizing

The MC68340 dynamically interprets the port size of the addressed device during each bus cycle, allowing operand transfers to or from 8- and 16-bit ports. During an operand transfer cycle, the slave device signals its port size (byte or word) and indicates completion of the bus cycle to the MC68340 through the use of the DSACKx inputs. Refer to Table 3-3 for DSACKx encoding.

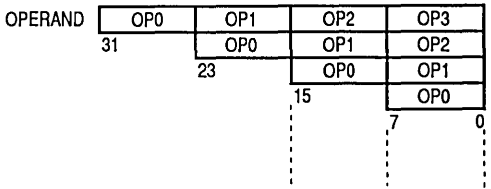
Table 3-3. DSACKx Codes and Results

DSACK1	DSACK0	Result
1 (Negated)	1 (Negated)	Insert Wait States in Current Bus Cycle
1 (Negated)	0 (Asserted)	Complete Cycle — Data Bus Port Size Is 8 Bits
0 (Asserted)	1 (Negated)	Complete Cycle — Data Bus Port Size Is 16 Bits
0 (Asserted)	0 (Asserted)	Reserved — Defaults to 16-Bit Port Size Can Be Used for 32-Bit DMA Cycles

For example, if the MC68340 is executing an instruction that reads a long-word operand from a 16-bit port, the MC68340 latches the 16 bits of valid data and runs another bus cycle to obtain the other 16 bits. The operation for an 8-bit port is similar but requires four read cycles. The addressed device uses DSACKx to indicate the port width. For instance, a 16-bit device always returns DSACKx for a 16-bit port (regardless of whether the bus cycle is a byte or word operation).

Dynamic bus sizing requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 16-bit port must reside on data bus bits 15–0, and an 8-bit port must reside on data bus bits 15–8. This requirement minimizes the number of bus cycles needed to transfer data to 8- and 16-bit ports and ensures that the MC68340 correctly transfers valid data.

The MC68340 always attempts to transfer the maximum amount of data on all bus cycles; for a word operation, it always assumes that the port is 16 bits wide when beginning the bus cycle. The bytes of operands are designated as shown in Figure 3-2. The most significant byte of a long-word operand is OP0, and OP3 is the least significant byte. The two bytes of a word-length operand are OP0 (most significant) and OP1. The single byte of a byte-length operand is OP0. These designations are used in the figures and descriptions that follow.



Case	Transfer Case	SIZ1	SIZ0	A0	$\overline{DSACK1}$	$\overline{DSACK0}$	Data Bus D15 – D8	Data Bus D7 – D0
(a)	Byte to Byte	0	1	X	1	0	OP0	(OP0)
(b)	Byte to Word (Even)	0	1	0	0	X	OP0	(OP0)
(c)	Byte to Word (Odd)	0	1	1	0	X	(OP0)	OP0
(d)	Word to Byte (Aligned)	1	0	0	1	0	OP0	(OP1)
(e)	Word to Byte (Misaligned)*	1	0	1	1	0	OP0	(OP0)
(f)	Word to Word (Aligned)	1	0	0	0	X	OP0	OP1
(g)	Word to Word (Misaligned)*	1	0	1	0	X	(OP0)	OP0
(h)	3 Byte to Byte (Aligned)*	1	1	0	1	0	OP0	(OP1)
(i)	3 Byte to Byte (Misaligned)*	1	1	1	1	0	OP0	(OP0)
(j)	3 Byte to Word (Aligned)*	1	1	0	0	X	OP0	OP1
(k)	3 Byte to Word (Misaligned)*	1	1	1	0	X	(OP0)	OP0
(l)	Long Word to Byte (Aligned)	0	0	0	1	0	OP0	(OP1)
(m)	Long Word to Byte (Misaligned)*	0	0	1	1	0	OP0	(OP0)
(n)	Long Word to Word (Aligned)	0	0	0	0	X	OP0	OP1
(o)	Long Word to Word (Misaligned)*	0	0	1	0	X	(OP0)	OP0

NOTES:

1. Operands in parentheses are ignored by the MC68340 during read cycles.
2. Misaligned and 3 byte transfer cases, identified by an asterisk, are not supported by the MC68340.
3. A 3-byte to byte transfer does occur as the second byte transfer of a long-word to byte port transfer.

Figure 3-2. MC68340 Interface to Various Port Sizes

Figure 3-2 shows the required organization of data ports on the MC68340 bus for both 8- and 16-bit devices. The four bytes shown in Figure 3-2 are connected through the internal data bus and data multiplexer to the external data bus. The data multiplexer establishes the necessary connections for different combinations of address and data sizes. The multiplexer takes the two bytes of the 16-bit bus and routes them to their required positions. The positioning of bytes is determined by the size (SIZ1 and SIZ0) and address (A0) outputs. The SIZ1 and SIZ0 outputs indicate the number of bytes to be transferred during the current bus cycle, as listed in Table 3-1. The number of bytes transferred during a write or read bus cycle is equal to or less than the size indicated by the SIZ1 and SIZ0 outputs, depending on port width. For example, during the first bus cycle of a long-word transfer to a word port, the size outputs indicate that four bytes are to be transferred although only two bytes are moved on that bus cycle.

The address line A0 also affects the operation of the data multiplexer. During an operand transfer, A31–A1 indicate the word base address of that portion of the operand to be accessed, and A0 indicates the byte offset from the base. Figure 3-2 lists the bytes required on the data bus for read cycles. The entries shown as OPn are portions of the requested operand that are read or written during that bus cycle and are defined by SIZ1, SIZ0, and A0 for the bus cycle. The transfer cases marked misaligned are not generated by the MC68340.

3.2.2 Misaligned Operands

In this architecture, the basic operand size is 16 bits. Operand misalignment refers to whether an operand is aligned on a word boundary or overlaps the word boundary, determined by address line A0. When A0 is low, the address is even and is a word and byte boundary. When A0 is high, the address is odd and is a byte boundary only. A byte operand is properly aligned at any address; a word or long-word operand is misaligned at an odd address.

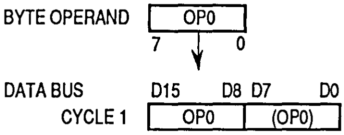
At most, each bus cycle can transfer a word of data aligned on a word boundary. If the MC68340 transfers a long-word operand over a 16-bit port, the most significant operand word is transferred on the first bus cycle, and the least significant operand word is transferred on a following bus cycle.

The CPU32 restricts all operands (both data and instructions) to be aligned. That is, word and long-word operands must be located on a word or long-word boundary, respectively. The only type of transfer that can be performed to an odd address is a single-byte transfer, referred to as an odd-byte transfer. If a misaligned access is attempted, the CPU32 generates an address error exception and enters exception processing. Refer to **SECTION 5 CPU32** for more information on exception processing.

3.2.3 Operand Transfer Cases

The following cases are examples of the allowable alignments of operands to ports.

3.2.3.1 BYTE OPERAND TO 8-BIT PORT, EVEN (A0=0). The MC68340 drives the address bus with the desired address and the size pins to indicate a single-byte operand.

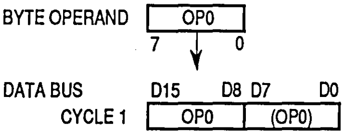


SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
0	1	X	1	0

For a read operation, the slave responds by placing data on bits 15-8 of the data bus, asserting $\overline{\text{DSACK0}}$ and negating $\overline{\text{DSACK1}}$ to indicate an 8-bit port. The MC68340 then reads the operand byte from bits 15–8 and ignores bits 7–0.

For a write operation, the MC68340 drives the single-byte operand on both bytes of the data bus because it does not know the port size until the $\overline{\text{DSACKx}}$ signals are read. The slave device reads the byte operand from bits 15–8 and places the operand in the specified location. The slave then asserts $\overline{\text{DSACK0}}$ to terminate the bus cycle.

3.2.3.2 BYTE OPERAND TO 16-BIT PORT, EVEN (A0=0). The MC68340 drives the address bus with the desired address and the size pins to indicate a single-byte operand.

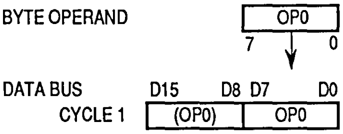


SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
0	1	0	0	X

For a read operation, the slave responds by placing data on bits 15–8 of the data bus and asserting $\overline{\text{DSACK1}}$ to indicate a 16-bit port. The MC68340 then reads the operand byte from bits 15–8 and ignores bits 7–0.

For a write operation, the MC68340 drives the single-byte operand on both bytes of the data bus because it does not know the port size until the $\overline{\text{DSACKx}}$ signals are read. The slave device reads the operand from bits 15–8 of the data bus and uses the address to place the operand in the specified location. The slave then asserts $\overline{\text{DSACK1}}$ to terminate the bus cycle.

3.2.3.3 BYTE OPERAND TO 16-BIT PORT, ODD (A0=1). The MC68340 drives the address bus with the desired address and the size pins to indicate a single-byte operand.

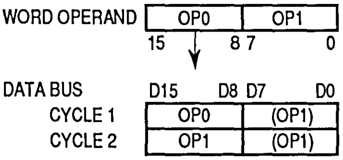


	SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
CYCLE 1	0	1	1	0	X

For a read operation, the slave responds by placing data on bits 7–0 of the data bus and asserting $\overline{\text{DSACK1}}$ to indicate a 16-bit port. The MC68340 then reads the operand byte from bits 7–0 and ignores bits 15–8.

For a write operation, the MC68340 drives the single-byte operand on both bytes of the data bus because it does not know the port size until the $\overline{\text{DSACKx}}$ signals are read. The slave device reads the operand from bits 7–0 of the data bus and uses the address to place the operand in the specified location. The slave then asserts $\overline{\text{DSACK1}}$ to terminate the bus cycle.

3.2.3.4 WORD OPERAND TO 8-BIT PORT, ALIGNED. The MC68340 drives the address bus with the desired address and the size pins to indicate a word operand.

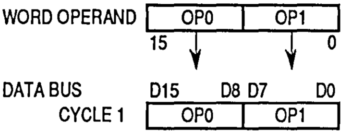


	SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
CYCLE 1	1	0	0	1	0
CYCLE 2	0	1	1	1	0

For a read operation, the slave responds by placing the most significant byte of the operand on bits 15–8 of the data bus and asserting $\overline{\text{DSACK0}}$ to indicate an 8-bit port. The MC68340 reads the most significant byte of the operand from bits 15–8 and ignores bits 7–0. The MC68340 then decrements the transfer size counter, increments the address, and reads the least significant byte of the operand from bits 15–8 of the data bus.

For a write operation, the MC68340 drives the word operand on bits 15–0 of the data bus. The slave device then reads the most significant byte of the operand from bits 15–8 of the data bus and asserts $\overline{\text{DSACK0}}$ to indicate that it received the data but is an 8-bit port. The MC68340 then decrements the transfer size counter, increments the address, and writes the least significant byte of the operand to bits 15–8 of the data bus.

3.2.3.5 WORD OPERAND TO 16-BIT PORT, ALIGNED. The MC68340 drives the address bus with the desired address and the size pins to indicate a word operand.

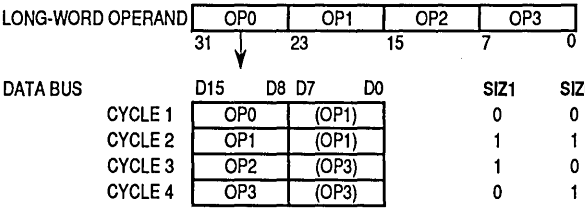


SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
1	0	0	0	X

For a read operation, the slave responds by placing the data on bits 15–0 of the data bus and asserting $\overline{\text{DSACK1}}$ to indicate a 16-bit port. When $\overline{\text{DSACK1}}$ is asserted, the MC68340 reads the data on the data bus and terminates the cycle.

For a write operation, the MC68340 drives the word operand on bits 15–0 of the data bus. The slave device then reads the entire operand from bits 15–0 of the data bus and asserts $\overline{\text{DSACK1}}$ to terminate the bus cycle.

3.2.3.6 LONG-WORD OPERAND TO 8-BIT PORT, ALIGNED. The MC68340 drives the address bus with the desired address and the size pins to indicate a long-word operand.



	SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
CYCLE 1	0	0	0	1	0
CYCLE 2	1	1	1	1	0
CYCLE 3	1	0	0	1	0
CYCLE 4	0	1	1	1	0

For a read operation, shown in Figure 3-3, the slave responds by placing the most significant byte of the operand on bits 15–8 of the data bus and asserting DSACK0 to indicate an 8-bit port. The MC68340 reads the most significant byte of the operand (byte 0) from bits 15–8 and ignores bits 7–0. The MC68340 then decrements the transfer size counter, increments the address, initiates a new cycle, and reads byte 1 of the operand from bits 15–8 of the data bus. The MC68340 repeats the process of decrementing the transfer size counter, incrementing the address, initiating a new cycle, and reading a byte to transfer the remaining two bytes.

For a write operation, shown in Figure 3-4, the MC68340 drives the two most significant bytes of the operand on bits 15–0 of the data bus. The slave device

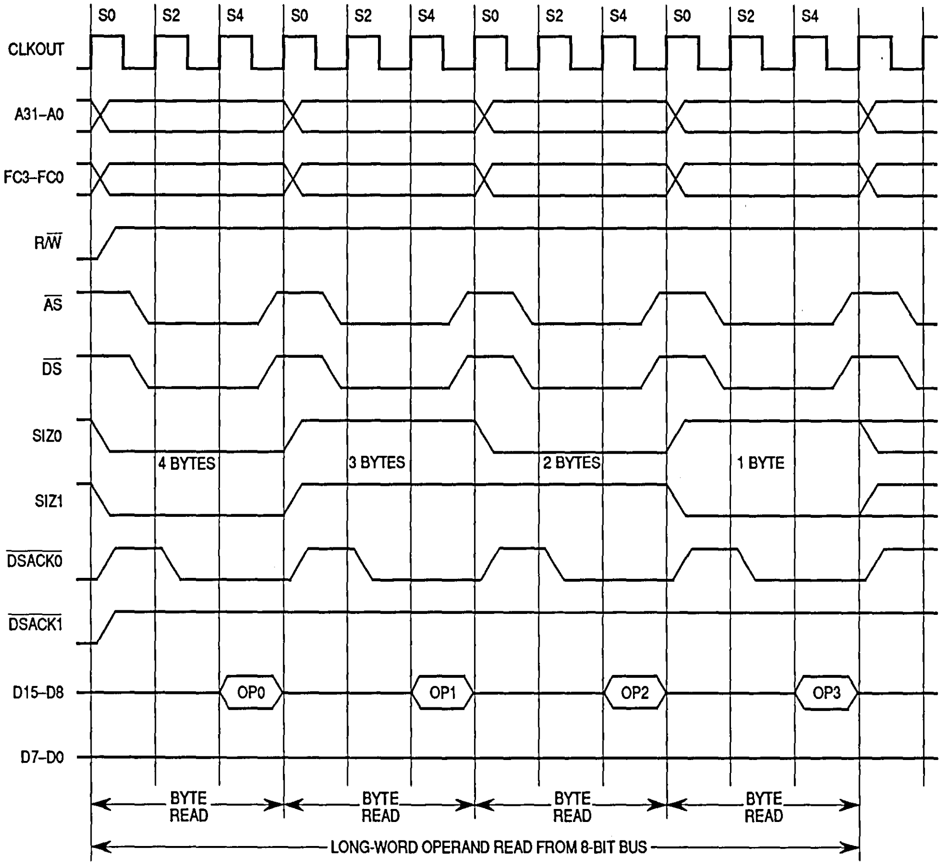


Figure 3-3. Long-Word Operand Read Timing from 8-Bit Port

then reads only the most significant byte of the operand (byte 0) from bits 15–8 of the data bus and asserts DSACK0 to indicate reception and an 8-bit port. The MC68340 then decrements the transfer size counter, increments the address, and writes byte 1 of the operand to bits 15–8 of the data bus. The MC68340 continues to decrement the transfer size counter, increment the address, and write a byte to transfer the remaining two bytes to the slave device.

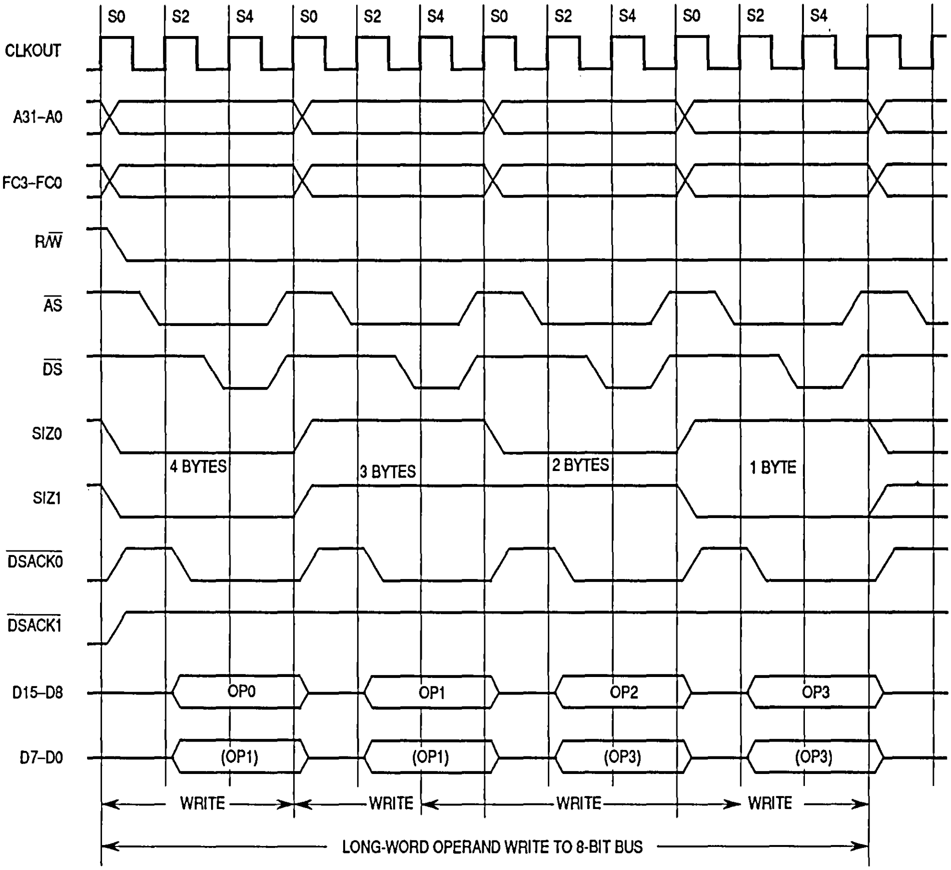
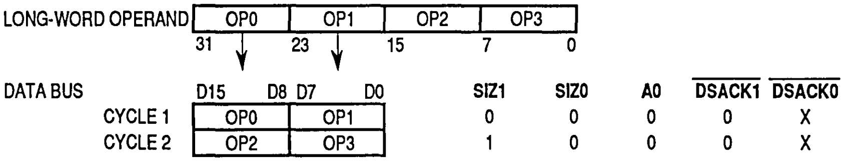


Figure 3-4. Long-Word Write Operand Timing to 8-Bit Port

3.2.3.7 LONG-WORD OPERAND TO 16-BIT PORT, ALIGNED.

Figure 3-5 shows both long-word and word read and write timing to a 16-bit port.



DATA BUS	D15 D8	D7 D0	SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
CYCLE 1	OP0	OP1	0	0	0	0	X
CYCLE 2	OP2	OP3	1	0	0	0	X

The MC68340 drives the address bus with the desired address and drives the size pins to indicate a long-word operand. For a read operation, the slave responds by placing the two most significant bytes of the operand on bits 15–0 of the data bus and asserting $\overline{\text{DSACK1}}$ to indicate a 16-bit port. The MC68340 reads the two most significant bytes of the operand (bytes 0 and 1) from bits 15–0. The MC68340 then decrements the transfer size counter, increments the address, initiates a new cycle, and reads bytes 2 and 3 of the operand from bits 15–0 of the data bus.

For a write operation, the MC68340 drives the two most significant bytes of the operand on bits 15–0 of the data bus. The slave device then reads the two most significant bytes of the operand (bytes 0 and 1) from bits 15–0 of the data bus and asserts $\overline{\text{DSACK1}}$ to indicate data reception and a 16-bit port. The MC68340 then decrements the transfer size counter by 2, increments the address by 2, and writes bytes 2 and 3 of the operand to bits 15–0 of the data bus.

3.2.4 Bus Operation

The MC68340 bus is asynchronous, allowing external devices connected to the bus to operate at clock frequencies different from the clock for the MC68340. Bus operation uses the handshake lines ($\overline{\text{AS}}$, $\overline{\text{DS}}$, $\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$) to control data transfers. $\overline{\text{AS}}$ signals a valid address on the address bus, and $\overline{\text{DS}}$ is used as a condition for valid data on a write cycle. Decoding the size outputs and lower address line A0 provides strobes that select the active portion of the data bus. The slave device (memory or peripheral) responds by placing the requested data on the correct portion of the data bus for a read cycle or by latching the data on a write cycle; the slave asserts the $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ combination that corresponds to the port size to terminate the cycle. Alternatively, the SIM can be programmed to assert the $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ combination internally and respond for the slave. If no slave responds or the access is invalid, external control logic may assert $\overline{\text{BERR}}$ or $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ to abort or retry the

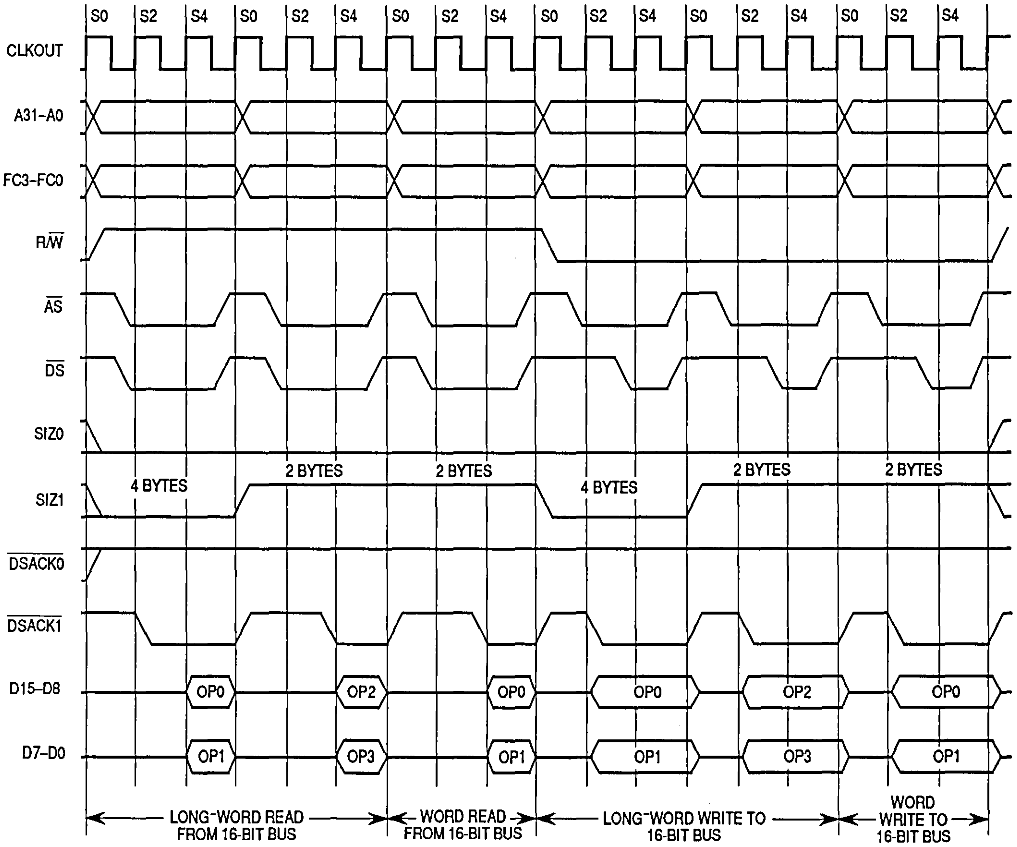


Figure 3-5. Long-Word and Word Read and Write Timing — 16-Bit Port

bus cycle, respectively. DSACKx can be asserted before the data from a slave device is valid on a read cycle. The length of time that DSACKx may precede data must not exceed a specified value in any asynchronous system to ensure that valid data is latched into the MC68340. (See MC68340/D, *MC68340 Technical Summary* for timing parameters.) Note that no maximum time is specified from the assertion of AS to the assertion of DSACKx. Although the MC68340 can transfer data in a minimum of three clock cycles when the cycle is terminated with DSACKx, the MC68340 inserts wait cycles in clock-period increments until DSACKx is recognized. BERR and/or HALT can be asserted after DSACK is asserted. BERR and/or HALT must be asserted within the time specified after DSACKx is asserted in any asynchronous system. If this maximum delay time is violated, the MC68340 may exhibit erratic behavior.

3.2.5 Synchronous Operation with DSACKx

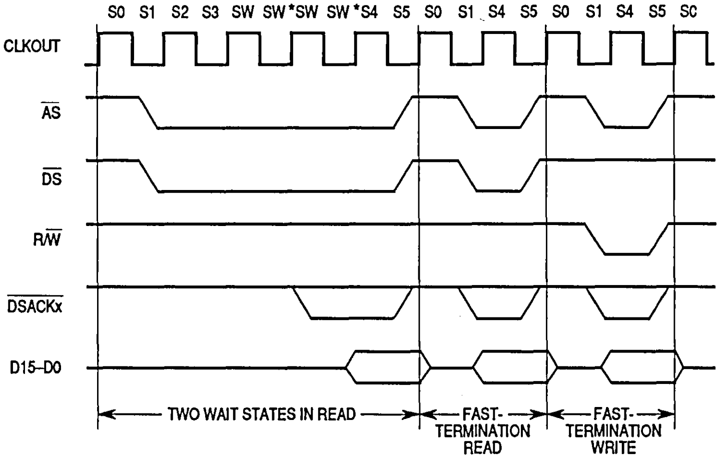
Although cycles terminated with DSACKx are classified as asynchronous, cycles terminated with DSACKx can also operate synchronously in that signals are interpreted relative to clock edges. The devices that use these cycles must synchronize the response to the MC68340 clock (CLKOUT) to be synchronous. Since the devices terminate bus cycles with DSACKx, the dynamic bus sizing capabilities of the MC68340 are available. The minimum cycle time for these cycles is also three clocks. To support systems that use the system clock to generate DSACKx and other asynchronous inputs, the asynchronous input setup time and the asynchronous input hold time are given. If the setup and hold times are met for the assertion or negation of a signal, such as DSACKx, the MC68340 is guaranteed to recognize that signal level on that specific falling edge of the system clock. If the assertion of DSACKx is recognized on a particular falling edge of the clock, valid data is latched into the MC68340 (for a read cycle) on the next falling clock edge if the data meets the data setup time. In this case, the parameter for asynchronous operation can be ignored. The timing parameters are described in MC68340/D, *MC68340 Technical Summary*.

If a system asserts DSACKx for the required window around the falling edge of S2 and obeys the proper bus protocol by maintaining DSACKx (and/or BERR/HALT) until and throughout the clock edge that negates AS (with the appropriate asynchronous input hold time), no wait states are inserted. The bus cycle runs at its maximum speed for bus cycles terminated with DSACKx (three clocks per cycle). When BERR (or BERR and HALT) is asserted after DSACKx, BERR (and HALT) must meet the appropriate setup time prior to the falling clock edge one clock cycle after DSACKx is recognized. This setup time is critical, and the MC68340 may exhibit erratic behavior if it is violated. When operating synchronously, the data-in setup and hold times for synchronous cycles may be used instead of the timing requirements for data relative to DS.

3.2.6 Fast-Termination Cycles

With an external device that has a fast access time, the chip-select circuit fast-termination enable (FTE) can provide a two-clock external bus transfer. Since the chip-select circuits are driven from the system clock, the bus cycle termination is inherently synchronized with the system clock. When fast termination is selected, the DD and PS bits of the corresponding address mask register are overridden. Fast termination can only be used with a 16-bit port and zero wait states. To use the fast-termination option, an external device should be fast enough to have data ready, within the specified setup time, by the falling edge of S4. Figure 3-6 shows the $\overline{\text{DSACKx}}$ timing for two wait states in read and a fast-termination read and write. When using the fast-termination option, $\overline{\text{DS}}$ is asserted only in a read cycle, not in a write cycle.

Refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for more information on chip selects.



* $\overline{\text{DSACKx}}$ only internally asserted for fast-termination cycles.

Figure 3-6. Fast-Termination Timing

3.3 DATA TRANSFER CYCLES

The transfer of data between the MC68340 and other devices involves the following signals:

- Address Bus A31–A0
- Data Bus D15–D0
- Control Signals

The address and data buses are both parallel, nonmultiplexed buses. The bus master moves data on the bus by issuing control signals, and the bus uses a handshake protocol to ensure correct movement of the data. In all bus cycles, the bus master is responsible for deskewing all signals it issues at both the start and end of the cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave devices. The following paragraphs define read, write, and read-modify-write cycle operations. Each bus cycle is defined as a succession of states that apply to the bus operation. These states are different from the MC68340 states described for the CPU32. The clock cycles used in the descriptions and timing diagrams of data transfer cycles are independent of the clock frequency. Bus operations are described in terms of external bus states.

3.3.1 Read Cycle

During a read cycle, the MC68340 receives data from a memory or peripheral device. If the instruction specifies a long-word or word operation, the MC68340 attempts to read two bytes at once. For a byte operation, the MC68340 reads one byte. The section of the data bus from which each byte is read depends on the operand size, address signal A0, and the port size. Refer to **3.2.1 Dynamic Bus Sizing** and **3.2.2 Misaligned Operands** for more information. Figure 3-7 is a flowchart of a word read cycle.

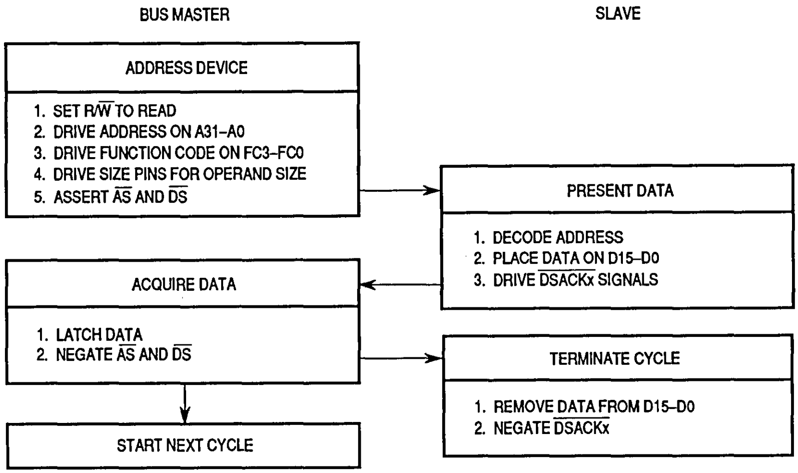

BUS MASTER
SLAVE
ADDRESS DEVICE
1. SET R/$\overline{W}$ TO READ
2. DRIVE ADDRESS ON A31–A0
3. DRIVE FUNCTION CODE ON FC3–FC0
4. DRIVE SIZE PINS FOR OPERAND SIZE
5. ASSERT $\overline{AS}$ AND $\overline{DS}$
PRESENT DATA
1. DECODE ADDRESS
2. PLACE DATA ON D15–D0
3. DRIVE $\overline{DSACKx}$ SIGNALS
ACQUIRE DATA
1. LATCH DATA
2. NEGATE $\overline{AS}$ AND $\overline{DS}$
TERMINATE CYCLE
1. REMOVE DATA FROM D15–D0
2. NEGATE $\overline{DSACKx}$
START NEXT CYCLE

Figure 3-7. Word Read Cycle Flowchart

State 0 — The read cycle starts in state 0 (S0). During S0, the MC68340 places a valid address on A31–A0 and valid function codes on FC3–FC0. The function codes select the address space for the cycle. The MC68340 drives R/$\overline{\mathrm{W}}$ high for a read cycle. SIZ1 and SIZ0 become valid, indicating the number of bytes requested for transfer.

State 1 — One-half clock later, in state 1 (S1), the MC68340 asserts $\overline{\mathrm{AS}}$ indicating a valid address on the address bus. The MC68340 also asserts $\overline{\mathrm{DS}}$ during S1. The selected device uses R/$\overline{\mathrm{W}}$, SIZ1 or SIZ0, A0, and $\overline{\mathrm{DS}}$ to place its information on the data bus. One or both of the bytes (D15–D8 and D7–D0) are selected by SIZ1, SIZ0, and A0. Concurrently, the selected device asserts $\overline{\mathrm{DSACKx}}$.

State 2 — As long as at least one of the $\overline{\mathrm{DSACKx}}$ signals is recognized on the falling edge of S2 (meeting the asynchronous input setup time requirement), data is latched on the falling edge of S4, and the cycle terminates.

State 3 — If $\overline{\mathrm{DSACKx}}$ is not recognized by the start of state 3 (S3), the MC68340 inserts wait states instead of proceeding to states 4 and 5. To ensure that wait states are inserted, both $\overline{\mathrm{DSACK1}}$ and $\overline{\mathrm{DSACK0}}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the MC68340 continues to sample $\overline{\mathrm{DSACKx}}$ on the falling edges of the clock until one is recognized.

State 4 — At the falling edge of state 4 (S4), the MC68340 latches the incoming data and samples $\overline{\mathrm{DSACKx}}$ to get the port size.

State 5 — The MC68340 negates $\overline{\mathrm{AS}}$ and $\overline{\mathrm{DS}}$ during state 5 (S5). It holds the address valid during S5 to provide address hold time for memory systems. R/$\overline{\mathrm{W}}$, SIZ1, SIZ0, and FC3–FC0 also remain valid throughout S5. The external device keeps its data and $\overline{\mathrm{DSACKx}}$ signals asserted until it detects the negation of $\overline{\mathrm{AS}}$ or $\overline{\mathrm{DS}}$ (whichever it detects first). The device must remove its data and negate $\overline{\mathrm{DSACKx}}$ within approximately one clock period after sensing the negation of $\overline{\mathrm{AS}}$ or $\overline{\mathrm{DS}}$. $\overline{\mathrm{DSACKx}}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

3.3.2 Write Cycle

During a write cycle, the MC68340 transfers data to memory or a peripheral device. Figure 3-8 is a flowchart of a write cycle operation for a word transfer.

State 0 — The write cycle starts in S0. During S0, the MC68340 places a valid address on A31–A0 and valid function codes on FC3–FC0. The function codes select the address space for the cycle. The MC68340 drives R/W low for a write cycle. SIZ1 and SIZ0 become valid, indicating the number of bytes to be transferred.

State 1 — One-half clock later, in S1, the MC68340 asserts AS, indicating a valid address on the address bus.

State 2 — During S2, the MC68340 places the data to be written onto D15–D0 and samples DSACKx at the end of S2.

State 3 — The MC68340 asserts DS during S3, indicating that data is stable on the data bus. As long as at least one of the DSACKx signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), the cycle terminates one clock later. If DSACKx is not recognized by the start of S3, the MC68340 inserts wait states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both DSACK1 and DSACK0 must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the MC68340 continues to sample DSACKx on the falling edges of the clock until one is recognized. The selected device uses R/W, SIZ1, SIZ0, and A0 to latch data from the appropriate byte(s) of D15–D8 and D7–D0. SIZ1, SIZ0, and A0 select the

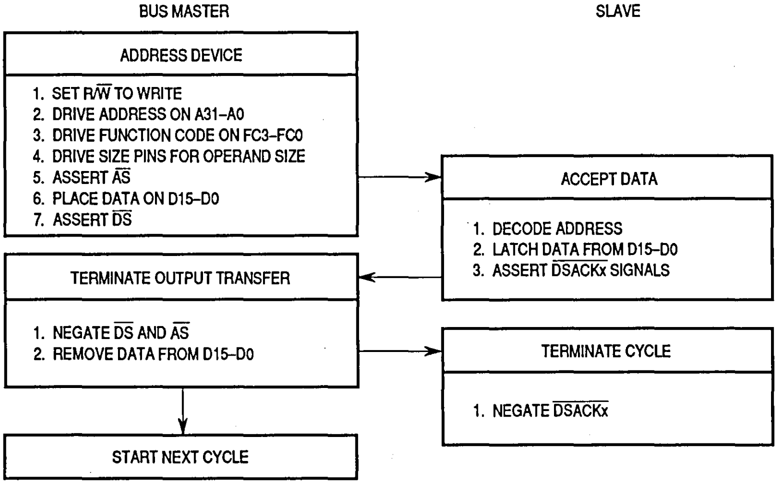


Figure 3-8. Write Cycle Flowchart

bytes of the data bus. If it has not already done so, the device asserts $\overline{\text{DSACKx}}$ to signal that it has successfully stored the data.

State 4 — The MC68340 issues no new control signals during S4.

State 5 — The MC68340 negates $\overline{\text{AS}}$ and $\overline{\text{DS}}$ during S5. It holds the address and data valid during S5 to provide address hold time for memory systems. R/$\overline{\text{W}}$, SIZ1, SIZ0, and FC3–FC0 also remain valid throughout S5. The external device must keep $\overline{\text{DSACKx}}$ asserted until it detects the negation of $\overline{\text{AS}}$ or $\overline{\text{DS}}$ (whichever it detects first). The device must negate $\overline{\text{DSACKx}}$ within approximately one clock period after sensing the negation of $\overline{\text{AS}}$ or $\overline{\text{DS}}$. $\overline{\text{DSACKx}}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

3.3.3 Read-Modify-Write Cycle

The read-modify-write cycle performs a read, conditionally modifies the data in the arithmetic logic unit, and may write the data out to memory. In the MC68340, this operation is indivisible, providing semaphore capabilities for multiprocessor systems. During the entire read-modify-write sequence, the MC68340 asserts $\overline{\text{RMC}}$ to indicate that an indivisible operation is occurring. The MC68340 does not issue a bus grant ($\overline{\text{BG}}$) signal in response to a bus request ($\overline{\text{BR}}$) signal during this operation. Figure 3-9 is an example of a functional timing diagram of a read-modify-write instruction specified in terms of clock periods.

State 0 — The MC68340 asserts $\overline{\text{RMC}}$ in S0 to identify a read-modify-write cycle. The MC68340 places a valid address on A31–A0 and valid function codes on FC3–FC0. The function codes select the address space for the operation. SIZ1 and SIZ0 become valid in S0 to indicate the operand size. The MC68340 drives R/$\overline{\text{W}}$ high for the read cycle.

State 1 — One-half clock later, in S1, the MC68340 asserts $\overline{\text{AS}}$ indicating a valid address on the address bus. The MC68340 also asserts $\overline{\text{DS}}$ during S1.

State 2 — The selected device uses R/$\overline{\text{W}}$, SIZ1, SIZ0, A0, and $\overline{\text{DS}}$ to place information on the data bus. Either or both of the bytes (D15–D8 and D7–D0) are selected by SIZ1, SIZ0, and A0. Concurrently, the selected device may assert $\overline{\text{DSACKx}}$.

State 3 — As long as at least one of the $\overline{\text{DSACKx}}$ signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), data is latched on the next falling edge of the clock, and the cycle terminates. If $\overline{\text{DSACKx}}$ is not recognized by the start of S3, the MC68340 inserts wait

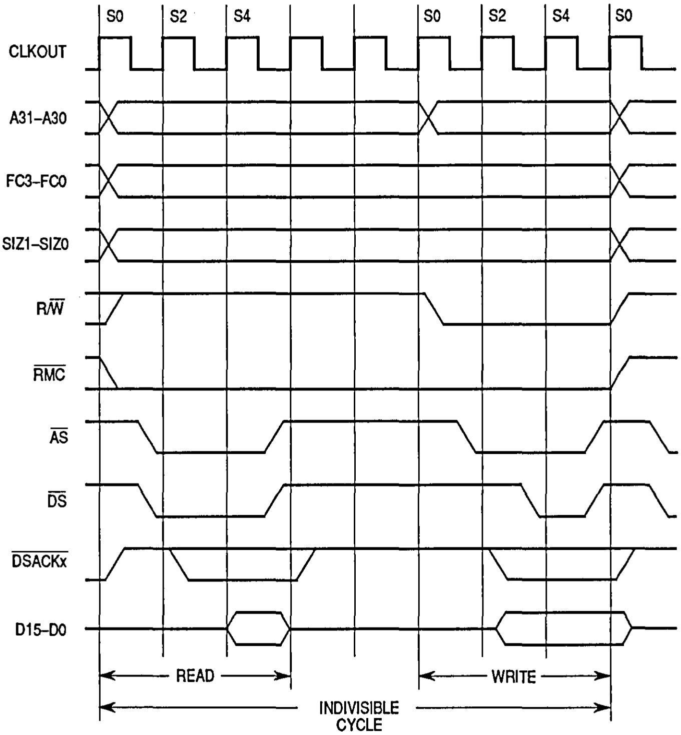


Figure 3-9. Read-Modify-Write Cycle Timing

states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both $\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the MC68340 continues to sample $\overline{\text{DSACKx}}$ on the falling edges of the clock until one is recognized.

State 4 — At the end of S4, the MC68340 latches the incoming data.

State 5 — The MC68340 negates $\overline{\text{AS}}$ and $\overline{\text{DS}}$ during S5. If more than one read cycle is required to read in the operand(s), S0–S5 are repeated for each read cycle. When finished reading, the MC68340 holds the address, R/$\overline{\text{W}}$, and FC3–FC0 valid in preparation for the write portion of the cycle. The external device keeps its data and $\overline{\text{DSACKx}}$ signals asserted until it detects the negation of $\overline{\text{AS}}$ or $\overline{\text{DS}}$ (whichever it detects first). The device must remove the data and negate $\overline{\text{DSACKx}}$ within approximately one clock period after sensing the negation of $\overline{\text{AS}}$ or $\overline{\text{DS}}$. $\overline{\text{DSACKx}}$ signals that remain asserted beyond this limit may be prematurely detected for the next portion of the operation.

Idle States — The MC68340 does not assert any new control signals during the idle states, but it may internally begin the modify portion of the cycle at this time. S0–S5 are omitted if no write cycle is required. If a write cycle is required, R/$\overline{\text{W}}$ remains in the read mode until S0 to prevent bus conflicts with the preceding read portion of the cycle; the data bus is not driven until S2.

State 0 — The MC68340 drives R/$\overline{\text{W}}$ low for a write cycle. Depending on the write operation to be performed, the address lines may change during S0.

State 1 — In S1, the MC68340 asserts $\overline{\text{AS}}$, indicating a valid address on the address bus.

State 2 — During S2, the MC68340 places the data to be written onto D15–D0.

State 3 — The MC68340 asserts $\overline{\text{DS}}$ during S3, indicating stable data on the data bus. As long as at least one of the $\overline{\text{DSACKx}}$ signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), the cycle terminates one clock later. If $\overline{\text{DSACKx}}$ is not recognized by the start of S3, the MC68340 inserts wait states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both $\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the MC68340 continues to sample $\overline{\text{DSACKx}}$ on the falling edges of the clock until one is recognized. The selected device uses R/$\overline{\text{W}}$, $\overline{\text{DS}}$, SIZ1, SIZ0, and A0 to latch data from the appropriate section(s) of D15–D8 and D7–D0. SIZ1, SIZ0, and A0 select the data bus sections. If it has not already done so, the device asserts $\overline{\text{DSACKx}}$ when it has successfully stored the data.

State 4 — The MC68340 issues no new control signals during S4.

State 5 — The MC68340 negates $\overline{\text{AS}}$ and $\overline{\text{DS}}$ during S5. It holds the address and data valid during S5 to provide address hold time for memory systems. R/$\overline{\text{W}}$ and FC3–FC0 also remain valid throughout S5. If more than one write cycle is required, states S0–S5 are repeated for each write cycle. The external device keeps $\overline{\text{DSACKx}}$ asserted until it detects the negation of $\overline{\text{AS}}$ or $\overline{\text{DS}}$ (whichever it detects first). The device must remove its data and negate $\overline{\text{DSACKx}}$ within approximately one clock period after sensing the negation of $\overline{\text{AS}}$ or $\overline{\text{DS}}$.

3.4 CPU SPACE CYCLES

FC3–FC0 select user and supervisor program and data areas. The area selected by function code FC3–FC0=$7 is classified as the CPU space. The breakpoint acknowledge, LPSTOP broadcast, module base address register access, and interrupt acknowledge cycles described in the following paragraphs use CPU space. The CPU space type, which is encoded on A19–A16 during a CPU space operation, indicates the function that the MC68340 is performing. On the MC68340, four of the encodings are implemented as shown in Figure 3-10. All unused values are reserved by Motorola for additional CPU space types.

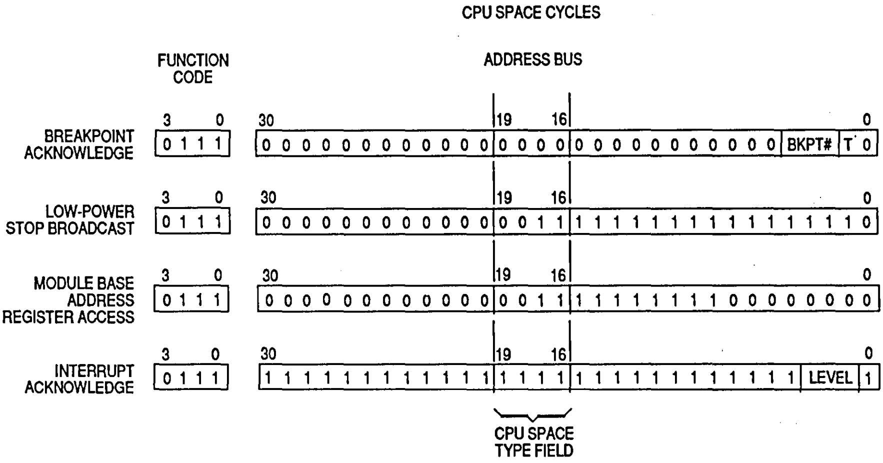


Figure 3-10. CPU Space Address Encoding

3.4.1 Breakpoint Acknowledge Cycle

The breakpoint acknowledge cycle allows external hardware to insert an instruction directly into the instruction pipeline as the program executes. The breakpoint acknowledge cycle is generated by the execution of a breakpoint instruction (BKPT) or the assertion of the breakpoint pin. The T-bit state (shown in Figure 3-10) differentiates a software breakpoint cycle from a hardware breakpoint cycle.

When a BKPT is executed, the MC68340 performs a word read from CPU space, type 0, at an address corresponding to the breakpoint number (bits [2–0] of the BKPT opcode) on A4–A2, and the T-bit (A1) is cleared. If this bus cycle is terminated with $\overline{\text{BERR}}$ (i.e., no instruction word available), the MC68340 then performs illegal instruction exception processing. If the bus cycle is terminated by $\overline{\text{DSACKx}}$, the MC68340 uses the data on D15–D0 (for 16-bit ports) or two reads from D15–D8 (for 8-bit ports) to replace the BKPT instruction in the internal instruction pipeline and then begins execution of that instruction.

When the CPU32 acknowledges breakpoint pin assertion with background mode disabled, the CPU32 performs a word read from CPU space, type 0, at an address corresponding to all ones on A4–A2 (BKPT#7), and the T-bit (A1) set. If this bus cycle is terminated by $\overline{\text{BERR}}$, the MC68340 performs hardware breakpoint exception processing. If this bus cycle is terminated by $\overline{\text{DSACKx}}$, the MC68340 ignores data on the data bus and continues execution of the next instruction.

NOTE

The $\overline{\text{BKPT}}$ pin is sampled on the same clock phase as data and is latched with data as it enters the CPU32 pipeline. If $\overline{\text{BKPT}}$ is asserted for only one bus cycle and a pipeline flush occurs before $\overline{\text{BKPT}}$ is detected by the CPU32, $\overline{\text{BKPT}}$ is ignored. To ensure detection of $\overline{\text{BKPT}}$ by the CPU32, $\overline{\text{BKPT}}$ can be asserted until a breakpoint acknowledge cycle is recognized.

The breakpoint operation flowchart is shown in Figure 3-11. Figures 3-12 and 3-13 show the timing diagrams for the breakpoint acknowledge cycle with instruction opcodes supplied on the cycle and with an exception signaled, respectively.

3.4.2 LPSTOP Broadcast Cycle

The LPSTOP broadcast cycle is generated by the CPU32 executing the LPSTOP instruction. The external bus interface must get a copy of the interrupt mask level from the CPU32, so the CPU32 performs a CPU space type 3 write with the mask level encoded on the data bus, as shown in the following figure. The CPU space type 3 cycle is shown externally, if the bus is available, to indicate to external devices that the MC68340 is going into low-power stop mode. The SIM provides $\overline{\text{DSACKx}}$ response to this cycle. A complete description of how the SIM responds to low-power stop mode is included in **SECTION 4 SYSTEM INTEGRATION MODULE.**

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	—	—	—	—	—	I2	I1	I0
RESET:																
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

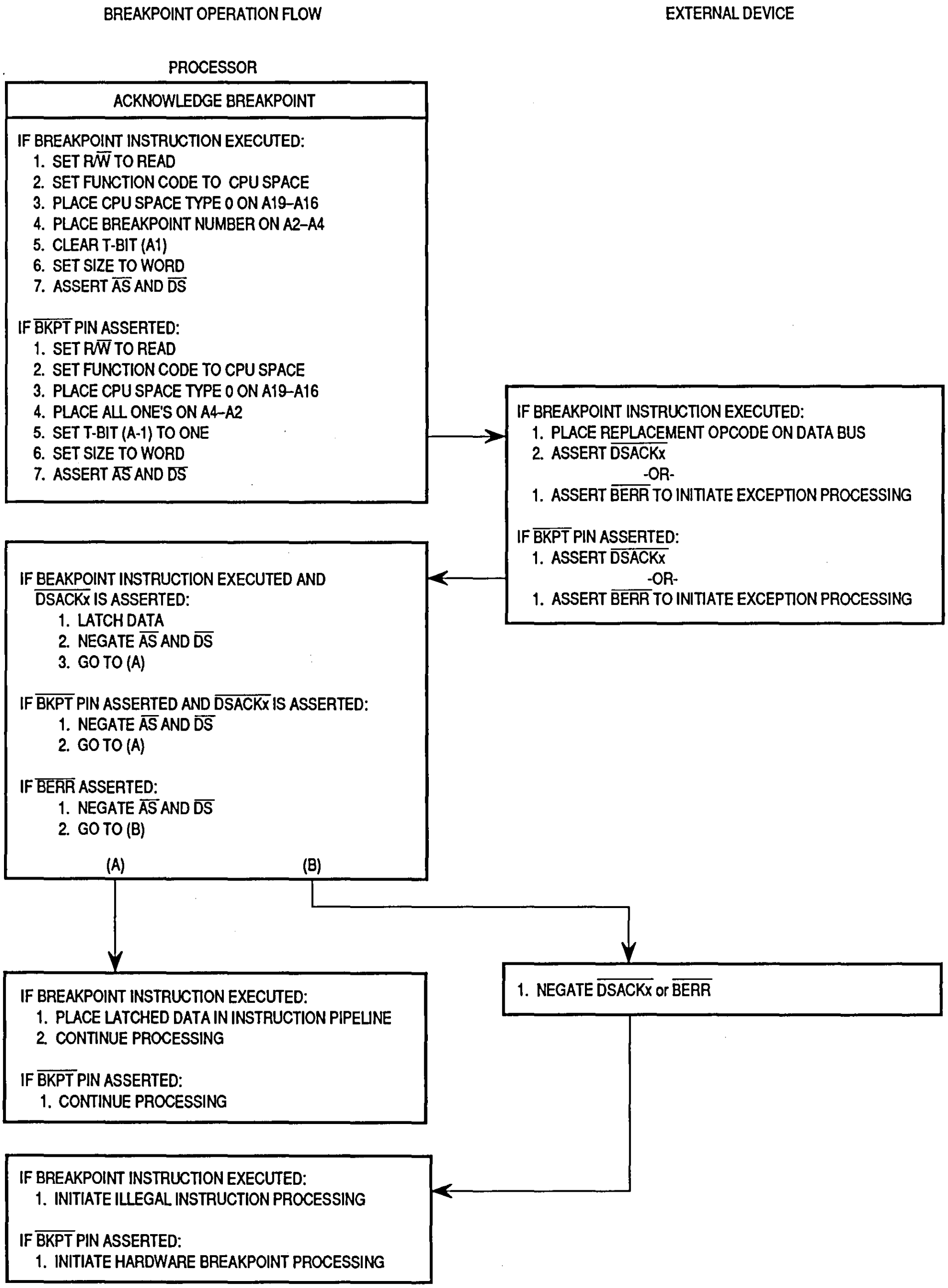

BREAKPOINT OPERATION FLOW
EXTERNAL DEVICE
PROCESSOR
ACKNOWLEDGE BREAKPOINT
IF BREAKPOINT INSTRUCTION EXECUTED:
1. SET R/$\overline{W}$ TO READ
2. SET FUNCTION CODE TO CPU SPACE
3. PLACE CPU SPACE TYPE 0 ON A19–A16
4. PLACE BREAKPOINT NUMBER ON A2–A4
5. CLEAR T-BIT (A1)
6. SET SIZE TO WORD
7. ASSERT $\overline{AS}$ AND $\overline{DS}$
IF $\overline{BKPT}$ PIN ASSERTED:
1. SET R/$\overline{W}$ TO READ
2. SET FUNCTION CODE TO CPU SPACE
3. PLACE CPU SPACE TYPE 0 ON A19–A16
4. PLACE ALL ONE'S ON A4–A2
5. SET T-BIT (A-1) TO ONE
6. SET SIZE TO WORD
7. ASSERT $\overline{AS}$ AND $\overline{DS}$
IF BREAKPOINT INSTRUCTION EXECUTED:
1. PLACE REPLACEMENT OPCODE ON DATA BUS
2. ASSERT $\overline{DSACKx}$
-OR-
1. ASSERT $\overline{BERR}$ TO INITIATE EXCEPTION PROCESSING
IF $\overline{BKPT}$ PIN ASSERTED:
1. ASSERT $\overline{DSACKx}$
-OR-
1. ASSERT $\overline{BERR}$ TO INITIATE EXCEPTION PROCESSING
IF BEAKPOINT INSTRUCTION EXECUTED AND $\overline{DSACKx}$ IS ASSERTED:
1. LATCH DATA
2. NEGATE $\overline{AS}$ AND $\overline{DS}$
3. GO TO (A)
IF $\overline{BKPT}$ PIN ASSERTED AND $\overline{DSACKx}$ IS ASSERTED:
1. NEGATE $\overline{AS}$ AND $\overline{DS}$
2. GO TO (A)
IF $\overline{BERR}$ ASSERTED:
1. NEGATE $\overline{AS}$ AND $\overline{DS}$
2. GO TO (B)
(A)
(B)
1. NEGATE $\overline{DSACKx}$ or $\overline{BERR}$
IF BREAKPOINT INSTRUCTION EXECUTED:
1. PLACE LATCHED DATA IN INSTRUCTION PIPELINE
2. CONTINUE PROCESSING
IF $\overline{BKPT}$ PIN ASSERTED:
1. CONTINUE PROCESSING
IF BREAKPOINT INSTRUCTION EXECUTED:
1. INITIATE ILLEGAL INSTRUCTION PROCESSING
IF $\overline{BKPT}$ PIN ASSERTED:
1. INITIATE HARDWARE BREAKPOINT PROCESSING

Figure 3-11. Breakpoint Operation Flowchart

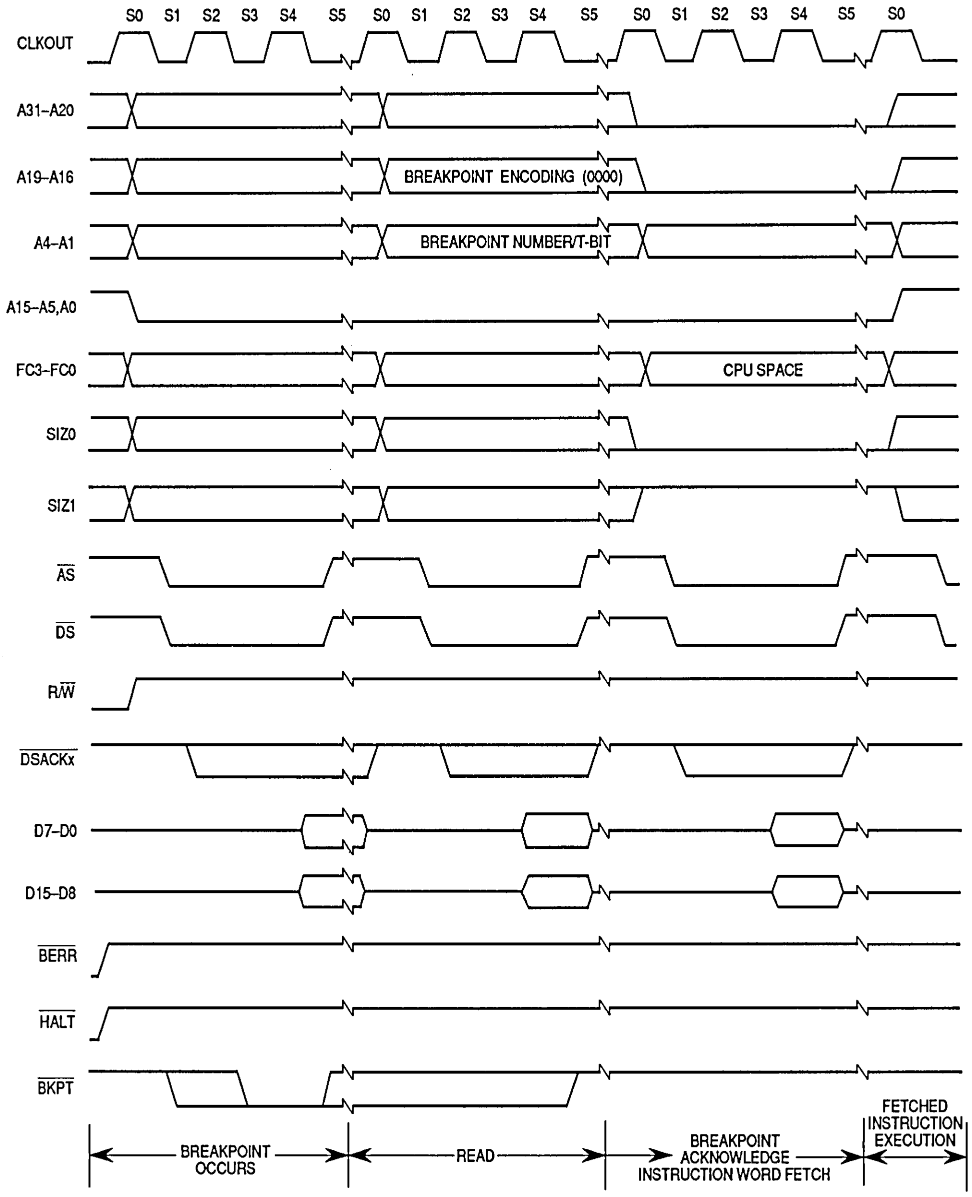


Figure 3-12. Breakpoint Acknowledge Cycle Timing (Opcode Returned)

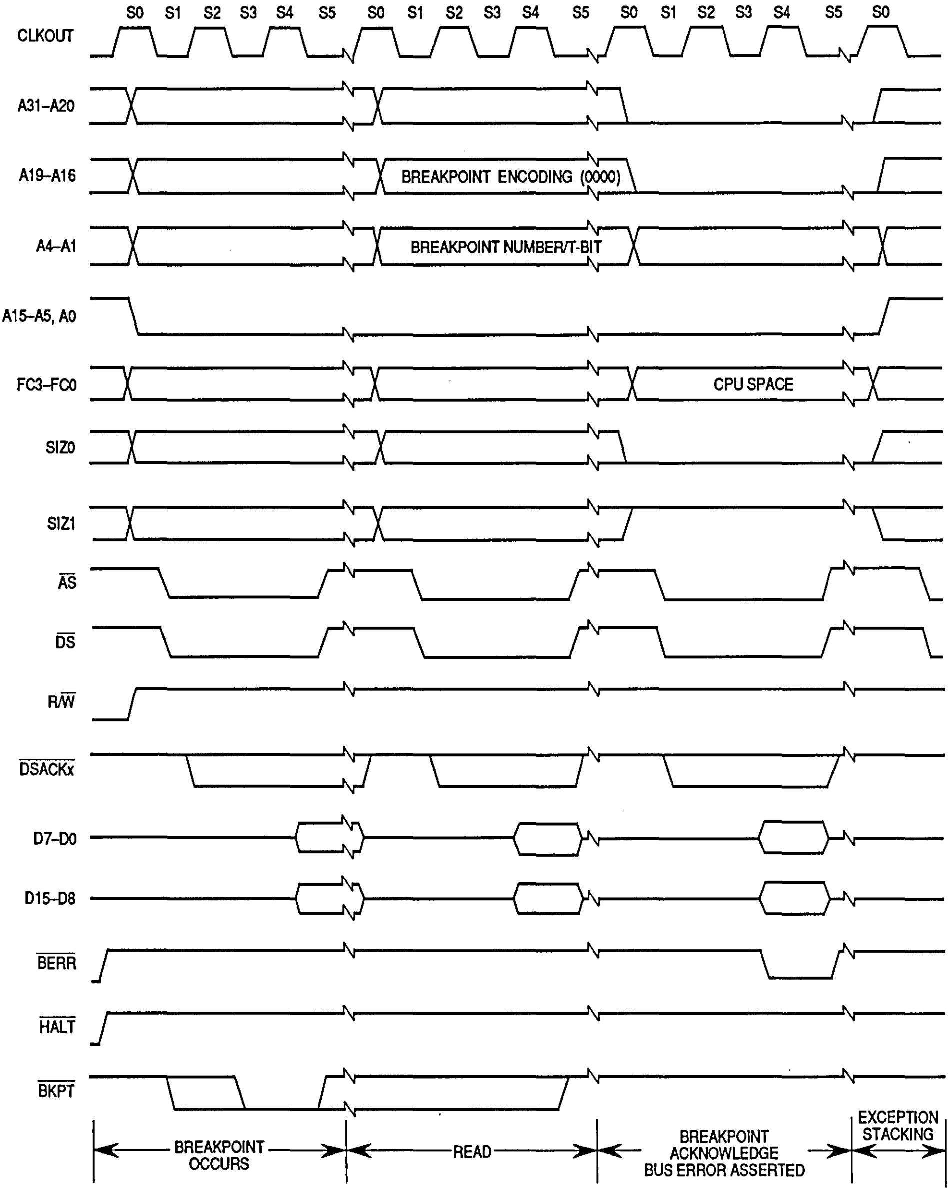


Figure 3-13. Breakpoint Acknowledge Cycle Timing (Exception Signaled)

3.4.3 Module Base Address Register Access

All internal module registers, including the SIM, occupy a single 4K-byte memory block that is relocatable along 4K-byte boundaries. The location is fixed by writing the desired base address of the SIM block to the module base address register using the MOVES instruction. The module base address register is only accessible in CPU32 space at address $03FF00. Refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for additional information on the module base address register.

3.4.4 Interrupt Acknowledge Bus Cycles

The CPU32 makes an interrupt pending in three cases. The first case occurs when a peripheral device signals the CPU32 (with the $\overline{\text{IRQ7}}$–$\overline{\text{IRQ1}}$ signals) that the device requires service and the internally synchronized value on these signals indicates a higher priority than the interrupt mask in the status register. The second case occurs when a transition has occurred in the case of a level 7 interrupt. A recognized level 7 interrupt must be removed for one clock cycle before a second level 7 can be recognized. The third case occurs if, upon return from servicing a level 7 interrupt, the request level stays at 7 and the processor mask level changes from 7 to a lower level, a second level 7 is recognized. The CPU32 takes an interrupt exception for a pending interrupt within one instruction boundary (after processing any other pending exception with a higher priority). The following paragraphs describe the various kinds of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing.

3.4.4.1 INTERRUPT ACKNOWLEDGE CYCLE — TERMINATED NORMALLY. When the CPU32 processes an interrupt exception, it performs an interrupt acknowledge cycle to obtain the number of the vector that contains the starting location of the interrupt service routine. Some interrupting devices have programmable vector registers that contain the interrupt vectors for the routines they use. The following paragraphs describe the interrupt acknowledge cycle for these devices. Other interrupting conditions or devices cannot supply a vector number and use the autovector cycle described in **3.4.4.2 AUTOVECTOR INTERRUPT ACKNOWLEDGE CYCLE.**

The interrupt acknowledge cycle is a read cycle. It differs from the read cycle described in **3.3.1 Read Cycle** in that it accesses the CPU address space. Specifically, the differences are as follows:

1. FC3–FC0 are set to seven (FC3/FC2/FC1/FC0 = 0111) for CPU address space.
2. A3, A2, and A1 are set to the interrupt request level, and the IACKx strobe corresponding to the current interrupt level is asserted. (Either the function codes and address signals or the IACKx strobes can be monitored to determine that an interrupt acknowledge cycle is in progress and the current interrupt level.)
3. The CPU space type field (A19–A16) is set to $F, the interrupt acknowledge code.
4. Other address signals (A31–A20, A15–A4, and A0) are set to one.
5. The SIZ0, SIZ1, and R/W signals are driven to indicate a single-byte read cycle. The responding device places the vector number on the least significant byte of the its data port (for an 8-bit port, the vector number must be on D15–D8; for a 16-bit port, the vector must be on D7–D0) during the interrupt acknowledge cycle. Beyond this, the cycle is terminated normally with DSACKx.

Figure 3-14 is a flowchart of the interrupt acknowledge cycle; Figure 3-15 shows the timing for an interrupt acknowledge cycle terminated with DSACKx.

3.4.4.2 AUTOVECTOR INTERRUPT ACKNOWLEDGE CYCLE. When the interrupting device cannot supply a vector number, it requests an automatically generated vector (autovector). Instead of placing a vector number on the data bus and asserting DSACKx, the device asserts AVEC to terminate the cycle. The DSACKx signals may not be asserted during an interrupt acknowledge cycle terminated by AVEC. The vector number supplied in an autovector operation is derived from the interrupt level of the current interrupt. When AVEC is asserted instead of DSACKx during an interrupt acknowledge cycle, the MC68340 ignores the state of the data bus and internally generates the vector number (the sum of the interrupt level plus 24 ($18)).

AVEC is multiplexed with CS0, controlled by the FIRQ bit in the SIM MCR (refer to **SECTION 4 SYSTEM INTEGRATION MODULE** for additional information). AVEC is only sampled during an interrupt acknowledge cycle. During all other cycles, AVEC is ignored. Additionally, AVEC can be internally generated for external devices by programming the autovector register. Seven distinct autovectors can be used, corresponding to the seven levels of interrupt available with signals IRQ7–IRQ1. Figure 3-16 shows the timing for an autovector operation.

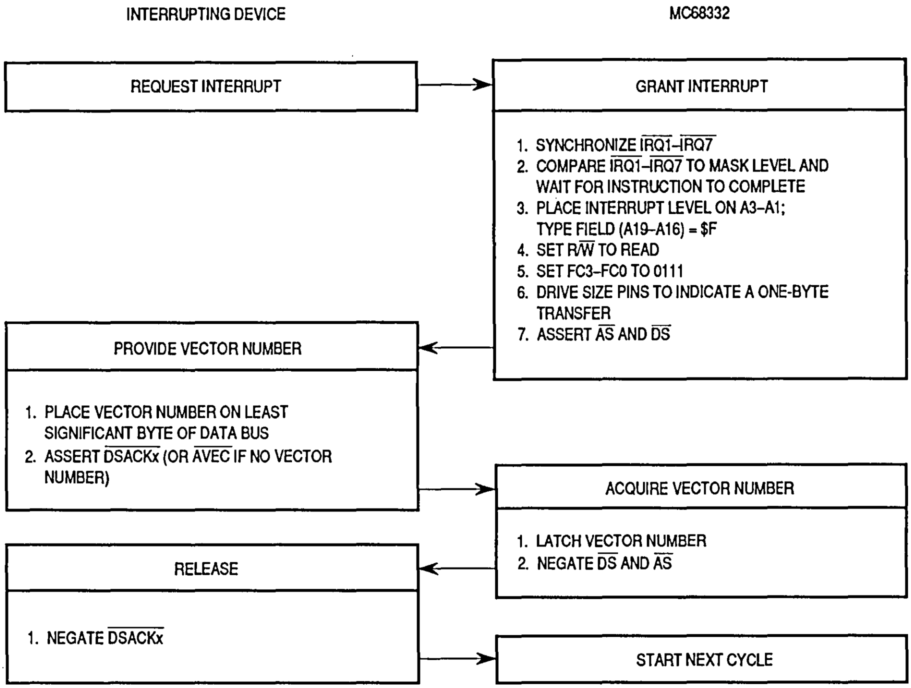

INTERRUPTING DEVICE
MC68332
REQUEST INTERRUPT
GRANT INTERRUPT
1. SYNCHRONIZE $\overline{IRQ1}$–$\overline{IRQ7}$
2. COMPARE $\overline{IRQ1}$–$\overline{IRQ7}$ TO MASK LEVEL AND WAIT FOR INSTRUCTION TO COMPLETE
3. PLACE INTERRUPT LEVEL ON A3–A1; TYPE FIELD (A19–A16) = $F
4. SET R/$\overline{W}$ TO READ
5. SET FC3–FC0 TO 0111
6. DRIVE SIZE PINS TO INDICATE A ONE-BYTE TRANSFER
7. ASSERT $\overline{AS}$ AND $\overline{DS}$
PROVIDE VECTOR NUMBER
1. PLACE VECTOR NUMBER ON LEAST SIGNIFICANT BYTE OF DATA BUS
2. ASSERT $\overline{DSACKx}$ (OR $\overline{AVEC}$ IF NO VECTOR NUMBER)
ACQUIRE VECTOR NUMBER
1. LATCH VECTOR NUMBER
2. NEGATE $\overline{DS}$ AND $\overline{AS}$
RELEASE
1. NEGATE $\overline{DSACKx}$
START NEXT CYCLE

Figure 3-14. Interrupt Acknowledge Cycle Flowchart

3.4.4.3 SPURIOUS INTERRUPT CYCLE. Requested interrupts, whether internal or external, are arbitrated internally. When no internal module (including the SIM, which responds for external requests) responds during an interrupt acknowledge cycle by arbitrating for the interrupt acknowledge cycle internally, the spurious interrupt monitor generates an internal bus error signal to terminate the vector acquisition. The MC68340 automatically generates the spurious interrupt vector number, 24, instead of the interrupt vector number in this case. When an external device does not respond to an interrupt acknowledge cycle with $\overline{\text{AVEC}}$ or $\overline{\text{DSACKx}}$, a bus monitor must assert $\overline{\text{BERR}}$, which results in the CPU32 taking the spurious interrupt vector. If $\overline{\text{HALT}}$ is also asserted, the MC68340 retries the interrupt acknowledge cycle instead of using the spurious interrupt vector.

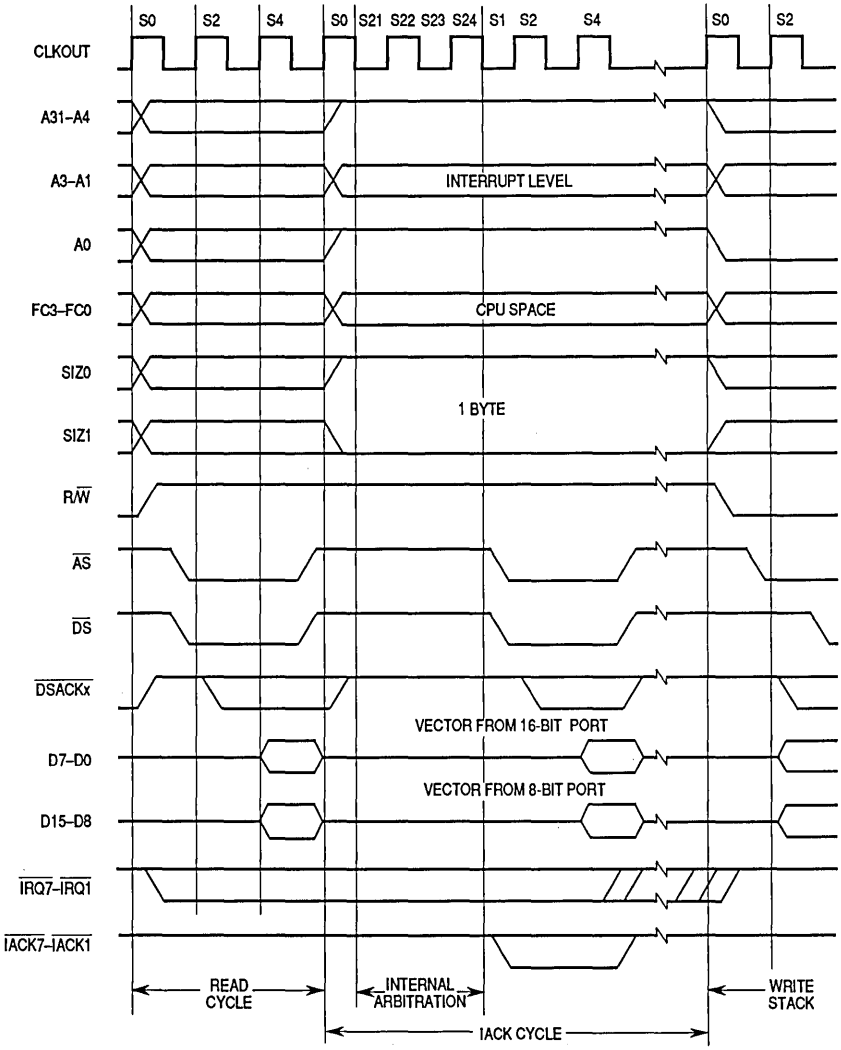


Figure 3-15. Interrupt Acknowledge Cycle Timing

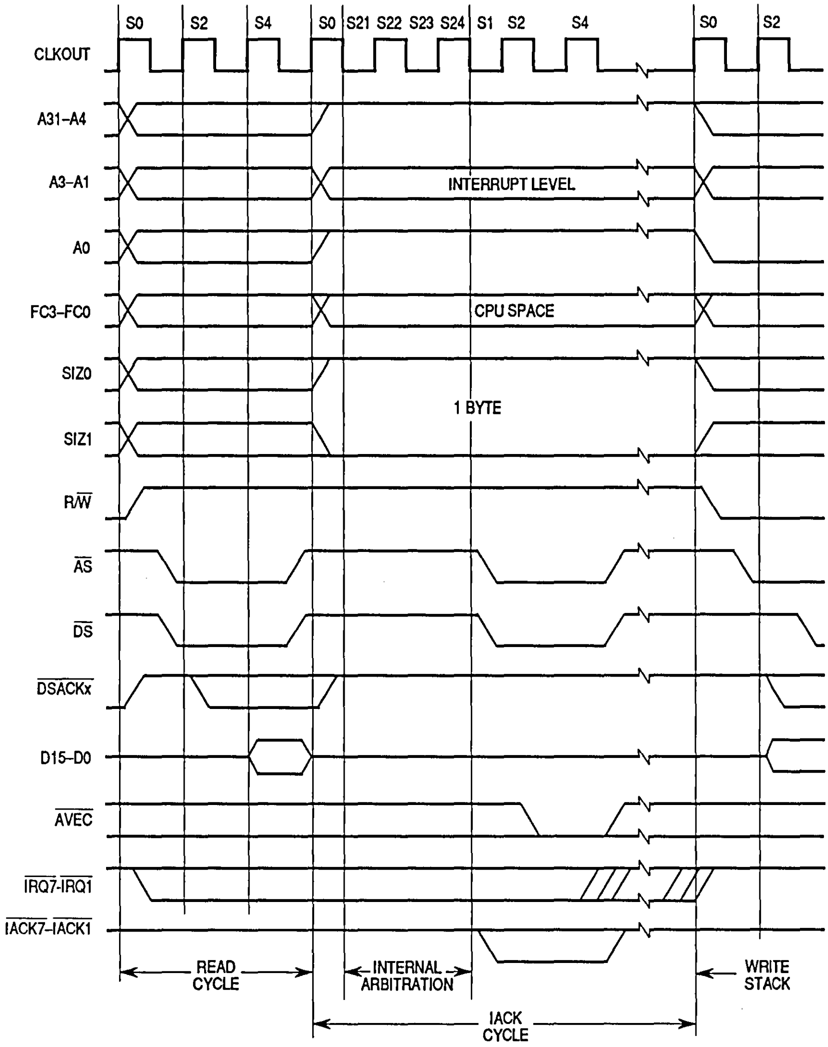


Figure 3-16. Autovector Operation Timing

3.5 BUS EXCEPTION CONTROL CYCLES

The bus architecture requires assertion of DSACKx from an external device to signal that a bus cycle is complete. Neither DSACKx nor AVEC is not asserted in the following cases:

- DSACKx/AVEC is programmed to respond internally.
- The external device does not respond.
- Various other application-dependent errors occur.

The MC68340 provides BERR when no device responds by asserting DSACKx/AVEC within an appropriate period of time after the MC68340 asserts AS. This mechanism allows the cycle to terminate and the MC68340 to enter exception processing for the error condition. HALT is also used for bus exception control. This signal can be asserted by an external device for debugging purposes to cause single bus cycle operation or, in combination with BERR, a retry of a bus cycle in error. To properly control termination of a bus cycle for a retry or a bus error condition, DSACKx, BERR, and HALT can be asserted and negated with the rising edge of the MC68340 clock. This precaution assures that when two signals are asserted simultaneously, the required setup and hold time for both is met for the same falling edge of the MC68340 clock. This or an equivalent precaution should be designed into the external circuitry to provide these signals. Alternatively, the internal bus monitor could be used. The acceptable bus cycle terminations for asynchronous cycles are summarized in relation to DSACKx assertion as follows (case numbers refer to Table 3-4):

Normal Termination — DSACKx is asserted; BERR and HALT remain negated (case 1).

Halt Termination — HALT is asserted at the same time or before DSACKx, and BERR remains negated (case 2).

Bus Error Termination — BERR is asserted in lieu of, at the same time, or before DSACKx (case 3) or after DSACKx (case 4), and HALT remains negated; BERR is negated at the same time or after DSACKx.

Retry Termination — HALT and BERR are asserted in lieu of, at the same time, or before DSACKx (case 5) or after DSACKx (case 6); BERR is negated at the same time or after DSACKx, and HALT may be negated at the same time or after BERR.

Table 3-4 shows various combinations of control signal sequences and the resulting bus cycle terminations. To ensure predictable operation, BERR and HALT should be negated according to the specifications in the MC68340/D, *MC68340 Technical Summary*. DSACKx, BERR, and HALT may be negated after

AS. If DSACKx or BERR remain asserted into S2 of the next bus cycle, that cycle may be terminated prematurely.

Table 3-4. DSACKx, BERR, and HALT Assertion Results

Case Num.	Control Signal	Asserted on Rising Edge of State		Result
		N	N+2	
1	DSACKx BERR HALT	A NA NA	S NA X	Normal cycle terminate and continue.
2	DSACKx BERR HALT	A NA A/S	S NA S	Normal cycle terminate and halt; continue when HALT negated
3	DSACKx BERR HALT	NA/A A NA	X S X	Terminate and take bus error exception, possibly deferred.
4	DSACKx BERR HALT	A NA NA	X A NA	Terminate and take bus error exception, possibly deferred.
5	DSACKx BERR HALT	NA/A A A/S	X S S	Terminate and retry when HALT negated.
6	DSACKx BERR HALT	A NA NA	X A A	Terminate and retry when HALT negated.

NOTE:
N—The number of current even bus state (e.g., S2, S4, etc.)
A—Signal is asserted in this bus state
NA—Signal is not asserted in this state
X—Don't care
S—Signal was asserted in previous state and remains asserted in this state

EXAMPLE A: A system uses a bus monitor timer to terminate accesses to an unpopulated address space. The timer asserts BERR after timeout (case 3).

EXAMPLE B: A system uses error detection and correction on RAM contents. The designer may:

1. Delay DSACKx until data is verified and assert BERR and HALT simultaneously to indicate to the MC68340 to automatically retry the error cycle (case 5) or, if data is valid, assert DSACKx (case 1).
2. Delay DSACKx until data is verified and assert BERR with or without DSACKx if data is in error (case 3). This initiates exception processing for software handling of the condition.
3. Return DSACKx prior to data verification; if data is invalid, BERR is asserted on the next clock cycle (case 4). This initiates exception processing for software handling of the condition.

4. Return DSACKx prior to data verification; if data is invalid, assert BERR and HALT on the next clock cycle (case 6). The memory controller can then correct the RAM prior to or during the automatic retry.

3.5.1 Bus Errors

BERR can be used to abort the bus cycle and the instruction being executed. BERR takes precedence over DSACKx provided it meets the timing constraints described in MC68340/D, *MC68340 Technical Summary.* If BERR does not meet these constraints, it may cause unpredictable operation of the MC68340. If BERR remains asserted into the next bus cycle, it may cause incorrect operation of that cycle. When BERR is issued to terminate a bus cycle, the MC68340 may enter exception processing immediately following the bus cycle, or it may defer processing the exception.

The instruction prefetch mechanism requests instruction words from the bus controller before it is ready to execute them. If a bus error occurs on an instruction fetch, the MC68340 does not take the exception until it attempts to use that instruction word. Should an intervening instruction cause a branch or should a task switch occur, the bus error exception does not occur. The bus error condition is recognized during a bus cycle in any of the following cases:

- DSACKx and HALT are negated, and BERR is asserted.
- HALT and BERR are negated, and DSACKx is asserted. BERR is then asserted within one clock cycle (HALT remains negated).
- BERR and HALT are asserted together, indicating a retry.

When the MC68340 recognizes a bus error condition, it terminates the current bus cycle in the normal way. Figure 3-17 shows the timing of a bus error for the case in which DSACKx is not asserted. Figure 3-18 shows the timing for a bus error that is asserted after DSACKx. Exceptions are taken in both cases. (Refer to **SECTION 5 CPU32** for details of bus error exception processing.)

In the second case, in which BERR is asserted after DSACKx is asserted, BERR must be asserted within the time specified for purely asynchronous operation, or it must be asserted and remain stable during the sample window around the next falling edge of the clock after DSACKx is recognized. If BERR is not stable at this time, the MC68340 may exhibit erratic behavior. BERR has priority over DSACKx. In this case, data may be present on the bus but may not be valid. This sequence can be used by systems that have memory error detection and correction logic and by external cache memories.

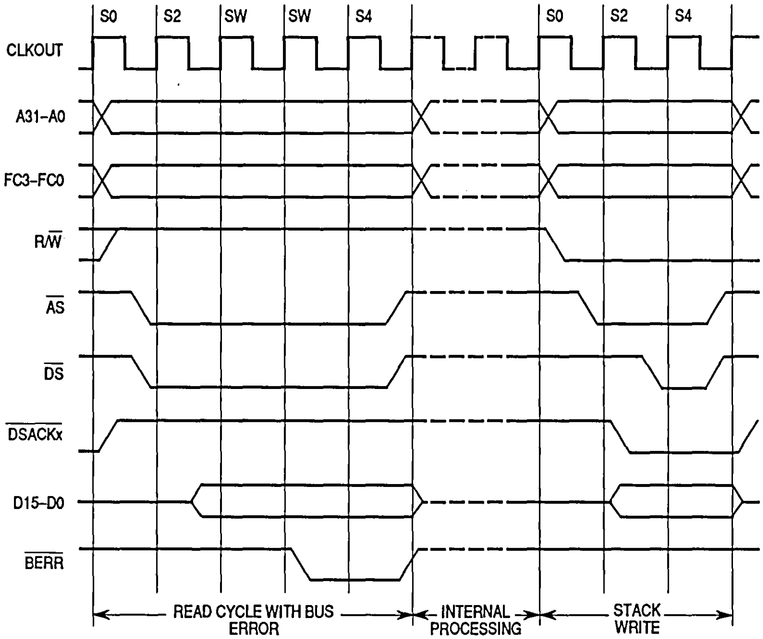


Figure 3-17. Bus Error without DSACKx

3.5.2 Retry Operation

When both BERR and HALT are asserted by an external device during a bus cycle, the MC68340 enters the retry sequence shown in Figure 3-19. A delayed retry, which is similar to the delayed bus error signal described previously, can also occur (see Figure 3-20). The MC68340 terminates the bus cycle, places the control signals in their inactive state, and does not begin another bus cycle until the BERR and HALT signals are negated by external logic. After a synchronization delay, the MC68340 retries the previous cycle using the same access information (address, function code, size, etc.). BERR should be negated before S2 of the retried cycle to ensure correct operation of the retried cycle.

The MC68340 retries any read or write cycle of a read-modify-write operation separately; RMC remains asserted during the entire retry sequence. Asserting BR along with BERR and HALT provides a relinquish and retry operation. The MC68340 does not relinquish the bus during a read-modify-write operation. Any device that requires the MC68340 to give up the bus and retry a bus cycle during a read-modify-write cycle must assert BERR and BR only (HALT must

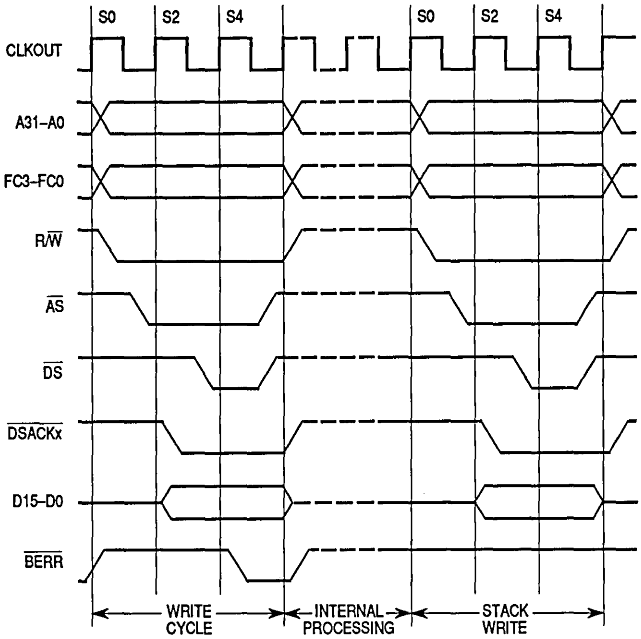


Figure 3-18. Late Bus Error with DSACKx

not be included). The bus error handler software should examine the read-modify-write bit in the special status word (refer to **SECTION 5 CPU32**) and take the appropriate action to resolve this type of fault when it occurs.

3.5.3 Halt Operation

When HALT is asserted and BERR is not asserted, the MC68340 halts external bus activity at the next bus cycle boundary (see Figure 3-21). HALT by itself does not terminate a bus cycle. Negating and reasserting HALT in accordance with the correct timing requirements provides a single-step (bus cycle to bus cycle) operation. HALT affects external bus cycles only; thus, a program that does not require use of the external bus may continue executing. The single-cycle mode allows the user to proceed through (and debug) external MC68340 operations, one bus cycle at a time. Since the occurrence of a bus error while HALT is asserted causes a retry operation, the user must anticipate retry cycles while debugging in the single-cycle mode. The single-step operation and the software trace capability allow the system debugger to trace single bus cycles, single instructions, or changes in program flow.

When the MC68340 completes a bus cycle with HALT asserted, D15–D0 is placed in the high-impedance state, and bus control signals are driven inactive (not high-impedance state); the address, function code, size, and read/write signals remain in the same state. The halt operation has no effect on bus arbitration (refer to **3.6 BUS ARBITRATION**). When bus arbitration occurs while the MC68340 is halted, the address and control signals are also placed in the high-impedance state. Once bus mastership is returned to the MC68340, if HALT is still asserted, the address, function code, size, and read/write signals are again driven to their previous states. The MC68340 does not service interrupt requests while it is halted.

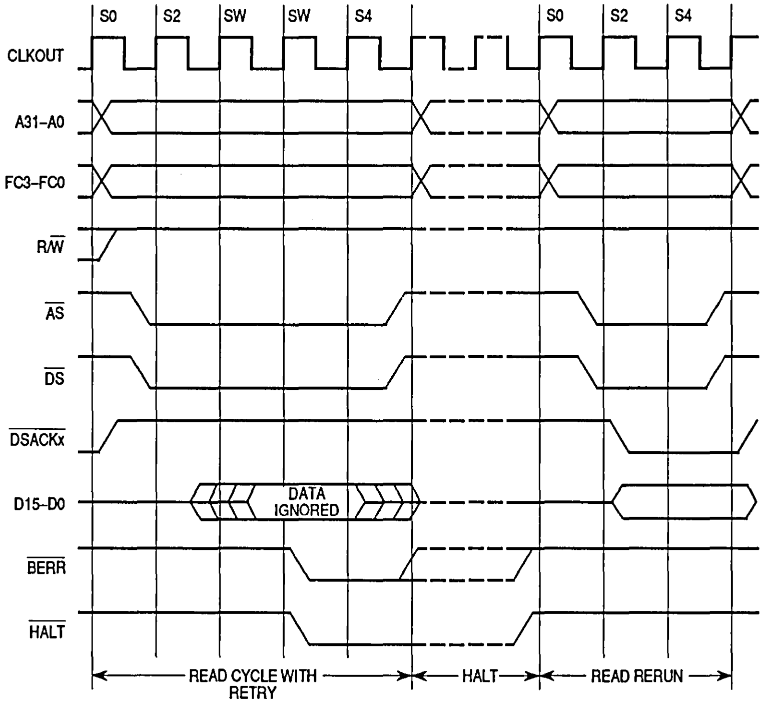


Figure 3-19. Retry Sequence

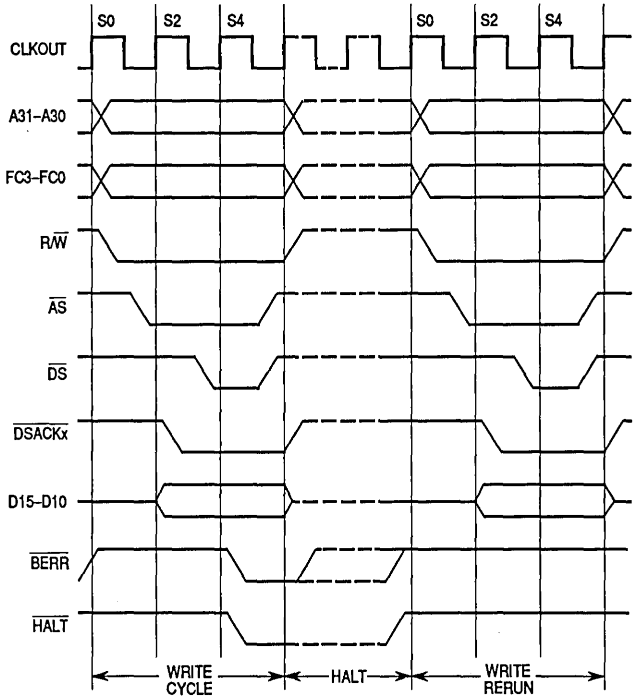


Figure 3-20. Late Retry Sequence

3.5.4 Double Bus Fault

A double bus fault results when a bus error or an address error occurs during the exception processing sequence for any of the following:

- A previous bus error
- A previous address error
- A reset

For example, the MC68340 attempts to stack several words containing information about the state of the machine while processing a bus error exception. If a bus error exception occurs during the stacking operation, the second error is considered a double bus fault. When a double bus fault occurs, the MC68340 halts and drives the HALT line low. Only a reset operation can restart a halted MC68340. However, bus arbitration can still occur (refer to **3.6 BUS ARBITRATION**). A second bus error or address error that occurs after exception processing has completed (during the execution of the exception handler routine, or later) does not cause a double bus fault. A bus cycle that is retried does not constitute a bus error or contribute to a double bus fault. The MC68340 continues to retry the same bus cycle as long as the external hardware requests it.

Reset can also be generated internally by the double bus fault monitor (see **SECTION 5 CPU32**).

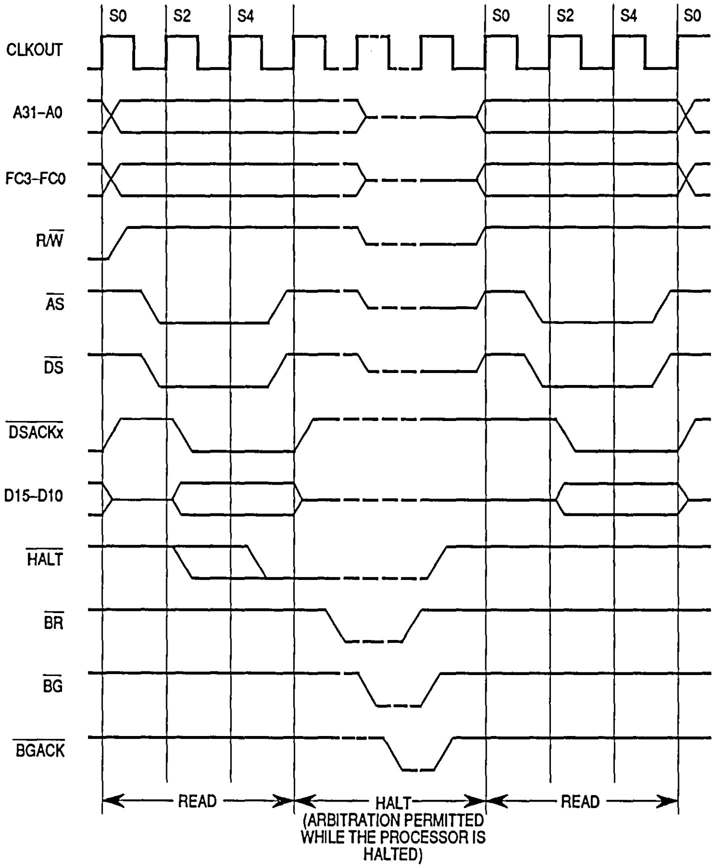


Figure 3-21. HALT Timing

3.6 BUS ARBITRATION

The bus design of the MC68340 provides for a single bus master at any one time, either the MC68340 or an external device. One or more of the external devices on the bus can have the capability of becoming bus master for the external bus, but not the MC68340 internal bus. Bus arbitration is the protocol by which an external device becomes bus master; the bus controller in the MC68340 manages the bus arbitration signals so that the MC68340 has the lowest priority. External devices that need to obtain the bus must assert the bus arbitration signals in the sequences described in the following paragraphs. Systems that include several devices that can become bus master require external circuitry to assign priorities to the devices, so that, when two or more external devices attempt to become bus master at the same time, the one having the highest priority becomes bus master first. The sequence of the protocol is as follows:

1. An external device asserts $\overline{\text{BR}}$.
2. The MC68340 asserts $\overline{\text{BG}}$ to indicate that the bus is available.
3. The external device asserts $\overline{\text{BGACK}}$ to indicate that it has assumed bus mastership.

$\overline{\text{BR}}$ may be issued any time during a bus cycle or between cycles. $\overline{\text{BG}}$ is asserted in response to $\overline{\text{BR}}$. To guarantee operand coherency, $\overline{\text{BG}}$ is only asserted at the end of an operand transfer. Additionally, $\overline{\text{BG}}$ is not asserted until the end of a read-modify-write operation (when $\overline{\text{RMC}}$ is negated) in response to a $\overline{\text{BR}}$ signal. When the requesting device receives $\overline{\text{BG}}$ and more than one external device can be bus master, the requesting device should begin whatever arbitration is required. When it assumes bus mastership, the external device asserts $\overline{\text{BGACK}}$ and maintains $\overline{\text{BGACK}}$ during the entire bus cycle (or cycles) for which it is bus master. The following conditions must be met for an external device to assume mastership of the bus through the normal bus arbitration procedure: 1) it must have received $\overline{\text{BG}}$ through the arbitration process, and 2) $\overline{\text{BGACK}}$ must be inactive, indicating that no other bus master has claimed ownership of the bus.

Figure 3-22 is a flowchart showing the detail involved in bus arbitration for a single device. This technique allows processing of bus requests during data transfer cycles.

$\overline{\text{BR}}$ is negated at the time that $\overline{\text{BGACK}}$ is asserted. This type of operation applies to a system consisting of the MC68340 and one device capable of bus mastership. In a system having a number of devices capable of bus mastership, the $\overline{\text{BR}}$ from each device can be wire-ORed to the MC68340. In such a system, more than one bus request could be asserted simultaneously. $\overline{\text{BG}}$ is negated a few clock cycles after the transition of $\overline{\text{BGACK}}$. However, if bus requests are

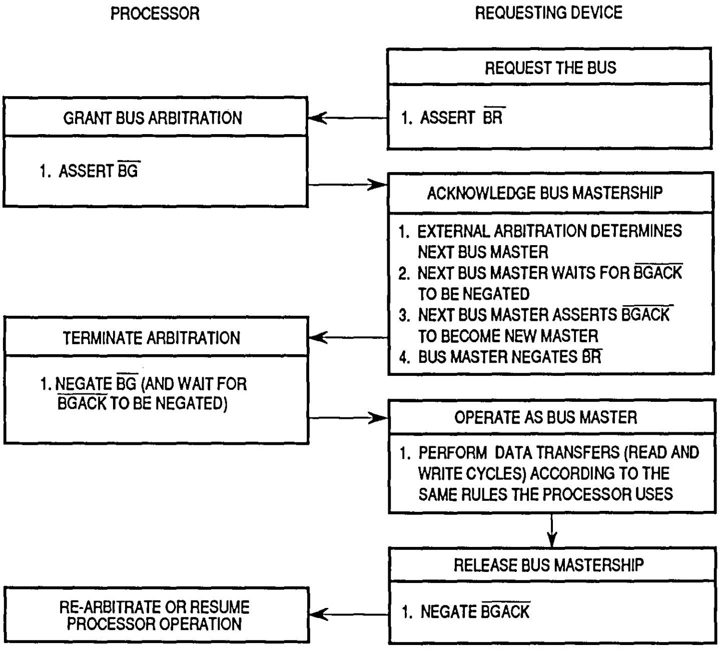

PROCESSOR
REQUESTING DEVICE
REQUEST THE BUS
1. ASSERT BR
GRANT BUS ARBITRATION
1. ASSERT BG
ACKNOWLEDGE BUS MASTERSHIP
1. EXTERNAL ARBITRATION DETERMINES NEXT BUS MASTER
2. NEXT BUS MASTER WAITS FOR BGACK TO BE NEGATED
3. NEXT BUS MASTER ASSERTS BGACK TO BECOME NEW MASTER
4. BUS MASTER NEGATES BR
TERMINATE ARBITRATION
1. NEGATE BG (AND WAIT FOR BGACK TO BE NEGATED)
OPERATE AS BUS MASTER
1. PERFORM DATA TRANSFERS (READ AND WRITE CYCLES) ACCORDING TO THE SAME RULES THE PROCESSOR USES
RELEASE BUS MASTERSHIP
1. NEGATE BGACK
RE-ARBITRATE OR RESUME PROCESSOR OPERATION

Figure 3-22. Bus Arbitration Flowchart for Single Request

still pending after the negation of $\overline{\text{BG}}$, the MC68340 asserts another $\overline{\text{BG}}$ within a few clock cycles after it was negated. This additional assertion of $\overline{\text{BG}}$ allows external arbitration circuitry to select the next bus master before the current bus master has finished using the bus. The following paragraphs provide additional information about the three steps in the arbitration process. Bus arbitration requests are recognized during normal processing, $\overline{\text{HALT}}$ assertion, and when the CPU32 has halted due to a double bus fault.

3.6.1 Bus Request

External devices capable of becoming bus masters request the bus by asserting $\overline{\text{BR}}$. This signal can be wire-ORed to indicate to the MC68340 that some external device requires control of the bus. The MC68340 is effectively at a lower bus priority level than the external device and relinquishes the bus after it has completed the current bus cycle (if one has started). If no BGACK is received while the $\overline{\text{BR}}$ is active, the MC68340 remains bus master once $\overline{\text{BR}}$ is negated. This prevents unnecessary interference with ordinary processing if the arbitration circuitry inadvertently responds to noise or if an external device determines that it no longer requires use of the bus before it has been granted mastership.

3.6.2 Bus Grant

This MC68340 supports operand coherency; thus, if an operand transfer requires multiple bus cycles, the MC68340 does not release the bus until the entire transfer is complete. The assertion of $\overline{\text{BG}}$ is therefore subject to the following constraints:

- The minimum time for $\overline{\text{BG}}$ assertion after $\overline{\text{BR}}$ is asserted depends on internal synchronization (see MC68340/D, *MC68340 Technical Summary*).
- During an external operand transfer, the MC68340 does not assert $\overline{\text{BG}}$ until after the last cycle of the transfer (determined by SIZx and $\overline{\text{DSACKx}}$).
- During an external operand transfer, the MC68340 does not assert $\overline{\text{BG}}$ as long as $\overline{\text{RMC}}$ is asserted.
- If both show cycle bits are asserted and the CPU32 is making internal accesses, the MC68340 does not assert $\overline{\text{BG}}$ until the CPU32 finishes the internal transfers. Otherwise, the external bus is granted away, and the CPU32 continues to execute internal bus transfers.
- If the show cycle bits are 10, the MC68340 does not assert $\overline{\text{BG}}$ to an external master.

Externally, the $\overline{\text{BG}}$ signal can be routed through a daisy-chained network or a priority-encoded network. The MC68340 is not affected by the method of arbitration as long as the protocol is obeyed.

3.6.3 Bus Grant Acknowledge

An external device cannot request and be granted the external bus while another device is the active bus master. A device that asserts $\overline{\text{BGACK}}$ remains the bus master until it negates $\overline{\text{BGACK}}$. $\overline{\text{BGACK}}$ should not be negated until all required bus cycles are completed. Bus mastership is terminated at the negation of $\overline{\text{BGACK}}$.

Once an external device receives the bus and asserts $\overline{\text{BGACK}}$, it should negate $\overline{\text{BR}}$. If $\overline{\text{BR}}$ remains asserted after $\overline{\text{BGACK}}$ is asserted, the MC68340 assumes that another device is requesting the bus and prepares to issue another $\overline{\text{BG}}$.

3.6.4 Bus Arbitration Control

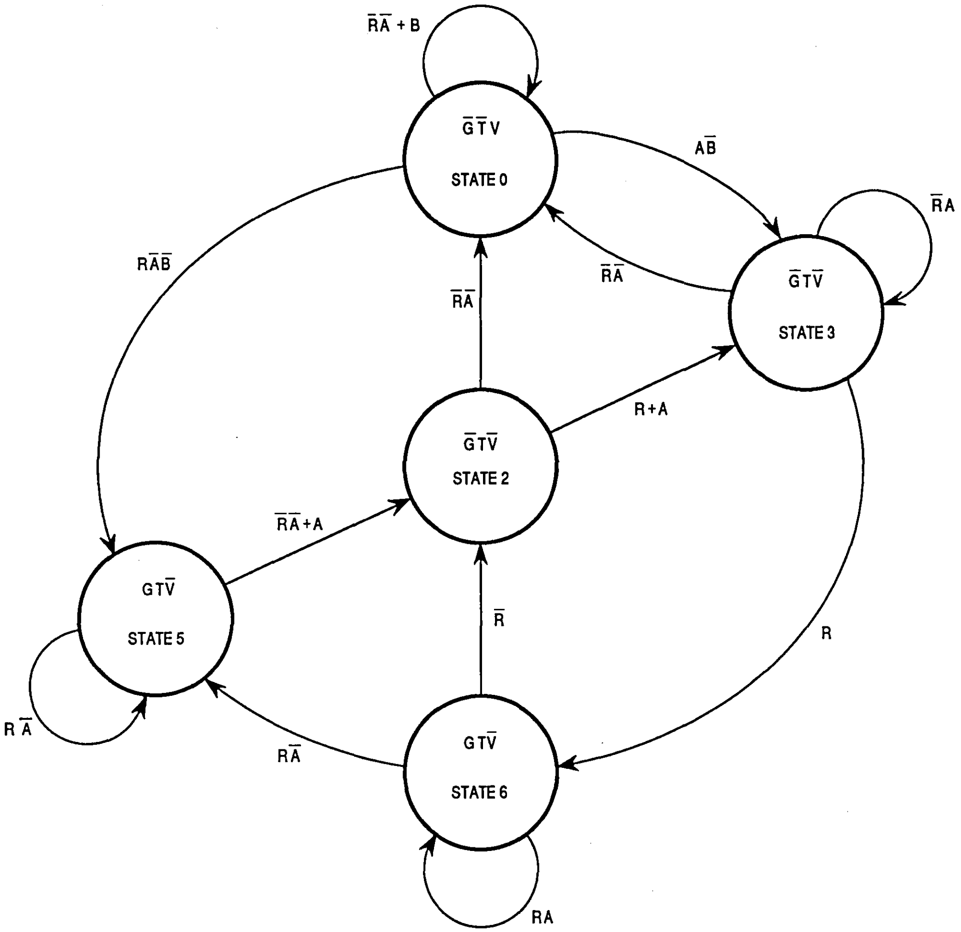
The bus arbitration control unit in the MC68340 is implemented with a finite state machine. As discussed previously, all asynchronous inputs to the MC68340 are internally synchronized in a maximum of two cycles of the clock. As shown in Figure 3-23, input signals labeled R and A are internally synchronized versions of BR and BGACK, respectively. The BG output is labeled G, and the internal high-impedance control signal is labeled T. If T is true, the address, data, and control buses are placed in the high-impedance state after the next rising edge following the negation of AS and RMC. All signals are shown in positive logic (active high) regardless of their true active voltage level. The state machine shown in Figure 3-23 does not have a state 1 or state 4.

State changes occur on the next rising edge of the clock after the internal signal is valid. The BG signal transitions on the falling edge of the clock after a state is reached during which G changes. The bus control signals (controlled by T) are driven by the MC68340 immediately following a state change, when bus mastership is returned to the MC68340. State 0, in which G and T are both negated, is the state of the bus arbiter while the MC68340 is bus master. R and A keep the arbiter in state 0 as long as they are negated. The MC68340 does not allow arbitration of the external bus during the RMC sequence. For the duration of this sequence, the MC68340 ignores the BR input. If mastership of the bus is required during an RMC operation, BERR must be used to abort the RMC sequence.

3.6.5 Show Cycles

The MC68340 can perform data transfers with its internal modules without using the external bus, but, when debugging, it is desirable to have address and data information appear on the external bus. These external bus cycles, called show cycles, are distinguished by the fact that AS is not asserted externally.

After reset, show cycles are disabled and must be enabled by writing to the SHEN bits in the module configuration register. When show cycles are disabled, the address bus, function codes, size, and read/write signals continue to reflect internal bus activity. The only differences are that AS and DS are not asserted externally and the external data bus remains in a high-impedance state. The following paragraphs are a state-by-state description of show cycles. Refer to MC68340/D, *MC68340 Technical Summary* for specific timing information.


$\overline{R}\,\overline{A}$ + B
$\overline{G}\,\overline{T}$ V
STATE 0
A$\overline{B}$
$\overline{R}$A
$\overline{G}$ T $\overline{V}$
STATE 3
R$\overline{A}\,\overline{B}$
$\overline{R}\,\overline{A}$
$\overline{R}\,\overline{A}$
$\overline{G}$ T $\overline{V}$
STATE 2
R+A
$\overline{R}\,\overline{A}$+A
G T $\overline{V}$
STATE 5
$\overline{R}$
R
R $\overline{A}$
R$\overline{A}$
G T $\overline{V}$
STATE 6
RA

R - BUS REQUEST
A - BUS GRANT ACKNOWLEDGE
B - BUS CYCLE IN PROGRESS

G - BUS GRANT
T - THREE-STATE SIGNAL TO BUS CONTROL
V - BUS AVAILABLE TO BUS CONTROL

Figure 3-23. Bus Arbitration State Diagram

State 0 — During state 0, the address and function codes become valid, R/$\overline{\text{W}}$ is driven to indicate a show read or write cycle, and the size pins indicate the number of bytes to transfer. During a read, the addressed peripheral is driving the data bus, and the user must take care to avoid bus conflicts.

State 41 — One-half clock cycle later, $\overline{\text{DS}}$ (rather than $\overline{\text{AS}}$) is asserted to indicate that address information is valid.

State 42 — No action occurs in state 2. The bus controller remains in state 2 until the internal read cycle is complete.

State 43 — $\overline{\text{DS}}$ is negated to indicate that show data is valid on the next falling edge of system clock. The external data bus drivers are enabled so that data becomes valid on the external bus as soon as it is available on the internal bus.

State 0 — The address, function codes, read/write, and size pins change to begin the next cycle. Data from the preceding cycle is valid through state 0.

3.7 RESET OPERATION

The MC68340 has reset control logic to determine the cause of reset, synchronize it if necessary, and assert the appropriate reset lines. The reset control logic can independently drive three different lines:

1. EXTRST (external reset) drives the external $\overline{\text{RESET}}$ pin.
2. CLKRST (clock reset) resets the clock module.
3. INTRST (internal reset) goes to all other internal circuits.

Table 3-5 summarizes the result of each reset source. Synchronous reset sources are not asserted until the end of the current bus cycle, whether or not $\overline{\text{RMC}}$ is asserted. The internal bus monitor is automatically enabled for synchronous resets; therefore, if the current bus cycle does not terminate normally, the bus monitor terminates it. Only single-byte or word transfers are guaranteed valid for synchronous resets. Asynchronous reset sources indicate a catastrophic failure, and the reset control logic immediately resets the system.

If an external device drives $\overline{\text{RESET}}$ low, the reset control logic holds reset asserted internally until the external $\overline{\text{RESET}}$ is released. When the reset control logic detects that external $\overline{\text{RESET}}$ is no longer being driven, it drives both internal and external reset low for an additional 512 cycles to guarantee this length of reset to the entire system.

If reset is asserted from any other source, the reset control logic asserts $\overline{\text{RESET}}$ for a minimum of 512 cycles until the source of reset is negated.

Table 3-5. Reset Source Summary

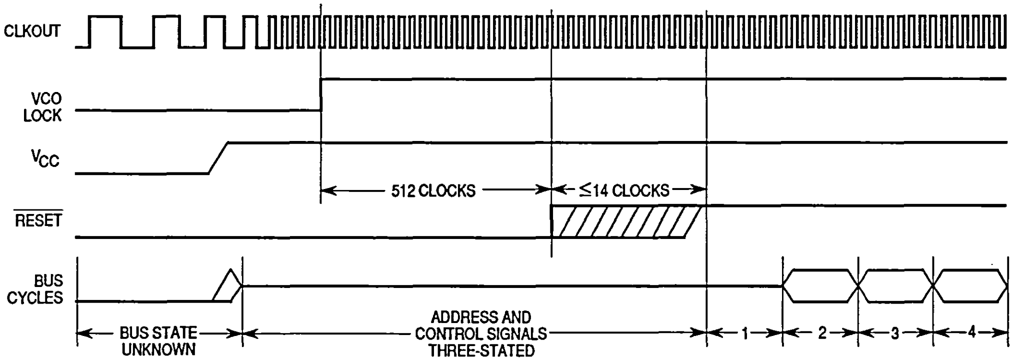
Type	Source	Timing	Reset Lines Asserted by Controller		
External	External	Synchronous	INTRST	CLKRST	EXTRST
Power-up	EBI	Asynchronous	INTRST	CLKRST	EXTRST
Software Watchdog	Sys Prot	Asynchronous	INTRST	CLKRST	EXTRST
Double Bus Fault	Sys Prot	Asynchronous	INTRST	CLKRST	EXTRST
Loss of Clock	Clock	Synchronous	INTRST	CLKRST	EXTRST
RESET Instruction	CPU32	Asynchronous	—	—	EXTRST

After any internal reset occurs, a 14-cycle rise time is allowed before testing for the presence of an external reset. If no external reset is detected, the CPU32 begins its vector fetch.

Figure 3-24 is a timing diagram of the power-up reset operation, showing the relationships between $\overline{\text{RESET}}$, V_{CC}, and bus signals. During the reset period, the entire bus three-states (except for non-three-stateble signals, which are driven to their inactive state). Once $\overline{\text{RESET}}$ negates, all control signals are driven to their inactive state, the data bus is in read mode, and the address bus is driven. After this, the first bus cycle for $\overline{\text{RESET}}$ exception processing begins.

$\overline{\text{RESET}}$ should be asserted for at least 590 clock periods to ensure that the MC68340 resets. Resetting the MC68340 causes any bus cycle in progress to terminate as if $\overline{\text{DSACKx}}$ or $\overline{\text{BERR}}$ had been asserted. In addition, the MC68340 appropriately initializes registers for a reset exception.

When a reset instruction is executed, the MC68340 drives the $\overline{\text{RESET}}$ signal for 512 clock cycles. In this case, the MC68340 resets the external devices of the system, and the internal registers of the MC68340 are unaffected. The external devices connected to the $\overline{\text{RESET}}$ signal are reset at the completion of the $\overline{\text{RESET}}$ instruction.



NOTES:

1. Internal start-up time.
2. SSP read here.
3. PC read here.
4. First instruction fetched here.

Figure 3-24. Initial Reset Operation Timing

SECTION 4
SYSTEM INTEGRATION MODULE

The MC68340 system integration module (SIM) consists of five submodules; four of these submodules control the system startup, initialization, configuration, and the external bus with a minimum of external devices. A fifth submodule provides IEEE 1149.1 boundary scan capabilities. The five submodules, shown in Figure 4-1, are as follows:

- System Configuration and Protection
- Clock Synthesizer
- Chip Selects
- External Bus Interface
- IEEE 1149.1 Test Access Port

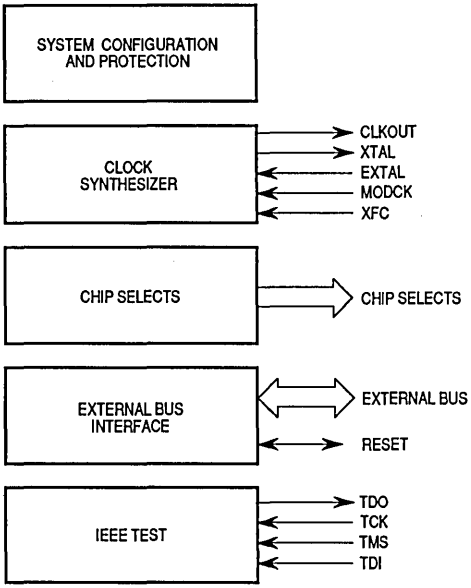


Figure 4-1. SIM Block Diagram

4.1 MODULE OVERVIEW

The system configuration and protection submodule controls system configuration and provides bus monitors and a software watchdog for system protection.

The clock synthesizer generates the clock signals used by the SIM as well as other modules and external devices.

The programmable chip-select submodule provides four chip-select signals. Each chip-select signal has an associated base address register and an address mask register that contain the programmable characteristics of that chip select.

The external bus interface (EBI) handles the transfer of information between the internal CPU32 and memory, peripherals, or other processing elements in the external address space. See **SECTION 3 BUS OPERATION** for further information.

The MC68340 includes dedicated user-accessible test logic that is fully compliant with the IEEE 1149.1 *Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to development of this standard under the sponsorship of the IEEE Test Technology Committee and Joint Test Action Group (JTAG). The MC68340 implementation supports circuit-board test strategies based on this standard. Refer to **SECTION 9 IEEE 1149.1 TEST ACCESS PORT** for additional information.

4.2 MODULE OPERATION

The following paragraphs describe the operation of the module base address register, the system configuration and protection, clock synthesizer, and chip-select submodules, and the external bus interface.

4.2.1 Module Base Address Register

The module base address register controls the location of all module registers (see **4.3.1 Module Base Address Register**). All internal module registers occupy a single 4K-byte memory block that is relocatable along 4K-byte boundaries. The location is fixed by writing the desired base address of the 4K-byte memory block to the module base address register using the MOVES instruction. In this manual, the offset from the base address is shown above the register diagram. The module base address register is the only exception since it resides at a fixed location in CPU space.

The location of the module registers, fixed within the relocatable 4K-byte memory block, is listed in Table 4-1.

Table 4-1. Location of Modules

Module	Address Range
SIM	000–07F
Timer 1	600–63F
Timer 2	640–67F
Serial	700–73F
DMA Channel 1	780–79F
DMA Channel 2	7A0–7BF

4.2.2 System Configuration and Protection Submodule

The SIM allows the user to control certain features of system configuration by writing bits in the module configuration register (MCR). This register also contains read-only status bits that show the state of the SIM.

All M68000 Family members are designed to provide maximum system safeguards. As an extension of the family, the MC68340 promotes the same basic concepts of safeguarded design present in all M68000 members. In addition, many functions that normally must be provided in external circuits are incorporated in this device. The following features are provided in the system configuration and protection submodule:

SIM Configuration

The SIM allows the user to configure the system according to the particular requirements. The functions include control of FREEZE and show cycle operation, the function of the $\overline{\text{CS3}}$–$\overline{\text{CS0}}$ signals, the access privilege of the supervisor/user registers, the level of interrupt arbitration, and automatic autovectoring for external interrupts.

Reset Status

The reset status register provides the user with information on the cause of the most recent reset. The possible causes include: external, power-up, software watchdog, double bus fault, loss of clock, and reset instruction.

Internal Bus Monitor

The SIM provides an internal bus monitor to monitor the data and size acknowledge (DSACK) response time for all internal bus accesses. An option allows the monitoring of internal-to-external bus accesses. Four selectable response times allow for variations in response speed of memory and peripherals used in the system. A bus error signal is asserted internally if the DSACK response limit is exceeded. $\overline{\text{BERR}}$ is not asserted externally. This function can be disabled.

Double Bus Fault Monitor

The double bus fault monitor causes a reset to occur if the internal HALT is asserted by the CPU32, indicating a double bus fault. This function can be disabled. See **SECTION 3 BUS OPERATION** for more information.

Spurious Interrupt Monitor

If no interrupt arbitration occurs during an interrupt acknowledge cycle (IACK), the bus error signal is asserted internally.

Software Watchdog

The software watchdog asserts reset or a level 7 interrupt (as selected by the system protection and control register) if the software fails to service the software watchdog for a designated period of time (i.e., because it is trapped in a loop or lost). There are eight selectable timeout periods. This function can be disabled.

Periodic Interrupt Timer

The SIM provides a timer to generate periodic interrupts. The periodic interrupt time period can vary from 122 µs to 15.94 s (with a 32.768-kHz crystal used to generate the system clock). This function can be disabled.

Figure 4-2 shows a block diagram of the system configuration and protection submodule.

4.2.2.1 SYSTEM CONFIGURATION. Aspects of the system configuration are controlled by the MCR and the autovector register (AVR). The configuration of port B is controlled by the combination of the FIRQ bit in the MCR and the port B pin assignment register. Port B pins can function as dedicated I/O lines, chip selects, or $\overline{\text{IRQx}}/\overline{\text{AVEC}}$.

For debug purposes, accesses to internal peripherals can be shown on the external bus. This function is called show cycles. The SHEN1, SHEN0 bits in the MCR control show cycles. Bus arbitration can be either enabled or disabled during show cycles.

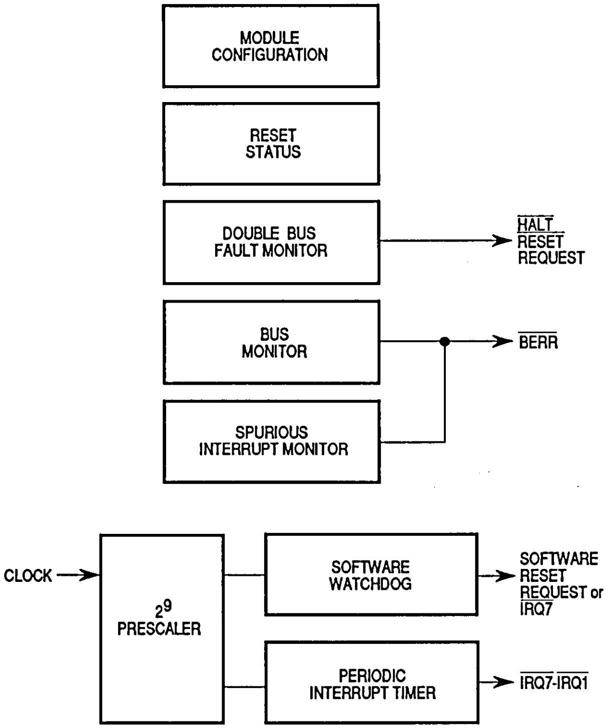


Figure 4-2. System Configuration and Protection Submodule

Arbitration for servicing interrupts is controlled by the value programmed into the interrupt arbitration (IARB) field of the MCR. Each module that generates interrupts, including the SIM, has an IARB field. (The SIM arbitrates for both its own interrupts and externally generated interrupts.) The value of the IARB field allows arbitration during an IACK cycle among modules that simultaneously generate the same interrupt level. No two modules should share the same IARB value.

There are eight arbitration levels for bus access, and the SIM is fixed at the highest level (level 7). The CPU32 is fixed at the lowest level (level 0). Only the SIM, the CPU32, and the direct memory access (DMA) module can be bus masters and arbitrate for the bus.

The AVR contains bits that correspond to external interrupt levels that require an autovector response. The SIM supports up to seven discrete external interrupt requests. If the bit corresponding to an interrupt level is set in the AVR, the SIM returns an autovector in response to the IACK cycle servicing that external interrupt request. Otherwise, external circuitry must either return an interrupt vector or assert the external AVEC signal.

4.2.2.2 INTERNAL BUS MONITOR. The internal bus monitor continually checks for the bus cycle termination response time by checking the DSACKx, BERR, and HALT status or the AVEC status during an IACK cycle. The monitor initiates BERR if the response time is excessive. The internal bus monitor cannot check the DSACKx response on the external bus unless the MC68340 is the bus master. The BME bit in the system protection control register (SYPCR) enables the internal bus monitor for internal-to-external bus cycles. If the system contains external bus masters whose bus cycles must be monitored, an external bus monitor can be implemented. In this case, the internal-to-external bus monitor option must be disabled. The bus monitor feature cannot be disabled for internal accesses to internal modules.

The bus cycle termination response time is measured in clock cycles, and the maximum-allowable response time is programmable. The bus monitor response time period ranges from 8 to 64 system clocks (see Table 4-8). These options are provided to allow for different response times of peripherals that might be used in the system.

4.2.2.3 DOUBLE BUS FAULT MONITOR. The double bus fault monitor responds to an assertion of HALT on the internal bus. Refer to **SECTION 3 BUS OPERATION** for more information. The DBF bit in the reset status register indicates that the last reset was caused by the double bus fault monitor. The double bus fault monitor reset can be inhibited by the DBFE bit in the SYPCR.

4.2.2.4 SPURIOUS INTERRUPT MONITOR. The spurious interrupt monitor issues BERR if no interrupt arbitration occurs during an IACK cycle. Normally, during an IACK cycle, one or more internal submodules recognize that the CPU32 is responding to their interrupt request(s) and arbitrate for the privilege of returning a vector or asserting AVEC. (The SIM reports and arbitrates for externally generated interrupts.) This feature cannot be disabled.

4.2.2.5 SOFTWARE WATCHDOG. Once enabled by the SWE bit in the MCR, the software watchdog requires a special service sequence to be executed on a periodic basis. If this periodic servicing action does not occur, the software watchdog times out and issues a reset or a level 7 interrupt (as programmed by the SWRI bit in the SYPCR). The address of the interrupt service routine for the software watchdog interrupt is stored in the software interrupt vector register (SWIV). Figure 4-3 shows a block diagram of the software watchdog as well as the clock control circuits for the periodic interrupt timer.

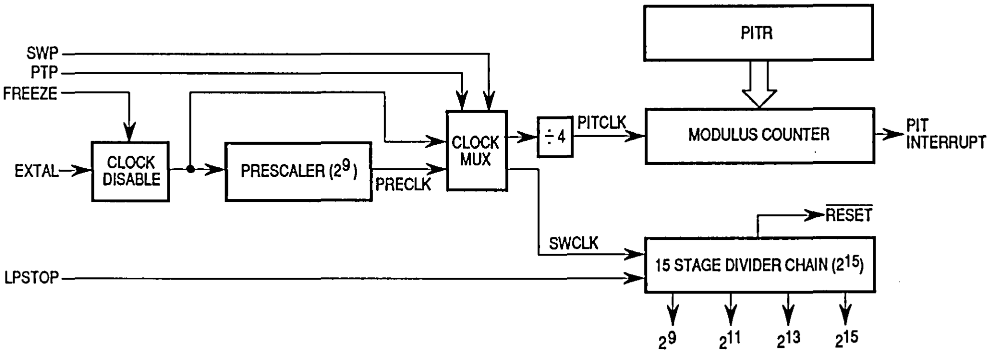


Figure 4-3. Software Watchdog

This mechanism protects the system against the possibility of the software becoming trapped in loops or running away. See Table 4-7 for a list of watchdog timeout periods. The watchdog clock rate is determined by the SWP bit in the periodic interrupt timer register (PITR) and the SWT bits in the SYPCR.

The software watchdog service sequence consists of the following two steps: write \$55 to the software service register (SWSR) and write \$AA to the SWSR. Both writes must occur in the order listed prior to the watchdog timeout, but any number of instructions or accesses to the SWSR can be executed between the two writes.

4.2.2.6 PERIODIC INTERRUPT TIMER. The periodic interrupt timer consists of an 8-bit modulus counter that is loaded with the value contained in the PITR (see Figure 4-3). The modulus counter is clocked by a signal derived from the buffered crystal oscillator (EXTAL) input pin unless an external frequency source is used. When an external frequency source is used (MODCK low during reset), the default state of the prescaler control bits (SWP and PTP) in the PITR is changed to enable both prescalers.

Either clock source (EXTAL or EXTAL÷512) is divided by four before driving the modulus counter (PITCLK). When the modulus counter value reaches zero, an interrupt is generated. The level of the generated interrupt is programmed into the PIRQL bits in the periodic interrupt control register (PICR). During the IACK cycle, the SIM places the periodic interrupt vector, programmed into the PIV bits in the PICR, onto the bus. The value of bits 7–0 in the PITR is then loaded again into the modulus counter, and the counting process starts over. If a new value is written to the PITR, this value is loaded into the modulus counter when the current count is completed.

4.2.2.6.1 Periodic Timer Period Calculation. The period of the periodic timer can be calculated using the following equation:

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{\dfrac{\text{EXTAL frequency/prescaler value}}{2^2}}$$

Solving the equation using a crystal frequency of 32.768 kHz with the prescaler disabled gives:

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{\dfrac{32768/1}{2^2}}$$

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{8192}$$

This gives a range from 122 µs, with a PITR value of \$01 (00000001 binary), to 31.128 ms, with a PITR value of \$FF (11111111 binary).

Solving the equation with the prescaler enabled (PTP=1) gives the following values:

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{\dfrac{32768/512}{2^2}}$$

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{16}$$

This gives a range from 62.5 ms, with a PITR value of \$01, to 15.94 s, with a PITR value of \$FF.

For fast calculation of periodic timer period using a 32.768-kHz crystal, the following equations can be used:

With prescaler disabled:

$$\text{programmable interrupt timer period} = \text{PITR } (122\ \mu\text{s})$$

With prescaler enabled:

$$\text{programmable interrupt timer period} = \text{PITR } (62.5\ \text{ms})$$

4.2.2.6.2 Using the Periodic Timer as a Real-Time Clock. The periodic interrupt timer can be used as a real-time clock interrupt by setting it up to generate an interrupt with a one-second period. Rearranging the periodic timer period equation to solve for the desired count value:

$$\text{PITR count value} = \frac{(\text{PIT period})\ (\text{EXTAL frequency})}{(\text{Prescaler value})\ (2^2)}$$

$$\text{PITR count value} = \frac{(1)\ (32768)}{(512)\ (2^2)}$$

$$\text{PITR count value} = 16\ (\text{decimal})$$

Therefore, when using a 32.768-kHz crystal, the PITR should be loaded with a value of \$10 with the prescaler enabled to generate interrupts at a one-second rate.

4.2.3 Clock Synthesizer

The clock synthesizer can operate from an on-chip phase-locked loop (PLL) and voltage-controlled oscillator (VCO), using an external crystal connected between the EXTAL and XTAL pins as a reference frequency source. A 32.768-kHz watch crystal provides an inexpensive reference, but the reference crystal frequency can be any frequency from 25 to 50 kHz. Additionally, the system clock frequency can be driven directly into the EXTAL pin (the XTAL pin must be left floating for this case).

When using a 32.768-kHz crystal, the system clock frequency is programmable (using the W, X, and Y bits in the SYNCR) from 131 kHz to the maximum clock frequency specified in MC68340/D, *MC68340 Technical Summary*, with a resolution of 131 kHz. A status bit (SLIMP) in the SYNCR indicates that a loss of crystal reference has been detected. The RSTEN bit controls whether a loss of crystal causes a system reset or causes the device to operate in limp mode. In limp mode, the VCO runs approximately one-half maximum speed, using an internal voltage reference. Another status bit (SLOCK) in the SYNCR indicates when the VCO has locked onto the desired frequency or if an external clock is being used.

A separate power pin (V_{CCSYN}) is used to allow the clock circuits to run with the rest of the device powered down and to provide increased noise immunity for the clock circuits. The source for V_{CCSYN} should be a quiet power supply with adequate external bypass capacitors placed as close as possible to the V_{CCSYN} pin to ensure a stable operating frequency. Figure 4-4 shows a block diagram of the clock submodule and typical values for the bypass and PLL external capacitors. The crystal manufacturer's documentation should be consulted for specific recommendations for external components.

4.2.3.1 PHASE COMPARATOR AND FILTER.

The phase comparator takes the output of the frequency divider and compares it to a reference signal from an external crystal. The result of this compare is low-pass filtered and used to control the VCO. The comparator also detects when the crystal oscillator stops running to initiate the limp mode for the system clock.

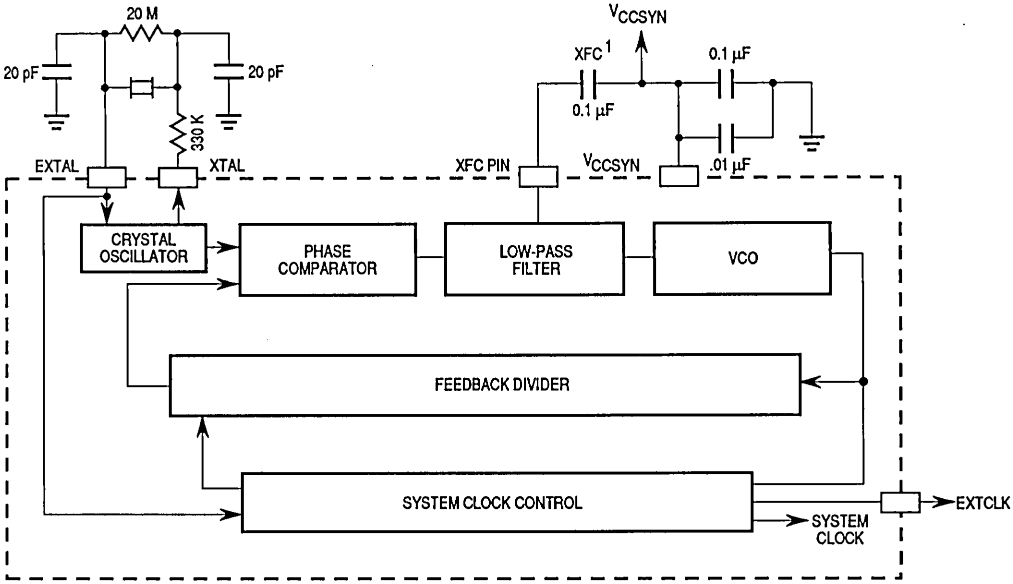
The PLL requires an external low-leakage filter capacitor, typically in the range from 0.01 to 0.1 µF, connected between the XFC and V_{CCSYN} pins. Smaller values of the external filter capacitor provide a faster response time for the PLL, and larger values provide greater frequency stability.

4.2.3.2 FREQUENCY DIVIDER.

The frequency divider circuits divide the VCO frequency down to the reference frequency for the phase comparator. The frequency divider consists of the following: 1) a 2-bit prescaler controlled by the W bit in the SYNCR and 2) a 6-bit modulo downcounter controlled by the Y bits in the SYNCR.

Several factors are important to the design of the system clock. The resulting system clock frequency must be within the limits specified for the device. The frequency of the system clock is given by the following equation:

$$F_{SYSTEM} = F_{CRYSTAL} \left(4(Y+1)2^{2W+X} \right)$$



NOTE 1: Must be low-leakage capacitor.

Figure 4-4. Clock Submodule Block Diagram

The maximum VCO frequency limit must also be observed. The VCO frequency is given by the following equation:

$$FVCO = F_{SYSTEM}{}^{(2-X)}$$

Since clearing the X-bit causes the VCO to run at twice the system frequency, the VCO upper frequency limit must be considered when programming the SYNCR. Both the system clock and VCO frequency limits are given in the MC68340/D, *MC68340 Technical Summary*. Table 4-2 lists some the frequencies available from various combinations of SYNCR bits with a reference frequency of 32.768 kHz.

Table 4-2. System Frequencies from 32.768-kHz Reference

Y	W=0; X=0	W=0; X=1	W=1; X=0	W=1; X=1
000000	131	262	524	1049
000101	786	1573	3146	6291
001010	1442	2884	5767	11534
001111	2097	4194	8389	16777
010100	2753	5505	11010	—
011001	3408	6816	13631	—
011111	4194	8389	16777	—
100011	4719	9437	—	—
101000	5374	10748	—	—
101101	6029	12059	—	—
110010	6685	13369	—	—
110111	7340	14680	—	—
111100	7995	15991	—	—
111111	8389	16777	—	—

NOTE: System frequencies are in kHz.

4.2.3.3 CLOCK CONTROL. The clock control circuits determine the source used for both internal and external clocks during special circumstances, such as low-power stop (LPSTOP) execution.

Table 4-3 summarizes the clock activity during LPSTOP, with MODCK=1 during reset. Any clock in the off state is held low. Two bits in the SYNCR (STEXT and STSIM) control clock activity during LPSTOP. Refer to **4.2.6 Low-Power Stop** for additional information.

Table 4-3. Clock Control Signals

Control Bits		Clock Outputs	
STSIM	**STEXT**	**SIMCLK**	**CLKOUT**
0	0	EXTAL	Off
0	1	EXTAL	EXTAL
1	0	VCO	Off
1	1	VCO	VCO

NOTE: SIMCLK runs the periodic interrupt, $\overline{\text{RESET}}$, and $\overline{\text{IRQx}}$ pin synchronizers in LPSTOP mode.

4.2.4 Chip-Select Submodule

Typical microprocessor systems require external hardware to provide select signals to external memory and peripherals. This device integrates these functions on-chip to provide the cost, speed, and reliability benefits of a higher level of integration. The chip-select submodule contains register pairs for each external chip-select signal. The pair consists of a base address register and an address mask register that define the characteristics of a single chip select. The register pair provides flexibility for a wide variety of chip-select functions.

4.2.4.1 PROGRAMMABLE FEATURES. The chip-select submodule supports the following programmable features:

Four Programmable Chip-Select Circuits

All four chip-select circuits are independently programmable from the same list of selectable features. Each chip-select circuit has an individual base address register and address mask register that contain the programmed characteristics of that chip select. The valid (V) bit of the base address register indicates that the register information for that chip select is valid. A global chip select allows address decode for a boot ROM before system initialization occurs.

Variable Block Sizes

The block size, starting from the specified base address, can vary in size from 256 bytes up to 4 Gbytes in 2^n increments. This size is specified in the address mask register.

Both 8- and 16-Bit Ports Supported

The 8-bit ports are accessible on both odd and even addresses when connected to data bus bits 15–8; the 16-bit ports can be accessed as odd bytes, even bytes, or even words. The port size is specified by the PS bits in the address mask register.

Write-Protect Capability

The WP bit in each base address register can restrict write access to its range of addresses.

Fast-Termination Option

Programming the FTE bit in the base address register for the fast-termination option causes the chip-select submodule to terminate the cycle by asserting the internal $\overline{\text{DSACKx}}$ early, providing a two-cycle external access. Peripherals using the fast-termination option must be 16 bits wide.

Internal $\overline{\text{DSACKx}}$ Generation for External Accesses with Programmable Wait States

The pin assignment register can be referenced for generating $\overline{\text{DSACKx}}$ with up to three wait states for a particular device using the DD bits in the address mask register.

Full 32-Bit Address Decode with Address Space Checking

All accesses privileges can be optionally checked through the use of the FC bits in the base address register and the FCM bits in the address mask register.

4.2.4.2 GLOBAL CHIP-SELECT OPERATION. Global chip-select operation allows address decode for a boot ROM before system initialization occurs. CS0 is the global chip-select output, and its operation differs from the other external chip-select outputs following reset. When the CPU32 begins fetching after reset, CS0 is asserted for every address, unless the module address base register is accessed or internal peripheral chip select is generated.

Global chip select provides a 16-bit port with three wait states, allowing a boot ROM to be located in any address space and still providing the stack pointer and program counter values at $00000000 and $00000004, respectively. CS0 operates in this manner until the V-bit is set in the $\overline{\text{CS0}}$ base address register. $\overline{\text{CS0}}$ can be programmed to continue decode for a range of addresses after the V-bit is set, provided the desired address range is first loaded into base address register 0. After the V-bit is set for $\overline{\text{CS0}}$, global chip select can only be restarted with a system reset.

A system can use an 8-bit boot ROM if an external 8-bit DSACK is generated which responds in two wait states or less.

4.2.5 External Bus Interface

This section describes port A and port B functions. Refer to **SECTION 3 BUS OPERATION** for more information about the external bus interface.

4.2.5.1 PORT A. Port A pins can be programmed to be either addresses A31–A24, discrete I/O pins, or $\overline{\text{IACKx}}$ pins. The port A pin assignment registers (PPARA1 and PPARA2) control the function of the port A pins as shown in Table 4-4. Each pin can be independently programmed. Upon reset, port A is configured as input pins. If the system uses these signals as addresses, pulldowns should be put on these signals to avoid indeterminate values until the port A registers can be programmed.

Table 4-4. Port A Pin Assignment Register Function

Signal	Pin Function		
	PPARA1 BIT = 0 PPARA2 BIT = 0	PPARA1 BIT = 1 PPARA2 BIT = X	PPARA1 BIT = 0 PPARA2 BIT = 1
A31	A31	PORT A7	$\overline{\text{IACK7}}$
A30	A30	PORT A6	$\overline{\text{IACK6}}$
A29	A29	PORT A5	$\overline{\text{IACK5}}$
A28	A28	PORT A4	$\overline{\text{IACK4}}$
A27	A27	PORT A3	$\overline{\text{IACK3}}$
A26	A26	PORT A2	$\overline{\text{IACK2}}$
A25	A25	PORT A1	$\overline{\text{IACK1}}$
A24	A24	PORT A0	—

4.2.5.2 PORT B. Port B pins can be programmed to be chip selects, $\overline{\text{IRQx}}$ and MODCK pins, or discrete I/O pins. These pins are multiplexed as shown in Figure 4-5. Selection of pin function is done by a combination of the port B pin assignment register (PPARB) and the FIRQ bit of the MCR (see Table 4-5). Each pin can be independently programmed. Upon reset, port B is configured as MODCK, $\overline{\text{IRQ7}}$, $\overline{\text{IRQ6}}$, $\overline{\text{IRQ5}}$, $\overline{\text{IRQ3}}$, and $\overline{\text{CS3}}$–$\overline{\text{CS0}}$.

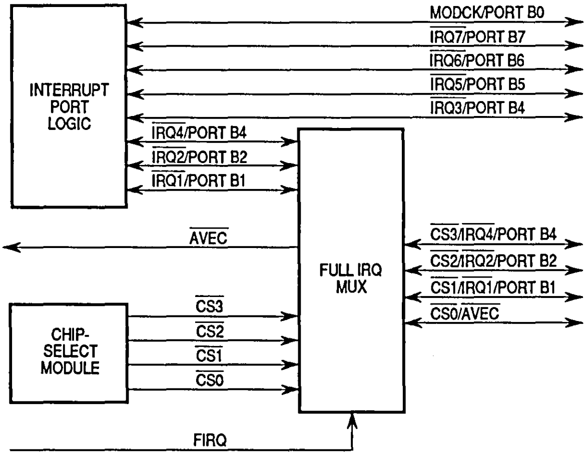


Figure 4-5. Full Interrupt Request Multiplexer

Table 4-5. Port B Pin Assignment Register and FIRQ Bit Function

Signal	Pin Function			
	FIRQ=0 PPARB BIT = 0	FIRQ = 0 PPARB BIT = 1	FIRQ = 1 PPARB BIT = 0	FIRQ = 1 PPARB BIT = 1
$\overline{\text{IRQ7}}$	PORT B7	$\overline{\text{IRQ7}}$	PORT B7	$\overline{\text{IRQ7}}$
$\overline{\text{IRQ6}}$	PORT B6	$\overline{\text{IRQ6}}$	PORT B6	$\overline{\text{IRQ6}}$
$\overline{\text{IRQ5}}$	PORT B5	$\overline{\text{IRQ5}}$	PORT B5	$\overline{\text{IRQ5}}$
$\overline{\text{CS3}}$	$\overline{\text{CS3}}$	$\overline{\text{CS3}}$	PORT B4	$\overline{\text{IRQ4}}$
$\overline{\text{IRQ3}}$	PORT B3	$\overline{\text{IRQ3}}$	PORT B3	$\overline{\text{IRQ3}}$
$\overline{\text{CS2}}$	$\overline{\text{CS2}}$	$\overline{\text{CS2}}$	PORT B2	$\overline{\text{IRQ2}}$
$\overline{\text{CS1}}$	$\overline{\text{CS1}}$	$\overline{\text{CS1}}$	PORT B1	$\overline{\text{IRQ1}}$
$\overline{\text{CS0}}$	$\overline{\text{CS0}}$	$\overline{\text{CS0}}$	$\overline{\text{AVEC}}$	$\overline{\text{AVEC}}$
MODCK	PORT B0	MODCK	PORT B0	MODCK

NOTE: MODCK has no function after reset.

4.2.6 Low-Power Stop

The LPSTOP mode provides reduced power consumption when the MC68340 is idle. All internal modules have no clock, except the SIM, which remains active. Operation of the SIM clock and CLKOUT during LPSTOP is controlled by the STSIM and STEXT bits in the SYNCR. Execution of the LPSTOP instruction disables the clock to the software watchdog in the low state. The software watchdog remains stopped until the LPSTOP state is ended and begins to run again on the next rising clock edge.

NOTE

When the CPU32 executes the STOP instruction (as opposed to LPSTOP), the software watchdog continues to run. If the software watchdog is enabled, it issues a reset or interrupt when timeout occurs.

The periodic interrupt timer does not respond to an LPSTOP instruction; thus, it can be used to exit LPSTOP as long as the interrupt request level is higher than the CPU32 interrupt mask level. To stop the periodic interrupt timer while in LPSTOP, the PITR must be loaded with a zero value before LPSTOP is executed. The bus monitor, double bus fault monitor, and spurious interrupt monitor are all inactive during LPSTOP.

4.2.7 Freeze

FREEZE is asserted by the CPU32 if a breakpoint is encountered with background mode enabled. Refer to **SECTION 5 CPU32** for more information on the background mode. When FREEZE is asserted, the double bus fault monitor and spurious interrupt monitor continue to operate normally. However, the software watchdog and the periodic interrupt timer may be affected. Setting the FRZ1 bit in the MCR disables the software watchdog when FREEZE is asserted, and setting the FRZ0 bit in the MCR disables the periodic interrupt timer when FREEZE is asserted.

4.3 PROGRAMMER'S MODEL

Figure 4-6 is a programmer's model (register map) of all registers in the SIM. For more information about a particular register, refer to the description for the module or submodule indicated in the right column. The ADDR (address) column indicates the offset of the register from the address stored in the base address register. The FC (function code) column indicates whether a register is restricted to supervisor access (S) or programmable to exist in either supervisor or user space (S/U).

In the registers discussed in the following pages, the number in the upper right-hand corner indicates the offset of the register from the address stored in the base address register. The numbers on the top line of the register represent the bit position in the register. The second line contains the mnemonic for the bit. The numbers below the register represent the bit values after reset. The access privilege is indicated in the lower right-hand corner.

ADDR	FC	15 8	7 0	
000	S	MODULE CONFIGURATION REGISTER (MCR)		SYSTEM PROTECTION
004	S	CLOCK SYNTHESIZER CONTROL REGISTER (SYNCR)		CLOCK
006	S	AUTOVECTOR REGISTER (AVR)	RESET STATUS REGISTER (RSR)	SYSTEM PROTECTION
010	S/U	RESERVED	PORT A DATA (PORTA)	EBI
012	S/U	RESERVED	PORT A DATA DIRECTION (DDRA)	EBI
014	S	RESERVED	PORT A PIN ASSIGNMENT 1 (PPRA1)	EBI
016	S	RESERVED	PORT A PIN ASSIGNMENT 2 (PPRA2)	EBI
018	S/U	RESERVED	PORT B DATA (PORTB)	EBI
01A	S/U	RESERVED	PORT B DATA (PORTB1)	EBI
01C	S/U	RESERVED	PORT B DATA DIRECTION (DDRB)	EBI
01E	S	RESERVED	PORT B PIN ASSIGNMENT (PPARB)	EBI
020	S	SW INTERRUPT VECTOR (SWIV)	SYSTEM PROTECTION CONTROL (SYPCR)	SYSTEM PROTECTION
022	S	PERIODIC INTERRUPT CONTROL REGISTER (PICR)		SYSTEM PROTECTION
024	S	PERIODIC INTERRUPT TIMING REGISTER (PITR)		SYSTEM PROTECTION
026	S	RESERVED	SOFTWARE SERVICE (SWSR)	SYSTEM PROTECTION
040	S	ADDRESS MASK 1 CS0		CHIP SELECT
042	S	ADDRESS MASK 2 CS0		CHIP SELECT
044	S	BASE ADDRESS 1 CS0		CHIP SELECT
046	S	BASE ADDRESS 2 CS0		CHIP SELECT
048	S	ADDRESS MASK 1 CS1		CHIP SELECT
04A	S	ADDRESS MASK 2 CS1		CHIP SELECT
04C	S	BASE ADDRESS 1 CS1		CHIP SELECT
04E	S	BASE ADDRESS 2 CS1		CHIP SELECT
050	S	ADDRESS MASK 1 CS2		CHIP SELECT
052	S	ADDRESS MASK 2 CS2		CHIP SELECT
054	S	BASE ADDRESS 1 CS2		CHIP SELECT
056	S	BASE ADDRESS 2 CS2		CHIP SELECT
058	S	ADDRESS MASK 1 CS3		CHIP SELECT
05A	S	ADDRESS MASK 2 CS3		CHIP SELECT
05C	S	BASE ADDRESS 1 CS3		CHIP SELECT
05E	S	BASE ADDRESS 2 CS3		CHIP SELECT

Figure 4-6. SIM Programming Model

4.3.1 Module Base Address Register

Module Base Address Register 1 $03FF00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	BA18	BA17	BA16
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CPU Space Only

Module Base Address Register 2 $03FF02

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BA15	BA14	BA13	BA12	0	0	0	0	0	0	0	0	0	0	0	V
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CPU Space Only

BA31–BA12 — Base Address Bits 31–12

The base address field is the upper 20 bits of the module base address register, providing for block starting locations in increments of 4K-bytes.

V — Valid Bit

This bit indicates when the contents of the module base address register are valid. The base address value is not used; therefore, all internal module registers are not accessible until the V-bit is set.

1 = Contents valid
0 = Contents not valid

NOTE

An access to this register does not affect external space since the cycle is not run externally.

4.3.2 System Configuration and Protection Registers

The following paragraphs provide descriptions of the system configuration and protection registers.

4.3.2.1 MODULE CONFIGURATION REGISTER (MCR). The MCR, which controls the SIM configuration, can be read or written at any time.

MCR $000

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	FRZ1	FRZ0	FIRQ	0	0	SHEN1	SHEN0	SUPV	0	0	0	IARB3	IARB2	IARB1	IARB0
RESET:	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	1

Supervisor Only

FRZ1 — Freeze Software Watchdog Enable

1 = When FREEZE is asserted, the software watchdog counters are disabled, preventing interrupts from occurring during software debug.

0 = When FREEZE is asserted, the software watchdog counters continue to run. See **4.2.7 Freeze** for more information.

FRZ0 — Freeze Periodic Interrupt Timer Enable

1 = When FREEZE is asserted, the periodic interrupt timer counters are disabled.

0 = When FREEZE is asserted, the periodic interrupt timer counters continue to operate as programmed.

FIRQ — Full Interrupt Request Mode

1 = Configures port B for seven interrupt request lines, autovector, and no external chip selects.

0 = Configures port B for four interrupt request lines and four external chip selects.

See **4.2.5.2 PORT B** for a description of the FIRQ bit and port B logic.

SHEN1, SHEN0 — Show Cycle Enable

These two control bits determine what the EBI does with the external bus during internal transfer operations. A show cycle allows internal transfers to be externally monitored. The address, data, and control signals, except for $\overline{AS}$, are driven externally. Table 4-6 lists all show cycle bit combinations, whether or not show cycle data is driven externally, and whether external bus arbitration can occur. If external bus arbitration is disabled, the EBI will not recognize an external bus request until arbitration is enabled again. Address information is always driven externally unless the external bus is relinquished. However, data is not driven and $\overline{DS}$ is not asserted for internal accesses unless show cycles are enabled. When both show cycle bits are set, an external bus request causes an internal master to finish its current cycle and no longer initiate cycles on the internal bus. The internal master resumes running cycles on the bus after $\overline{\text{BR}}$ and $\overline{\text{BGACK}}$ are negated. To prevent bus conflicts, external peripherals must not attempt to initiate cycles during show cycles with arbitration disabled.

Table 4-6. Show Cycle Control Bits

SHEN1	SHEN0	ACTION
0	0	Show cycles disabled, external arbitration enabled
0	1	Show cycles enabled, external arbitration disabled
1	0	Show cycles enabled, external arbitration enabled
1	1	Show cycles enabled, external arbitration enabled, internal activity halted by a bus grant

SUPV — Supervisor/User Data Space

The SUPV bit defines the SIM global registers as either supervisor data space or user (unrestricted) data space.

1 = The SIM registers defined as supervisor/user are restricted to supervisor data access (FC3–FC0 = \$5). An attempted user-space write is ignored and returns $\overline{\text{BERR}}$.

0 = The SIM registers defined as supervisor/user data are unrestricted (FC2 is a don't care).

IARB3–IARB0 — Interrupt Arbitration Bits 3–0

The reset value of IARB is \$F, allowing the SIM to arbitrate during an IACK cycle immediately after reset. The system software should initialize the IARB field to a value from \$F (highest priority) to \$1 (lowest priority). A value of \$0 prevents arbitration and causes all SIM interrupts, including external interrupts, to be discarded as extraneous.

4.3.2.2 AUTOVECTOR REGISTER (AVR). The AVR contains bits that correspond to external interrupt levels that require an autovector response. Setting a bit allows the SIM to assert an internal AVEC during the IACK cycle in response to the specified interrupt request level. This register can be read and written at any time.

AVR $006

7	6	5	4	3	2	1	0
AV7	AV6	AV5	AV4	AV3	AV2	AV1	AV0
RESET:							
0	0	0	0	0	0	0	0

Supervisor Only

4.3.2.3 RESET STATUS REGISTER (RSR).

The RSR contains a bit for each reset source to the SIM. A set bit indicates the last type of reset that occurred, and only one bit can be set in the register. The RSR is updated by the reset control logic when the SIM comes out of reset. This register can be read at any time; a write has no effect. For more information, see **SECTION 3 BUS OPERATION.**

RSR $007

7	6	5	4	3	2	1	0
EXT	POW	SW	DBF	0	LOC	SYS	0

Supervisor Only

EXT — External Reset

1 = The last reset was caused by an external signal driving $\overline{\text{RESET}}$.

POW — Power-Up Reset

1 = The last reset was caused by the power-up reset circuit.

SW — Software Watchdog Reset

1 = The last reset was caused by the software watchdog circuit.

DBF — Double Bus Fault Monitor Reset

1 = The last reset was caused by the double bus fault monitor.

LOC — Loss of Clock Reset

1 = The last reset was caused by a loss of frequency reference to the clock submodule. This reset can only occur if the RSTEN bit in the clock submodule is set and the VCO is enabled.

SYS — System Reset

1 = The last reset was caused by the CPU32 executing a reset instruction. The system reset does not load a reset vector or affect any internal CPU32 registers or SIM configuration registers, but does reset external devices and other internal modules.

4.3.2.4 SOFTWARE INTERRUPT VECTOR REGISTER (SWIV). The SWIV contains the 8-bit vector that is returned by the SIM during an IACK cycle in response to an interrupt generated by the software watchdog. This register can be read or written at any time. This register is set to the uninitialized vector, $0F, at reset.

SWIV $020

7	6	5	4	3	2	1	0
SWIV7	SWIV6	SWIV5	SWIV4	SWIV3	SWIV2	SWIV1	SWIV0
RESET:							
0	0	0	0	1	1	1	1

Supervisor Only

4.3.2.5 SYSTEM PROTECTION CONTROL REGISTER (SYPCR). The SYPCR controls the system monitors, the prescaler for the software watchdog, and the bus monitor timing. This register can be read at any time but can be written only once after reset.

SYPCR $021

7	6	5	4	3	2	1	0
SWE	SWRI	SWTI	SWT0	DBFE	BME	BMT1	BMT0
RESET:							
0	0	0	0	0	0	0	0

Supervisor Only

SWE — Software Watchdog Enable
1 = Software watchdog enabled
0 = Software watchdog disabled
See **4.2.2.5 SOFTWARE WATCHDOG** for more information.

SWRI — Software Watchdog Reset/Interrupt Select
1 = Software watchdog causes a system reset.
0 = Software watchdog causes a level 7 interrupt to the CPU32.

SWT1, SWT0 — Software Watchdog Timing

These bits, along with the SWP bit in the PITR, control the divide ratio used to establish the timeout period for the software watchdog. The software watchdog timeout period is given by the following formula:

$$\frac{1}{\text{EXTAL frequency/divide count}}$$

or

$$\frac{\text{divide count}}{\text{EXTAL frequency}}$$

The software watchdog timeout period, listed in Table 4-7, gives the formula to derive the software watchdog timeout for any clock frequency. The timeout periods are listed for a 32.768-kHz crystal used with the VCO and for a 16.777-MHz external oscillator.

Table 4-7. Deriving Software Watchdog Timeout

SWP	SWT1	SWT0	Software Timeout Period	32.768-kHz Crystal Period	16.777-MHz External Clock Period
0	0	0	2^9/EXTAL Input Frequency	15.6 ms	30 μs
0	0	1	2^{11}/EXTAL Input Frequency	62.5 ms	122 μs
0	1	0	2^{13}/EXTAL Input Frequency	250 ms	488 μs
0	1	1	2^{15}/EXTAL Input Frequency	1 s	1.45 μs
1	0	0	2^{18}/EXTAL Input Frequency	8 s	15.6 μs
1	0	1	2^{20}/EXTAL Input Frequency	32 s	62.5 μs
1	1	0	2^{22}/EXTAL Input Frequency	128 s	250 μs
1	1	1	2^{24}/EXTAL Input Frequency	512 s	1 μs

CAUTION

When the SWP and SWT bits are modified to select a software timeout other than the default, the software service sequence ($55 followed by $AA written to the software service register) must be performed before the new timeout period takes effect.

Refer to **4.2.2.5 SOFTWARE WATCHDOG** for more information.

DBFE — Double Bus Fault Monitor Enable

1 = Enable double bus fault monitor function

0 = Disable double bus fault monitor function

For more information, see **4.2.2.3 DOUBLE BUS FAULT MONITOR** and **SECTION 5 CPU32.**

BME — Bus Monitor External Enable

1 = Enable bus monitor function for an internal-to-external bus cycle.

0 = Disable bus monitor function for an internal-to-external bus cycle.

For more information see **4.2.2.2 INTERNAL BUS MONITOR.**

BMT — Bus Monitor Timing.

These bits select the timeout period for the bus monitor (see Table 4-8).

Table 4-8. BMT Encoding

BMT1	BMT0	Bus Monitor Timeout Period
0	0	64 system clocks (CLKOUT)
0	1	32 system clocks
1	0	16 system clocks
1	1	8 system clocks

4.3.2.6 PERIODIC INTERRUPT CONTROL REGISTER (PICR). The PICR contains the interrupt level and the vector number for the periodic interrupt request. This register can be read or written at any time. Bits 15–11 are unimplemented and always return zero; a write to these bits has no effect.

PICR $022

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	PIRQL2	PIRQL1	PIRQL0	PIV7	PIV6	PIV5	PIV4	PIV3	PIV2	PIV1	PIV0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Supervisor Only

PIRQL2–PIRQL0 — Periodic Interrupt Request Level

These bits contain the periodic interrupt request level. Table 4-9 lists which interrupt request level is asserted during an IACK cycle when a periodic interrupt is generated. The periodic timer continues to run when the interrupt is disabled.

Table 4-9. PIRQL Encoding

PIRQL2	PIRQL1	PIRQL0	Interrupt Request Level
0	0	0	Periodic Interrupt Disabled
0	0	1	Interrupt Request Level 1
0	1	0	Interrupt Request Level 2
0	1	1	Interrupt Request Level 3
1	0	0	Interrupt Request Level 4
1	0	1	Interrupt Request Level 5
1	1	0	Interrupt Request Level 6
1	1	1	Interrupt Request Level 7

PIV7–PIV0 — Periodic Interrupt Vector Bits 7–0

These bits contain the value of the vector generated during an IACK cycle in response to an interrupt from the periodic timer. When the SIM responds to the IACK cycle, the periodic interrupt vector from the PICR is placed on the bus. This vector number is multiplied by four to form the vector offset, which is added to the vector base register to obtain the address of the vector.

4.3.2.7 PERIODIC INTERRUPT TIMER REGISTER (PITR). The PITR contains control for prescaling the software watchdog and periodic timer as well as the count value for the periodic timer. This register can be read or written at any time. Bits 15–10 are not implemented and always return zero when read. A write does not affect these bits.

PITR $024

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	SWP	PTP	PITR7	PITR6	PITR5	PITR4	PITR3	PITR2	PITR1	PITR0
RESET:															
0	0	0	0	0	0	$\overline{\text{MODCK}}$	$\overline{\text{MODCK}}$	0	0	0	0	0	0	0	0

Supervisor Only

SWP — Software Watchdog Prescale

This bit controls the software watchdog clock source as shown in **4.3.2.5 SYSTEM PROTECTION CONTROL REGISTER (SYPCR).**

1 = Software watchdog clock prescaled by a value of 512
0 = Software watchdog clock not prescaled

The SWP reset value is the inverse of the MODCK bit state on the rising edge of reset.

PTP — Periodic Timer Prescaler Control

This bit contains the prescaler control for the periodic timer.

1 = Periodic timer clock prescaled by a value of 512
0 = Periodic timer clock not prescaled

The PTP reset value is the inverse of the MODCK bit state on the rising edge of reset.

PITR7–PITR0 — Periodic Interrupt Timer Register Bits 7–0

The remaining bits of the PITR contain the count value for the periodic timer. A zero value turns off the periodic timer.

4.3.2.8 SOFTWARE SERVICE REGISTER (SWSR). The SWSR is the location to which the software watchdog servicing sequence is written. The software watchdog can be enabled or disabled by the SWE bit in the SYPCR. SWSR can be written at any time but returns all zeros when read.

SWSR $027

7	6	5	4	3	2	1	0
SWSR7	SWSR6	SWSR5	SWSR4	SWSR3	SWSR2	SWSR1	SWSR0
RESET:							
0	0	0	0	0	0	0	0

Supervisor Only

4.3.3 Clock Synthesizer Control Register (SYNCR)

The SYNCR can be read or written only in supervisor mode. The reset state of SYNCR produces an operating frequency of 8.38 MHz when the PLL is referenced to a 32.768-kHz crystal. The system frequency is controlled by the frequency control bits in the upper byte of the SYNCR as follows:

$$F_{SYSTEM} = F_{CRYSTAL} \left(4(Y+1)2^{2W+X} \right)$$

SYNCR $004

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	X	Y5	Y4	Y3	Y2	Y1	Y0	RSVD	0	0	SLIMP	SLOCK	RSTEN	STSIM	STEXT
RESET:															
0	0	1	1	1	1	1	1	0	0	0	U	U	0	0	0

U = Unaffected by reset Supervisor Only

W — Frequency Control Bit

This bit controls the prescaler tap in the synthesizer feedback loop. Setting the bit increases the VCO speed by a factor of four, requiring a time delay for the VCO to relock (see equation for determining system frequency).

X — Frequency Control Bit

This bit controls a divide-by-two prescaler, which is not in the synthesizer feedback loop. Setting the bit doubles the system clock speed without changing the VCO speed, as specified in the equation for determining system frequency; therefore, no delay is incurred to relock the VCO.

Y5–Y0 — Frequency Control Bits

The Y-bits, with a value from 0–63, control the modulus downcounter in the synthesizer feedback loop, causing it to divide by the value of Y+1 (see the equation for determining system frequency). Changing these bits requires a time delay for the VCO to relock.

RSVD — Reserved

This bit is reserved for factory testing.

SLIMP — Limp Mode

1 = A loss of crystal reference has been detected, and the VCO is running at approximately one-half maximum speed, determined from an internal voltage reference.

0 = External crystal frequency is at VCO reference.

SLOCK — Synthesizer Lock

1 = VCO has locked onto the desired frequency (or system clock is driven externally).

0 = VCO is enabled but has not yet locked.

RSTEN — Reset Enable

1 = Loss of crystal causes a system reset.

0 = Loss of crystal causes the VCO to operate at a nominal speed without external reference (limp mode), and the device continues to operate at that speed.

STSIM — Stop Mode System Integration Clock

1 = When the LPSTOP instruction is executed, the SIM clock is driven from the VCO.

0 = When the LPSTOP instruction is executed, the SIM clock is driven from the crystal oscillator, and the VCO is turned off to conserve power.

STEXT — Stop Mode External Clock

1 = When the LPSTOP instruction is executed, the external clock pin (CLKOUT) is driven from the SIM clock as determined by the STSIM bit.

0 = When the LPSTOP instruction is executed, the external clock is held low to conserve power.

4.3.4 Chip-Select Registers

The following paragraphs provide descriptions of the registers in the chip-select submodule.

4.3.4.1 BASE ADDRESS REGISTERS. There are four base address registers in the chip-select submodule, one for each chip-select signal.

Base Address 1 $044, $04C, $054, $05C

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	BA18	BA17	BA16
RESET:	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Supervisor Only

Base Address 2 $046, $04E, $056, $05E

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	FC3	FC2	FC1	FC0	WP	FTE	0	V
RESET:	U	U	U	U	U	U	U	U	U	U	U	U	U	U	0	0

U = Unaffected by reset Supervisor Only

BA31–BA8 — Base Address Bits 31–8

The base address field, the upper 24 bits of each base address register, provides for block sizes in increments of 256 bytes. The base address field (and the function code field) is compared to the address on the address bus to determine if a chip select should be generated.

FC3–FC0 — Function Code Bits 3–0

This field can be used to specify access to a certain address space type.

WP — Write Protect

This bit can restrict write accesses to the address range in a base address register. An attempt to write to the range of addresses specified in a base address register that has this bit set returns $\overline{\text{BERR}}$.

1 = Only read accesses allowed
0 = Either read or write allowed

FTE — Fast-Termination Enable

This bit causes the submodule to terminate the cycle early with a word-sized $\overline{\text{DSACKx}}$, giving a fast two-clock external access. When clear, all external cycles are at least three clocks. If fast termination is enabled, the DD and PS bits of the corresponding address mask register are overridden (see **SECTION 3 BUS OPERATION**).

1 = Fast-termination cycle enabled
0 = Fast-termination cycle disabled (termination determined by DD and PS bits)

V — Valid Bit

This bit indicates that the contents of its base address register and address mask register pair are valid. The programmed chip selects do not assert until the V-bit is set.

1 = Contents valid
0 = Contents not valid

4.3.4.2 ADDRESS MASK REGISTERS. There are four address mask registers in the chip-select submodule, one for each chip-select signal.

Address Mask 1 $040, $048, $050, $058

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	AM31	AM30	AM29	AM28	AM27	AM26	AM25	AM24	AM23	AM22	AM21	AM20	AM19	AM18	AM17	AM16
RESET:	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Supervisor Only

Address Mask 2 $042, $04A, $052, $05A

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AM5	AM14	AM13	AM12	AM11	AM10	AM9	AM8	FCM3	FCM2	FCM1	FCM0	DD1	DD2	PS1	PS0
RESET:	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor Only

AM31–AM8 — Address Mask Bits 31–8

The address mask field, the upper 24 bits of each address mask register, provides for masking any of the corresponding bits in the associated base address register. By masking the address bits independently, external devices of different size address ranges can be used. Any set bit masks the corresponding address bit. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. This field can be read or written at any time.

FCM3–FCM0 — Function Code Mask Bits 3–0

This field can be used to mask certain function code bits, allowing more than one address space type to be assigned to a chip select. Any set bit masks the corresponding function code bit.

DD1,DD0 — DSACK Delay Bits 1 and 0

This field determines the number of wait states added before $\overline{\text{DSACKx}}$ is returned for that entry. The port size field must be programmed for a $\overline{\text{DSACKx}}$ response, or the DD bits have no significance. Table 4-10 lists the encoding for the DD bits.

Table 4-10. DD Encoding

DD1	DD0	Response
0	0	Zero Wait State
0	1	One Wait State
1	0	Two Wait States
1	1	Three Wait States

PS1, PS0 — Port Size Bits 1 and 0

This field determines whether a given chip select responds with $\overline{\mathrm{DSACKx}}$ and, if so, what port size is returned. Table 4-11 lists the encoding for the PS bits.

Table 4-11. PS Encoding

PS1	PS0	Mode
0	0	Reserved
0	1	16-Bit Port
1	0	8-Bit Port
1	1	No $\overline{\mathrm{DSACKx}}$ Response

An example of the address mask register for 128 kbytes in user space with a 16-bit port requiring one wait state is as follows:

Address Mask 1 = $0001
Address Mask 2 = $FF15

4.3.5 External Bus Interface Control

The following paragraphs describe the registers that control the I/O pins used with the external bus interface. Refer to the **SECTION 3 BUS OPERATION** for more information about the external bus interface. For a list of pin numbers used with port A and port B, see the pinout diagram in **SECTION 12 ORDERING INFORMATION AND MECHANICAL DATA. SECTION 2 SIGNAL DESCRIPTIONS** shows a block diagram of the port control circuits.

4.3.5.1 PORT A PIN ASSIGNMENT REGISTER 1 (PPARA1).

PPARA1 selects between an address and discrete I/O function for the port A pins. Any set bit defines the corresponding pin to be an I/O pin, controlled by the port A data and data direction registers. Any cleared bit defines the corresponding pin to be an address bit as defined in the following register diagram. Bits set in this register override the configuration setting of PPARA2. The all-ones reset value of PPARA1 configures it as an input port. This register can be read or written at any time.

PPARA1 $015

7	6	5	4	3	2	1	0
PRTA7 (A31)	PRTA6 (A30)	PRTA5 (A29)	PRTA4 (A28)	PRTA3 (A27)	PRTA2 (A26)	PRTA1 (A25)	PRTA0 (A24)
RESET:							
1	1	1	1	1	1	1	1

Supervisor Only

4.3.5.2 PORT A PIN ASSIGNMENT REGISTER 2 (PPARA2).

PPARA2 selects between an address and $\overline{\text{IACKx}}$ function for the port A pins. Any set bit defines the corresponding pin to be an $\overline{\text{IACKx}}$ output pin. Any cleared bit defines the corresponding pin to be an address bit as defined in the register diagram. Any set bits in PPARA1 override the configuration set in PPARA2. Bit 0 has no function in this register because there is no level-zero interrupt. This register can be read or written at any time.

PPARA2 $017

7	6	5	4	3	2	1	0
PRTA7 (A31)	IACK6 (A30)	IACK5 (A29)	IACK4 (A28)	IACK3 (A27)	IACK2 (A26)	IACK1 (A25)	0
RESET:							
0	0	0	0	0	0	0	0

Supervisor Only

The $\overline{\text{IACKx}}$ signals are asserted if a bit in PPARA2 is set and the CPU32 services an external interrupt at the corresponding level. $\overline{\text{IACKx}}$ signals have the same timing as address signals.

4.3.5.3 PORT A DATA DIRECTION REGISTER (DDRA). DDRA controls the direction of the pin drivers when the pins are configured as I/O. Any set bit configures the corresponding pin as an output. Any cleared bit configures the corresponding pin as an input. This register affects only pins configured as discrete I/O. This register can be read or written at any time.

DDRA $013

7	6	5	4	3	2	1	0
DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0
RESET:							
0	0	0	0	0	0	0	0

Supervisor/User

4.3.5.4 PORT A DATA REGISTER (PORTA). PORTA affects only pins configured as discrete I/O. A write to the port A data register is stored in the internal data latch, and, if any port A pin is configured as an output, the value stored for that bit is driven on the pin. A read of the port A data register returns the value at the pin only if the pin is configured as discrete input. Otherwise, the value read is the value stored in the internal data latch. This register can be read or written at any time.

PORTA $011

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0
RESET:							
U	U	U	U	U	U	U	U

Supervisor/User

4.3.5.5 PORT B PIN ASSIGNMENT REGISTER (PPARB). PPARB controls the function of each port B pin. Any set bit defines the corresponding pin to be an interrupt request or chip select as defined in Table 4-5. Any cleared bit defines the corresponding pin to be an I/O pin (or chip select if the FIRQ bit of the MCR is zero) controlled by the port B data and data direction registers. The MODCK signal has no function after reset. This register can be read or written at any time.

PPARB $01F

7	6	5	4	3	2	1	0
PPARB7 (IRQ7)	PPARB6 (IRQ6)	PPARB5 (IRQ5)	PPARB4 (IRQ4)	PPARB3 (IRQ3)	PPARB2 (IRQ2)	PPARB1 (IRQ1)	PPARB0 (MODCK)
RESET:							
1	1	1	1	1	1	1	1

Supervisor Only

4.3.5.6 PORT B DATA DIRECTION REGISTER (DDRB). DDRB controls the direction of the pin drivers when the pins are configured as I/O. Any set bit configures the corresponding pin as an output. Any cleared bit configures the corresponding pin as an input. This register affects only pins configured as discrete I/O. This register can be read or written at any time.

DDRB $01D

	7	6	5	4	3	2	1	0
	DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0
RESET:	0	0	0	0	0	0	0	0

Supervisor/User

4.3.5.7 PORT B DATA REGISTER (PORTB, PORTB1). This is a single register that can be accessed at two different addresses. The port B data register affects only those pins configured as discrete I/O. A write is stored in the internal data latch, and, if any port B pin is configured as an output, the value stored for that bit is driven on the pin. A read of this register returns the value stored in the register only if the pin is configured as a discrete output. Otherwise, the value read is the value of the pin. This register can be read or written at any time.

PORTB, PORTB1 $019, 01B

	7	6	5	4	3	2	1	0
	P7	P6	P5	P4	P3	P2	P1	P0
RESET:	U	U	U	U	U	U	U	U

Supervisor/User

SECTION 5
CPU32

The CPU32, the first-generation instruction processing module of the M68300 Family, is based on the industry-standard MC68000 core processor. It has many features of the MC68010 and MC68020 as well as unique features suited for high-performance controller applications. The CPU32 provides a significant performance increase over the MC68000 CPU, yet maintains source code and binary code compatibility with the M68000 Family.

5.1 OVERVIEW

The CPU32 is designed to interface to the intermodule bus (IMB), allowing interaction with other IMB submodules. In this manner, integrated processors can be developed that contain useful peripherals on-chip. This integration provides high-speed accesses among the IMB submodules, increasing system performance.

Another advantage of the CPU32 is low power consumption. The CPU32 is implemented in high-speed complementary metal-oxide semiconductor (HCMOS) technology, providing low power use during normal operation. During periods of inactivity, the low-power stop (LPSTOP) instruction can be executed, shutting down the CPU32 and other IMB submodules, greatly reducing power consumption.

Ease of programming is an important consideration when using a microcontroller. The CPU32 instruction format reflects a predominate register-memory interaction philosophy. All data resources are available to all operations that require them. The programming model includes eight multifunction data registers and seven general-purpose addressing registers. The data registers readily support 8-bit (byte), 16-bit (word), and 32-bit (long-word) operand lengths for all operations. Address manipulation is supported by word and long-word operations. Although the program counter (PC) and stack pointers (SP) are special-purpose registers, they are also available for most data addressing activities. Ease of program checking and diagnosis is enhanced by trace and trap capabilities at the instruction level.

As controller applications become more complex and control programs become larger, high-level language (HLL) will become the system designer's choice in programming languages. HLL aids in the rapid development of complex algorithms with less error, and is readily portable. The CPU32 instruction set will efficiently support HLL.

5.1.1 Features

Features of the CPU32 are as follows:

- Fully Upward-Object-Code Compatible with M68000 Family
- Virtual Memory Implementation
- Loop Mode of Instruction Execution
- Fast Multiply, Divide, and Shift Instructions
- Fast Bus Interface with Dynamic Bus Port Sizing
- Improved Exception Handling for Controller Applications
- Additional Addressing Modes
  - — Scaled Index
  - — Address Register Indirect with Base Displacement and Index
  - — Expanded PC Relative Modes
  - — 32-Bit Branch Displacements
- Instruction Set Additions
  - — High-Precision Multiply and Divide
  - — Trap On Condition Codes
  - — Upper and Lower Bounds Checking
- Enhanced Breakpoint Instruction
- Trace on Change of Flow
- Table Lookup and Interpolate Instruction
- Low-Power Stop Instruction
- Hardware Breakpoint Signal, Background Mode
- Fully Static Implementation

A block diagram of the CPU32 is shown in Figure 5-1. The major blocks depicted operate in a highly independent fashion that maximizes concurrency of operation while managing the essential synchronization of instruction execution and bus operation. The bus controller loads instructions from the data bus into the decode unit. The sequencer and control unit provide overall chip control, managing the internal buses, registers, and functions of the execution unit.

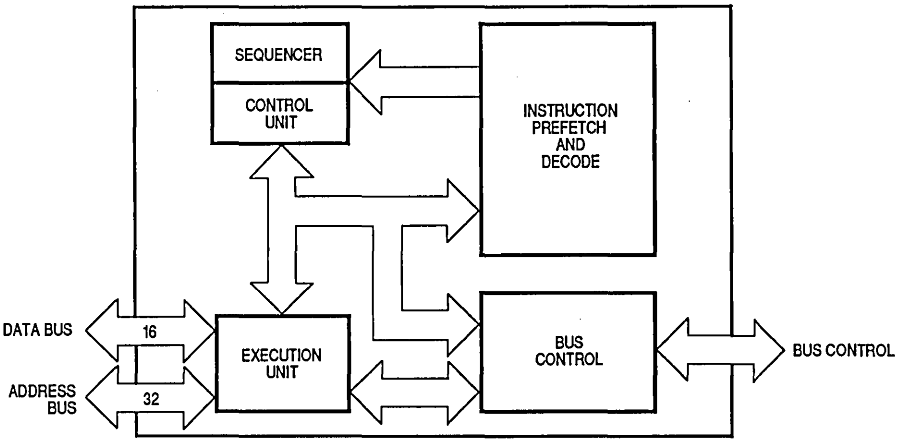


Figure 5-1. CPU32 Block Diagram

5.1.2 Virtual Memory

A system that supports virtual memory has a limited amount of high-speed physical memory that can be accessed directly by the processor and maintains an image of a much larger "virtual" memory on a secondary storage device. When the processor attempts to access a location in the virtual memory map that is not resident in physical memory, a page fault occurs. The access to that location is temporarily suspended while the necessary data is fetched from secondary storage and placed in physical memory. The suspended access is then restarted or continued. The CPU32 uses instruction restart, which requires that only a small portion of the internal machine state be saved. After correcting the fault, the machine state is restored, and the instruction is refetched and restarted. This process is completely transparent to the application program.

5.1.3 Loop Mode Instruction Execution

The CPU32 has several features that provide efficient execution of program loops. One of these features is the DBcc looping primitive instruction. To increase the performance of the CPU32, a loop mode has been added to the processor. The loop mode is used by any single-word instruction that does not change the program flow. Loop mode is implemented in conjunction with the DBcc instruction. Figure 5-2 shows the required form of an instruction loop for the processor to enter loop mode.

The loop mode is entered when the DBcc instruction is executed and the loop displacement is −4. Once in loop mode, the processor performs only the data cycles associated with the instruction and suppresses all instruction fetches. The termination condition and count are checked after each execution of the data operations of the looped instruction. The CPU32 automatically exits the loop mode on interrupts or other exceptions.

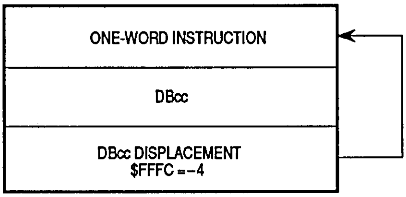


Figure 5-2. Loop Mode Instruction Sequence

5.1.4 Vector Base Register

The vector base register (VBR) contains the base address of the 1024-byte exception vector table, which consists of 256 exception vectors. Exception vectors contain the memory addresses of routines that begin execution at the completion of exception processing. These routines perform a series of operations appropriate for the corresponding exceptions. Because the exception vectors contain memory addresses, each consists of one long word, except for the reset vector. The reset vector consists of two long words: the address used to initialize the supervisor SP and the address used to initialize the PC.

The address of an interrupt exception vector is derived from an 8-bit vector number and the VBR. The vector numbers for some exceptions are obtained from an external device; other numbers are supplied automatically by the processor. The processor multiplies the vector number by four to calculate the vector offset, which is added to the VBR. The sum is the memory address of the vector. All exception vectors are located in supervisor data space, except the reset vector, which is located in supervisor program space. Only the initial reset vector is fixed in the processor's memory map; once initialization is complete, there are no fixed assignments. Since the VBR provides the base address of the vector table, the vector table can be located anywhere in memory; it can even be dynamically relocated for each task that is executed by an operating system. Refer to **5.6 EXCEPTION PROCESSING** for additional details.

31	0
VECTOR BASE REGISTER (VBR)	

5.1.5 Exception Handling

The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary internal copy of the status register is made, and the status register is set for exception processing. During the second step, the exception vector is determined; during the third step, the current processor context is saved. During the fourth step, a new context is obtained, and the processor then proceeds with instruction processing.

Exception processing saves the most volatile portion of the current context by pushing it on the supervisor stack. This context is organized in a format called the exception stack frame. This information always includes the status register and PC context of the processor when the exception occurred. To support generic handlers, the processor places the vector offset in the exception stack frame. The processor also marks the frame with a frame format. The format field allows the return-from-exception (RTE) instruction to identify what information is on the stack so that it may be properly restored.

5.1.6 Addressing Modes

Addressing in the CPU32 is register oriented. Most instructions allow the results of the specified operation to be placed either in a register or directly in memory; this flexibility eliminates the need for extra instructions to store register contents in memory.

The seven basic addressing modes are as follows:

- Register Direct
- Register Indirect
- Register Indirect with Index
- Program Counter Indirect with Displacement
- Program Counter Indirect with Index
- Absolute
- Immediate

Included in the register indirect addressing modes are the capabilities to postincrement, predecrement, and offset. The PC relative mode also has index and offset capabilities. In addition to these addressing modes, many instructions implicitly specify the use of the status register, SP and/or PC. Addressing is explained fully in **5.3 DATA ORGANIZATION AND ADDRESSING CAPABILITIES.**

5.1.7 Instruction Set

The instruction set of the CPU32 is very similar to that of the MC68020 (see Table 5-1). Two new instructions have been added to facilitate controller applications: low-power stop (LPSTOP) and table lookup and interpolate (TBL). The following M68020 instructions are *not implemented* on the CPU32:

BFxxx — Bit Field Instructions (BFCHG, BFCLR, BFEXTS, BFEXTU, BFFFO, BFINS, BFSET, BFTST)
CALLM, RTM — Call Module, Return Module
CAS, CAS2 — Compare and Set (Read-Modify-Write Instructions)
cpxxx — Coprocessor Instructions (cpBcc, cpDBcc, cpGEN, cpRESTORE, cpSAVE, cpScc, cpTRAPcc)
PACK, UNPK — Pack, Unpack BCD Instructions

The CPU32 traps on unimplemented instructions or illegal effective addressing modes, allowing user-supplied code to emulate unimplemented capabilities or to define special-purpose functions. However, Motorola reserves the right to use all currently unimplemented instruction operation codes for future M68000 core enhancements.

5.1.7.1 TABLE LOOKUP AND INTERPOLATE INSTRUCTIONS. To maximize throughput for real-time applications, reference data is often "particulated" and stored in memory for quick access. The storage of each data point would require an inordinate amount of memory. The table instruction requires only a sample of data points stored in the array, thus reducing memory requirements. Intermediate values are recovered with this instruction via linear interpolation. The results may be rounded by a round-to-nearest algorithm.

5.1.7.2 LOW-POWER STOP INSTRUCTION. In applications where power consumption is a consideration, the CPU32 forces the device into a low-power standby mode when immediate processing is not required. The low-power stop mode is entered by executing the LPSTOP instruction. The processor will remain in this mode until a user-specified (or higher) interrupt level or reset occurs.

Table 5-1. Instruction Set Summary

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BGND	Enter Background Mode
BKPT	Breakpoint
BRA	Branch Always
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CHK	Check Register Against Bounds
CHK2	Check Register against Upper and Lower Bounds
CLR	Clear Operand
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EXG	Exchange Registers
EXT, EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LPSTOP	Low-Power Stop
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVE CCR	Move Condition Code Register
MOVE SR	Move to/from Status Register
MOVE USP	Move User Stack Pointer
MOVEA	Move Address
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral Data
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS, MULS.L	Signed Multiply
MULU, MULU.L	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Ones Complement
OR	Logical Inclusive OR
ORI	Logical Inclusive OR Immediate
PEA	Push Effective Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTR	Return and Restore Codes
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract Address
SUBI	Subtract Immediate
SUBQ	Subtract Quick
SUBX	Subtract with Extend
SWAP	Swap Register Words
TAS	Test Operand and Set
TBLS,TBLSN	Table Lookup and Interpolate (Signed)
TBLU, TBLUN	Table Lookup and Interpolate (Unsigned)
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink

5.1.8 Processing States

The processor is always in one of four processing states: normal, exception, halted, or background. The normal processing state is that associated with instruction execution; the bus is used to fetch instructions and operands and to store results. The exception processing state is associated with interrupts, trap instructions, tracing, and other exception conditions. The exception may be internally generated explicitly by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, a bus error, or a reset. The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. The background processing state is initiated by breakpoints, execution of special instructions, or a double bus fault. Background processing allows interactive debugging of the system via a simple serial interface. Refer to **5.5 PROCESSING STATES** for details.

5.1.9 Privilege States

The processor operates at one of two levels of privilege — user or supervisor. The supervisor level has higher privileges than the user level. Not all instructions are permitted to execute in the lower privileged user level, but all instructions are available at the supervisor level. This scheme allows the supervisor to protect system resources from uncontrolled access. The processor uses the privilege level indicated by the S bit in the status register to select either the user or supervisor privilege level and either the user stack pointer (USP) or supervisor stack pointer (SSP) for stack operations.

5.2 ARCHITECTURE SUMMARY

The CPU32 architecture includes several important features that provide both power and versatility to the user. The CPU32 is source and object code compatible with the M68000 and MC68010. All user state programs can be executed unchanged.

The major CPU32 features are as follows:

- 32-Bit Internal Data Path and Arithmetic Hardware
- 32-Bit Address Bus Supported by 32-Bit Calculations
- Rich Instruction Set
- Eight 32-Bit General-Purpose Data Registers

- Seven 32-Bit General-Purpose Address Registers
- Separate User and Supervisor Stack Pointers
- Separate User and Supervisor State Address Spaces
- Separate Program and Data Address Spaces
- Many Data Types
- Flexible Addressing Modes
- Full Interrupt Processing
- Expansion Capability

5.2.1 Programming Model

The programming model of the CPU32 consists of two groups of registers: user model and supervisor model that correspond to the user and supervisor privilege levels. Executing at the user privilege level, user programs can only use the registers of the user model. Executing at the supervisor level, system software can use both the control registers of the supervisor level and the user model registers to perform supervisor functions.

As shown in the programming models (see Figures 5-3 and 5-4), the CPU32 has 16 32-bit general-purpose registers, a 32-bit program counter, one 32-bit supervisor stack pointer, a 16-bit status register, two 3-bit alternate function code registers, and a 32-bit vector base register. The user programming model remains unchanged from previous M68000 Family microprocessors. The supervisor programming model, which supplements the user programming model, is used exclusively by the CPU32 system programmers who utilize the supervisor privilege level to implement sensitive operating system functions. The supervisor programming model contains all the controls to access and enable the special features of the CPU32. All application software, written to run at the nonprivileged user level, migrates to the CPU32 from any M68000 platform without modification.

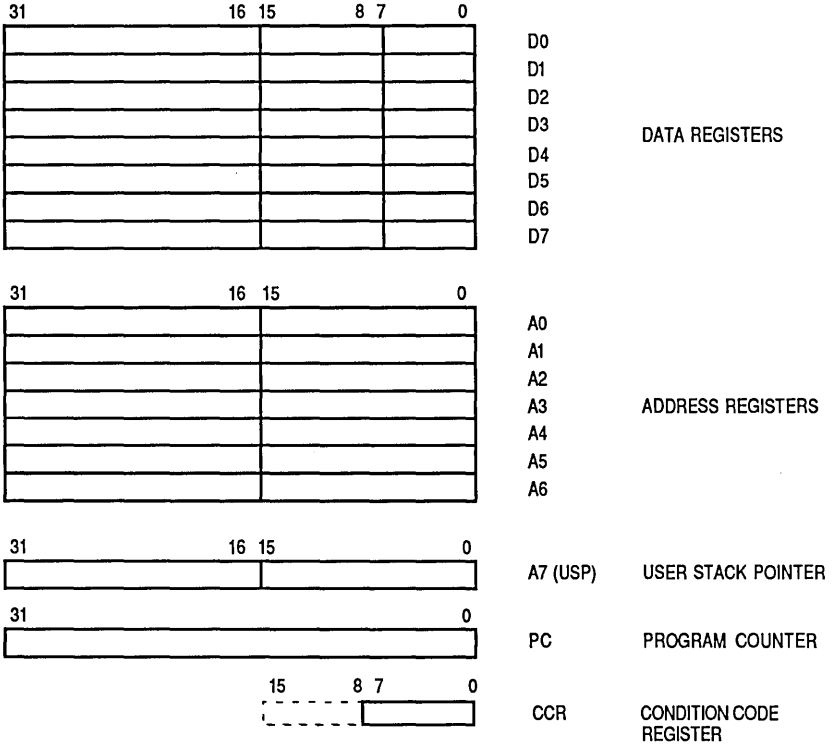


Figure 5-3. User Programming Model

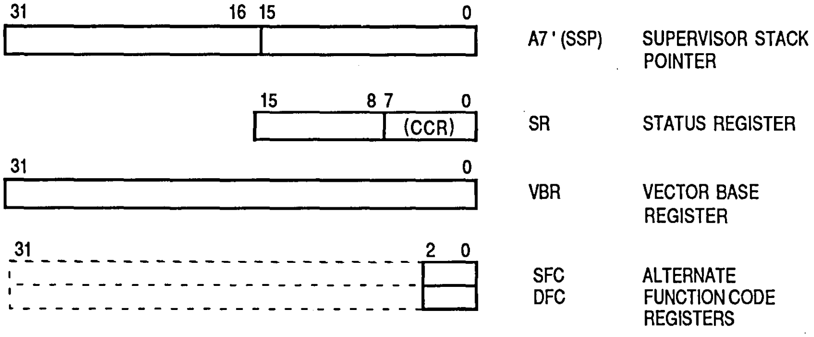


Figure 5-4. Supervisor Programming Model Supplement

5.2.2 Registers

Registers D7–D0 are used as data registers for bit, byte (8-bit), word (16-bit), long-word (32-bit), and quad-word (64-bit) operations. Registers A6–A0 and the user and supervisor stack pointers are address registers that may be used as software stack pointers or base address registers. Register A7 (shown as A7 and A7′ in Figures 5-3 and 5-4) is a register designation that applies to the user stack pointer in the user privilege level and to the supervisor stack pointer in the supervisor privilege level. In addition, the address registers may be used for word and long-word operations. All 16 general-purpose registers (D7–D0, A7–A0) may be used as index registers.

The program counter (PC) contains the address of the next instruction to be executed by the CPU32. During instruction execution and exception processing, the processor automatically increments the contents of the PC or places a new value in the PC, as appropriate.

The status register (SR) stores the processor status (see Figure 5-5). The SR contains the condition codes that reflect the results of a previous operation and can be used for conditional instruction execution in a program. The condition codes are extend (X), negative (N), zero (Z), overflow (V), and carry (C). The user byte containing the condition codes is the only portion of the SR information available in the user privilege level; it is referenced as the condition code register (CCR) in user programs. In the supervisor privilege level, software can access the full status register, including the interrupt priority mask (three bits), as well as additional control bits. These bits put the processor in one of two trace modes (T1, T0) and in user or supervisor privilege level (S).

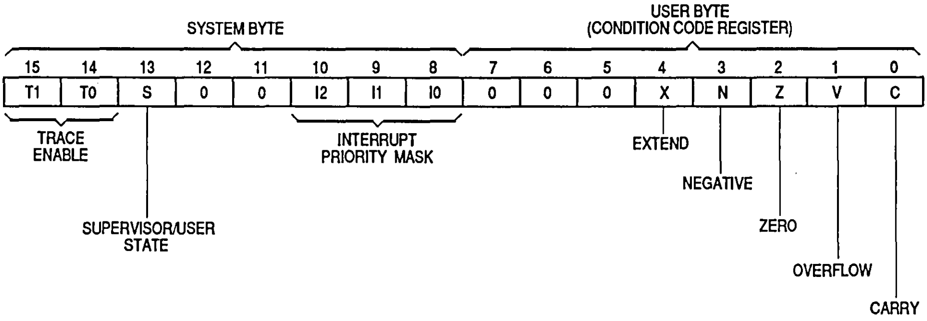


Figure 5-5. Status Register

The vector base register (VBR) contains the base address of the exception vector table in memory. The displacement of an exception vector is added to the value in this register to access the vector table.

Alternate function code registers (SFC and DFC) contain 3-bit function codes. Function codes can be considered extensions of the 32-bit linear address that optionally provide as many as eight 4-Gbyte address spaces. Function codes are automatically generated by the processor to select address spaces for data and program at the user and supervisor privilege levels and to select a CPU address space used for processor functions (such as breakpoint and interrupt acknowledge cycles). Registers SFC and DFC are used by the MOVE instructions to explicitly specify the function codes of the memory address.

5.2.3 Data Types

Six basic data types are supported:

- Single Bits
- Binary-Coded Decimal (BCD) Digits
- Byte Integers (8 bits)
- Word Integers (16 bits)
- Long-word Integers (32 bits)
- Quad-Word Integers (64 bits)

5.2.3.1 ORGANIZATION IN REGISTERS. The eight data registers can store data operands of 1, 8, 16, 32, and 64 bits and addresses of 16 or 32 bits. The seven address registers and the two stack pointers are used for address operands of 16 or 32 bits. The PC is 32 bits wide.

5.2.3.1.1 Data Registers. Each data register is 32 bits wide. Byte operands occupy the low-order 8 bits, word operands, the low-order 16 bits, and long-word operands, the entire 32 bits. When a data register is used as either a source or destination operand, only the appropriate low-order byte or word (in byte or word operations, respectively) is used or changed; the remaining high-order portion is neither used nor changed. The least significant bit (LSB) of a long-word integer is addressed as bit 0, and the most significant bit (MSB) is addressed as bit 31. Figure 5-6 shows the organization of various types of data in the data registers.

Quad-word data consists of two long words: for example, the product of 32-bit multiply or the quotient of 32-bit divide operations (signed and unsigned). Quad words may be organized in any two data registers without restrictions on order or pairing. There are no explicit instructions for the management of this data type; however, the MOVEM instruction can be used to move a quad word into or out of the registers.

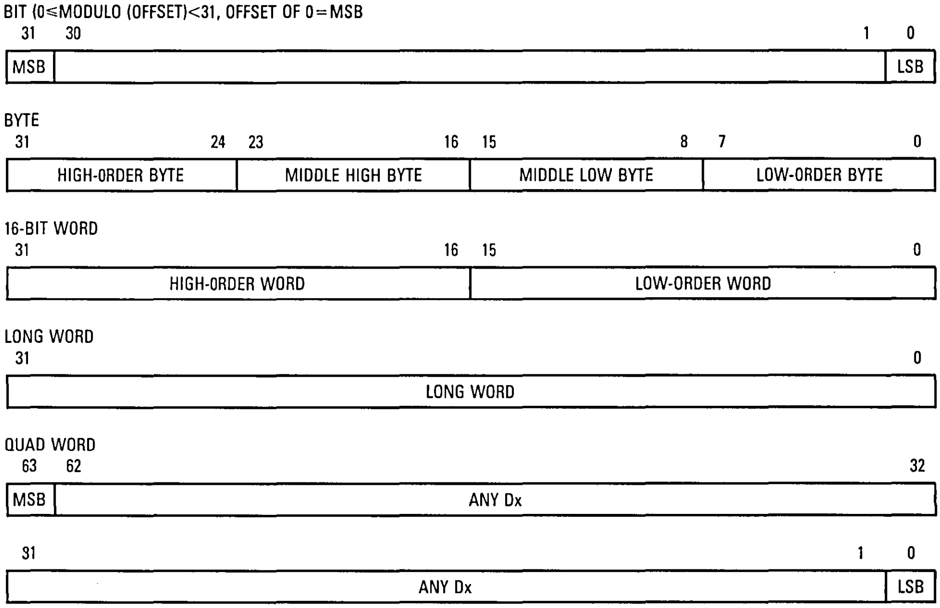


Figure 5-6. Data Organization in Data Registers

BCD data represents decimal numbers in binary form. Although many BCD codes have been devised, the BCD instructions of the M68000 Family support formats in which the four LSBs consist of a binary number having the numeric value of the corresponding decimal number. In this BCD format, a byte contains one digit; the four LSBs contain the binary value, and the four MSBs are undefined. ABCD, SBCD, and NBCD operate on two BCD digits which are manually packed into a single byte.

5.2.3.1.2 Address Register. Each address register and stack pointer is 32 bits wide and holds a 32-bit address. Address registers cannot be used for byte-sized operands. Therefore, when an address register is used as a source operand, either the low-order word or the entire long-word operand is used, depending upon the operation size. When an address register is used as the destination operand, the entire register is affected, regardless of the operation size. If the source operand is a word size, it is first sign extended to 32 bits, and then used in the operation to an address register destination. Address registers are used primarily for addresses and to support address computation. The instruction set includes instructions that add to, subtract from, compare, and move the contents of address registers. Figure 5-7 shows the organization of addresses in address registers.

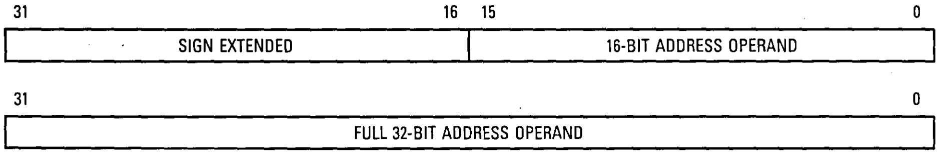


Figure 5-7. Address Organization in Address Registers

5.2.3.1.3 Control Registers. The control registers described in this section contain control information for supervisor functions and vary in size. With the exception of the user portion of the SR (CCR), they are accessed only by instructions at the supervisor privilege level.

The SR shown in Figure 5-5 is 16 bits wide. Only 11 bits of the SR are defined; all undefined values are reserved by Motorola for future definition. The undefined bits are read as zeros and should be written as zeros for future compatibility. The lower byte of the SR is the CCR. Operations to the CCR can be performed at the supervisor or user privilege level. All operations to the SR and CCR are word-size operations, but for all CCR operations, the upper byte is read as all zeros and is ignored when written, regardless of privilege level.

The alternate function code registers (SFC and DFC) are 32-bit registers with only bits 2–0 implemented that contain the address space values (FC2–FC0) for the read or write operand of the MOVES instruction. The MOVEC instruction is used to transfer values to and from the alternate function code registers. These are long-word transfers; the upper 29 bits are read as zeros and are ignored when written.

5.2.3.2 Organization in Memory. Memory is organized on a byte-addressable basis in which lower addresses correspond to higher order bytes. The address, N, of a long-word data item corresponds to the address of the most significant byte of the highest order word. The lower order word is located at address N+2, leaving the least significant byte at address N+3 (see Figure 5-3). The CPU32 requires long-word and word data as well as instruction words to be aligned on word boundaries (see Figure 5-8). Data misalignment is not supported.

BIT DATA
1 BYTE = 8 BITS

INTEGER DATA
1 BYTE = 8 BITS

15		8	7		0
MSB	BYTE 0	LSB	MSB	BYTE 1	LSB
MSB	BYTE 2	LSB	MSB	BYTE 3	LSB

WORD = 16 BITS

15		0
MSB	WORD 0	LSB
MSB	WORD 1	LSB
MSB	WORD 2	LSB

LONG WORD = 32 BITS

15		0
MSB	LONG WORD 0 (HIGH ORDER)	
	LONG WORD 0 (LOW ORDER)	LSB
MSB	LONG WORD 1 (HIGH ORDER)	
	LONG WORD 1 (LOW ORDER)	LSB
MSB	LONG WORD 2 (HIGH ORDER)	
	LONG WORD 2 (LOW ORDER)	LSB

ADDRESS
1 ADDRESS = 32 BITS

15		0
MSB	ADDRESS 0 (HIGH ORDER)	
	ADDRESS 0 (LOW ORDER)	LSB
MSB	ADDRESS 1 (HIGH ORDER)	
	ADDRESS 1 (LOW ORDER)	LSB
MSB	ADDRESS 2 (HIGH ORDER)	
	ADDRESS 2 (LOW ORDER)	LSB

Most Significant Bit
LSB = Least Significant Bit

DECIMAL DATA
BCD DIGITS = 1 BYTE

15		12	11		8	7		4	3		0
MSD	BCD0			BCD1	LSD	MSD	BCD2			BCD3	LSD
MSD	BCD4			BCD5	LSD	MSD	BCD6			BCD7	LSD

MSD = Most Significant Digit
LSD = Least Significant Digit

Figure 5-8. Memory Operand Addressing

5.3 DATA ORGANIZATION AND ADDRESSING CAPABILITIES

The addressing mode of an instruction can specify the value of an operand (with an immediate operand), a register that contains the operand (with the register direct addressing mode), or how the effective address (EA) of an operand in memory is derived.

Figure 5-9 shows the general format of the single EA instruction operation word. The EA field specifies the addressing mode for an operand that can use one of the numerous defined modes. The designation is composed of two 3-bit fields: mode and register. The value in the mode field selects one mode or a set of addressing modes. The register field specifies a register for the mode or a submode for modes that do not use registers.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	EFFECTIVE ADDRESS					
										MODE			REGISTER		

Figure 5-9. Single EA Instruction Operation Word

Many instructions imply the addressing mode for one of the operands. The formats of these instructions include appropriate fields for operands that use only one addressing mode.

The EA field may require additional information to fully specify the operand address. This additional information, called the EA extension, is contained in an additional word or words and is considered part of the instruction. Refer to **5.3.4.4 EFFECTIVE ADDRESS ENCODING SUMMARY** for a description of the extension word formats.

When the addressing mode uses a register, the register field of the operation word specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

5.3.1 Program and Data References

An M68000 Family processor separates memory references into two classes, each with a complete logical address range. The first class is program references, which includes primarily references to opcodes and extension words. The other class is data references. Operand reads are from the data space with two exceptions: 1) immediate operands embedded in the instruction stream and 2) operands addressed relative to the current program counter. Operands satisfying either of these two exceptions are classified as program space references. All operand writes are to data space.

5.3.2 Notation Conventions

EA—Effective address
An—Address register n
Example: A3 is address register 3
Dn—Data register n
Example: D5 is data register 5
Rn—Any register, data or address
Xn.SIZE*SCALE —
Index register n (data or address),
Index size (W for word, L for long word),
Scale factor (1, 2, 4, or 8 for none, word, long-word, or quad-word scaling)
PC—Program counter
SR—Status register
SP—Stack pointer
CCR—Condition code register
USP—User stack pointer
SSP—Supervisor stack pointer
d_n—Displacement value, n bits wide
bd—Base displacement
L—Long-word size
W—Word size
B—Byte size
()—Identify an indirect address in a register

5

5.3.3 Implicit Reference

Some instructions make implicit reference to the program counter, the system stack pointer, the user stack pointer, the supervisor stack pointer, or the status register. Table 5-2 enumerates these instructions and the registers involved:

Table 5-2. Implicit Reference Instructions

Instruction	Implicit Registers
ANDI to CCR	SR
ANDI to SR	SR
BRA	PC
BSR	PC, SP
CHK (exception)	SSP, SR
CHK2 (exception)	SSP, SR
DBcc	PC
DIVS (exception)	SSP, SR
DIVU (exception)	SSP, SR
EORI to CCR	SR
EORI to SR	SR
JMP	PC
JSR	PC, SP
LINK	SP
LPSTOP	SR
MOVE CCR	SR
MOVE SR	SR
MOVE USP	USP
ORI to CCR	SR
ORI to SR	SR
PEA	SP
RTD	PC, SP
RTE	PC, SP, SR
RTR	PC, SP, SR
RTS	PC, SP
STOP	SR
TRAP (exception)	SSP, SR
TRAPV (exception)	SSP, SR
UNLK	SP

5.3.4 Effective Address

Most instructions specify the location of an operand by a field in the operation word called an EA field. The EA is composed of two 3-bit subfields: mode specification field and register specification field. Each of the address modes is selected by a particular value in the mode specification subfield of the EA. The EA field may require further information to fully specify the operand. This information, called the EA extension, is in a following word or words and is considered part of the instruction (see **5.3.1 Program and Data References**).

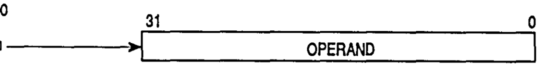
5.3.4.1 REGISTER DIRECT MODE.

These EA modes specify that the operand is in one of the 16 multifunction registers.

5.3.4.1.1 Data Register Direct.

In the data register direct mode, the operand is in the data register specified by the EA register field.

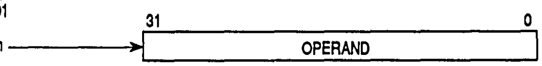
GENERATION:	EA = Dn
ASSEMBLER SYNTAX:	Dn
MODE:	000
REGISTER:	n
DATA REGISTER:	Dn
NUMBER OF EXTENSION WORDS:	0



5.3.4.1.2 Address Register Direct.

In the address register direct mode, the operand is in the address register specified by the EA register field.

GENERATION:	EA = An
ASSEMBLER SYNTAX:	An
MODE:	001
REGISTER:	n
DATA REGISTER:	An
NUMBER OF EXTENSION WORDS:	0



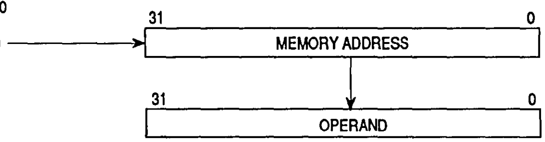
5.3.4.2 MEMORY ADDRESSING MODES.

These EA modes specify the address of the memory operand.

5.3.4.2.1 Address Register Indirect.

In the address register indirect mode, the operand is in memory, and the address of the operand is in the address register specified by the register field.

GENERATION:	EA = (An)
ASSEMBLER SYNTAX:	(An)
MODE:	010
REGISTER:	n
ADDRESS REGISTER:	An
MEMORY ADDRESS:	
NUMBER OF EXTENSION WORDS:	0

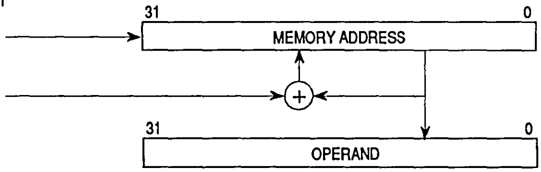


5.3.4.2.2 Address Register Indirect with Postincrement.

In the address register indirect with postincrement mode, the operand is in memory, and the address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four, depending on the size of the operand: byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer aligned to a word boundary.

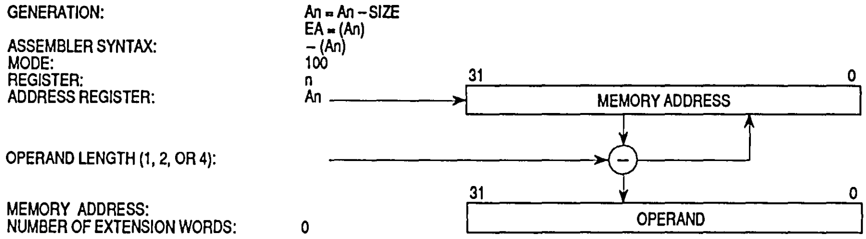
GENERATION:	EA = (An)
	An = An + SIZE

ASSEMBLER SYNTAX:	(An) +
MODE:	011
REGISTER:	n
ADDRESS REGISTER:	An
OPERAND LENGTH (1, 2, OR 4):	
MEMORY ADDRESS:	
NUMBER OF EXTENSION WORDS:	0



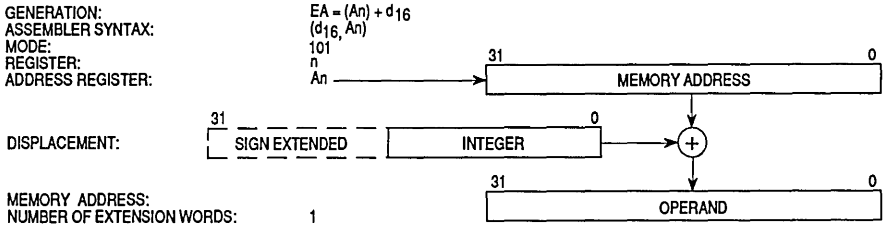
5.3.4.2.3 Address Register Indirect with Predecrement.

In the address register indirect with predecrement mode, the operand is in memory, and the address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four, depending on the operand size: byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer aligned to a word boundary.



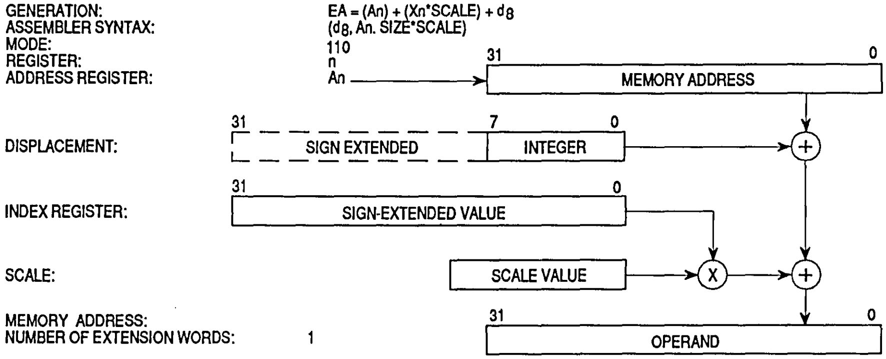
GENERATION: An = An − SIZE
EA = (An)
ASSEMBLER SYNTAX: − (An)
MODE: 100
REGISTER: n
ADDRESS REGISTER: An
OPERAND LENGTH (1, 2, OR 4):
MEMORY ADDRESS:
NUMBER OF EXTENSION WORDS: 0

5.3.4.2.4 Address Register Indirect with Displacement. In the address register indirect with displacement mode, the operand is in memory. The address of the operand is the sum of the address in the address register plus the sign-extended 16-bit displacement integer in the extension word. Displacements are always sign extended to 32 bits before being used in EA calculations.



GENERATION: EA = (An) + d_{16}
ASSEMBLER SYNTAX: (d_{16}, An)
MODE: 101
REGISTER: n
ADDRESS REGISTER: An
DISPLACEMENT:
MEMORY ADDRESS:
NUMBER OF EXTENSION WORDS: 1

5.3.4.2.5 Address Register Indirect with Index (8-Bit Displacement). This mode requires one extension word that contains the index register indicator and an 8-bit displacement. The index register indicator includes size and scale information. In this mode, the operand is in memory. The address of the operand is the sum of the contents of the address register, the sign-extended displacement value in the low-order eight bits of the extension word, and the sign-extended contents of the index register (possibly scaled). The user must specify the displacement, the address register, and the index register in this mode.



GENERATION: EA = (An) + (Xn*SCALE) + d_8
ASSEMBLER SYNTAX: (d_8, An. SIZE*SCALE)
MODE: 110
REGISTER: n
ADDRESS REGISTER: An
DISPLACEMENT:
INDEX REGISTER:
SCALE:
MEMORY ADDRESS:
NUMBER OF EXTENSION WORDS: 1

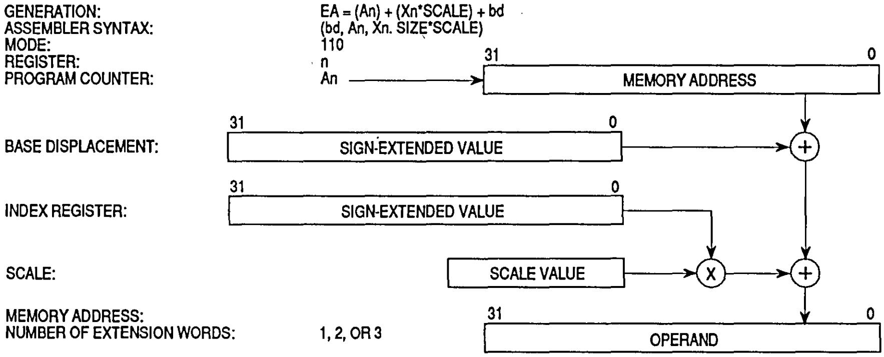
This address mode can have either of two different formats of extension. The brief format requires one word of extension and provides fast indexed addressing; the full format provides a number of options in size of displacements. Both formats use an index operand. The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low-order eight bits of the extension word, and the index operand. The reference is classed as a data reference, except for the JMP and JSR instructions. The index operand is specified "Ri.sz*scl".

"Ri" specifies a general data or address register to be used as the index register. The index operand is derived from the index register. The index register is a data register if bit [15] = 0 in the first extension word and is an address register if bit [15] = 1. The register number of the index register is given by bits [14:12] of the extension word.

The term "sz" refers to index size and may be either:"W" or "L". Index size is given by bit [11] of the extension word; if bit [11] = 0, the index value is the sign-extended low-order word integer of the index register ("W"); if bit [11] = 1, the index value is the long-word integer in the index register ("L").

The term "scl" refers to index scale selection and may be 1, 2, 4, or 8. The index value is scaled according to the scaling selection in bits [10:9] to derive the index operand. Scale selections 00, 01, 10, or 11 select scaling of the index value by 1, 2, 4, or 8, respectively.

5.3.4.2.6 Address Register Indirect with Index (Base Displacement). This mode requires an index register indicator and an optional 16- or 32-bit sign-extended base displacement. The index register indicator includes size and scale information. In this mode, the operand is in memory. The address of the operand is the sum of the contents of the address register, the scaled contents of the sign-extended index register, and the base displacement.



GENERATION: EA = (An) + (Xn*SCALE) + bd
ASSEMBLER SYNTAX: (bd, An, Xn. SIZE*SCALE)
MODE: 110
REGISTER: n
PROGRAM COUNTER: An

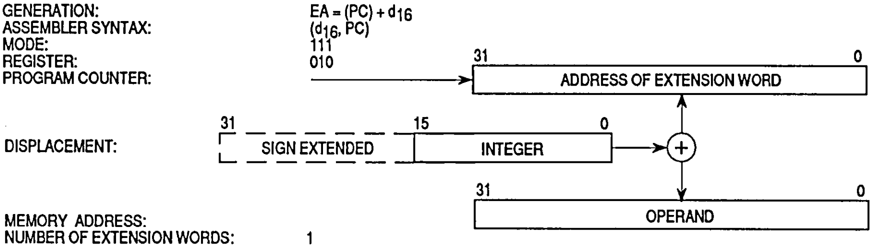
NUMBER OF EXTENSION WORDS: 1, 2, OR 3

5.3.4.3 SPECIAL ADDRESSING MODES.

These special addressing modes do not use the register field to specify a register number but rather to specify a submode.

5.3.4.3.1 Program Counter Indirect with Displacement.

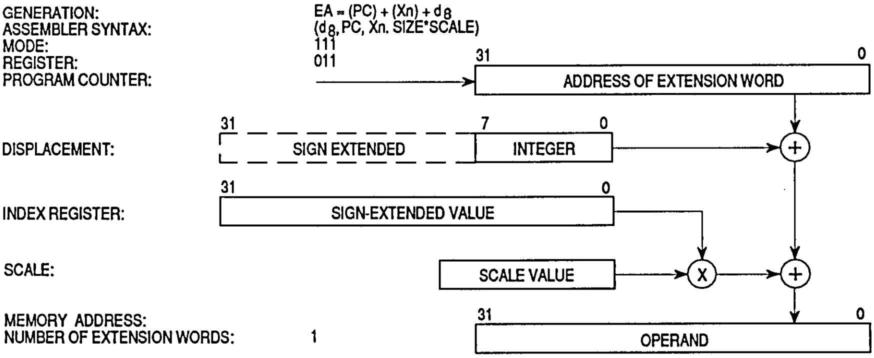
In this mode, the operand is in memory. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference is a program space reference and is only allowed for read accesses.



GENERATION: EA = (PC) + d_{16}
ASSEMBLER SYNTAX: (d_{16}, PC)
MODE: 111
REGISTER: 010
PROGRAM COUNTER:
DISPLACEMENT:
MEMORY ADDRESS:
NUMBER OF EXTENSION WORDS: 1

5.3.4.3.2 Program Counter Indirect with Index (8-Bit Displacement).

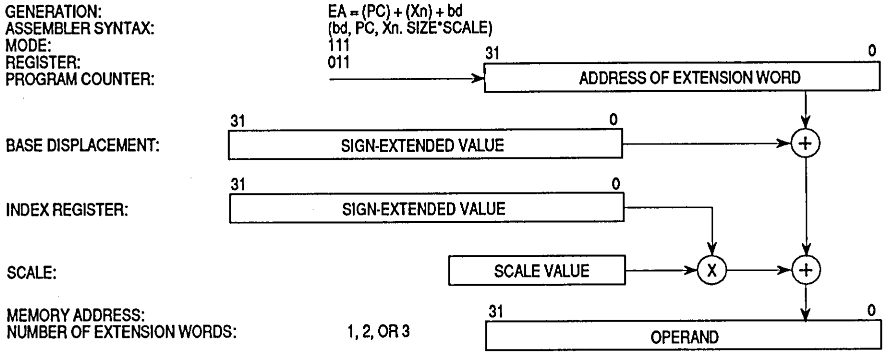
This mode is similar to the mode described in **5.3.4.2.5 Address Register Indirect with Index (8-Bit Displacement)**, but the program counter is used as the base register. The operand is in memory. The address of the operand is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the sized, scaled, and sign-extended index operand. The value in the program counter is the address of the extension word. This reference is a program space reference and is only allowed for reads. The user must include the displacement, the program counter, and the index register when specifying this addressing mode.



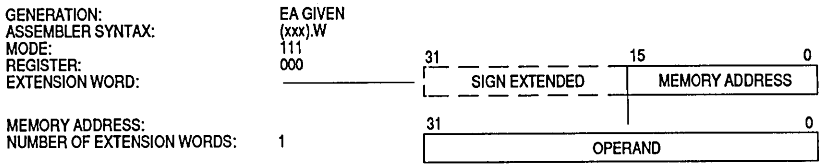
GENERATION: EA = (PC) + (Xn) + d_8
ASSEMBLER SYNTAX: (d_8, PC, Xn. SIZE*SCALE)
MODE: 111
REGISTER: 011
PROGRAM COUNTER:
DISPLACEMENT:
INDEX REGISTER:
SCALE:
MEMORY ADDRESS:
NUMBER OF EXTENSION WORDS: 1

5.3.4.3.3 Program Counter Indirect with Index (Base Displacement). This mode is similar to the mode described in **5.3.4.2.6 Address Register Indirect with Index (Base Displacement)**, but the program counter is used as the base register. It requires an index register indicator and an optional 16- or 32-bit sign-extended base displacement. The operand is in memory. The address of the operand is the sum of the contents of the program counter, the scaled contents of the sign-extended index register, and the base displacement. The value of the program counter is the address of the first extension word. The reference is a program space reference and is only allowed for read accesses.

In this mode, the program counter, the index register, and the displacement are all optional. However, the user must supply the assembler notation "ZPC" (zero value is taken for the program counter) to indicate that the program counter is not used. This scheme allows the user to access the program space without using the program counter in calculating the EA. The user can access the program space with a data register indirect access by placing ZPC in the instruction and specifying a data register (Dn) as the index register.



5.3.4.3.4 Absolute Short Address. In this addressing mode, the operand is in memory, and the address of the operand is in the extension word. The 16-bit address is sign extended to 32 bits before it is used.

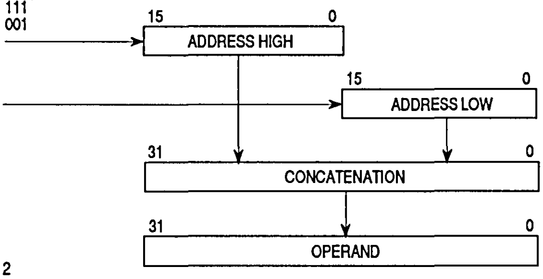


5.3.4.3.5 Absolute Long Address. In this mode, the operand is in memory, and the address of the operand occupies the two extension words following the instruction word in memory. The first extension word contains the high-order part of the address; the low-order part of the address is the second extension word.

GENERATION: EA GIVEN
ASSEMBLER SYNTAX: (xxx).L
MODE: 111
REGISTER: 001
FI RST EXTENSION WORD:

SECOND EXTENSION WORD:

MEMORY ADDRESS:
NUMBER OF EXTENSION WORDS: 2



5.3.4.3.6 Immediate Data. In this addressing mode, the operand is in one or two extension words:

Byte Operation

The operand is in the low-order byte of the extension word.

Word Operation

The operand is in the extension word.

Long-word Operation

The high-order 16 bits of the operand are in the first extension word; the low-order 16 bits are in the second extension word.

GENERATION:	OPERAND GIVEN
ASSEMBLER SYNTAX:	#XXX
MODE:	111
REGISTER:	100
NUMBER OF EXTENSION WORDS:	1 OR 2

5.3.4.4 EFFECTIVE ADDRESS ENCODING SUMMARY. Most of the addressing modes use one of the three formats shown in Figure 5-10. The single EA instruction is in the format of the instruction word. The encoding of the mode field of this word selects the addressing mode. The register field contains the general register number or a value that selects the addressing mode when the mode field contains '111'. Some indexed or indirect modes use the instruction word followed by the brief format extension word. Other indexed or indirect modes consist of the instruction word and the full format of extension words. The longest instruction for the CPU32 contains six extension words. It is a MOVE instruction with full format extension words for both the source and destination EAs and with 32-bit base displacements for both addresses.

SINGLE EA INSTRUCTION FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	EFFECTIVE ADDRESS MODE			REGISTER		

BRIEF FORMAT EXTENSION WORD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	SCALE		0	DISPLACEMENT							

FULL FORMAT EXTENSION WORD(S)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	SCALE		1	BS	IS	BD SIZE		0	I/IS		
BASE DISPLACEMENT (0, 1, OR 2 WORDS)															

Field	**Definition**	**Field**	**Definition**
Instruction Register	General Register Number	BS	Base Register Suppress
Extensions Register	Index Register Number		0=Base Register Added
D/A	Index Register Type		1=Base Register Suppressed
	0=Dn	IS	Index Suppressed
	1=An		0=Evaluate and Add Index Operand
W/L	Word/Long-Word Index Size		1=Suppress Index Operand
	0=Sign-Extended Word	BD SIZE	Base Displacement Size
	1=Long Word		00=Reserved
Scale	Scale Factor		01=Null Displacement
	00=1		10=Word Displacement
	01=2		11=Long Word Displacement
	10=4	I/IS*	Index/Indirect Selection
	11=8		Indirect and Indexing Operand Determined in Conjunction with Bit 6, Index Suppress

**Memory indirect addressing will cause illegal instruction trap; must be=000 if IS=1.

Figure 5-10. EA Specification Formats

Grouped according to the use of the mode, EA modes can be classified as follows:

Data A data addressing EA mode is one that refers to data operands.

Memory A memory addressing EA mode is one that refers to memory operands.

Alterable An alterable addressing EA mode is one that refers to alterable (writable) operands.

Control A control addressing EA mode is one that refers to memory operands without an associated size.

These categories are sometimes combined, forming new categories that are more restrictive. Two combined classifications are alterable memory or alterable data. The former category refers to those addressing modes that are both alterable and memory addresses; the latter category refers to addressing modes that are both alterable and data addresses. Table 5-3 lists the categories to which each of the EA modes belong.

Table 5-3. EA Mode Categories

Address Modes	Mode	Register	Data	Memory	Control	Alterable	Assembler Syntax
Data Register Direct	000	reg. no.	X	—	—	X	Dn
Address Register Direct	001	reg. no.	—	—	—	X	An
Address Register Indirect	010	reg. no.	X	X	X	X	(An)
Address Register Indirect with Postincrement	011	reg. no.	X	X	—	X	(An)+
Address Register Indirect with Predecrement	100	reg. no.	X	X	—	X	–(An)
Address Register Indirect with Displacement	101	reg. no.	X	X	X	X	(d_{16},An)
Address Register Indirect with Index (8-Bit Displacement)	110	reg. no.	X	X	X	X	(d_8,An,Xn)
Address Register Indirect with Index (Base Displacement)	110	reg. no.	X	X	X	X	(bd,An,Xn)
Absolute Short	111	000	X	X	X	X	(xxx).W
Absolute Long	111	001	X	X	X	X	(xxx).L
Program Counter Indirect with Displacement	111	010	X	—	X	X	(d_{16},PC)
Program Counter Indirect with Index (8-Bit Displacement)	111	011	X	—	X	X	(d_8,PC,Xn)
Program Counter Indirect with Index (Base Displacement)	111	011	X	—	X	X	(bd,PC,Xn)
Immediate	111	100	X	X	—	—	#⟨data⟩

5.3.5 Programming View of Addressing Modes

Extensions to the indexed addressing modes, indirection, and full 32-bit displacements provide additional programming capabilities for the CPU32, MC68020, MC68030, and MC68040. The following paragraphs describe addressing techniques that exploit these capabilities and summarize the addressing modes from a programming point of view.

5.3.5.1 ADDRESSING CAPABILITIES. In the CPU32, MC68020, MC68030, and MC68040, setting the base register suppress (BS) bit in the full format extension word (see Figure 5-10) suppresses use of the base address register in calculating the EA, allowing any index register to be used in place of the base register. In this way, a data register indirect form (Dn) can be used since any of the data registers can be an index register. Additionally, since either a data register or an address register can be used, the mode could be called register indirect (Rn). This addressing mode is an extension to the M68000 Family because the CPU32, MC68020, MC68030, and MC68040 can use both the data registers and the address registers to address memory. The capability of specifying the size and scale of an index register (Xn.SIZE*SCALE) in these modes provides additional addressing flexibility. Using the SIZE parameter, either the entire contents of the index register can be used, or the least significant word can be sign extended to provide a 32-bit index value (refer to Figure 5-11).

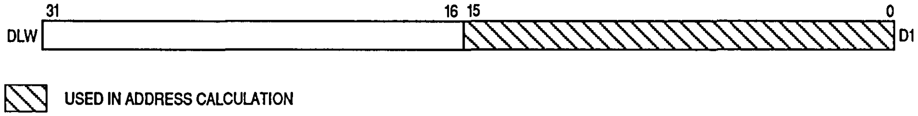


Figure 5-11. SIZE in the Index Selection

For the CPU32, MC68020, MC68030, and MC68040, the register indirect modes can be extended further. Since displacements can be 32 bits wide, they can represent absolute addresses or the results of expressions that contain absolute addresses. This scheme allows the general register indirect form to be (bd,Rn) or (bd,An,Rn) when the base register is not suppressed. Thus, an absolute address can be directly indexed by one or two registers (refer to Figure 5-12).

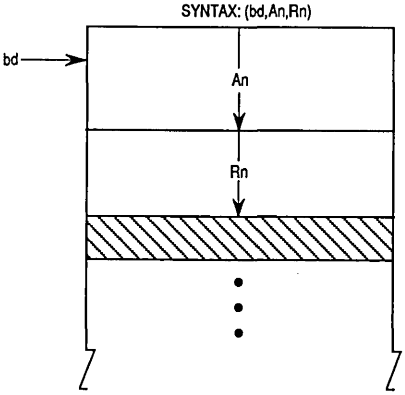
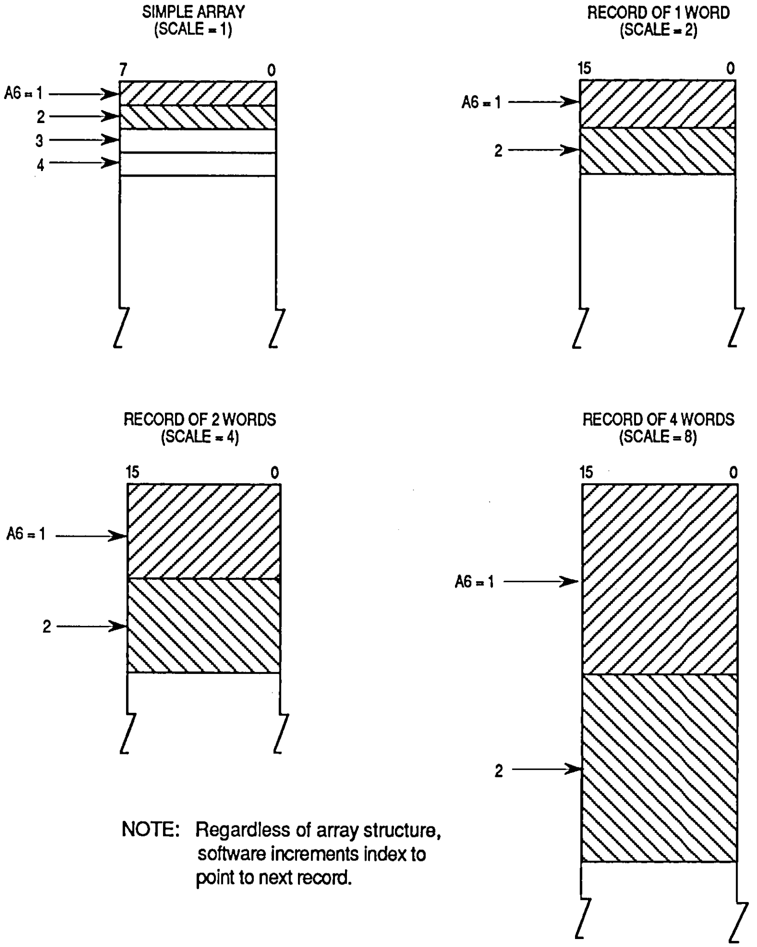


Figure 5-12. Absolute Address with Indexes

The indirect suppressed index register mode (see Figure 5-13) uses the contents of register An as an index to the pointer located at the address specified by the displacement. The actual data item is at the address in the selected pointer.

```
SYNTAX: MOVE.W (A5,A6.L*SCALE),(A7)
  WHERE:
    A5 = ADDRESS OF ARRAY STRUCTURE
    A6 = INDEX NUMBER OF ARRAY ITEM
    A7 = STACK POINTER
```



NOTE: Regardless of array structure, software increments index to point to next record.

Figure 5-13. Addressing Array Items

Scaling provides an optional shifting of the value in an index register to the left by zero, one, two, or three bits before using it in the EA calculation (the actual value in the index register remains unchanged). This is equivalent to multiplying the register by one, two, four, or eight for direct subscripting into an array of elements of corresponding size using an arithmetic value residing in any of the 16 general-purpose registers. Scaling does not add to the EA calculation time. However, when combined with the appropriate derived modes, scaling produces additional capabilities. Arrayed structures can be addressed absolutely and then subscripted; for example, (bd,Rn*SCALE). Optionally, an address register that contains a dynamic displacement can be included in the address calculation (bd,An,Rn*SCALE). Another variation that can be derived is (An,Rn*SCALE). In the first case, the array address is the sum of the contents of a register and a displacement (see Figure 5-13). In the second example, An contains the address of an array and Rn contains a subscript.

5.3.5.2 GENERAL ADDRESSING MODE SUMMARY. The addressing modes described in the previous paragraphs are derived from specific combinations of options in the indexing mode or a selection of two alternate addressing modes. For example, the addressing mode called register indirect (Rn) assembles as the address register indirect if the register is an address register. If Rn is a data register, the assembler uses the address register indirect with index mode, using the data register as the indirect register, and suppresses the address register by setting the base suppress bit in the EA specification. Assigning an address register as Rn provides higher performance than using a data register as Rn. Another case is (bd,An), which selects an addressing mode based on the size of the displacement. If the displacement is 16 bits or less, the address register indirect with displacement mode (d16,An) is used. When a 32-bit displacement is required, the address register indirect with index (bd,An,Xn) is used with the index register suppressed.

It is useful to examine the derived addressing modes available to a programmer (without regard to the CPU32 EA mode actually encoded) because the programmer need not be concerned about these decisions. The assembler can choose the more efficient addressing mode to encode.

5.3.6 M68000 Family Addressing Capability

Programs can be easily transported from one member of the M68000 Family to another member in an upward compatible fashion. The user object code of each early member of the family is upward compatible with newer members and can be executed on the newer microprocessor without change. The address extension word(s) are encoded with information that allows the CPU32 to distinguish the new address extensions to the basic M68000 Family architecture. The address extension words for the MC68000/MC68008/ MC68010 and CPU32/ MC68020/MC68030/MC68040 microprocessors are shown in Figure 5-14.

MC68000/MC68008/MC68010
ADDRESS EXTENSION WORD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	0	0	0	DISPLACEMENT INTEGER							

D/A: 0 = Data Register Select
1 = Address Register Select

W/L: 0 = Word-Size Operation
1 = Long-Word-Size Operation

CPU32/MC68020/MC68030/MC68040
EXTENSION WORD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	SCALE		0	DISPLACEMENT INTEGER							

D/A: 0 = Data Register Select
1 = Address Register Select

W/L: 0 = Word-Size Operation
1 = Long-Word-Size Operation

SCALE: 00 = Scale Factor 1 (Compatible with MC68000)
01 = Scale Factor 2 (Extension to MC68000)
10 = Scale Factor 4 (Extension to MC68000)
11 = Factor 8 (Extension to MC68000)

Figure 5-14. M68000 Family Address Extension Words

The encoding for SCALE used by the CPU32/MC68020/MC68030/MC68040 is a compatible extension of the M68000 architecture. A value of zero for SCALE is the same encoding for both extension words; thus, software that uses this encoding is both upward and downward compatible across all processors in the product line. However, the other values of SCALE are not found in both extension formats; therefore, while software can be easily migrated in an upward compatible direction, only nonscaled addressing is supported in a downward fashion. If the MC68000 were to execute an instruction that encoded a scaling factor, the scaling factor would be ignored and would not access the desired memory address.

The earlier microprocessors have no knowledge of the extension word formats implemented by newer processors, and, while they do detect illegal instructions, they do not decode invalid encodings of the extension words as exceptions.

5.3.7 Other Data Structures

In addition to supporting the array data structure with the index addressing mode, M68000 processors also support stack and queue data structures with the address register indirect postincrement and predecrement addressing modes. A stack is a last-in-first-out (LIFO); a queue is a first-in-first-out (FIFO) list. When data is added to a stack or queue, it is pushed onto the structure; when it is removed, it is 'popped' or pulled from the structure. The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through use of addressing modes.

5.3.7.1 SYSTEM STACK. Address register 7 (A7) is the system stack pointer (SP). The SP is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S bit in the status register. If the S bit indicates the supervisor state, the SSP is the SP, and the USP cannot be referenced as an address register. If the S bit indicates the user state, the USP is the active SP, and the SSP cannot be referenced. Each system stack fills from high memory to low memory. The address mode –(SP) creates a new item on the active system stack, and the address mode (SP)+deletes an item from the active system stack.

The program counter is saved on the active system stack on subroutine calls and is restored from the active system stack on returns. However, during the processing of traps and interrupts, both the program counter and the status register are saved on the supervisor stack. Thus, the correct execution of the supervisor state code is not dependent on the behavior of user code, and user programs may use the USP arbitrarily.

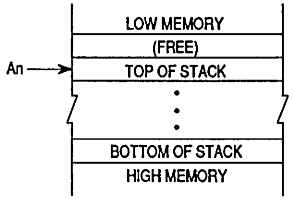
To keep data on the system stack aligned properly, data entry on the stack is restricted so that data is always put in the stack on a word boundary. Thus, byte data is pushed on or pulled from the system stack in the high-order half of the word; the low-order half is unchanged.

5.3.7.2 USER STACKS. The user can implement stacks with the address register indirect with postincrement and predecrement addressing modes. With address register An (n=0–6), the user can implement a stack that is filled either from high to low memory or from low to high memory. Important considerations are as follows:

- Use the predecrement mode to decrement the register before its contents are used as the pointer to the stack.
- Use the postincrement mode to increment the register after its contents are used as the pointer to the stack.
- Maintain the SP correctly when byte, word, and long-word items are mixed in these stacks.

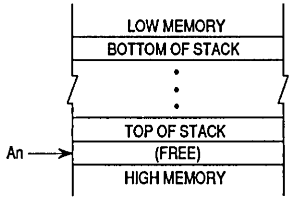
To implement stack growth from high to low memory, use −(An) to push data on the stack and (An)+ to pull data from the stack.

For this type of stack, after either a push or a pull operation, register An points to the top item on the stack. This scheme is illustrated as follows:



To implement stack growth from low to high memory, use (An)+ to push data on the stack and −(An) to pull data from the stack.

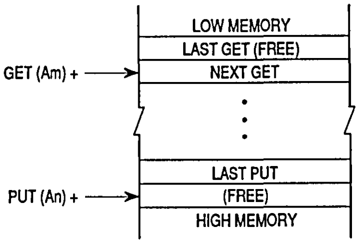
In this case, after either a push or pull operation, register An points to the next available space on the stack. This scheme is illustrated as follows:



5.3.7.3 QUEUES. The user can implement queues with the address register indirect with postincrement or predecrement addressing modes. Using a pair of address registers (two of A0–A6), the user can implement a queue which is filled either from high to low memory or from low to high memory. Two registers are used because queues are pushed from one end and pulled from the other. One register, An, contains the 'put' pointer; the other register, Am, contains the 'get' pointer.

To implement growth of the queue from low to high memory, use (An)+ to put data into the queue and (Am)+ to get data from the queue.

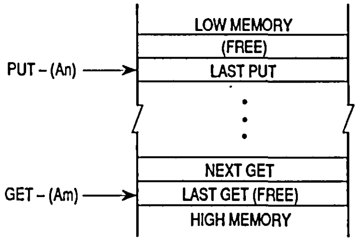
After a 'put' operation, the 'put' address register points to the next available space in the queue, and the unchanged 'get' address register points to the next item to be removed from the queue. After a 'get' operation, the 'get' address register points to the next item to be removed from the queue, and the unchanged 'put' address register points to the next available space in the queue, which is illustrated as follows:



To implement the queue as a circular buffer, the relevant address register should be checked and adjusted, if necessary, before performing the 'put' or 'get' operation. The address register is adjusted by subtracting the buffer length (in bytes) from the register contents.

To implement growth of the queue from high to low memory, use –(An) to put data into the queue and –(Am) to get data from the queue.

After a 'put' operation, the 'put' address register points to the last item placed in the queue, and the unchanged 'get' address register points to the last item removed from the queue. After a 'get' operation, the 'get' address register points to the last item removed from the queue, and the unchanged 'put' address register points to the last item placed in the queue, which is illustrated as follows:



To implement the queue as a circular buffer, the 'get' or 'put' operation should be performed first, and then the relevant address register should be checked and adjusted, if necessary. The address register is adjusted by adding the buffer length (in bytes) to the register contents.

5.4 INSTRUCTION SET

This subsection describes the set of instructions provided in the CPU32 and demonstrates their use. For a more detailed description of the instructions, refer to M68000 PM/AD, *Programmer's Reference Manual*.

The CPU32 instructions include machine functions for all the following operations:

- Data Movement
- Arithmetic Operations
- Logical Operations
- Shifts and Rotates
- Bit Manipulation
- Conditionals and Branches
- System Control

The large instruction set encompasses a complete range of capabilities and, combined with the enhanced addressing modes, provides a flexible base for program development.

5.4.1 M68000 Family Compatibility

It is the philosophy of the M68000 Family that all user-mode programs can execute unchanged on a more advanced processor and that supervisor-mode programs and exception handlers should require only minimal alteration.

The CPU32 can be thought of as an intermediate member of the M68000 Family. Object code from an MC68000 or MC68010 may be executed on the CPU32, and many of the instruction and addressing mode extensions of the MC68020 are also supported.

5.4.1.1 NEW INSTRUCTIONS. Two instructions have been added to the M68000 instruction set for use in controller applications. These are the low-power stop (LPSTOP) and the table lookup and interpolate (TBL) commands.

5.4.1.1.1 Low-Power Stop (LPSTOP). In applications where power consumption is a consideration, the CPU32 can force the device into a low-power standby mode when immediate processing is not required. The low-power mode is entered by executing the LPSTOP instruction. The processor remains in this mode until a user-specified or higher level interrupt or a reset occurs.

5.4.1.1.2 Table Lookup and Interpolate (TBL). To maximize throughput for real-time applications, reference data is often precalculated and stored in memory for quick access. The storage of sufficient data points can require an inordinate amount of memory. The TBL instruction uses linear interpolation to recover intermediate values from a sample of data points, thus conserving memory.

When the TBL instruction is executed, the CPU32 looks up two table entries bounding the desired result and performs a linear interpolation between them. Byte, word, and long-word operand sizes are supported. The result is rounded according to the round-to-nearest algorithm. Optionally, byte and word results can be left unrounded and returned along with the fractional portion of the calculated result. Software can make use of this extra precision to reduce the cumulative error in complex calculations. See **5.4.5 Using the Table Instruction** for examples.

5.4.1.2 UNIMPLEMENTED INSTRUCTIONS. The ability to trap on unimplemented instructions allows user-supplied code to emulate unimplemented capabilities or to define special-purpose functions. However, Motorola reserves the right to use all currently unimplemented instruction operation codes for future M68000 enhancements. See **5.6.2.8 ILLEGAL OR UNIMPLEMENTED INSTRUCTIONS** for more details.

5.4.2 Instruction Format and Notation

All instructions consist of at least one word. Some instructions can have as many as seven words, as shown in Figure 5-15. The first word of the instruction, called the operation word, specifies the instruction length and the operation to be performed. The remaining words, called extension words, further specify the instruction and operands. These words may be immediate operands, extensions to the effective address mode specified in the operation word, branch displacements, bit number, special register specifications, trap operands, or argument counts.

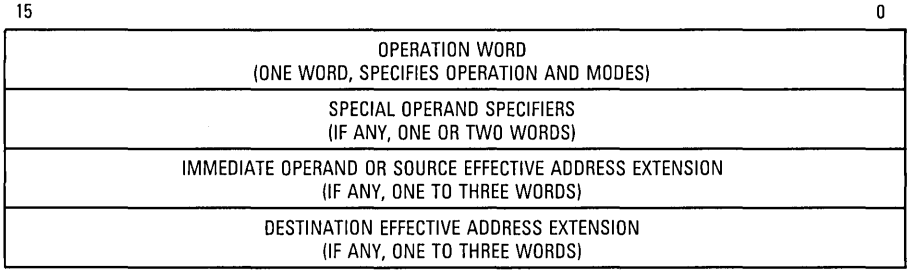

15
0
OPERATION WORD
(ONE WORD, SPECIFIES OPERATION AND MODES)
SPECIAL OPERAND SPECIFIERS
(IF ANY, ONE OR TWO WORDS)
IMMEDIATE OPERAND OR SOURCE EFFECTIVE ADDRESS EXTENSION
(IF ANY, ONE TO THREE WORDS)
DESTINATION EFFECTIVE ADDRESS EXTENSION
(IF ANY, ONE TO THREE WORDS)

Figure 5-15. Instruction Word General Format

Besides the operation code, which specifies the function to be performed, an instruction defines the location of every operand for the function. Instructions specify an operand location in one of three ways:

• Register Specification	A register field of the instruction contains the number of the register.
• Effective Address	An effective address field of the instruction contains address mode information.
• Implicit Reference	The definition of an instruction implies the use of specific registers.

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used. See **5.3 DATA ORGANIZATION AND ADDRESSING CAPABILITIES** for detailed register information.

Except where noted, the following notations are used:

An	= any address register, A7–A0
Dn	= any data register, D7–D0
Rn	= any address or data register
CCR	= condition code register (lower byte of status register)
cc	= condition codes from CCR
SR	= status register
SP	= active stack pointer
USP	= user stack pointer
SSP	= supervisor stack pointer
DFC	= destination function code register
SFC	= source function code register
Rc	= control register (VBR, SFC, DFC)
d	= displacement; d_{16} is a 16-bit displacement
⟨ea⟩	= effective address
list	= list of registers (for example, D3–D0)
#⟨data⟩	= immediate data; a literal integer
label	= assembly program label
[7]	= bit 7 of an operand
[31:24]	= bits 31–24 of operand (high-order byte of a register)
X	= extend (X) bit in CCR
N	= negative (N) bit in CCR
V	= overflow (V) bit in CCR
C	= carry (C) bit in CCR
+	= arithmetic addition or postincrement
−	= arithmetic subtraction or predecrement
×	= arithmetic multiplication
/	= arithmetic division or conjunction symbol
~	= invert; operand is logically complemented
Λ	= logical AND
V	= logical OR
⊕	= logical exclusive OR
Dc	= data register D7–D0 used during compare
Du	= D7–D0 used during update
Dr, Dq	= data registers, remainder or quotient of divide
Dh, Dl	= data registers, high- or low-order 32 bits of product
MSW	= most significant word
LSW	= least significant word
FC	= function code
{R/W}	= read or write indicator
[An]	= address extensions

In the description of an operation, a destination operand is placed to the right of source operands and is indicated by an arrow (➧).

5.4.3 Instruction Summary

All CPU32 instructions are summarized in Table 5-4.

Table 5-4. Instruction Set Summary

Opcode	Operation	Syntax
ABCD	Source$_{10}$ + Destination$_{10}$ + X ➧ Destination	ABCD Dy,Dx ABCD −(Ay),−(Ax)
ADD	Source + Destination ➧ Destination	ADD ⟨ea⟩,Dn ADD Dn,⟨ea⟩
ADDA	Source + Destination ➧ Destination	ADDA ⟨ea⟩,An
ADDI	Immediate Data + Destination ➧ Destination	ADDI #⟨data⟩,⟨ea⟩
ADDQ	Immediate Data + Destination ➧ Destination	ADDQ #⟨data⟩,⟨ea⟩
ADDX	Source + Destination + X ➧ Destination	ADDX Dy,Dx ADDX −(Ay),−(Ax)
AND	SourceΛDestination ➧ Destination	AND ⟨ea⟩,Dn AND Dn,⟨ea⟩
ANDI	Immediate DataΛDestination ➧ Destination	ANDI #⟨data⟩,⟨ea⟩
ANDI to CCR	SourceΛCCR ➧CCR	ANDI #⟨data⟩,CCR
ANDI to SR	If supervisor state the SourceΛSR ➧ SR else TRAP	ANDI #⟨data⟩,SR
ASL,ASR	Destination Shifted by ⟨count⟩ ➧ Destination	ASd Dx,Dy ASd #⟨data⟩,Dy ASd ⟨ea⟩
Bcc	If (condition true) then PC + d ➧ PC	Bcc ⟨label⟩
BCHG	~(⟨number⟩ of Destination) ➧ Z; ~(⟨number⟩ of Destination) ➧ ⟨bit number⟩ of Destination	BCHG Dn,⟨ea⟩ BCHG #⟨data⟩,⟨ea⟩
BCLR	~(⟨bit number⟩ of Destination) ➧ Z; 0 ➧ ⟨bit number⟩ of Destination	BCLR Dn,⟨ea⟩ BCLR #⟨data⟩,⟨ea⟩
BGND	If (background mode enabled) then enter background mode else Format/Vector offset ➧ −(SSP) PC ➧ −(SSP) SR ➧ −(SSP) (Vector) ➧ PC	BGND
BKPT	Run breakpoint acknowledge cycle; TRAP as illegal instruction	BKPT #⟨data⟩
BRA	PC + d ➧ PC	BRA ⟨label⟩
BSET	~(⟨bit number⟩ of Destination) ➧ Z; 1 ➧ ⟨bit number⟩ of Destination	BSET Dn,⟨ea⟩ BSET #⟨data⟩,⟨ea⟩
BSR	SP − 4 ➧ SP; PC ➧ (SP); PC + d ➧ PC	BSR ⟨label⟩

Table 5-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
BTST	–(⟨bit number⟩ of Destination) ➧ Z;	BTST Dn,⟨ea⟩ BTST #⟨data⟩,⟨ea⟩
CHK	If Dn < 0 or Dn > Source then TRAP	CHK ⟨ea⟩,Dn
CHK2	If Rn < lower bound or Rn > upper bound then TRAP	CHK2 ⟨ea⟩,Rn
CLR	0 ➧ Destination	CLR ⟨ea⟩
CMP	Destination — Source ➧ cc	CMP ⟨ea⟩,Dn
CMPA	Destination — Source	CMPA ⟨ea⟩,An
CMPI	Destination — Immediate Data	CMPI #⟨data⟩,⟨ea⟩
CMPM	Destination — Source ➧ cc	CMPM (Ay)+,(Ax)+
CMP2	Compare Rn < lower-bound or Rn > upper-bound and Set Condition Codes	CMP2 ⟨ea⟩,Rn
DBcc	If condition false then (Dn – 1 ➧ Dn; If Dn ≠ –1 then PC + d ➧ PC)	DBcc Dn,⟨label⟩
DIVS DIVSL	Destination/Source ➧ Destination	DIVS.W ⟨ea⟩,Dn 32/16 ➧ 16r:16q DIVS.L ⟨ea⟩,Dq 32/32 ➧ 32q DIVS.L ⟨ea⟩,Dr:Dq 64/32 ➧ 32r:32q DIVSL.L ⟨ea⟩,Dr:Dq 32/32 ➧ 32r:32q
DIVU DIVUL	Destination/Source ➧ Destination	DIVU.W ⟨ea⟩,Dn 32/16 ➧ 16r:16q DIVU.L ⟨ea⟩,Dq 32/32 ➧ 32q DIVU.L ⟨ea⟩,Dr:Dq 64/32 ➧ 32r:32q DIVUL.L ⟨ea⟩,Dr:Dq 32/32 ➧ 32r:32q
EOR	Source ⊕ Destination ➧ Destination	EOR Dn,⟨ea⟩
EORI	Immediate Data ⊕ Destination ➧ Destination	EORI #⟨data⟩,⟨ea⟩
EORI to CCR	Source ⊕ CCR ➧ CCR	EORI #⟨data⟩,CCR
EORI to SR	If supervisor state the Source ⊕ SR ➧ SR else TRAP	EORI #⟨data⟩,SR
EXG	Rx ⬌ Ry	EXG Dx,Dy EXG Ax,Ay EXG Dx,Ay EXG Ay,Dx
EXT EXTB	Destination Sign-Extended ➧ Destination	EXT.W Dn extend byte to word EXT.L Dn extend word to long word EXTB.L Dn extend byte to long word

ILLEGAL	SSP – 2 ➧ SSP; Vector Offset ➧ (SSP);
	SSP – 4 ➧ SSP; PC ➧ (SSP);
	SSp – 2 ➧ SSP; SR ➧ (SSP);
	Illegal Instruction Vector Address ➧ PC
		ILLEGAL

JMP	Destination Address ➧ PC	JMP ⟨ea⟩
JSR	SP – 4 ➧ SP; PC ➧ (SP) Destination Address ➧ PC	JSR ⟨ea⟩
LEA	⟨ea⟩ ➧ An	LEA ⟨ea⟩,An

Table 5-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
LINK	SP−4 ▶ SP; An ▶ (SP) SP ▶ An, SP+d ▶ SP	LINK An,#⟨displacement⟩
LPSTOP	If supervisor state Immediate Data ▶ SR Interrupt Mask ▶ External Bus Interface (EBI) STOP else TRAP	
LSL,LSR	Destination Shifted by ⟨count⟩ ▶ Destination	LSd[1] Dx,Dy LSd[1] #⟨data⟩,Dy LSd[1] ⟨ea⟩
MOVE	Source ▶ Destination	MOVE ⟨ea⟩,⟨ea⟩
MOVEA	Source ▶ Destination	MOVEA ⟨ea⟩,An
MOVE from CCR	CCR ▶ Destination	MOVE CCR,⟨ea⟩
MOVE to CCR	Source ▶ CCR	MOVE ⟨ea⟩,CCR
MOVE from SR	If supervisor state then SR ▶ Destination else TRAP	MOVE SR,⟨ea⟩
MOVE to SR	If supervisor state then Source ▶ SR else TRAP	MOVE ⟨ea⟩,SR
MOVE USP	If supervisor state then USP ▶ An or An ▶ USP else TRAP	MOVE USP,An MOVE An,USP
MOVEC	If supervisor state then Rc ▶ Rn or Rn ▶ Rc else TRAP	MOVEC Rc,Rn MOVEC Rn,Rc
MOVEM	Registers ▶ Destination Source ▶ Registers	MOVEM register list,⟨ea⟩ MOVEM ⟨ea⟩,register list
MOVEP	Source ▶ Destination	MOVEP Dx,(d,Ay) MOVEP (d,Ay),Dx
MOVEQ	Immediate Data ▶ Destination	MOVEQ #⟨data⟩,Dn
MOVES	If supervisor state then Rn ▶ Destination [DFC] or Source [SFC] ▶ Rn else TRAP	MOVES Rn,⟨ea⟩ MOVES ⟨ea⟩,Rn
MULS	Source × Destination ▶ Destination	MULS.W ⟨ea⟩,Dn 16×16 ▶ 32 MULS.L ⟨ea⟩,Dl 32×32 ▶ 32 MULS.L ⟨ea⟩,Dh:Dl 32×32 ▶ 64
MULU	Source × Destination ▶ Destination	MULU.W ⟨ea⟩,Dn 16×16 ▶ 32 MULU.L ⟨ea⟩,Dl 32×32 ▶ 32 MULU.L ⟨ea⟩,Dh:Dl 32×32 ▶ 64
NBCD	0−(Destination_{10})−X ▶ Destination	NBCD ⟨ea⟩
NEG	0−(Destination) ▶ Destination	NEG ⟨ea⟩
NEGX	0−(Destination)−X ▶ Destination	NEGX ⟨ea⟩
NOP	None	NOP

Table 5-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
NOT	~Destination → Destination	NOT ⟨ea⟩
OR	Source V Destination → Destination	OR ⟨ea⟩,Dn OR Dn,⟨ea⟩
ORI	Immediate Data V Destination → Destination	ORI #⟨data⟩,⟨ea⟩
ORI to CCR	Source V CCR → CCR	ORI #⟨data⟩,CCR
ORI to SR	If supervisor state then Source V SR → SR else TRAP	ORI #⟨data⟩,SR
PEA	Sp−4 → SP; ⟨ea⟩ → (SP)	PEA ⟨ea⟩
RESET	If supervisor state then Assert $\overline{\text{RESET}}$ else TRAP	RESET
ROL,ROR	Destination Rotated by ⟨count⟩ → Destination	ROd[1] Rx,Dy ROd[1] #⟨data⟩,Dy ROd[1] ⟨ea⟩
ROXL,ROXR	Destination Rotated with X by ⟨count⟩ → Destination	ROXd[1] Dx,Dy ROXd[1] #⟨data⟩,Dy ROXd[1] ⟨ea⟩
RTD	(SP) → PC; SP+4+d →'SP	RTD #⟨displacement⟩

RTE	If supervisor state
	the (SP) → SR; SP+2 → SP; (SP) → PC;
	SP+4 → SP;
	restore state and deallocate stack according to (SP)
	else TRAP
		RTE

RTR	(SP) → CCR; SP+2 → SP; (SP) → PC; SP+4 → SP	RTR
RTS	(SP) → PC; SP+4 → SP	RTS
SBCD	Destination$_{10}$−Source$_{10}$−X → Destination	SBCD Dx,Dy SBCD −(Ax),−(Ay)
Scc	If Condition True then 1s → Destination else 0s → Destination	Scc ⟨ea⟩
STOP	If supervisor state then Immediate Data → SR; STOP else TRAP	STOP #⟨data⟩
SUB	Destination−Source → Destination	SUB ⟨ea⟩,Dn SUB Dn,⟨ea⟩
SUBA	Destination−Source → Destination	SUBA ⟨ea⟩,An
SUBI	Destination−Immediate Data → Destination	SUBI #⟨data⟩,⟨ea⟩
SUBQ	Destination−Immediate Data → Destination	SUBQ #⟨data⟩,⟨ea⟩
SUBX	Destination−Source−X → Destination	SUBX Dx,Dy SUBX −(Ax),−(Ay)
SWAP	Register [31:16] ↔ Register [15:0]	SWAP Dn
TAS	Destination Tested → Condition Codes; 1 → bit 7 of Destination	TAS ⟨ea⟩

Table 5-4. Instruction Set Summary (Concluded)

Opcode	Operation	Syntax
TBLS	ENTRY(n)+{(ENTRY(n+1)−ENTRY(n))*Dx[7:0]}/256 ➧ Dx	TBLS.⟨size⟩ ⟨ea⟩, Dx TBLS.⟨size⟩ Dym:Dyn, Dx
TBLSN	ENTRY(n)•256+{(ENTRY(n+1)−ENTRY(n))•Dx[7:0]} ➧ Dx	TBLSN.⟨size⟩ ⟨ea⟩,Dx TBLSN.⟨size⟩ Dym:Dyn, Dx
TBLU	ENTRY(n)+{(ENTRY(n+1)−ENTRY(n))*Dx[7:0]}/256 ➧ Dx	TBLU.⟨size⟩ ⟨ea⟩,Dx TBLU.⟨size⟩ Dym:Dyn, Dx
TBLUN	ENTRY(n)•256+{(ENTRY(n+1)−ENTRY(n))•Dx[7:0]} ➧ Dx	TBLUN.⟨size⟩ ⟨ea⟩,Dx TBLUN.⟨size⟩ Dym:Dyn,Dx

TRAP	SSP−2 ➧ SSP; Format/Offset ➧ (SSP);
	SSP−4 ➧ SSP; PC ➧ (SSP); SSP−2 ➧ SSP;
	SR ➧ (SSP); Vector Address ➧ PC
		TRAP #⟨vector⟩

TRAPcc	If cc then TRAP	TRAPcc TRAPcc.W #⟨data⟩ TRAPcc.L #⟨data⟩
TRAPV	If V then TRAP	TRAPV
TST	Destination Tested ➧ Condition Codes	TST ⟨ea⟩
UNLK	An ➧ SP; (SP) ➧ An; SP+4 ➧ SP	UNLK An

NOTE 1: d is direction, L or R.

The instructions form a set of tools to perform the following operations:

Data Movement	Bit Manipulation
Integer Arithmetic	Binary-Coded Decimal Arithmetic
Logical	Program Control
Shift and Rotate	System Control

The complete range of instruction capabilities combined with the addressing modes described previously provide flexibility for program development.

5.4.3.1 CONDITION CODE REGISTER. The condition code register portion of the status register contains five bits that indicate the result of a processor operation. Table 5-5 lists the effect of each instruction on these bits. The carry bit and the multiprecision extend bit are separate in the M68000 Family to simplify programming techniques that use them. Refer to Table 5-9 as an example.

Table 5-5. Condition Code Computations

Operations	X	N	Z	V	C	Special Definition
ABCD	*	U	?	U	?	C = Decimal Carry $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$
ADDX	*	*	?	?	?	$V = Sm \wedge Dm \wedge \overline{Rm} \vee \overline{Sm} \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge Dm \vee \overline{Rm} \wedge Dm \vee Sm \wedge \overline{Rm}$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	—	*	*	0	0	
CHK	—	*	U	U	U	
CHK2, CMP2	—	U	?	U	?	$Z = (R = LB) \vee (R = UB)$ $C = (LB < = UB) \wedge (IR < LB) \vee (R > UB)) \vee$ $(UB < LB) \wedge (R > UB) \wedge (R < LB)$
SUB, SUBI, SUBQ	*	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
SUBX	*	*	?	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
CMP, CMPI, CMPM	—	*	*	?	?	$V = \overline{Sm} \wedge Dm \wedge \overline{Rm} \vee Sm \wedge \overline{Dm} \wedge Rm$ $C = Sm \wedge \overline{Dm} \vee Rm \wedge \overline{Dm} \vee Sm \wedge Rm$
DIVS, DIVU	—	*	*	?	0	V = Division Overflow
MULS, MULU	—	*	*	?	0	V = Multiplication Overflow
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{Ro}$
NEG	*	*	*	?	?	$V = Dm \wedge Rm$ $C = Dm \vee Rm$
NEGX	*	*	?	?	?	$V = Dm \wedge Rm$ $C = Dm \vee Rm$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
ASL	*	*	*	?	?	$V = Dm \wedge (\overline{Dm-1} \vee \ldots \vee \overline{Dm-r}) \vee \overline{Dm} \wedge (DM-1$ $\vee \ldots + Dm-r)$ $C = \overline{Dm-r+1}$
ASL (r=0)	—	*	*	0	0	
LSL, ROXL	*	*	*	0	?	$C = Dm-r+1$
LSR (r=0)	—	*	*	0	0	
ROXL (r=0)	—	*	*	0	?	C = X
ROL	—	*	*	0	?	$C = Dm-r+1$
ROL (r=0)	—	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	$C = Dr-1$
ASR, LSR (r=0)	—	*	*	0	0	

Table 5-5. Condition Code Computations (Continued)

Operations	X	N	Z	V	C	Special Definition
ROXR (r=0)	—	*	*	0	?	C = X
ROR	—	*	*	0	?	C = Dr−1
ROR (r=0)	—	*	*	0	0	

NOTE:

— = Not Affected
U = Undefined, Result Meaningless
? = Other — See Special Definition
* = General Case
X = C
N = Rm
Z = $\overline{\text{Rm}}$ Λ . . . Λ $\overline{\text{R0}}$
Sm = Source Operand — Most Significant Bit
Dm = Destination Operand — Most Significant Bit
Rm = Result Operand — Most Significant Bit
R = Register Tested
n = Bit Number
r = Shift Count
LB = Lower Bound
UB = Upper Bound
Λ = Boolean AND
V = Boolean OR
$\overline{\text{Rm}}$ = NOT Rm

5.4.3.2 DATA MOVEMENT INSTRUCTIONS. The MOVE instruction with its associated addressing modes is the basic means of transferring and storing address and data. MOVE instructions transfer byte, word, and long-word operands from memory to memory, memory to register, register to memory, and register to register. Address movement instructions (MOVE or MOVEA) transfer word and long-word operands and ensure that only valid address manipulations are executed. In addition to the general MOVE instructions, there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), move quick (MOVEQ), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), and unlink stack (UNLK). Table 5-6 is a summary of the data movement operations.

Table 5-6. Data Movement Operations

Instruction	Operand Syntax	Operand Size	Operation
EXG	Rn, Rn	32	Rn → Rn
LEA	⟨ea⟩, An	32	⟨ea⟩ → An
LINK	An, #⟨d⟩	16, 32	SP −4 → SP, An → (SP); SP → An, SP+d → SP
MOVE MOVEA	⟨ea⟩, ⟨ea⟩ ⟨ea⟩, An	8, 16, 32 16,32 → 32	source → destination
MOVEM	list, ⟨ea⟩ ⟨ea⟩, list	16, 32 16, 32 → 32	listed registers → destination source → listed registers
MOVEP	Dn, (d_{16},An) (d_{16},An), Dn	16, 32	Dn[31:24] → (An + d); Dn[23:16] → (An+d+2); Dn[15:8] → (An+d+4); Dn[7:0] → (An+d+6) (An+d) → Dn[31:24]; (An+d+2) → Dn[23:16] (An+d+4) → Dn[15:8]; (An+d+6) → Dn[7:0]
MOVEQ	#⟨data⟩, Dn	8 → 32	immediate data → destination
PEA	⟨ea⟩	32	SP −4 → SP; ⟨ea⟩ → (SP)
UNLK	An	32	An → SP; (SP) → An; SP+4 → SP

5.4.3.3 INTEGER ARITHMETIC OPERATIONS. The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP, CMPM, CMP2), clear (CLR), and negate (NEG). The instruction set includes ADD, CMP, and SUB instructions for both address and data operations with all operand sizes valid for data operations. Address operands consist of 16 or 32 bits. The clear and negate instructions apply to all sizes of data operands.

Signed and unsigned MUL and DIV instructions include:

- Word multiply to produce a long-word product
- Long-word multiply to produce a long-word or quad-word product
- Division of a long-word dividend by a word divisor (word quotient and word remainder)
- Division of a long-word or quad-word dividend by a long-word divisor (long-word quotient and long-word remainder)

A set of extended instructions provides multiprecision and mixed-size arithmetic. These instructions are add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX). Refer to Table 5-7 for a summary of the integer arithmetic operations.

Table 5-7. Integer Arithmetic Operations

Instruction	Operand Syntax	Operand Size	Operation
ADD	Dn, ⟨ea⟩	8, 16, 32	source + destination ▶ destination
	⟨ea⟩, Dn	8, 16, 32	
ADDA	⟨ea⟩, An	16, 32	
ADDI	#⟨data⟩, ⟨ea⟩	8, 16, 32	immediate data + destination ▶ destination
ADDQ	#⟨data⟩, ⟨ea⟩	8, 16, 32	
ADDX	Dn, Dn	8, 16, 32	source + destination + X ▶ destination
	−(An), −(An)	8, 16, 32	
CLR	⟨ea⟩	8, 16, 32	0 ▶ destination
CMP	⟨ea⟩, Dn	8, 16, 32	(destination — source), CCR shows results
CMPA	⟨ea⟩, An	16, 32	
CMPI	#⟨data⟩, ⟨ea⟩	8, 16, 32	(destination — immediate data), CCR shows results
CMPM	(An)+, (An)+	8, 16, 32	(destination — source), CCR shows results
CMP2	⟨ea⟩, Rn	8, 16, 32	lower bound< = Rn< = upper bound
DIVS/DIVU	⟨ea⟩, Dn	32/16 ▶ 16:16	destination/source ▶ destination (signed or unsigned)
	⟨ea⟩, Dr:Dq	64/32 ▶ 32:32	
	⟨ea⟩, Dq	32/32 ▶ 32	
DIVSL/DIVUL	⟨ea⟩, Dr:Dq	32/32 ▶ 32:32	
EXT	Dn	8 ▶ 16	sign extended destination ▶ destination
	Dn	16 ▶ 32	
EXTB	Dn	8 ▶ 32	
MULS/MULU	⟨ea⟩, Dn	16×16 ▶ 32	source*destination ▶ destination (signed or unsigned)
	⟨ea⟩, Dl	32×32 ▶ 32	
	⟨ea⟩, Dh:Dl	32×32 ▶ 64	
NEG	⟨ea⟩	8, 16, 32	0 — destination ▶ destination
NEGX	⟨ea⟩	8, 16, 32	0 — destination — X ▶ destination
SUB	⟨ea⟩, Dn	8, 16, 32	destination — source ▶ destination
	Dn, ⟨ea⟩	8, 16, 32	
SUBA	⟨ea⟩, An	16, 32	
SUBI	#⟨data⟩, ⟨ea⟩	8, 16, 32	destination — immediate data ▶ destination
SUBQ	#⟨data⟩, ⟨ea⟩	8, 16, 32	
SUBX	Dn, Dn	8, 16, 32	destination — source — X ▶ destination
	−(An), −(An)	8, 16, 32	
TBLS/TBLU	⟨ea⟩, Dn	8, 16, 32	Dyn — Dym ▶ temp
	Dym:Dyn, Dn		[temp*Dx (7:0)]) ▶ temp
			Dym*256) + temp ▶ Dn
TBLSN/	⟨ea⟩, Dn	8, 16, 32	Dyn — Dyn ▶ Temp
TBLUN	Dym:Dyn, Dn		(temp*Dn [7:0])/256 ▶ temp
			Dym + temp ▶ Dn

5.4.3.4 LOGICAL INSTRUCTIONS. The logical operation instructions (AND, OR, EOR, and NOT) perform logical operations with all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. The TST instruction arithmetically compares the operand with zero, placing the result in the condition code register. Table 5-8 summarizes the logical operations.

Table 5-8. Logical Operations

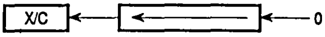
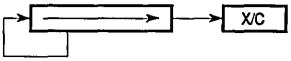
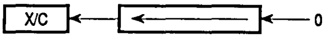
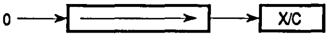
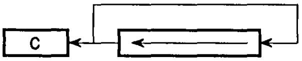
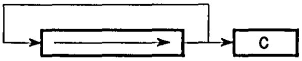
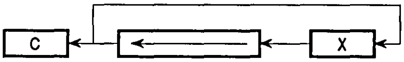
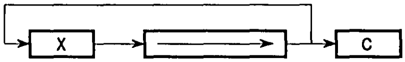
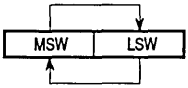
Instruction	Operand Syntax	Operand Size	Operation
AND	⟨ea⟩,Dn Dn, ⟨ea⟩	8, 16, 32 8, 16, 32	source Λ destination ➧ destination 8, 16, 32
ANDI	#⟨data⟩,⟨ea⟩	8, 16, 32	immediate data Λ destination ➧ destination
EOR	Dn,⟨ea⟩	8, 16, 32	source ⊕ destination ➧ destination
EORI	#⟨data⟩,⟨ea⟩	8, 16, 32	immediate data ⊕ destination ➧ destination
NOT	⟨ea⟩	8, 16, 32	~ destination ➧ destination
OR	⟨ea⟩,Dn Dn,⟨ea⟩	8, 16, 32 8, 16, 32	source V destination ➧ destination
ORI	#⟨data⟩,⟨ea⟩	8, 16, 32	immediate data V destination ➧ destination
TST	⟨ea⟩	8, 16, 32	source −0 to set condition codes

5.4.3.5 SHIFT AND ROTATE INSTRUCTIONS. The arithmetic shift instructions, ASR and ASL, and logical shift instructions, LSR and LSL, provide shift operations in both directions. The ROR, ROL, ROXR, and ROXL instructions perform rotate (circular shift) operations, with and without the extend bit. All shift and rotate operations can be performed on either registers or memory.

Register shift and rotate operations shift all operand sizes. The shift count may be specified in the instruction operation word (to shift from 1–8 places) or in a register (modulo 64 shift count).

Memory shift and rotate operations shift word-length operands one bit position only. The SWAP instruction exchanges the 16-bit halves of a register. Performance of shift/rotate instructions is enhanced so that use of the ROR and ROL instructions with a shift count of eight allows fast byte swapping. Table 5-9 is a summary of the shift and rotate operations.

Table 5.9. Shift and Rotate Operations

Instruction	Operand Syntax	Operand Size	Operation
ASL	Dn,Dn #⟨data⟩,Dn ⟨ea⟩	8, 16, 32 8, 16, 32 16	
ASR	Dn,Dn #⟨data⟩,Dn ⟨ea⟩	8, 16, 32 8, 16, 32 16	
LSL	Dn,Dn #⟨data⟩,Dn ⟨ea⟩	8, 16, 32 8, 16, 32 16	
LSR	Dn,Dn #⟨data⟩,Dn ⟨ea⟩	8, 16, 32 8, 16, 32 16	
ROL	Dn,Dn #⟨data⟩,Dn ⟨ea⟩	8, 16, 32 8, 16, 32 16	
ROR	Dn,Dn #⟨data⟩,Dn ⟨ea⟩	8, 16, 32 8, 16, 32 16	
ROXL	Dn,Dn #⟨data⟩,Dn ⟨ea⟩	8, 16, 32 8, 16, 32 16	
ROXR	Dn,Dn #⟨data⟩,Dn ⟨ea⟩	8, 16, 32 8, 16, 32 16	
SWAP	Dn	16	

5.4.3.6 BIT MANIPULATION INSTRUCTIONS. Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). All bit manipulation operations can be performed on either registers or memory. The bit number is specified as immediate data or in a data register. Register operands are 32 bits long, and memory operands are 8 bits long. Table 5-10 is a summary of bit manipulation instructions.

Table 5-10. Bit Manipulation Operations

Instruction	Operand Syntax	Operand Size	Operation
BCHG	Dn,⟨ea⟩ #⟨data⟩,⟨ea⟩	8, 32 8, 32	~ (⟨bit number⟩ of destination) ▶ Z ▶ bit of destination
BCLR	Dn,⟨ea⟩ #⟨data⟩,⟨ea⟩	8, 32 8, 32	~ (⟨bit number⟩ of destination) ▶ Z; 0 ▶ bit of destination
BSET	Dn,⟨ea⟩ #⟨data⟩,⟨ea⟩	8, 32 8, 32	~ (⟨bit number⟩ of destination) ▶ Z; 1 ▶ bit of destination
BTST	Dn,⟨ea⟩ #⟨data⟩,⟨ea⟩	8, 32 8, 32	~ (⟨bit number⟩ of destination) ▶ Z

5.4.3.7 BINARY-CODED DECIMAL (BCD) INSTRUCTIONS. Five instructions support operations on BCD numbers. The arithmetic operations on packed BCD numbers are add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). Table 5-11 is a summary of the BCD operations.

Table 5-11. Binary-Coded Decimal Operations

Instruction	Operand Syntax	Operand Size	Operation
ABCD	Dn,Dn –(An),–(An)	8 8	source$_{10}$+destination$_{10}$+X ▶ destination
NBCD	⟨ea⟩	8	0 — destination$_{10}$ – X ▶ destination
SBCD	Dn,Dn –(An),–(An)	8 8	destination$_{10}$ – source$_{10}$ – X ▶ destination

5.4.3.8 PROGRAM CONTROL INSTRUCTIONS. A set of subroutine call and return instructions and conditional and unconditional branch instructions perform program control operations. Table 5-12 summarizes these instructions.

Table 5-12. Program Control Operations

Instruction	Operand Syntax	Operand Size	Operation
		Conditional	
Bcc	⟨label⟩	8, 16, 32	if condition true, then PC+d ▶ PC
DBcc	Dn,⟨label⟩	16	if condition false, then Dn − 1 ▶ Dn if Dn ≠ − 1, then PC+d ▶ PC+d ▶ CP
Scc	⟨ea⟩	8	if condition true, then destination bits are set to 1; else destination bits are cleared to 0
		Unconditional	
BRA	⟨label⟩	8, 16, 32	PC+d ▶ PC
BSR	⟨label⟩	8, 16, 32	SP − 4 ▶ SP; PC ▶ (SP); PC+d ▶ PC
JMP	⟨ea⟩	none	destination ▶ PC
JSR	⟨ea⟩	none	SP − 4 ▶ SP; PC ▶ (SP); destination ▶ PC
NOP	none	none	PC+2 ▶ PC
		Returns	
RTD	#⟨d⟩	16	(SP) ▶ PC; SP+4+d ▶ SP
RTR	none	none	(SP) ▶ CCR; SP+2 ▶ SP; (SP) ▶ PC; SP+4 ▶ SP
RTS	none	none	(SP) ▶ PC; SP+4 ▶ SP

To specify conditions for change in program control, condition codes must be substituted for the letters "cc" in conditional program control opcodes. Condition test mnemonics are given below. Refer to **5.4.3.10 CONDITION TESTS** for detailed information on condition codes.

CC—Carry clear
CS—Carry set
EQ—Equal
F—False*
GE—Greater or equal
GT—Greater than
HI—High
LE—Less or equal
LS—Low or same
LT—Less than
MI—Minus
NE—Not equal
PL—Plus
T—True
VC—Overflow clear
VS—Overflow set

*Not applicable to the Bcc instruction.

5.4.3.9 SYSTEM CONTROL INSTRUCTIONS. Privileged instructions, trapping instructions, and instructions that use or modify the condition code register provide system control operations. All of these instructions cause the processor to flush the instruction pipeline. Table 5-13 summarizes the instructions. The preceding list of condition tests also applies to the TRAPcc instruction. Refer to **5.4.3.10 CONDITION TESTS** for detailed information on condition codes.

Table 5-13. System Control Operations

Instruction	Operand Syntax	Operand Size	Operation
			Privileged
ANDI	#⟨data⟩,SR	16	immediate data Λ SR ➧ SR
EORI	#⟨data⟩,SR	16	immediate data ⊕ SR ➧ SR
MOVE	⟨ea⟩,SR SR,⟨ea⟩	16 16	source ➧ SR SR ➧ destination
MOVE	USP,An An,USP	32 32	USP ➧ An An ➧ USP
MOVEC	Rc,Rn Rn,Rc	32 32	Rc ➧ Rn Rn ➧ Rc
MOVES	Rn,⟨ea⟩ ⟨ea⟩,Rn	8, 16, 32	Rn ➧ destination using DFC source using SFC ➧ Rn
ORI	#⟨data⟩,SR	16	immediate data V SR ➧ SR
RESET	none	none	assert $\overline{\text{RESET}}$
RTE	none	none	(SP) ➧ SR; SP+2 ➧ SP; (SP) ➧ PC; SP+4 ➧ SP; restore stack according to format
STOP	#⟨data⟩	16	immediate data ➧ SR; STOP
LPSTOP	#⟨data⟩	none	immediate data ➧ SR; interrupt mask ➧ EBI; STOP
			Trap Generating
BKPT	#⟨data⟩	none	if breakpoint cycle acknowledged, then execute returned operation word, else trap as illegal instruction
BGND	none	none	if background mode enabled, then enter background mode else format/vector offset ➧ −(SSP); PC ➧ −(SSP); SR ➧ −(SSP); (vector) ➧ PC
CHK	⟨ea⟩,Dn	16, 32	if Dn <0 or Dn <(ea), then CHK exception
CHK2	⟨ea⟩,Rn	8, 16, 32	if Rn <lower bound or Rn> upper bound, then CHK exception
ILLEGAL	none	none	SSP −2 ➧ SSP; vector offset ➧ (SSP); SSP −4 ➧ SSP; PC ➧ (SSP); SSP −2 ➧ SSP; SR ➧ (SSP); illegal instruction vector address ➧ PC
TRAP	#⟨data⟩	none	SSP −2 ➧ SSP; format and vector offset ➧ (SSP) SSP −4 ➧ SSP; PC ➧ (SSP); SR ➧ (SSP); vector address ➧ PC
TRAPcc	none #⟨data⟩	none 16, 32	if cc true, then TRAP exception
TRAPV	none	none	if V then take overflow TRAP exception
			Condition Code Register
ANDI	#⟨data⟩,CCR	8	immediate data Λ CCR ➧ CCR
EORI	#⟨data⟩,CCR	8	immediate data ⊕ CCR ➧ CCR
MOVE	⟨ea⟩,CCR CCR,⟨ea⟩	16 16	source ➧ CCR CCR ➧ destination
ORI	#⟨data⟩,CCR	8	immediate data V CCR ➧ CCR

5.4.3.10 CONDITION TESTS. Conditional program control instructions and the TRAPcc instruction execute on the basis of condition tests. A condition test is the evaluation of a logical expression related to the state of the CCR bits. If the result is one, the condition is true. If the result is zero, the condition is false. For example, the T condition is always true, and the EQ condition is true only if the Z-bit condition code is true. Table 5-14 lists each condition test.

Table 5-14. Condition Tests

Mnemonic	Condition	Encoding	Test
T	True	0000	1
F*	False	0001	0
HI	High	0010	$\overline{C} \bullet \overline{Z}$
LS	Low or Same	0011	$\overline{C} + \overline{Z}$
CC(HS)	Carry Clear	0100	$\overline{C}$
CS(LO)	Carry Set	0101	C
NE	Not Equal	0110	$\overline{Z}$
EQ	Equal	0111	Z
VC	Overflow Clear	1000	$\overline{V}$
VS	Overflow Set	1001	V
PL	Plus	1010	$\overline{N}$
MI	Minus	1011	N
GE	Greater or Equal	1100	$N \bullet V + \overline{N} \bullet \overline{V}$
LT	Less Than	1101	$N \bullet \overline{V} + \overline{N} \bullet V$
GT	Greater Than	1110	$N \bullet V \bullet \overline{Z} + \overline{N} \bullet \overline{V} \bullet \overline{Z}$
LE	Less or Equal	1111	$Z + N \bullet \overline{V} + \overline{N} \bullet V$

*Not available for the Bcc instruction.

• = Boolean AND
$\underline{+}$ = Boolean OR
$\overline{N}$ = Boolean NOT N

5.4.4 Using the Table Lookup and Interpolate Instruction

There are four table lookup and interpolate instructions. TBLS returns a signed, rounded byte, word, or long-word result; TBLSN returns a signed, unrounded byte, word, or long-word result. TBLU returns an unsigned, rounded byte, word, or long-word result; TBLUN returns an unsigned, unrounded byte, word, or long-word result. All four instructions support two types of interpolation data: an n-element table stored in memory, and a 2-element range stored in a pair of data registers. The latter form provides a means of performing surface (3D) interpolation between two previously calculated linear interpolations.

The following examples show how a programmer can compress tables and use fewer interpolation levels between table entries. Example 1 (see Figure 5-16) demonstrates table lookup and interpolation for a 257-entry table, allowing up to 256 interpolation levels between entries. Example 2 (see Figure 5-17 reduces table length for the same data to four entries. Example 3 (see Figure 5-18) demonstrates using an 8-bit independent variable with an instruction.

Two additional examples show how TBLSN can reduce cumulative error when multiple table lookup and interpolation operations are used in a calculation. Example 4 demonstrates adding the results of three table interpolations. Example 5 illustrates using TBLSN in surface interpolation.

5.4.4.1 TABLE EXAMPLE 1: STANDARD USAGE. The table consists of 257 word entries. As shown in Figure 5-16, the function is linear within the range 32768 $\leqslant X \leqslant$ 49152. Table entries within this range are as given in Table 5-15.

Table 5-15. Standard Usage Entries

Entry Number	X Value	Y Value
128*	32768	1311
162	41472	1659
163	41728	1669
164	41984	1679
165	42240	1690
192*	49152	1966

*These values are the end points of the range. All entries between these points fall on the line.

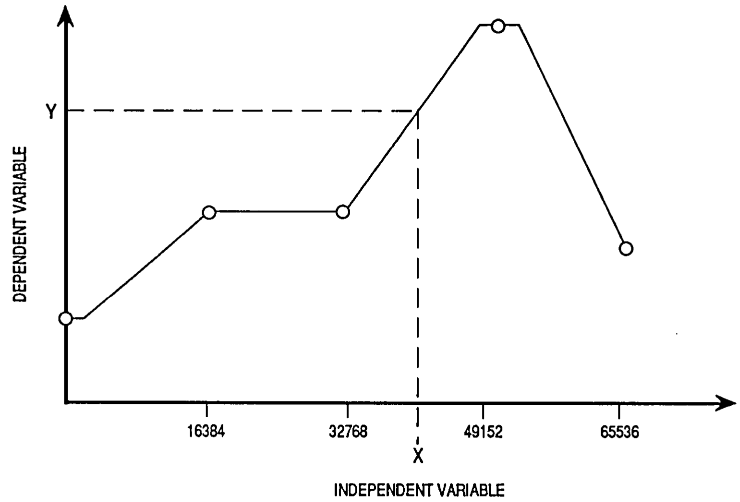


Figure 5-16. Table Example 1

The table instruction is executed with the following bit pattern in Dx:

31 16	15															0
NOT USED	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0

Table Entry Offset ➧ Dx [8:15] = \$A3 = 163
Interpolation Fraction ➧ Dx [0:7] = \$80 = 128

Using this information, the table instruction calculates dependent variable Y:

$$Y = 1669 + (128\ (1679 - 1669))/256 = 1674$$

5.4.4.2 TABLE EXAMPLE 2: COMPRESSED TABLE. In Example 2 (see Figure 5-17), the data from Example 1 has been compressed by limiting the maximum value of the independent variable. Instead of the range 0 ≤ X = 65535, X is limited to 0 ≤ X ≤ 1023. The table has been compressed to only 5 entries, but up to 256 levels of interpolation are allowed between entries.

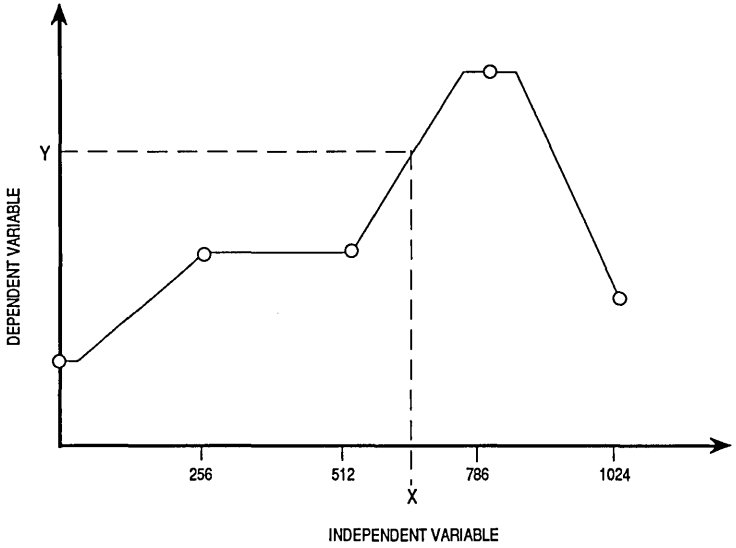


Figure 5-17. Table Example 2

CAUTION

Extreme table compression with many levels of interpolation is possible only with highly linear functions. The table entries within the range of interest are given in Table 5-16.

Table 5-16. Compressed Table Entries

Entry Number	X Value	Y Value
2	512	1311
3	786	1966

Since the table is reduced from 257 to 5 entries, independent variable X must be scaled appropriately. In this case, the scaling factor is 64, and the scaling is done by a single instruction:

```
LSR.W #6,Dx
```

Thus, Dx now contains the following bit pattern:

31 … 16	15															0
NOT USED	0	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0

Table Entry Offset Dx [8:15] = \$02 = 2
Interpolation Fraction Dx [0:7] = \$8E = 142

Using this information, the table instruction calculates dependent variable Y:

$$Y = 1331 + (142\ (1966 - 1311))/256 = 1674$$

The function chosen for Examples 1 and 2 is linear between data points. Had another function been used, interpolated values might not have been identical.

5.4.4.3 TABLE EXAMPLE 3: 8-BIT INDEPENDENT VARIABLE. This example shows how to use a table instruction within an interpolation subroutine. Independent variable X is calculated as an 8-bit value, allowing 16 levels of interpolation on a 17-entry table. X is passed to the subroutine, which returns an 8-bit result. The subroutine uses the data given in Table 5-17, based on the function shown in Figure 5-18.

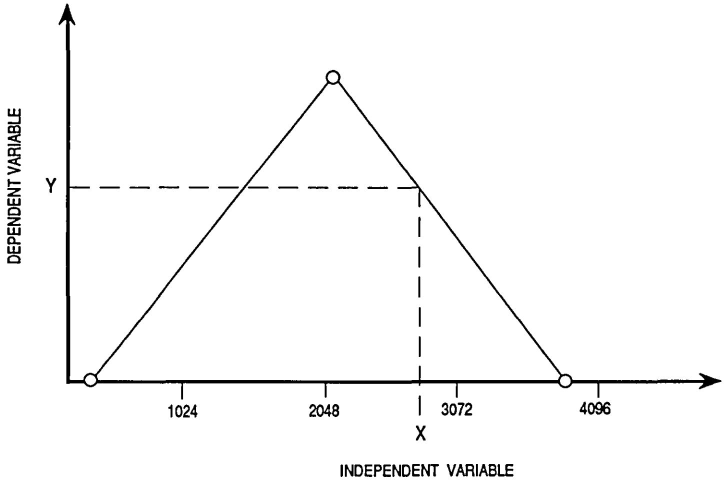


Figure 5-18. Table Example 3

Table 5-17. 8-Bit Independent Variable Entries

X (Subroutine)	X (Instruction)	Y
0	0	0
1	256	16
2	512	32
3	768	48
4	1024	64
5	1280	80
6	1536	96
7	1792	112
8	2048	128
9	2304	112
10	2560	96
11	2816	80
12	3072	64
13	3328	48
14	3584	32
15	3840	16
16	4096	0

The first column is the value passed to the subroutine, the second column is the value expected by the table instruction, and the third column is the result returned by the subroutine.

The following value has been calculated for independent variable X:

31 16	15															0
NOT USED	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1

Since X is an 8-bit value, the upper four bits are used as a table offset, and the lower four bits are used as an interpolation fraction. The following results are obtained from the subroutine:

Table Entry Offset ➧ Dx [4:7] = \$B = 11
Interpolation Fraction ➧ Dx [0:3] = \$D = 13

Thus, Y is calculated as follows:

$$Y = 80 + (13\ (64 - 80))/16 = 67$$

If the 8-bit value for X were used directly by the table instruction, interpolation would be incorrectly performed between entries 0 and 1. Data must be shifted to the left four places before use:

LSL.W #4, Dx

The new range for X is $0 \leqslant X \leqslant 4096$; however, since a left shift fills the least significant digits of the word with zeros, the interpolation fraction can only have one of 16 values.

After the shift operation, Dx contains the following value:

31 16	15															0
NOT USED	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0

Execution of the table instruction using the new value in Dx yields:

Table Entry Offset ➧ Dx [8:15] = \$0B = 11
Interpolation Fraction ➧ Dx [0:7] = \$D0 = 208

Thus, Y is calculated as follows:

$$Y = 80 + (208\ (64 - 80))/256 = 67$$

5.4.4.4 TABLE EXAMPLE 4: MAINTAINING PRECISION. In this example, three table lookup and interpolation (TLI) operations are performed, and the results are summed. The calculation is done once with the result of each TLI rounded before addition and once with only the final result rounded. Assume that the result of the three interpolations are as follows (a "." indicates the binary radix point).

TLI # 1	0010 0000 . 0111 0000
TLI # 2	0011 1111 . 0111 0000
TLI # 3	0000 0001 . 0111 0000

First, the results of each TLI are rounded with the TBLS round-to-nearest-even algorithm. The following values would be returned by TBLS:

TLI # 1	0010 0000 .
TLI # 2	0011 1111 .
TLI # 3	0000 0001 .

Summing, the following result is obtained:

$$\begin{array}{r} 0010\ 0000\ . \\ 0011\ 1111\ . \\ \underline{0000\ 0001\ .} \\ 0110\ 0000\ . \end{array}$$

Now, using the same TLI results, the sum is first calculated and then rounded according to the same algorithm:

$$\begin{array}{r} 0010\ 0000\ .\ 0111\ 0000 \\ 0011\ 1111\ .\ 0111\ 0000 \\ \underline{0000\ 0001\ .\ 0111\ 0000} \\ 0110\ 0001\ .\ 0101\ 0000 \end{array}$$

Rounding yields:

$$0110\ 0001\ .$$

The second result is preferred. The following code sequence illustrates how addition of a series of table interpolations can be performed without loss of precision in the intermediate results:

```
L0:
 TBLSN.B  ⟨ea⟩, Dx
 TBLSN.B  ⟨ea⟩, Dx
 TBLSN.B  ⟨ea⟩, Dl
 ADD.L    Dx, Dm     Long addition avoids problems with carry
 ADD.L    Dm, Dl
 ASR.L    #8, Dl     Move radix point
 BCC.B    L1         Fraction MSB in carry
 ADDQ.B   #1, Dl
L1: . . .
```

5.4.4.5 TABLE EXAMPLE 5: SURFACE INTERPOLATIONS. The various forms of table can be used to perform surface (3D) TLIs. However, since the calculation must be split into a series of 2D TLIs, losing precision in the intermediate results is possible. The following code sequence, incorporating both TBLS and TBLSN, eliminates this possibility:

```
L0:
 MOVE.W   Dx, Dl      Copy entry number and fraction number
 TBLSN.B  ea, Dx
 TBLSN.B  ea, Dl
 TBLS.W   Dx:Dl, Dm   Surface interpolation, with round
 ASR.L    #8, Dm      Read just the result
 BCC.B    L1          No round necessary
 ADDQ.B   #1, Dl      Half round up
L1: . . .
```

Before execution of this code sequence, Dx must contain fraction and entry numbers for the two TLIs, and Dm must contain the fraction for surface interpolation. The ⟨ea⟩ fields in the TBLSN instructions point to consecutive columns in a 3D table. The TBLS size parameter must be word if the TBLSN size parameter is byte, and must be long word if TBLSN is word. Increased size is necessary because a larger number of significant digits is needed to accommodate the scaled fractional results of the 2D TLI.

5.4.5 Nested Subroutine Calls

The LINK instruction pushes an address onto the stack, saves the stack address at which the address is stored, and reserves an area of the stack for use. Using this instruction in a series of subroutine calls will generate a linked list of stack frames.

The UNLK instruction removes a stack frame from the end of the list by loading an address into the stack pointer and pulling the value at that address from the stack. When the instruction operand is the address of the link address at the bottom of a stack frame, the effect is to remove the stack frame from both the stack and the linked list.

5.4.6 Pipeline Synchronization with the NOP Instruction

Although the no operation (NOP) instruction performs no visible operation, it does force synchronization of the instruction pipeline since all previous instructions must complete execution before the NOP begins.

5.5 PROCESSING STATES

This subsection describes the processing states of the CPU32. It includes a functional description of the bits in the supervisor portion of the status register and an overview of processor response to exception conditions.

5.5.1 State Transitions

The processor is in normal, background, or exception processing state unless halted.

When the processor fetches instructions and operands or executes instructions, it is in the normal processing state. The stopped state, which the processor enters when a STOP or LPSTOP instruction is executed, is a special case of the normal state in which no further bus cycles are generated.

Background state is an alternate operational mode used for system debugging. Refer to **5.7 DEVELOPMENT SUPPORT** for more information.

Exception processing refers specifically to the transition from normal processing of a program to normal processing of system routines, interrupt routines, and other exception handlers. Exception processing includes the stack operations, the exception vector fetch, and the filling of the instruction pipeline caused by an exception. Exception processing ends when execution of an exception handler routine begins. Refer to **5.6 EXCEPTION PROCESSING** for comprehensive information.

A catastrophic system failure occurs if the processor detects a bus error or generates an address error while in the exception processing state. This type of failure halts the processor. For example, if a bus error occurs during exception processing caused by a bus error, the CPU32 assumes that the system is not operational and halts. Only a reset can restart a halted processor. The halted condition should not be confused with the stopped condition. After the processor executes a STOP or LPSTOP instruction, execution of instructions can resume when a trace, interrupt, or reset exception occurs.

5.5.2 Privilege Levels

To protect system resources, the processor can operate with either of two levels of access — user or supervisor. Supervisor level is more privileged than user level. All instructions are available at the supervisor level, but execution of some instructions is not permitted at the user level. There are separate stack pointers for each level. The S-bit in the status register indicates privilege level and determines which stack pointer is used for stack operations. The processor identifies each bus access (supervisor or user mode) via function codes to enforce supervisor and user access levels.

In a typical system, most programs execute at the user level. User programs can access only their own code and data areas and are restricted from accessing other information. The operating system executes at the supervisor privilege level, has access to all resources, performs the overhead tasks for the user level programs, and coordinates their activities.

5.5.2.1 SUPERVISOR PRIVILEGE LEVEL. If the S-bit in the status register is set, supervisor privilege level applies, and all instructions are executable. The bus cycles generated for instructions executed in supervisor level are normally classified as supervisor references, and the values of the function codes on FC3–FC0 refer to supervisor address spaces.

All exception processing is performed at the supervisor level. All bus cycles generated during exception processing are supervisor references, and all stack accesses use the supervisor stack pointer.

Instructions that have important system effects can only be executed at supervisor level. For instance, user programs are not permitted to execute STOP, LPSTOP, or RESET instructions. To prevent a user program from gaining privileged access, except in a controlled manner, instructions that can alter the S-bit in the status register are privileged. The TRAP #n instruction provides controlled user access to operating system services.

5.5.2.2 USER PRIVILEGE LEVEL. If the S-bit in the status register is cleared, the processor executes instructions at the user privilege level. The bus cycles for an instruction executed at the user privilege level are classified as user references, and the values of the function codes on FC3–FC0 specify user address spaces. While the processor is at the user level, implicit references to the system stack pointer and explicit references to address register seven (A7) refer to the user stack pointer (USP).

5.5.2.3 CHANGING PRIVILEGE LEVEL. To change from user privilege level to supervisor privilege level, a condition that causes exception processing must occur. When exception processing begins, the current values in the status register, including the S-bit , are saved on the supervisor stack, and then the

S-bit is set, enabling supervisory access. Execution continues at supervisor level until exception processing is complete.

To return to user access level, a system routine must execute one of the following instructions: MOVE to SR, ANDI to SR, EORI to SR, ORI to SR, or RTE. These instructions execute only at supervisor privilege level and can modify the S-bit of the status register. After these instructions execute, the instruction pipeline is flushed, then refilled from the appropriate address space.

The RTE instruction causes a return to a program that was executing when an exception occurred. When RTE is executed, the exception stack frame saved on the supervisor stack can be restored in either of two ways.

If the frame was generated by an interrupt, breakpoint, trap, or instruction exception, the status register and program counter are restored to the values saved on the supervisor stack, and execution resumes at the restored program counter address, with access level determined by the S-bit of the restored status register.

If the frame was generated by a bus error or an address error exception, the entire processor state is restored from the stack.

5.6 EXCEPTION PROCESSING

An exception is a special condition that pre-empts normal processing. Exception processing is the transition from normal mode program execution to normal mode execution of a routine that deals with an exception. Efficient exception processing is a critical constraint on system function. This section discusses system resources, exception processing sequence, and specific features of individual exception processing routines.

5.6.1 Exception Vectors

An exception vector is the address of a routine that handles an exception. The vector base register (VBR) contains the base address of a 1024-byte exception vector table, which consists of 256 exception vectors — 64 vectors are defined by the processor, and 192 vectors are reserved for user definition as interrupt vectors. Except for the reset vector, each vector in the table is one long word in length. The reset vector is two long words in length. Refer to Table 5-18 for information on vector assignment.

CAUTION

Since there is no protection on the 64 processor-defined vectors, external devices can access vectors reserved for internal purposes. This practice is strongly discouraged.

All exception vectors, except the reset vector, are located in supervisor data space. The reset vector is located in supervisor program space. Only the initial reset vector is fixed in the processor memory map. When initialization is complete, there are no fixed assignments. Since the VBR stores the vector table base address, the table can be located anywhere in memory. It can also be dynamically relocated for each task executed by an operating system.

Each vector is assigned an 8-bit number. Vector numbers for some exceptions are obtained from an external device; others are supplied by the processor. The processor multiplies the vector number by four to calculate vector offset, then adds the offset to the contents of the VBR. The sum is the memory address of the vector.

Table 5-18. Exception Vector Assignments

Vector Number	Vector Offset			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial Stack Pointer
1	4	004	SP	Reset: Initial Program Counter
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK, CHK2 Instructions
7	28	01C	SD	TRAPcc, TRAPV Instructions
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12	48	030	SD	Hardware Breakpoint
13	52	034	SD	(Reserved for Coprocessor Protocol Violation)
14	56	038	SD	Format Error and Uninitialized Interrupt
15	60	03C	SD	Format Error and Uninitialized Interrupt
16–23	64	040	SD	(Unassigned, Reserved)
	92	05C		—
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32–47	128	080	SD	TRAP Instruction Vectors (0–15)
	188	0BC		—
48–58	192	0C0	SD	(Reserved for Coprocessor)
	232	0E8		—
59–63	236	0EC	SD	(Unassigned, Reserved)
	252	0FC		—
64–255	256	100	SD	User-Defined Vectors — (192)
	1020	3FC		

5.6.1.1 TYPES OF EXCEPTIONS. An exception can be due to internal or external causes.

An internal exception can be generated by an instruction or by an error. The TRAP, TRAPcc, TRAPV, BKPT, CHK, CHK2, RTE, and DIV instructions can cause exceptions during normal execution. Illegal instructions, instruction fetches from odd addresses, word or long-word operand accesses from odd addresses, and privilege violations also cause internal exceptions.

Sources of external exception include interrupts, breakpoints, bus errors, and reset requests. Interrupts are requests for processor action from peripheral devices. Breakpoints are used to support development equipment. Bus error and reset are used for access control and processor restart.

5.6.1.2 EXCEPTION PROCESSING SEQUENCE. For all exceptions other than a reset exception, exception processing occurs in the following sequence. Refer to **5.6.2.1 RESET** for details of reset processing.

As exception processing begins, the processor makes an internal copy of the status register. After the copy is made, the processor state bits in the status register are changed — the S-bit is set, establishing supervisor access level, and bits T1 and T0 are cleared, disabling tracing. For reset and interrupt exceptions, the interrupt priority mask is also updated.

Next, the exception number is obtained. For interrupts, the number is fetched from CPU space $F (the bus cycle is an interrupt acknowledge). For all other exceptions, internal logic provides a vector number.

Next, current processor status is saved. An exception stack frame is created and placed on the supervisor stack. All stack frames contain copies of the status register and the program counter for use by the RTE instruction. The type of exception and the context in which the exception occurs determine what other information is stored in the stack frame.

Finally, the processor prepares to resume normal execution of instructions. The exception vector offset is determined by multiplying the vector number by four, and the offset is added to the contents of the VBR to determine displacement into the exception vector table. The exception vector is loaded into the program counter. If no other exception is pending, the processor will resume normal execution at the new address in the program counter.

5.6.1.3 EXCEPTION STACK FRAME. During exception processing, the most volatile portion of the current context is saved on the top of the supervisor stack. This context is organized in a format called the exception stack frame.

The exception stack frame always includes the contents of status register and program counter at the time the exception occurred. To support generic handlers, the processor also places the vector offset in the exception stack frame and marks the frame with a format code. The format field allows an RTE instruction to identify stack information so that it can be properly restored.

The general form of the exception stack frame is illustrated in Figure 5-19. Although some formats are peculiar to a particular M68000 Family processor, format 0000 is always legal and always indicates that only the first four words of a frame are present. See **5.6.4 CPU32 Stack Frames** for a complete discussion of exception stack frames.

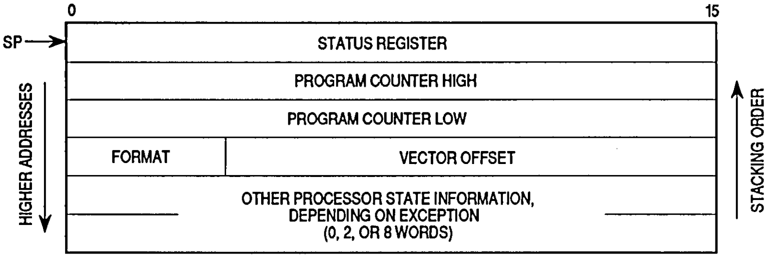


Figure 5-19. Exception Stack Frame

5.6.1.4 MULTIPLE EXCEPTIONS. Each exception has been assigned a priority based on its relative importance to system operation. Priority assignments are shown in Table 5-19. Group zero (0) exceptions have the highest priorities. Group four (4) exceptions have the lowest priorities. Exception processing for exceptions that occur simultaneously is done by priority, from highest to lowest.

Table 5-19. Exception Priority Groups

Group/ Priority	Exception and Relative Priority	Characteristics
0	Reset	Aborts all processing (instruction or exception); does not save old context.
1.1 1.2	Address Error Bus Error	Suspends processing (instruction or exception); saves internal context.
2	BKPT#n, CHK, CHK2, Divide by Zero, RTE, TRAP#n, TRAPcc, TRAPV	Exception processing is a part of instruction execution.
3	Illegal Instruction, Line A, Unimplemented Line F, Privilege Violation	Exception processing begins before instruction is executed.
4.1 4.2 4.3	Trace Hardware Breakpoint Interrupt	Exception processing begins when current instruction or previous exception processing is completed.

It is important to be aware of the difference between exception processing mode and execution of an exception handler. Each exception has an assigned vector, which points to an associated handler routine. Exception processing includes steps described in **5.6.1.2 EXCEPTION PROCESSING SEQUENCE** but does not include execution of handler routines, which is done in normal mode.

When the CPU32 completes exception processing, it is ready to begin either exception processing for a pending exception or execution of a handler routine. Priority assignment governs the order in which exception processing occurs, not the order in which exception handlers are executed.

As a general rule, when simultaneous exceptions occur, the handler routines for lower priority exceptions are executed before the handler routines for higher priority exceptions. For example, consider the arrival of an interrupt during execution of a TRAP instruction, while tracing is enabled. Trap exception processing (2) is done first, followed immediately by exception processing for the trace (4.1), and then by exception processing for the interrupt (4.3). Each exception places a new context on the stack. When the processor resumes normal instruction execution, it is vectored to the interrupt handler, which returns to the trace handler, which returns to the trap handler.

There are special cases to which the general rule does not apply. The reset exception will always be the first exception handled, since reset clears all other exceptions. It is also possible for high-priority exception processing to begin before low-priority exception processing is complete. For example, if a bus error occurs during trace exception processing, the bus error will be processed and handled before trace exception processing is completed.

5.6.2 Processing of Specific Exceptions

The following paragraphs provide details concerning sources of specific exceptions, how each arises, and how each is processed.

5.6.2.1 RESET. Assertion of $\overline{\text{RESET}}$ by external hardware or assertion of the internal $\overline{\text{RESET}}$ signal by an internal module causes a reset exception. The reset exception has the highest priority of any exception. Reset is used for system initialization and for recovery from catastrophic failure. The reset exception aborts any processing in progress when it is recognized, and that processing cannot be recovered. Reset performs the following operations:

1. Clears T0 and T1 in the status register to disable tracing.
2. Sets the S-bit in the status register to establish supervisor privilege.

3. Sets the interrupt priority mask to the highest priority level (%111).
4. Initializes the vector base register to zero ($00000000).
5. Generates a vector number to reference the reset exception vector.
6. Loads the first long word of the vector into the interrupt stack pointer.
7. Loads the second long word of the vector into the program counter.

Figure 5-20 is a flowchart of the reset exception.

After initial instruction prefetches, normal program execution begins at the address in the program counter. The reset exception does not save the value of either the program counter or the status register.

If a bus error or address error occurs during a reset exception processing sequence, a double bus fault occurs. The processor halts, and the $\overline{\text{HALT}}$ signal is asserted to indicate the halted condition.

Execution of the RESET instruction does not cause a reset exception nor affect any internal register, but it does cause the CPU32 to assert the $\overline{\text{RESET}}$ signal, resetting all internal and external peripherals.

5.6.2.2 BUS ERROR. A bus error exception occurs when an assertion of the $\overline{\text{BERR}}$ signal is acknowledged. $\overline{\text{BERR}}$ can be asserted by one of three sources:

1. External logic by assertion of the $\overline{\text{BERR}}$ input pin
2. Direct assertion of the internal $\overline{\text{BERR}}$ signal by an internal module
3. Direct assertion of the internal $\overline{\text{BERR}}$ signal by the on-chip bus monitor after detecting a no-response condition

Bus error exception processing begins when the processor attempts to use information from an aborted bus cycle.

When the aborted bus cycle is an instruction prefetch, the processor will not initiate exception processing unless the prefetched information is used. For example, if a branch instruction flushes an aborted prefetch, that word is not accessed, and no exception occurs.

When the aborted bus cycle is a data access, the processor initiates exception processing immediately, except in the case of released operand writes. Released write bus errors are delayed until the next instruction boundary or until another operand access is attempted.

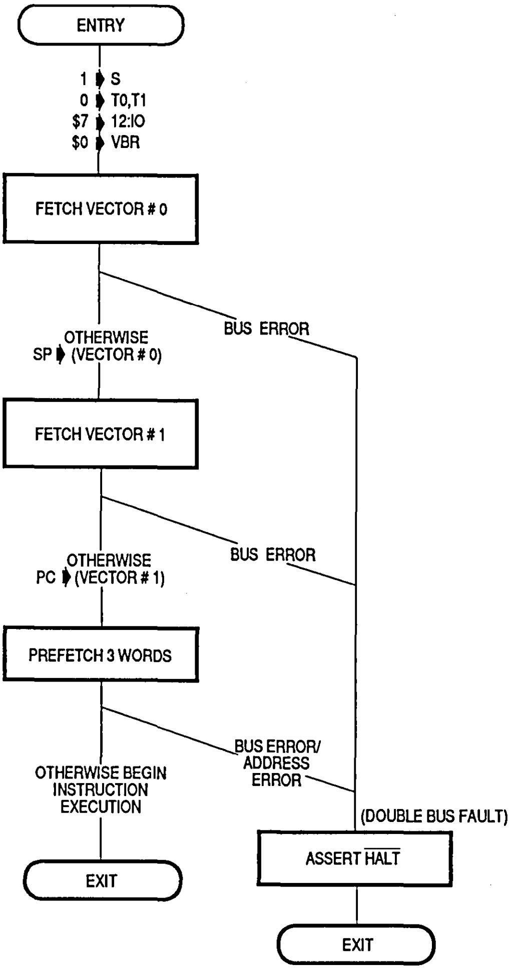


Figure 5-20. Reset Operation Flowchart

Exception processing for bus error exceptions follows the regular sequence, but context preservation is more involved than for other exceptions because a bus exception can be initiated while an instruction is executing. Several bus error stack format organizations are utilized to provide additional information regarding the nature of the fault.

First, any register altered by a faulted-instruction effective address calculation is restored to its initial value. Then a special status word (SSW) is placed on the stack. The SSW contains specific information about the aborted access —

size, type of access (read or write), bus cycle type, and function code are saved. Finally, fault address, bus error exception vector number, program counter value, and a copy of the status register are saved.

If a bus error occurs during exception processing for a bus error, an address error, a reset, or while the processor is loading stack information during RTE execution, the processor halts. This simplifies isolation of catastrophic system failure by preventing processor interaction with stacks and memory. Only assertion of $\overline{\text{RESET}}$ can restart a halted processor.

5.6.2.3 ADDRESS ERROR.

Address error exceptions occur when the processor attempts to access an instruction, word operand, or long-word operand at an odd address. The effect is much the same as an internally generated bus error.

Address error exception processing begins when the processor attempts to use information from the aborted bus cycle.

If the aborted cycle is a data space access, exception processing begins when the processor attempts to use the data, except in the case of a released operand write. Released write exceptions are delayed until the next instruction boundary or attempted operand access.

An address exception on a branch to an odd address is delayed until the program counter is changed. No exception occurs if the branch is not taken.

The exception processing sequence is the same as that for bus error, except that the vector number refers to the address exception vector. The fault address and return program counter value placed in the exception stack frame are the odd address. The current instruction program counter points to the instruction that caused the exception.

If an address error occurs during exception processing for a bus error, another address error, or a reset, the processor halts.

5.6.2.4 INSTRUCTION TRAPS.

Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution or from use of specific trapping instructions. Traps are generally used to handle abnormal conditions that arise in control routines.

The TRAP instruction, which always forces an exception, is useful for implementing system calls for user programs. The TRAPcc, TRAPV, CHK, and CHK2 instructions force exceptions when a program detects a run-time error. The DIVS and DIVU instructions force an exception if a division operation is attempted with a divisor of zero.

Exception processing for traps follows the regular sequence. If tracing is enabled when an instruction that causes a trap begins execution, a trace exception will be generated by the instruction, but the trap handler routine will not be traced (the trap exception will be processed first, then the trace exception).

The vector number for the TRAP instruction is internally generated; part of the number comes from the instruction itself. The trap vector number, program counter value, and a copy of the status register are saved on the supervisor stack. The saved program counter value is the address of the instruction that follows the instruction which generated the trap. For all instruction traps other than TRAP, a pointer to the instruction causing the trap is also saved in the fifth and sixth words of the exception stack frame.

5.6.2.5 SOFTWARE BREAKPOINTS. To support hardware emulation, the CPU32 must provide a means of inserting breakpoints into target code and of clearly announcing when a breakpoint is reached.

The MC68000 and MC68008 can detect an illegal instruction inserted at a breakpoint when the processor fetches from the illegal instruction exception vector location. Since the VBR on the CPU32 allows relocation of exception vectors, the exception vector address is not a reliable indication of a breakpoint. CPU32 breakpoint support is provided by extending the function of a set of illegal instructions ($4848–$484F).

When a breakpoint instruction is executed, the CPU32 performs a read from CPU space $0, at a location corresponding to the breakpoint number. If this bus cycle is terminated by $\overline{\text{BERR}}$, the processor performs illegal instruction exception processing. If the bus cycle is terminated by $\overline{\text{DSACK}}$, the processor uses the data returned to replace the breakpoint in the instruction pipeline and begins execution of that instruction. See **SECTION 3 BUS OPERATION** for a description of CPU space operations.

5.6.2.6 HARDWARE BREAKPOINTS. The CPU32 recognizes hardware breakpoint requests. Hardware breakpoint requests do not force immediate exception processing but are left pending. An instruction breakpoint is not made pending until the instruction corresponding to the request is executed.

A pending breakpoint can be acknowledged between instructions or at the end of exception processing. To acknowledge a breakpoint, the CPU performs a read from CPU space $0 at location $1E (see **SECTION 3 BUS OPERATION**).

If the bus cycle terminates normally, instruction execution continues with the next instruction as if no breakpoint request occurred. If the bus cycle is terminated by $\overline{\text{BERR}}$, the CPU begins exception processing.

Exception processing follows the regular sequence. Vector number 12 (offset $30) is internally generated. The program counter of the currently executing instruction, the program counter of the next instruction to execute, and a copy of the status register are saved on the supervisor stack. Data returned during this bus cycle is ignored.

5.6.2.7 FORMAT ERROR. The processor checks certain data values for control operations. The validity of the stack format code and, in the case of a bus cycle fault format, the version number of the processor that generated the frame are checked during execution of the RTE instruction. This check ensures that the program does not make erroneous assumptions about information in the stack frame.

If the format of the control data is improper, the processor generates a format error exception. This exception saves a four-word format exception frame and then vectors through vector table entry number 14. The stacked program counter is the address of the RTE instruction that discovered the format error.

5.6.2.8 ILLEGAL OR UNIMPLEMENTED INSTRUCTIONS. An instruction is illegal if it contains a word bit pattern that does not correspond to the bit pattern of the first word of a legal CPU32 instruction, if it is a MOVEC instruction that contains an undefined register specification field in the first extension word, or if it contains an indexed addressing mode extension word with bits [5:4] = 00 or bits [3:0] ≠ 0000.

If an illegal instruction is fetched during instruction execution, an illegal instruction exception occurs. This facility allows the operating system to detect program errors or to emulate instructions in software.

Word patterns with bits [15:12] = 1010 (referred to as A-line opcodes) are unimplemented instructions. A separate exception vector (vector 10, offset $28) is given to unimplemented instructions to permit efficient emulation.

Word patterns with bits [15:12] = 1111 (referred to as F-line opcodes) are used for M68000 Family instruction set extensions. They can generate an unimplemented instruction exception caused by the first extension word of the instruction or by the addressing mode extension word. A separate F-line emulation vector (vector 11, offset $2C) is used for the exception vector.

All unimplemented instructions are reserved for use by Motorola for enhancements and extensions to the basic M68000 architecture. Opcode pattern $4AFC is defined to be illegal on all M68000 Family members. Those customers requiring the use of an unimplemented opcode for synthesis of "custom instructions," operating system calls, etc. should use this opcode.

Exception processing for illegal and unimplemented instructions is similar to that for traps. The instruction is fetched and decoding is attempted. When the processor determines that execution of an illegal instruction is being attempted, exception processing begins. No registers are altered.

Exception processing follows the regular sequence. The vector number is generated to refer to the illegal instruction vector or, in the case of an unimplemented instruction, to the corresponding emulation vector. The illegal instruction vector number, current program counter, and a copy of the status register are saved on the supervisor stack, with the saved value of the program counter being the address of the illegal or unimplemented instruction.

5.6.2.9 PRIVILEGE VIOLATIONS. To provide system security, certain instructions can be executed only at the supervisor access level. An attempt to execute one of these instructions at the user level will cause an exception. The privileged exceptions are as follows:

- AND Immediate to SR
- EOR Immediate to SR
- LPSTOP
- MOVE from SR
- MOVE to SR
- MOVE USP
- MOVEC
- MOVES
- OR Immediate to SR
- RESET
- RTE
- STOP

Exception processing for privilege violations is nearly identical to that for illegal instructions. The instruction is fetched and decoded. If the processor determines that a privilege violation has occurred, exception processing begins before instruction execution. Exception processing follows the regular sequence. The vector number is generated to reference the privilege violation vector. Privilege violation vector number, current program counter, and status register are saved on the supervisor stack. The saved program counter value is the address of the first word of the instruction causing the privilege violation.

5.6.2.10 TRACING. To aid in program development, M68000 processors include a facility to allow tracing of instruction execution. CPU32 tracing also has the ability to change program flow. In trace mode, a trace exception is generated after each instruction executes, allowing a debugging program to monitor the execution of a program under test. The T1 and T0 bits in the supervisor portion of the status register are used to control tracing.

When T[1:0]=00, tracing is disabled, and instruction execution proceeds normally (see Table 5-20).

Table 5-20. Tracing Control

T1	T0	Tracing Function
0	0	No Tracing
0	1	Trace on Change of Flow
1	0	Trace on Instruction Execution
1	1	(Undefined; Reserved)

When T[1:0]=01 at the beginning of instruction execution, a trace exception is generated if the program counter changes sequence during execution. All branches, jumps, subroutine calls, returns, and status register manipulations can be traced in this way. If a branch is not taken, an exception is not generated.

When T[1:0]=10 at the beginning of instruction execution, a trace exception is generated when execution is complete. If the instruction is not executed, either because an interrupt is taken or because the instruction is illegal, unimplemented, or privileged, an exception is not generated.

At the present time, T[1:0]=11 is an undefined condition. It is reserved by Motorola for future use.

Exception processing for trace starts at the end of normal processing for the traced instruction and before the start of the next instruction. Exception processing follows the regular sequence (tracing is disabled so that the trace exception itself is not traced). A vector number is generated to reference the trace exception vector. The address of the instruction that caused the trace exception, the trace exception vector offset, the current program counter, and a copy of the status register are saved on the supervisor stack. The saved value of the program counter is the address of the next instruction to be executed.

A trace exception can be viewed as an extension to the function of any instruction. If a trace exception is generated by an instruction, the execution of that instruction is not complete until the trace exception processing associated with it is also complete.

If an instruction is aborted by a reset, bus error, or address error exception, trace exception is deferred until normal execution resumes. An RTE from a bus error or address error will not be traced because of the possibility of continuing the instruction from the fault.

If an instruction is executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If an instruction forces an exception, the forced exception is processed before the trace exception.

If an instruction is executed and a breakpoint is pending upon completion of the instruction, the trace exception is processed before the breakpoint.

If an attempt is made to execute an illegal, unimplemented, or privileged instruction while tracing is enabled, no trace exception will occur because the instruction is not executed. This condition is particularly important to an emulation routine that performs an instruction function, adjusts the stacked program counter to beyond the unimplemented instruction, and then returns. The status register on the stack must be checked to determine if tracing is on before the return is executed. If tracing is on, trace exception processing must be emulated so that the trace exception handler can account for the emulated instruction.

Tracing affects normal operation of the STOP and LPSTOP instructions. If either instruction begins execution with T1 set, a trace exception will be taken after the instruction loads the status register. Upon return from the trace handler routine, execution will continue with the instruction following STOP (LPSTOP), and the processor will not enter the stopped condition.

5.6.2.11 INTERRUPTS. There are seven levels of interrupt priority and 192 assignable interrupt vectors within each exception vector table. Judicious use of multiple vector tables and hardware chaining will permit a virtually unlimited number of peripherals to interrupt the processor.

Interrupt recognition and subsequent processing are based on internal interrupt request signals ($\overline{\text{IRQ7}}$–$\overline{\text{IRQ1}}$) and the current priority set in status register priority mask I[2:0]. Interrupt request level zero ($\overline{\text{IRQ7}}$–$\overline{\text{IRQ1}}$ negated) indicates that no service is requested. When an interrupt of level one through six is requested via $\overline{\text{IRQ6}}$–$\overline{\text{IRQ1}}$, the processor compares the request level with the interrupt mask to determine whether the interrupt should be processed. Interrupt requests are inhibited for all priority levels less than or equal to the current priority. Level seven interrupts are nonmaskable.

$\overline{\text{IRQ7}}$–$\overline{\text{IRQ1}}$ are synchronized and debounced by input circuitry on consecutive rising edges of the processor clock. To be valid, an interrupt request must be held constant for at least two consecutive clock periods.

Interrupt requests do not force immediate exception processing but are left pending. A pending interrupt is detected between instructions or at the end of exception processing — all interrupt requests must be held asserted until they are acknowledged by the CPU. If the priority of the interrupt is greater than the current priority level, exception processing begins.

Exception processing follows the regular sequence until after tracing is suppressed. Priority level is then set to the level of the interrupt, and the processor fetches a vector number from the interrupting device (CPU space $F). The fetch bus cycle is classified as an interrupt acknowledge, and the encoded level number of the interrupt is placed on the address bus.

If an interrupting device requests automatic vectoring, the processor generates a vector number (25–31) determined by the interrupt level number.

If the response to the interrupt acknowledge bus cycle is a bus error, the interrupt is taken to be spurious, and the spurious interrupt vector number (24) is generated.

The exception vector number, program counter, and status register are saved on the supervisor stack. The saved value of the program counter is the address of the instruction that would have executed had the interrupt not occurred.

Priority level seven interrupt is a special case. Level seven interrupts are nonmaskable. These requests are transition sensitive to eliminate redundant servicing and concomitant stack overflow. Transition sensitive means that the level seven input must change state before the CPU will detect an interrupt.

A level seven interrupt is generated each time the interrupt request level changes to level seven, and each time the priority mask changes from seven to a lower number while the request level remains at seven.

Many M68000 peripherals provide for programmable interrupt vector numbers to be used in the system interrupt request/acknowledge mechanism. If the vector number is not initialized after reset and if the peripheral must acknowledge an interrupt request, the peripheral should return the uninitialized interrupt vector number (15). Refer to **SECTION 3 BUS OPERATION** for more information on interrupt acknowledge cycles.

5.6.2.12 RETURN FROM EXCEPTION. When exception stacking operations are complete, the processor begins normal mode handler execution for the last exception processed. After the exception handler has executed, the processor must restore the system context in existence prior to the exception. The RTE instruction is designed to accomplish this task.

When RTE is executed, the processor examines the stack frame on top of the supervisor stack to determine if it is valid and determines what type of context restoration must be performed. See **5.6.4 CPU32 Stack Frames** for a description of stack frames.

For a normal four-word frame, the processor updates the status register and program counter with data pulled from the stack, increments the supervisor stack pointer by eight, and resumes normal instruction execution. For a six-word frame, the status register and program counter are updated from the stack, the active supervisor stack pointer is incremented by 12, and normal instruction execution resumes.

For a bus fault frame, the format value on the stack is first checked for validity. In addition, the version number on the stack must match the version number of the processor that is attempting to read the stack frame. The version number is located in the most significant byte (bits [15:8]) of the internal register word at location SP+$14 in the stack frame. A validity check is used to ensure that data in a multiple processor system will be properly interpreted by an RTE instruction.

If a frame is invalid, a format error exception is taken. If it is inaccessible, a bus error exception is taken. Otherwise, the processor reads the entire frame into the proper internal registers, deallocates the stack (12 words), and resumes normal processing. Bus error frames for faults during exception processing require the RTE instruction to rewrite the faulted stack frame. If an error occurs during any of the bus cycles required by rewrite, the processor will halt.

If a format error occurs during RTE execution, the processor will create a normal four-word fault stack frame below the frame that it was attempting to use. If a bus error occurs, a bus cycle stack frame will be created. The faulty stack frame remains intact so that it may be examined and repaired by an exception handler or used by a different type processor (e.g., an MC68010, MC68020, or a future M68000 processor) in a multiprocessor system.

5.6.3 Fault Recovery

There are four phases of recovery from a fault: recognizing the fault, saving the processor state, repairing the fault (if possible), and restoring the processor state. Saving and restoring the processor state are described in the following paragraphs.

The stack contents are identified by the SSW. In addition to identifying the fault type represented by the stack frame, the SSW contains the internal processor state corresponding to the fault.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TP	MV	0	TR	B1	B0	RR	RM	IN	RW	LG	SIZ		FUNC		

TP—BERR Frame Type
MV—MOVEM in Progress
TR—Trace Pending
B1—Breakpoint Channel 1 Pending
B0—Breakpoint Channel 0 Pending
RR—Rerun Write Cycle after RTE
RM—Faulted Cycle Was Read-Modify-Write
IN—Instruction/Other
RW—Read/Write of Faulted Bus Cycle
LG—Original Operand Size Was Long Word
SIZ—Remaining Size of Faulted Bus Cycle
FUNC—Function Code of Faulted Bus Cycle

The TP field defines the class of the faulted bus operation. Two BERR exception frame types are defined to support faults on prefetch and operand accesses and exception frame stacking:

0 = Operand or prefetch bus fault
1 = Exception processing bus fault

MV is set when the operand transfer portion of the MOVEM instruction is in progress at the time of a bus fault. If a prefetch bus fault occurs while refetching the MOVEM opcode and extension word, both the MV and IN bits will be set in the stacked SSW.

0 = MOVEM was not in progress when fault occurred
1 = MOVEM was in progress when fault occurred

TR indicates that a trace exception was pending when a bus error exception was processed. The instruction that generated the trace will not be restarted upon return from the exception handler. This includes MOVEM and released write bus errors indicated by the assertion of either MV or RR in the SSW.

0 = Trace not pending
1 = Trace pending

B1 indicates that a breakpoint exception was pending on channel 1 (external breakpoint source) when a bus error exception was processed. Pending breakpoint status is stacked, regardless of the type of bus error exception being processed.

0 = Breakpoint not pending
1 = Breakpoint pending

B0 indicates that a breakpoint exception was pending on channel 0 (internal breakpoint source) when the bus error exception was processed. Pending breakpoint status is stacked, regardless of the type of bus error exception being processed.

0 = Breakpoint not pending
1 = Breakpoint pending

RR will be set in the SSW if the faulted bus cycle was a released write. If the write is completed (rerun) in the exception handler, the RR bit should be cleared before executing RTE. The bus cycle will be rerun if the RR bit is set upon return from the exception handler.

0 = Faulted cycle was a read, RMW, or unreleased write
1 = Faulted cycle was a released write

Faulted RMW bus cycles set the RM bit in the SSW. This bit is ignored during unstacking.

0 = Faulted cycle was a non-RMW cycle
1 = Faulted cycle was either the read or write of an RMW cycle

Instruction prefetch faults are distinguished from operand (both read and write) faults by the SSW IN bit. If IN is cleared, the error was on an operand cycle; if IN is set, the error was on an instruction prefetch. IN is ignored during unstacking.

0 = Operand
1 = Prefetch

Read and write bus cycles are distinguished by the SSW RW. Read bus cycles will set this bit, and write bus cycles will clear this bit. This bit is reloaded into the bus controller if the RR bit is set during unstacking.

0 = Faulted cycle was an operand write
1 = Faulted cycle was a prefetch or operand read

An original operand size of long word is conveyed in the SSW LG bit. LG is cleared if the original size of the operand was byte or word; SIZ will indicate the original (and remaining) size. LG is set if the original size of the operand was long word; SIZ will indicate the remaining size at the time of the fault. LG is ignored during unstacking.

0 = Original operand size was byte or word
1 = Original operand size was long word

The operand size remaining when the fault was detected is available in the SIZ field of the SSW. This field does not indicate the initial size of the operand. It does not necessarily indicate the proper status of a dynamically sized bus cycle. Dynamic sizing occurs at the external bus and is transparent to the CPU32. The byte size is shown only when the original operand was a byte. This field is reloaded into the bus controller if the RR bit is set during unstacking. The SIZ field is encoded as follows:

00 = Long word
01 = Byte
10 = Word
11 = Unused, reserved

The function code for the faulted cycle is stacked in the FUNC field of the SSW, which is a copy of FC2–FC0 for the faulted bus cycle. This field is reloaded into the bus controller if the RR bit is set during unstacking. All unused bits are stacked as zeros and are ignored during unstacking. Further discussion of the SSW is included in **5.6.3.1 TYPES OF FAULTS.**

5.6.3.1 TYPES OF FAULTS. An efficient implementation of instruction restart dictates that faults on some bus cycles be treated differently than faults on other bus cycles. The CPU32 defines four fault types: released write faults, faults during exception processing, faults during MOVEM operand transfer, and faults on any other bus cycle.

5.6.3.1.1 Type I: Released Write Faults. CPU32 instruction pipelining causes final instruction write to overlap the execution of the following instruction. A write that is overlapped is called a released write. Since the machine context is lost for the instruction that queued, the write is lost as soon as the following instruction starts. It is impossible to restart the faulted instruction.

Released write faults are taken at the next instruction boundary. The stacked program counter is that of the next unexecuted instruction. If a subsequent instruction attempts an operand access while a released write fault is pending, the instruction is aborted, and the write fault is acknowledged. This action prevents stale data from being used by the instruction.

The SSW for a released write fault contains the following bit pattern:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	TR	B1	B0	1	0	0	0	LG	SIZ		FUNC		

TR, B1, and B0 are set if the corresponding exception is pending when the BERR exception is taken. Status regarding the faulted bus cycle is reflected in the SSW LG, SIZ, and FUNC fields.

The remainder of the stack contains the program counter of the next unexecuted instruction, the current status register, the address of the faulted memory location, and the contents of the data buffer which was to be written to memory. This data is written on the stack in the format depicted in Figure 5-21.

5.6.3.1.2 Type II: Prefetch, Operand, RMW, and MOVEP Faults. The majority of BERR exceptions are included in this category — all instruction prefetches, all operand reads, all RMW cycles, and all operand accesses resulting from execution of MOVEP (except the last write of a MOVEP Rn,⟨ea⟩ or the last write of MOVEM, which are type I faults). The TAS, MOVEP, and MOVEM instructions account for all operand writes not considered released.

All type II faults cause an immediate exception that aborts the current instruction. Any registers that were altered as the result of an effective address calculation (i.e., postincrement or predecrement) are restored prior to processing the bus cycle fault.

The SSW for faults in this category contains the following bit pattern:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	B1	B0	0	RM	IN	RW	LG	SIZ		FUNC		

The trace pending bit is always cleared since the instruction will be restarted upon return from the handler. Saving a pending exception on the stack would result in a trace exception being taken prior to restarting the instruction. If the exception handler does not alter the stacked SR trace bits, the trace is requeued when the instruction is started.

The breakpoint pending bits are stacked in the SSW, even though the instruction is restarted upon return from the handler. This procedure avoids problems with bus state analyzer equipment that has been programmed to breakpoint only the first access to a specific location or to count accesses to that location. If this response is not desired, the exception handler can clear the bits before return. The RM, IN, RW, LG, FUNC, and SIZ fields all reflect the type of bus cycle that caused the fault. If the bus cycle was an RMW, the RM bit will be set, and the RW bit will show whether the fault was on a read or write.

5.6.3.1.3 Type III: Faults during MOVEM Operand Transfer. Bus faults that occur as a result of MOVEM operand transfer are classified as type III faults. MOVEM instruction prefetch faults are type II faults.

Type III faults cause an immediate exception that aborts the current instruction. None of the registers altered during execution of the faulted instruction are restored prior to execution of the fault handler. This includes any register predecremented as a result of the effective address calculation or any register overwritten during instruction execution. Since postincremented registers are not updated until the end of an instruction, the register retains its preinstruction value unless overwritten by operand movement.

The SSW for faults in this category contains the following bit pattern:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	TR	B1	B0	RR	0	IN	RW	LG	SIZ		FUNC		

MV is set, indicating that MOVEM should be continued from the point where the fault occurred upon return from the exception handler. TR, B1, and B0 are set if a corresponding exception is pending when the BERR exception is taken. IN is set if a bus fault occurs while refetching an opcode or an extension word during instruction restart. RW, LG, SIZ, and FUNC all reflect the type of bus cycle that caused the fault. All write faults have the RR bit set to indicate that the write should be rerun upon return from the exception handler.

The remainder of the stack frame contains sufficient information to continue MOVEM with the operand transfer following a faulted transfer. The next operand to be transferred, incremented, or decremented is stored in the faulted address location ($08). The stacked transfer counter is set to 16 minus the number of transfers attempted (including the faulted cycle). Refer to Figure 5-21 for the stacking format.

5.6.3.1.4 Type IV: Faults during Exception Processing. The fourth type of fault occurs during exception processing. If the exception is a second address or bus error, the machine halts in the double bus fault condition. However, if the exception is one that causes a four- or six-word stack frame to be written, a bus cycle fault frame is written below the faulted exception stack frame.

The SSW for a fault within an exception contains the following bit pattern:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	TR	B1	B0	0	0	0	1	LG	SIZ		FUNC		

TR, B1, and B0 are set if a corresponding exception is pending when the BERR exception is taken.

The contents of the faulted exception stack frame are included in the bus fault stack frame. The pre-exception status register and the format/vector word of the faulted frame are stacked. The type of exception can be determined from the format/vector word. If the faulted exception stack frame contains six words, the program counter of the instruction that caused the initial exception is also stacked. This data is placed on the stack in the format shown in Figure 5-22. The return address from the initial exception is stacked for RTE .

5.6.3.2 CORRECTING A FAULT. Fault correction methods are discussed in the following paragraphs.

There are two ways to complete a faulted released write (Type I) bus cycle. The first is to use a software handler. The second is to rerun the bus cycle via RTE.

Type II fault handlers must terminate with RTE, but specific requirements must also be met before an instruction is restarted.

There are three varieties of Type III operand fault recovery. The first is completion of an instruction in software. The second is conversion to Type II with restart via RTE. The third is continuation from the fault via RTE.

5.6.3.2.1 (Type I) Completing Released Writes via Software. To complete a bus cycle in software, a handler must first read the SSW function code field to determine the appropriate address space, then access the fault address pointer in that space, and finally transfer data from the stacked image of the output buffer to the fault address.

Because the CPU32 has a 16-bit internal data bus, long operands require two bus accesses. A fault during the second long-operand access causes the LG bit in the SSW to be set. The SIZ field indicates remaining operand size. If operand coherency is important, the complete operand must be rewritten. After a long operand is rewritten, the RR bit must be cleared. Failure to clear the RR bit can cause RTE to rerun the bus cycle. Following rewrite, it is not necessary to adjust the program counter (or other stack contents) before executing RTE.

5.6.3.2.2 (Type I) Completing Released Writes via RTE. An exception handler can use the RTE instruction to complete a faulted bus cycle. When RTE executes, the fault address, data output buffer, program counter, and status register are restored from the stack. Any pending breakpoint or trace exceptions, as indicated by TR, B1, and B0 in the stacked SSW, are requeued during SSW restoration. The RR bit in the SSW is checked during the unstacking operation; if it is set, the RW, FUNC, and SIZ fields are used to rerun the released write cycle.

To maintain long-word operand coherence, stack contents must be adjusted prior to RTE execution. The fault address must be decremented by two if LG is set and SIZ indicates a remaining byte or word. SIZ must be set to long. All other fields should be left unchanged. The bus controller uses the modified fault address and SIZ field to rerun the complete released write cycle.

NOTE

Manipulating the stacked SSW can cause unpredictable results because RTE checks the RR bit but does not check the RW bit. The rerun bus cycle may not be a write or it may not access the same address space as the original bus cycle. If the rerun bus cycle is a read, returned data will be ignored.

5.6.3.2.3 (Type II) Correcting Faults via RTE. Instructions aborted due to a type II fault are restarted upon return from the exception handler. A fault handler must establish safe restart conditions. If a fault is due to a nonresident page in a demand-paged virtual memory configuration, the fault address must be read from the stack, and the appropriate page retrieved. An RTE instruction terminates the exception handler. After unstacking the machine state, the instruction is refetched and restarted.

5.6.3.2.4 (Type III) Correcting Faults via Software. Sufficient information is contained in the stack frame to complete an instruction in software. After a fault is corrected, the faulted bus cycle must be rerun. The following steps are required to complete an instruction through software:

1. Read the MOVEM opcode and extension from locations pointed to by stack frame PC and PC+2. The effective address need not be recalculated since the next operand address is saved in the stack frame. However, the opcode effective address field must be examined to determine how to update the address register and program counter when the instruction is complete.

2. Adjust the mask to account for operands already transferred. Subtract the stacked operand transfer count from 16 to obtain the number of operands transferred. Scan the mask using this count value. Each time a set bit is found, clear it and decrement the counter. When the count is zero, the mask is ready for use.

3. Adjust the operand address. If the predecrement addressing mode is in effect, subtract the operand size from the stacked value; otherwise, add the operand size to the stacked value.

4. Scan the mask for set bits. Read/write the selected register from/to the operand address as each bit is found.

5. As each operand is transferred, clear the mask bit and increment (decrement) the operand address. When all bits in the mask are cleared, all operands have been transferred.

6. If the addressing mode is predecrement or postincrement, update the register to complete the execution of the instruction.

7. If the TR bit is set in the stacked SSW, create a six-word stack frame and execute the trace handler. If either B1 or B0 in the SSW is set, create another six-word stack frame and execute the hardware breakpoint handler.

8. Deallocate the stack and return control to the faulted program.

5.6.3.2.5 (Type III) Correcting Faults by Conversion and Restart. In some situations, it may be necessary to rerun all the operand transfers for a faulted instruction rather than continue from a faulted operand. When a fault occurs after an operand has transferred, that transfer is not "undone". However, these memory locations are accessed a second time when the instruction is restarted.

Clearing the SSW MV bit will convert a type III fault into a type II fault. All type II exceptions restart upon return from the exception handler.

NOTE

> If a register used in an effective address calculation is overwritten before a fault occurs, an incorrect effective address is calculated upon instruction restart.

5.6.3.2.6 (Type III) Correcting Faults via RTE. The preferred method of MOVEM bus fault recovery is to correct the cause of the fault and then execute an RTE instruction without altering the stack contents. The RTE recognizes that MOVEM was in progress when a fault occurred, restores the appropriate machine state, refetches the instruction, repeats the faulted transfer, and continues the instruction.

MOVEM is the only instruction continued upon return from an exception handler. Although the instruction is refetched, the effective address is not recalculated, and the mask is rescanned the same number of times as before the fault. Modifying the code prior to RTE can cause unexpected results.

5.6.3.2.7 (Type IV) Correcting Faults via Software. BERR exceptions can occur during exception processing while the processor is fetching an exception vector or while it is stacking. The same stack frame and SSW are used in both cases, but each has a distinct fault address. The BERR stack frame format/vector word identifies the type of faulted exception and the contents of the remainder of the frame. A fault address corresponding to the stacked format/vector word indicates that the processor could not obtain the address of the exception handler.

A BERR exception handler should execute RTE after correcting a fault. RTE restores the internal machine state, fetches the address of the original exception handler, recreates the original exception stack frame, and resumes execution at the exception handler address.

If the fault is intractable, the exception handler should rewrite the faulted exception stack frame at SP+$14+$06 and then jump directly to the original exception handler. The stack frame can be generated from the information in the BERR frame: the pre-exception status register (SP+$0C), the format/vector word (SP+$0E), and, if the frame being written is a six-word frame, the program counter of the instruction causing the exception (SP+$10). The return program counter value is available at SP+$02.

A stacked fault address equal to the current stack pointer indicates that stacking was successfully completed even though a stacking bus error occurred. This is an extremely improbable occurrence, but the CPU32 supports recovery from it. Once the exception handler determines that the fault has been corrected, recovery can proceed as described previously. If the fault cannot be corrected, move the supervisor stack to another area of memory, copy all valid stack frames to the new stack, create a faulted exception frame on top of the stack, and resume execution at the exception handler address.

5.6.4 CPU32 Stack Frames

The CPU32 generates three different stack frames: the four-word, six-word frames and twelve-word BERR frames.

5.6.4.1 NORMAL FOUR-WORD STACK FRAME. This stack frame is created by interrupt, format error, TRAP #n, illegal instruction, A-line and F-line emulator trap, and privilege violation exception. Depending on the exception type, the program counter value is either the address of the next instruction to be executed or the address of the instruction that caused the exception (see Figure 5-21).

This stack frame is created by instruction-related traps, which include CHK, CHK2, TRAPcc, TRAPV, and zero division, and by trace exceptions. The faulted instruction program counter value is the address of the instruction that caused the exception. The current program counter value (the address to which RTE returns) is the address of the next instruction to be executed.

	15				0
SP ➧	STATUS REGISTER				
+$02	PROGRAM COUNTER HIGH				
	PROGRAM COUNTER LOW				
+$06	0	0	0	0	VECTOR OFFSET

Figure 5-21. Format $0 — Four-Word Stack Frame

Hardware breakpoints also utilize this format. The faulted instruction program counter value is the address of the instruction executing when the breakpoint is sensed. Usually this is the address of the instruction that caused the breakpoint, but, because released writes can overlap a following instruction, the faulted instruction program counter may point to an instruction following the instruction that caused the breakpoint. The current program counter value (the address to which RTE returns) is the address of the next instruction to be executed (see Figure 5-22).

	15				0
SP →	STATUS REGISTER				
+$02	NEXT INSTRUCTION PROGRAM COUNTER HIGH				
	NEXT INSTRUCTION PROGRAM COUNTER LOW				
+$06	0	0	1	0	VECTOR OFFSET
+$08	FAULTED INSTRUCTION PROGRAM COUNTER HIGH				
	FAULTED INSTRUCTION PROGRAM COUNTER LOW				

Figure 5-22. Format $2 — Six-Word Stack Frame

5.6.4.2 BERR STACK FRAME.

This stack frame is created when a bus cycle fault is detected. The CPU32 BERR stack frame differs significantly from the equivalent stack frames of other M68000 Family members. The BERR stack frame is 12 words in length. There are three variations of the frame, each distinguished by different values in the SSW TP and MV fields.

The bus operation in progress at the time of a fault is conveyed by the SSW.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TP	MV	0	TR	B1	B0	RR	RM	IN	RW	LG	SIZ		FUNC		

An internal transfer count register appears at location SP+14 in all BERR stack frames. The register contains an 8-bit microcode revision number, and, for type III faults, an 8-bit transfer count. Register format is shown in the following illustration:

15	8	7	0
MICROCODE REVISION NUMBER		TRANSFER CODE	

The CPU32 checks the microcode revision number when it restores a BERR stack frame via RTE. In a multiprocessor system, this check ensures that the processor using the stacked information is at the same revision level as the processor that created it.

The transfer count is ignored unless the MV bit in the stacked SSW is set. If the MV bit is set, the least significant byte of the internal register is reloaded into the MOVEM transfer counter during RTE execution.

For faults occurring during normal instruction execution (both prefetches and non-MOVEM operand accesses) SSW [TP:MV] = 00. Stack frame format is shown in Figure 5-23.

	15				0
SP ➧	STATUS REGISTER				
+$02	RETURN PROGRAM COUNTER HIGH				
	RETURN PROGRAM COUNTER LOW				
+$06	1	1	0	0	VECTOR OFFSET
+$08	FAULTED ADDRESS HIGH				
	FAULTED ADDRESS LOW				
+$0C	DBUF HIGH				
	DBUF LOW				
+$10	CURRENT INSTRUCTION PROGRAM COUNTER HIGH				
	CURRENT INSTRUCTION PROGRAM COUNTER LOW				
+$14	INTERNAL TRANSFER COUNT REGISTER				
+$16	0	0	SPECIAL STATUS WORD		

Figure 5-23. Format $C — BERR Stack for Prefetches and Operands

Faults that occur during the operand portion of the MOVEM instruction.are identified by SSW [TP:MV] = 01. Stack frame format is shown in Figure 5-24.

	15				0
SP ➧	STATUS REGISTER				
+$02	RETURN PROGRAM COUNTER HIGH				
	RETURN PROGRAM COUNTER LOW				
+$06	1	1	0	0	VECTOR OFFSET
+$08	FAULTED ADDRESS HIGH				
	FAULTED ADDRESS LOW				
+$0C	DBUF HIGH				
	DBUF LOW				
+$10	CURRENT INSTRUCTION PROGRAM COUNTER HIGH				
	CURRENT INSTRUCTION PROGRAM COUNTER LOW				
+$14	INTERNAL TRANSFER COUNT REGISTER				
+$16	0	1	SPECIAL STATUS WORD		

Figure 5-24. Format $C — BERR Stack on MOVEM Operand

When a bus error occurs during exception processing, SSW [TP:MV]=10. The frame shown in Figure 5-25 is written below the faulting frame. Stacking begins at the address pointed to by SP 6 (SP value is the value before initial stacking on the faulted frame).

The frame can have either four or six words, depending on the type of error. Four-word stack frames do not include the faulted instruction program counter (the internal transfer count register is located at SP+$10 and the SSW is located at SP+$12).

The fault address of a dynamically sized bus cycle is the upper byte. There is no indication which of the two bytes caused the error.

	15				0
SP ➧	STATUS REGISTER				
+$02	NEXT INSTRUCTION ROGRAM COUNTER HIGH				
	NEXT INSTRUCTION PROGRAM COUNTER LOW				
+$06	1	1	0	0	VECTOR OFFSET
+$08	FAULTED ADDRESS HIGH				
	FAULTED ADDRESS LOW				
+$0C	PRE-EXCEPTION STATUS REGISTER				
	FAULTED EXCEPTION FORMAT/VECTOR WORD				
+$10	FAULTED INSTRUCTION PROGRAM COUNTER HIGH (SIX WORD FRAME ONLY)				
	FAULTED INSTRUCTION PROGRAM COUNTER LOW (SIX WORD FRAME ONLY)				
+$14	INTERNAL TRANSFER COUNT REGISTER				
+$16	1	0	SPECIAL STATUS WORD		

Figure 5-25. Format $C — Four- and Six-Word BERR Stack

5.7 DEVELOPMENT SUPPORT

All M68000 Family members include the following to facilitate applications development:

Trace on Instruction Execution — M68000 processors include an instruction-by-instruction tracing facility as an aid to program development; however, the MC68020, MC68030, and CPU32 also allow tracing only those instructions causing a change in program flow. In the trace mode, a trace exception is generated after each instruction is executed, allowing a debugger program to monitor the execution of a program under test. See **5.6.2.10 TRACING** for more information.

Breakpoint Instruction — An emulator may insert software breakpoints into the target code to indicate when a breakpoint has occurred. On the MC68010, MC68020, MC68030, and CPU32, this function is provided via illegal instructions ($4848–$484F) that serve as breakpoint instructions. See **5.6.2.5 SOFTWARE BREAKPOINTS** for more information.

Unimplemented Instruction Emulation — When an attempt is made to execute an illegal instruction, an illegal instruction exception occurs. Unimplemented instructions (F-line, A-line, . . .) utilize separate exception vectors to permit efficient emulation of unimplemented instructions in software. See **5.6.2.8 ILLEGAL OR UNIMPLEMENTED INSTRUCTIONS** for more information.

5.7.1 CPU32 Integrated Development Support

The CPU32 not only incorporates all the previous features but also provides additional features that aid development tools in advancing support for integrated system development. These additions include background debug mode, deterministic opcode tracking, hardware breakpoints, and internal visibility in the single-chip environment.

5.7.1.1 BACKGROUND DEBUG MODE (BDM) OVERVIEW. Microprocessor systems generally provide a debugger, implemented in software, for system analysis at the lowest level. The BDM on the CPU32 is unique in that the debugger is implemented in CPU microcode. Registers can be viewed and/or altered, memory can be read or written to, and test features can be invoked. Incorporating these capabilities on-chip simplifies the environment in which an in-circuit emulator operates. The traditional in-circuit emulator configuration (see Figure 5-26) removes the processor from the target, replacing it with hardware resident in the emulator. An expensive cable provides the communication path between the target system and the emulator. By contrast, with an integrated debugger, the traditional emulator configuration can be replaced by a bus state analyzer (BSA) (see Figure 5-27). The advantage of this configuration is twofold: 1) the processor remains in the target hardware, serving as its own emulation processor and 2) this integration reduces cost by eliminating the cable. The BSA provides a means for monitoring target processor operation; the on-chip debugger provides the mechanism for altering the operating environment. Many of the problems experienced with the classic emulator configuration are minimized: i.e., limitations on high-frequency operation, AC and DC parametric mismatches, and restrictions on cable length.

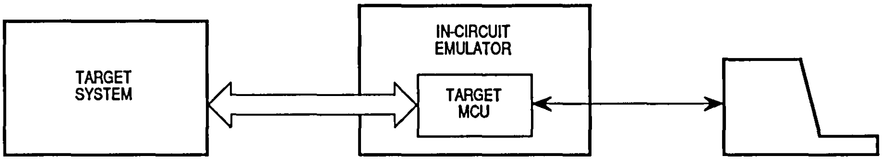


Figure 5-26. Traditional In-Circuit Emulator Diagram

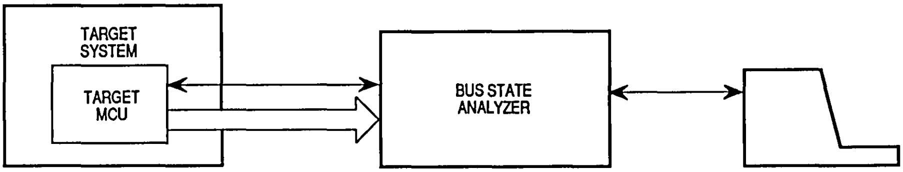


Figure 5-27. Bus State Analyzer Configuration

5.7.1.2 DETERMINISTIC OPCODE TRACKING OVERVIEW. CPU32 function code outputs are augmented by two supplementary signals to monitor the instruction pipeline. The instruction pipe ($\overline{\text{IPIPE}}$) output indicates the start of each new instruction and each mid-instruction pipeline advance. The instruction fetch ($\overline{\text{IFETCH}}$) output identifies those bus cycles in which the operand data is loaded into the instruction pipeline. Pipeline flushes are also signaled with $\overline{\text{IFETCH}}$. Monitoring these two signals allows a BSA to synchronize to the instruction stream and monitor the activity. Refer to **5.7.3 Deterministic Opcode Tracking** for a complete description.

5.7.1.3 ON-CHIP HARDWARE BREAKPOINT OVERVIEW. An external breakpoint input and on-chip hardware breakpoint allow a breakpoint trap on any memory access. Off-chip address comparators preclude breakpoints on internal accesses unless show cycles are enabled. Breakpoints on instruction prefetches, which are ultimately flushed from the instruction pipeline, are not acknowledged; operand breakpoints are always acknowledged. Acknowledged breakpoints optionally initiate exception processing or BDM. See **5.6.2.6 HARDWARE BREAKPOINTS** for more information.

5.7.2 Background Debug Mode (BDM)

BDM is an alternate CPU32 operating mode in which normal instruction execution is suspended while special microcode performs the functions of a debugger. BDM is initiated by one of several sources: externally generated breakpoints, internal peripheral breakpoints, the background (BGND) instruction, or catastrophic exception conditions. While in BDM, the CPU32 ceases fetching instructions via the parallel bus and, instead, accepts commands via a dedicated serial interface. A high-speed, SPI-type serial link provides BDM communication between the CPU32 and the development system. Figure 5-28 illustrates a block diagram of the BDM.

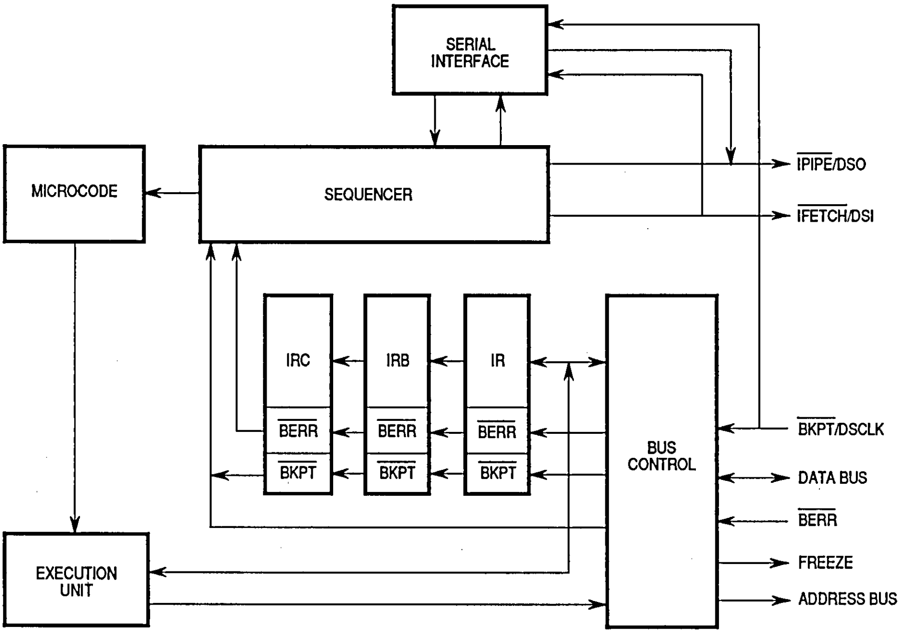


Figure 5-28. BDM Block Diagram

5.7.2.1 ENABLING BDM. Accidentally entering BDM in a nondevelopment environment could inadvertently lock up the CPU32 since the serial command interface would probably not be available. For this reason, BDM is enabled during reset via the breakpoint ($\overline{\text{BKPT}}$) signal. When $\overline{\text{BKPT}}$ is asserted (low) at the rising edge on $\overline{\text{RESET}}$, BDM operation is enabled until the next system reset. A high $\overline{\text{BKPT}}$ signal at the trailing edge of $\overline{\text{RESET}}$ disables BDM, and all sources of entry revert to their normal operation. $\overline{\text{BKPT}}$ is relatched on each rising transition of $\overline{\text{RESET}}$.

$\overline{\text{BKPT}}$ is synchronized internally; therefore, the signal must be held low for at least two clock cycles prior to the negation of $\overline{\text{RESET}}$. Special care must be taken in the design of the BDM enable logic. If the hold time on $\overline{\text{BKPT}}$ (after the trailing edge of RESET) extends into the first bus cycle, the possibility exists that the bus cycle could be inadvertently tagged with a breakpoint.

5.7.2.2 BDM SOURCES. Once BDM has been enabled, any of several sources are capable of causing the transition from normal operation into BDM. These sources include 1) external breakpoint hardware, 2) the BGND instruction, 3) double bus fault, and 4) internal peripheral breakpoints. If BDM is not enabled when the exception condition occurs, the exception is processed normally. Table 5-21 summarizes the processing of each source for both the enabled and disabled cases. As depicted in the table, the BKPT instruction never causes a transition into the BDM of operation.

Table 5-21. BDM Source Summary

Source	BDM Enabled	BDM Disabled
BKPT	Background	Breakpoint Exception
Double Bus Fault	Background	Halted
BGND Instruction	Background	Illegal Instruction
BKPT Instruction	Opcode Substitution Illegal Instruction	Opcode Substitution Illegal Instruction

5.7.2.2.1 External BKPT Signal. Once enabled, BDM is initiated whenever assertion of BKPT is acknowledged. If BDM is disabled, a breakpoint exception (vector $0C) is acknowledged. Timing on the BKPT input is the same as that for read cycle data with respect to the trailing edge of data strobe. A breakpoint acknowledge bus cycle is not run when entering BDM.

5.7.2.2.2 BGND Instruction. An illegal instruction, $4AFA, is reserved for use by development tools. The CPU32 defines $4AFA (BGND) to be a BDM entry point when BDM is enabled. If BDM is disabled, an illegal instruction trap is acknowledged. Illegal instruction traps are discussed in **5.6.2.8 ILLEGAL OR UNIMPLEMENTED INSTRUCTIONS.**

5.7.2.2.3 Double Bus Fault. Two bus faults in succession, or a double bus fault, normally indicates that a catastrophic error has occurred within the system, resulting in the suspension of instruction execution. When this error condition occurs during initial system debug (e.g., a fault in the reset logic), further debugging is impossible until the situation is corrected. Through BDM, the fault can be bypassed temporarily, the cause of the problem can be determined, and the effects of the problem can be corrected. Should BDM be disabled, a double bus fault causes the processor to terminate instruction execution until reset.

5.7.2.2.4 Peripheral Breakpoints. Peripherals capable of requesting breakpoints do so by asserting the $\overline{\text{BKPT}}$ signal. With respect to the CPU32, the operation of peripheral breakpoints is identical to that of external breakpoints. Consult the appropriate peripheral user's manual for additional details on the generation of peripheral breakpoints.

5.7.2.3 ENTERING BDM. Upon detecting a breakpoint or double bus fault or upon decoding a BGND instruction, the processor suspends instruction execution and asserts the FREEZE output. This action is the first indication that the processor has entered BDM (see Figure 5-29). Once FREEZE has been asserted, the CPU32 enables the serial communication hardware and awaits the first command.

As part of the process of entering BDM, the CPU32 writes a unique value into temporary register A (ATEMP), indicating the source that caused the transition. By issuing a read system register command as the initial command, the user can poll the register and determine the source (see Table 5-22).

Table 5-22. Polling the BDM Entry Source

Source	ATEMP [31:16]	ATEMP [15:0]
Double Bus Fault	SSW	$FFFF
BGND Instruction	$0000	$0001
Hardware Breakpoint	$0000	$0000

ATEMP is used in most debugger commands for temporary storage; therefore, it is imperative that the read system register (RSREG) command be the first command issued after the transition into BDM.

A double bus fault during the initial stack pointer/program counter (SP/PC) fetch sequence is further distinguished by a value of $FFFFFFFF in the current instruction PC. At no other time will the processor write an odd value into this register.

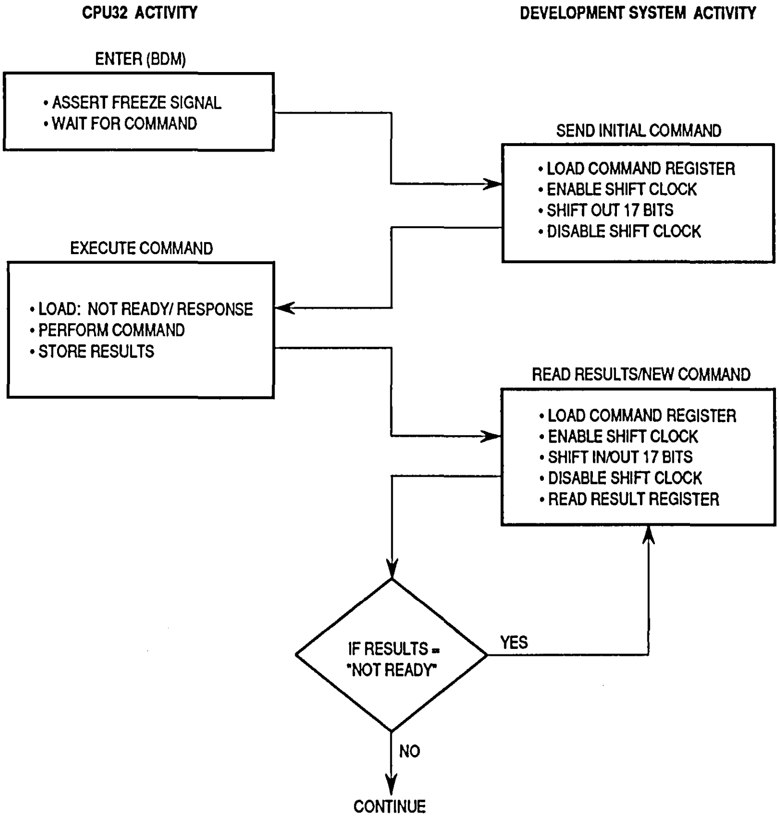


Figure 5-29. BDM Command Execution Flowchart

5.7.2.4 COMMAND EXECUTION. As each command is accumulated in the serial shifter, the microcode routine corresponding to that command is executed. If the command can complete without additional serial traffic, it does. However, if addresses or operands are required, the microcode reads each word as it is assembled by the serial interface. The CPU32 then performs the desired operation, including any necessary memory or register accesses. Result operands are loaded into the output shift register to be shifted out as the next command is read. This process is repeated for each instruction until the CPU32 returns to the normal operating mode.

5.7.2.5 BACKGROUND MODE REGISTERS. The following paragraphs describe three special-purpose registers used in BDM.

5.7.2.5.1 Fault Address Register (FAR). The FAR contains the address of the faulted bus cycle immediately following a bus or address error. This address remains available until overwritten by a subsequent bus cycle. Following a double bus fault, the FAR contains the address of the last bus cycle. The address of the first fault (if one occurred) is not visible to the user.

5.7.2.5.2 Return Program Counter (RPC). The RPC points to the location from which fetching will commence at the transition from BDM into normal mode. This register should be accessed to change the flow of a program under development. Changing the RPC to an odd value causes an address error, which is generated when fetching begins.

5.7.2.5.3 Current Instruction Program Counter (PCC). The PCC holds the pointer to the first word of the last instruction executed prior to the transition into BDM. Due to the pipelined nature of the CPU32, the instruction pointed to by the PCC may not be the instruction that caused the BDM transition. An example is a breakpoint on a released write. The bus cycle may extend into as many as two subsequent instructions before stalling the instruction sequencer. A breakpoint asserted during this cycle will not be acknowledged until the end of the instruction executing at the completion of the bus cycle.

5.7.2.6 RETURNING FROM BDM. BDM is terminated when a resume execution (GO) or call user code (CALL) command is received. Both GO and CALL flush the instruction pipeline and prefetch instructions from the location pointed to by the current PC. The current PC and the memory space referred to by the status register SUPV bit reflect any changes made during BDM. FREEZE is negated prior to initiating the first prefetch. Upon negation of FREEZE, the serial subsystem is disabled, and the signals revert to $\overline{\text{IPIPE}}/\overline{\text{IFETCH}}$ functionality.

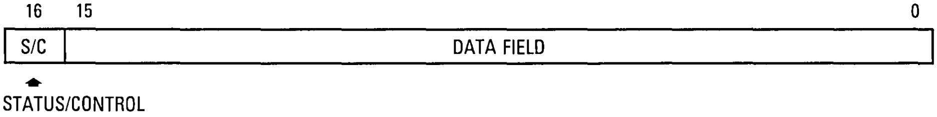
5.7.2.7 SERIAL INTERFACE. Communication with the CPU32 during BDM sessions occurs via a dedicated serial interface, which shares pins with other development features. The $\overline{\text{BKPT}}$ signal becomes the serial clock (DSCLK); serial input data ($\overline{\text{DSI}}$) is received on $\overline{\text{IFETCH}}$, and serial output data (DSO) is transmitted on $\overline{\text{IPIPE}}$.

The serial interface is a full-duplex synchronous protocol similar to the serial peripheral interface protocol. The development system serves as the master of the serial link since it is responsible for the generation of DSCLK. By deriving

DSCLK from the CPU32 system clock, the design of the development system serial logic is unhindered by the operating frequency of the target processor. Operable frequency range of the serial clock is from DC to one-half the processor system clock frequency.

The serial interface operates in a full-duplex mode — that is, data is both transmitted and received simultaneously by the master and slave devices. In general, data transitions occur on the falling edge of the DSCLK and are stable by the following rising edge of DSCLK. Data is transmitted most significant bit first and is latched on the rising edge of DSCLK.

The serial data word is 17 bits wide — 16 data bits and a status/control bit.



For CPU-generated messages, bit 16 indicates the message status as shown in Table 5-23.

Table 5-23. CPU-Generated Message Encoding

Bit 16	Data	Message Type
0	xxxx	Valid Data Transfer
0	FFFF	Command Complete; Status OK
1	0000	Not Ready with Response; Come Again
1	0001	BERR Terminated Bus Cycle; Data Invalid
1	FFFF	Illegal Command

Command and data transfers initiated by the development system should clear bit 16. The current implementation ignores this bit; however, Motorola reserves the right to use this bit for future enhancements.

5.7.2.7.1 CPU32 Serial Logic. The CPU32 serial logic block diagram, pictured in the left-hand portion of Figure 5-30, consists of the transmit and receive shift registers and control logic containing the synchronization logic, serial clock generation circuitry, and a received bit counter.

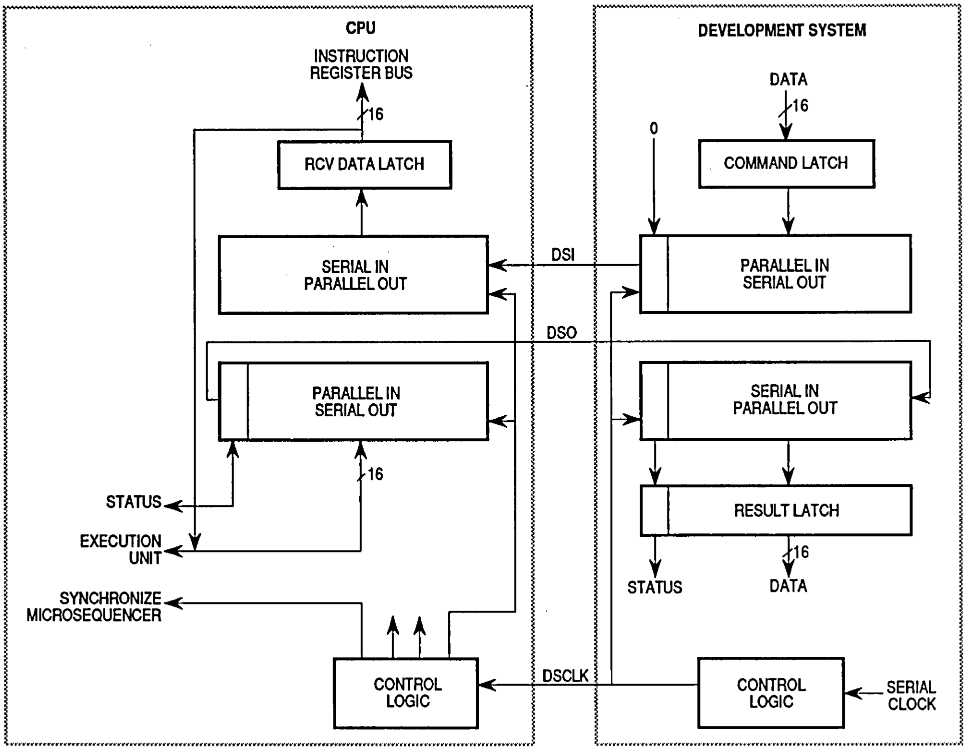


Figure 5-30. Debug Serial I/O Block Diagram

Both DSCLK and DSI are synchronized to the on-chip clocks, thereby minimizing the chance of propagating metastable states into the serial state machine. Data is sampled during the high phase of CLKOUT. At the falling edge of CLKOUT, the sampled value is made available to the internal logic. Thus, the minimum hold time on DSI with respect to DSCLK is one full period of CLKOUT.

The serial state machine begins a sequence of events based on the rising edge of the synchronized DSCLK (see Figure 5-31). The synchronized serial data is transferred to the input shift register, and the received bit counter is decremented. One-half clock period later, the output shift register is updated, bringing the next output bit to the DSO signal. DSO changes relative to the rising edge of DSCLK and does not necessarily remain stable until the falling edge of DSCLK.

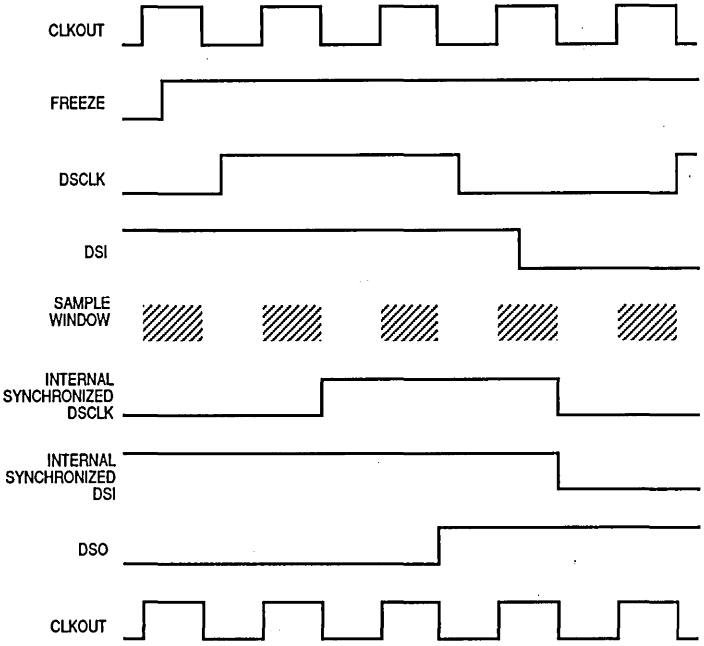


Figure 5-31. Serial Interface Timing Diagram

One clock period after the synchronized DSCLK has been seen internally, the updated counter value is checked. If the counter has reached zero, the receive data latch is updated from the input shift register. At this same time, the output shift register is reloaded with the "not ready/come again" response. Once the receive data latch has been loaded, the CPU32 is released to act on the new data. Response data overwrites the "not ready" response when the CPU32 has completed the current operation.

Data written into the output shift register appears immediately on the DSO signal. In general, this action changes the state of the signal from a high ("not ready" response status bit) to a low (valid data status bit) logic level. However, this level change only occurs if the command completes successfully. Error conditions overwrite the "not ready" response with the appropriate response that also has the status bit set.

A user may take advantage of the state change on DSO to signal hardware that the next serial transfer may begin. A timeout of sufficient length should also be incorporated into the design to trap error conditions that do not change the state of DSO. Hardware interlocks in the CPU32 prevent result data from corrupting serial transfers in progress.

5.7.2.7.2 Development System Serial Logic. The development system, as the master of the serial data link, must supply the serial clock. However, normal and BDM operations could inadvertently interact if the dual operating modes are not properly considered when designing the clock generator.

Breakpoint requests are made by asserting $\overline{\text{BKPT}}$ to the low state using one of two methods. The predominant method (one described thus far) is to assert $\overline{\text{BKPT}}$ during the single bus cycle for which the exception is desired. A second method is to assert $\overline{\text{BKPT}}$ and continue asserting it until the CPU32 responds by asserting FREEZE. This method is useful for forcing a transition into BDM when the bus is not being monitored. Each BKPT assertion method requires a slightly different approach in the design of the serial logic to avoid spurious serial clocks.

Figure 5-32 represents the timing required for asserting $\overline{\text{BKPT}}$ during a single bus cycle. Figure 5-33 depicts the timing of the $\overline{\text{BKPT}}$/FREEZE method. In both cases, the serial clock is left high after the final shift of each transfer. This technique eliminates the possibility of accidentally tagging the prefetch initiated at the conclusion of a BDM session. As mentioned previously, all timing within the CPU32 is derived from the rising edge of the clock; the falling edge is effectively ignored.

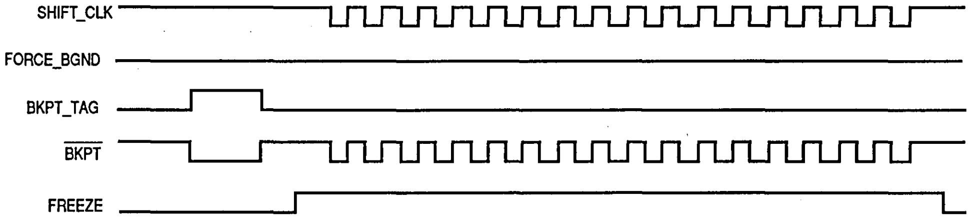


Figure 5-32. BKPT Timing for Single Bus Cycle

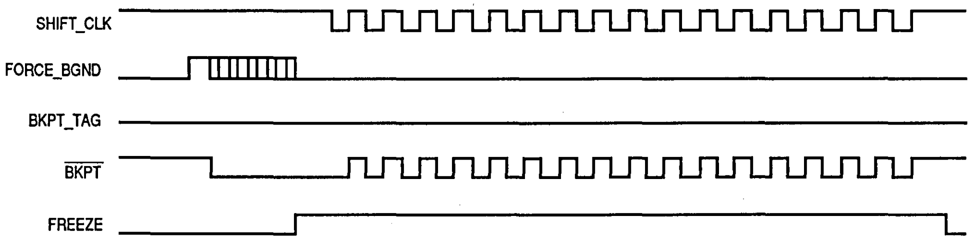


Figure 5-33. BKPT Timing for Forcing BDM

Figure 5-34 represents a sample circuit providing for both BKPT assertion methods. As the name implies, FORCE–BGND is used to force a transition into BDM by the assertion of $\overline{\text{BKPT}}$. FORCE–BGND can be a short pulse or can remain asserted until FREEZE is asserted. Once asserted, the set-reset latch holds $\overline{\text{BKPT}}$ low until the first SHIFT–CLK is applied.

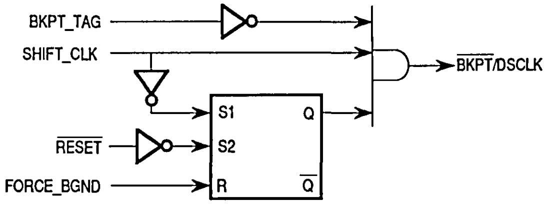


Figure 5-34. BKPT/DSCLK Logic Diagram

BKPT—TAG should be timed to the bus cycles since it is not latched. If extended past the assertion of FREEZE, the negation of BKPT–TAG appears to the CPU32 as the first DSCLK.

DSCLK is the gated serial clock. Normally high, it pulses low for each bit to be transferred. At the end of the seventeenth clock period, it returns high until the start of the next transmission. Clock frequency is implementation dependent and may range from DC to the maximum specified frequency. Although performance considerations might dictate a hardware implementation, software solutions are not precluded provided serial bus timing is maintained.

5.7.2.8 COMMAND SET. The following paragraphs describe the command set available in BDM.

5.7.2.8.1 Command Format. The following standard bit format is utilized by all BDM commands.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPERATION						0	R/W	OP SIZE		0	0	A/D	REGISTER		

EXTENSION WORD(S)

Operation Field:

Commands are distinguished by the operation field. This 6-bit field provides for a maximum of 64 unique commands.

R/W Field:

The direction of the operand transfer is specified in this field. When this bit is set, the operation direction is from the CPU32 to the development system. When this bit is clear, data is written into the CPU32 or memory from the development system.

Operand Size:

For sized operations, this field specifies the operand data size. All addresses are expressed as 32-bit absolute values. The size field is encoded as listed in Table 5-24.

Table 5-24. Size Field Encoding

Encoding	Operand Size
00	Byte
01	Word
10	Long
11	Reserved

Address/Data (A/D) Field:

Used by commands that operate on address and data registers, the A/D field determines whether the register field specifies a data or address register. A one indicates an address register; a zero selects a data register. For other commands, this field may be interpreted differently.

Register Field:

In most commands, this field specifies the register number when operating on an address or data register.

Extension Words (as required):

Some commands require immediate data or addresses in the form of extension words. Addresses require two extension words each since addressing capability is limited to absolute long. Immediate data can be either one or two words. Byte and word data each require a single extension word; longword data requires two words. Operands and addresses are transferred most significant word first. At this time, no command requires an extension word to fully specify the operation to be performed (i.e., single-word commands only).

5.7.2.8.2 Command Sequence Diagrams. A command sequence diagram illustrates the serial bus traffic for each command. Each bubble in the diagram represents a single 17-bit transfer across the bus. The top half in each diagram corresponds to the data transmitted by the development system to the CPU32; likewise, the bottom half corresponds to the data returned by the CPU32 in response to the development system commands. Command and result transactions are overlapped to minimize latency.

The command sequence diagram in Figure 5-35 demonstrates the use of these diagrams. The cycle in which the command is issued contains the command mnemonic issued by the development system (in this example, read memory location). During the same cycle, the CPU32 is responding with either the lowest order results of the previous command or with a command complete status if no other results were required.

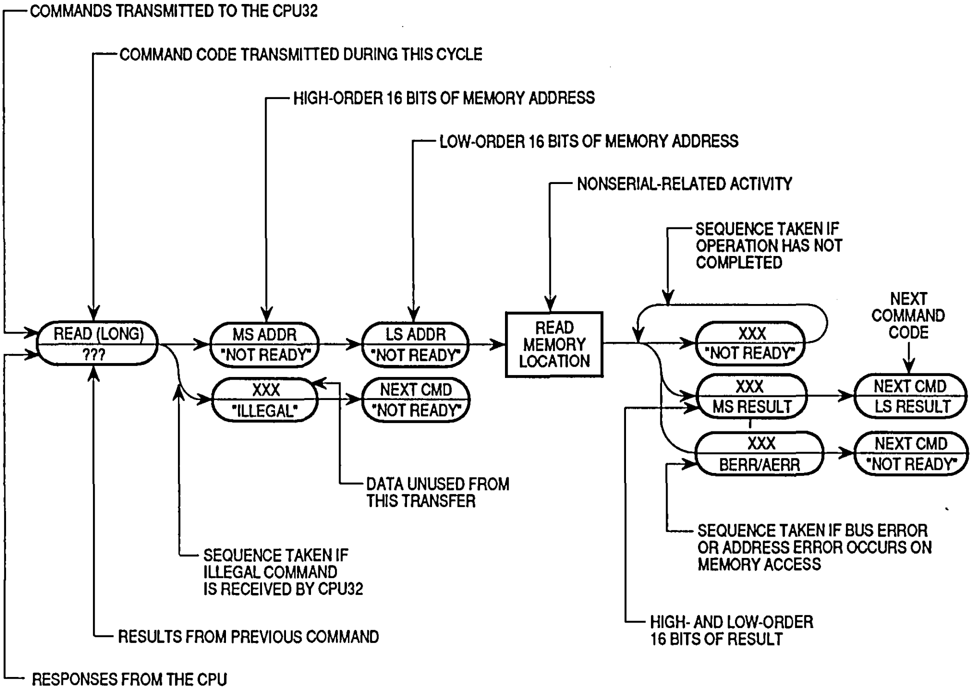

COMMANDS TRANSMITTED TO THE CPU32
COMMAND CODE TRANSMITTED DURING THIS CYCLE
HIGH-ORDER 16 BITS OF MEMORY ADDRESS
LOW-ORDER 16 BITS OF MEMORY ADDRESS
NONSERIAL-RELATED ACTIVITY
SEQUENCE TAKEN IF OPERATION HAS NOT COMPLETED
NEXT COMMAND CODE
READ (LONG)
???
MS ADDR
"NOT READY"
LS ADDR
"NOT READY"
READ MEMORY LOCATION
XXX
"NOT READY"
XXX
"ILLEGAL"
NEXT CMD
"NOT READY"
XXX
MS RESULT
NEXT CMD
LS RESULT
XXX
BERR/AERR
NEXT CMD
"NOT READY"
DATA UNUSED FROM THIS TRANSFER
SEQUENCE TAKEN IF BUS ERROR OR ADDRESS ERROR OCCURS ON MEMORY ACCESS
SEQUENCE TAKEN IF ILLEGAL COMMAND IS RECEIVED BY CPU32
HIGH- AND LOW-ORDER 16 BITS OF RESULT
RESULTS FROM PREVIOUS COMMAND
RESPONSES FROM THE CPU

Figure 5-35. Command Sequence Diagram Example

During the second cycle of the diagram, the development system supplies the high-order 16 bits of the memory address. The CPU32 returns the "not ready" response unless the received command was decoded as unimplemented, in which case the response data is the illegal command encoding. If an illegal command response occurs, the development system should retransmit the command.

NOTE

> The "not ready" response can be ignored except in those cases when a memory bus cycle is in progress. In all other cases, the CPU32 can accept a new serial transfer with eight system clock periods.

In the third cycle, the development system supplies the low-order 16 bits of the memory address. The CPU32 always returns the "not ready" response in this cycle. At the completion of the third cycle, the CPU32 initiates the memory read operation. Any serial transfers that begin while the memory access is in progress return the "not ready" response.

The results are returned in the two serial transfer cycles following the completion of the memory access. The data transmitted to the CPU32 during the final transfer is the opcode for the following command. Should the memory access generate either a bus or address error, an error status is returned in place of the result data.

5.7.2.8.3 Command Set Summary. The BDM command set is summarized in Table 5-25. Detailed descriptions of each command can be found in subsequent paragraphs.

Table 5-25. BDM Command Summary

Command	Mnemonic	Description
Read A/D Register	RAREG/RDREG	Read the selected address or data register and return the results via the serial interface.
Write A/D Register	WAREG/WDREG	The data operand is written to the specified address or data register.
Read System Register	RSREG	The specified system control register is read. All registers that can be read in supervisor mode can be read in BDM.
Write System Register	WSREG	The operand data is written into the specified system control register.
Read Memory Location	READ	Read the sized data at the memory location specified by the long-word address. The source function code (SFC) register determines the address space accessed.
Write Memory Location	WRITE	Write the operand data to the memory location specified by the long-word address. The destination function code (DFC) register determines the address space accessed.
Dump Memory Block	DUMP	Used in conjunction with the READ command to dump large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. Subsequent operands are retrieved with the DUMP command.
Fill Memory Block	FILL	Used in conjunction with the WRITE command to fill large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. Subsequent operands are written with the FILL command.
Resume Execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC.
Call User Code	CALL	Current PC is stacked at the location of the current SP. Instruction execution begins at user patch code.
Reset Peripherals	RST	Asserts RESET for 512 clock cycles. The CPU32 is **not** reset by this command. Synonymous with the CPU32 RESET instruction.
No Operation	NOP	NOP performs no operation and may be used as a null command.

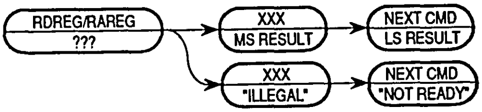
5.7.2.8.4 Read A/D Register (RAREG/RDREG).

Read the selected address or data register and return the results via the serial interface.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1	1	0	0	0	A/D	REGISTER		

Command Sequence:



Operand Data:
None

Result Data:
The contents of the selected register are returned as a long-word value. The data is returned most significant word first.

NOTE

Accesses to register A7 follow the supervisor (S) bit at the time BDM was entered. If S=0, A7 corresponds to the user SP; if S=1, A7 corresponds to the supervisor SP. BDM writes to the SR, which affect the S-bit, have no effect on the selection of A7. Use the RSREG/WSREG commands to directly access a specific SP.

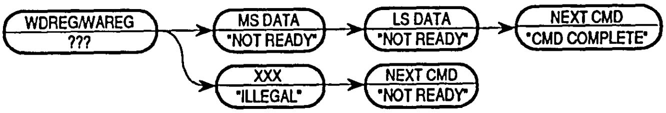
5.7.2.8.5 Write A/D Register (WAREG/WDREG).

The operand (long-word) data is written to the specified address or data register. All 32 bits of the register are altered by the write.

Command Format:

15	14	13	12	11	10	9.	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0	1	0	0	0	A/D	REGISTER		

Command Sequence:



Operand Data:

The long-word data is written into the specified address or data register. The data is supplied most significant word first.

Result Data:

Command complete status ($0FFFF) is returned when the register write has completed.

NOTE

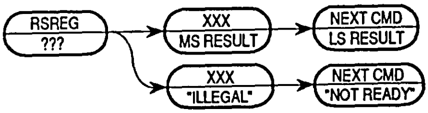
Accesses to register A7 follow the S-bit at the time BDM was entered. If S=0, A7 corresponds to the user SP; if S=1, A7 corresponds to the supervisor SP. BDM writes to SR, which affect the S-bit, have no effect on the selection of A7. Use the RSREG/WSREG commands to directly access a specific SP.

5.7.2.8.6 Read System Register (RSREG). The specified system control register is read. All registers that can be read in supervisor mode can be read in BDM. Several internal temporary registers are also accessible.

Command Format:

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

Command Sequence:



Operand Data:

None

Result Data:

Always returns 32 bits of data, regardless of the size of the register being read. If the register is less than 32 bits, the result is returned zero extended.

Register Field:

The system control register is specified by the register field (see Table 5-26).

Table 5-26. Register Field for RSREG

System Register	Select Code
Return Program Counter (RPC)	0000
Current Instruction Program Counter (PCC)	0001
Status Register (SR)	1011
User Stack Pointer (USP)	1100
Supervisor Stack Pointer (SSP)	1101
Source Function Code Register (SFC)	1110
Destination Function Code Register (DFC)	1111
Temporary Register A (ATEMP)	1000
Fault Address Register (FAR)	1001
Vector Base Register (VBR)	1010

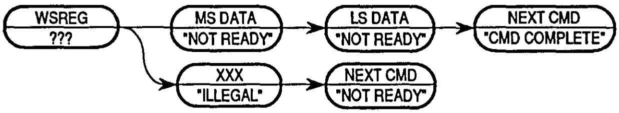
5.7.2.8.7 Write System Register (WSREG).

The operand data is written into the specified system control register. All registers that can be written in supervisor mode can be written in BDM. Several internal temporary registers are also accessible.

Command Format:

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

Command Sequence:



Operand Data:

The data to be written into the register is always supplied as a 32-bit long word. If the written register is less than 16 bits, the least significant word is used.

Result Data:

"Command complete" status is returned when the register write is completed.

Register Field:

The system control register is specified by the register field (see Table 5-27). The FAR is a read-only register; any write to this register is ignored.

Table 5-27. Register Field for WSREG

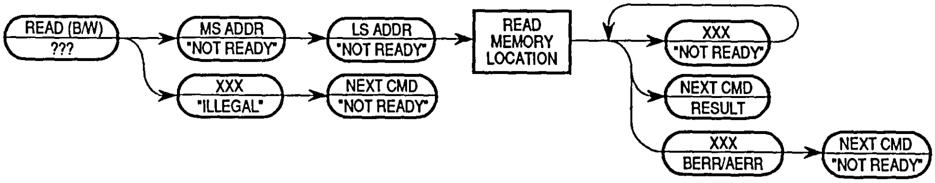
System Register	Select Code
Return Program Counter (RPC)	0000
Current Instruction Program Counter (PCC)	0001
Status Register (SR)	1011
User Stack Pointer (USP)	1100
Supervisor Stack Pointer (SSP)	1101
Source Function Code Register (SFC)	1110
Destination Function Code Register (DFC)	1111
Temporary Register A (ATEMP)	1000
Fault Address Register (FAR)	1001
Vector Base Register (VBR)	1010

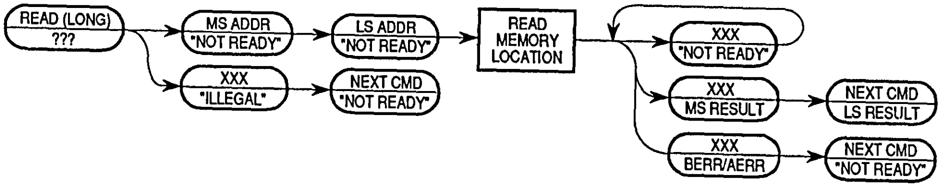
5.7.2.8.8 Read Memory Location (READ). Read the sized data at the memory location specified by the long-word address. Only absolute addressing is supported. The source function code (SFC) register determines the address space accessed. Valid data sizes include byte, word, or long word.

Command Format:

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

Command Sequence:





Operand Data:

The single operand is the long-word address of the requested memory location.

Result Data:

The requested data is returned as either a word or long word. Byte data is returned in the least significant byte of a word result with the upper byte cleared. Word results return 16 bits of significant data; long-word results return 32 bits.

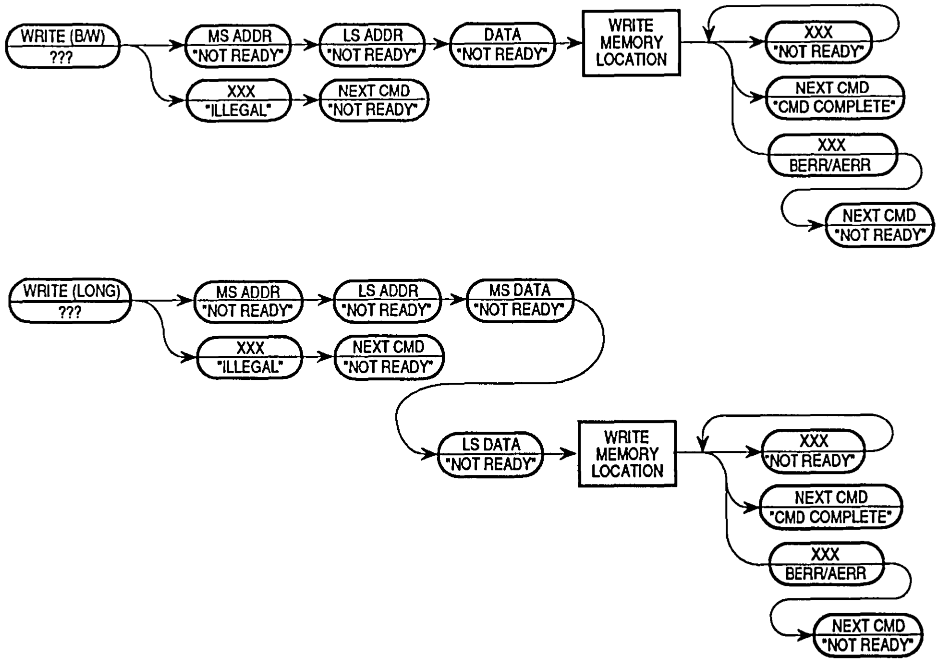
A successful read operation returns data bit 16 cleared; whereas, if a bus or address error is encountered, the returned data is $10001.

5.7.2.8.9 Write Memory Location (WRITE). Write the operand data to the memory location specified by the long-word address. The destination function code (DFC) register determines the address space accessed. Only absolute addressing is supported. Valid data sizes include byte, word, and long word.

Command Format:

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

Command Sequence:



Operand Data:

Two operands are required for this instruction. The first operand is a long-word absolute address specifying the location to which the operand data is to be written. The second operand is the data. Byte data is transmitted as a 16-bit word, justified in the least significant byte; 16- and 32-bit operands are transmitted as 16 and 32 bits, respectively.

Result Data:

Successful write operations return a status of $0FFFF. Bus or address errors on the write cycle are indicated by the assertion of bit 16 in the status message and by a data pattern of $0001.

5.7.2.8.10 Dump Memory Block (DUMP). DUMP is used in conjunction with the READ command to dump large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. Subsequent operands are retrieved with the DUMP command. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register. Subsequent DUMP commands use this address, increment it by the current operand size, and store the updated address back in the temporary register.

NOTE

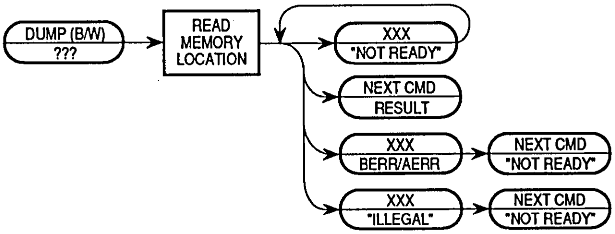
> The DUMP command does not check to see that a valid address is present in the temporary register. Therefore, DUMP is a valid command only when preceded by another DUMP or by a READ command; otherwise, the results are undefined.

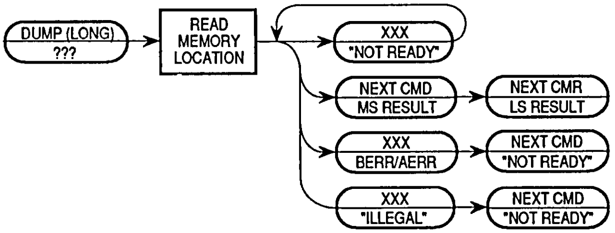
The size field is examined each time a DUMP command is given, allowing the operand size to be altered dynamically.

Command Format:

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

Command Sequence:





Operand Data:
None

Result Data:
The requested data is returned as either a word or long word. Byte data is returned in the least significant byte of a word result. Word results return 16 bits of significant data; long-word results return 32 bits. Status of the read operation is returned as in the READ command: $0xxxx for success, $10001 for bus or address errors.

5.7.2.8.11 Fill Memory Block (FILL).

FILL is used in conjunction with the WRITE command to fill large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. Subsequent operands are written with the FILL command. The initial address is incremented by the operand size (1, 2, or 4) and is saved in a temporary register. Subsequent FILL commands use this address, increment it by the current operand size, and store the updated address back in the temporary register.

NOTE

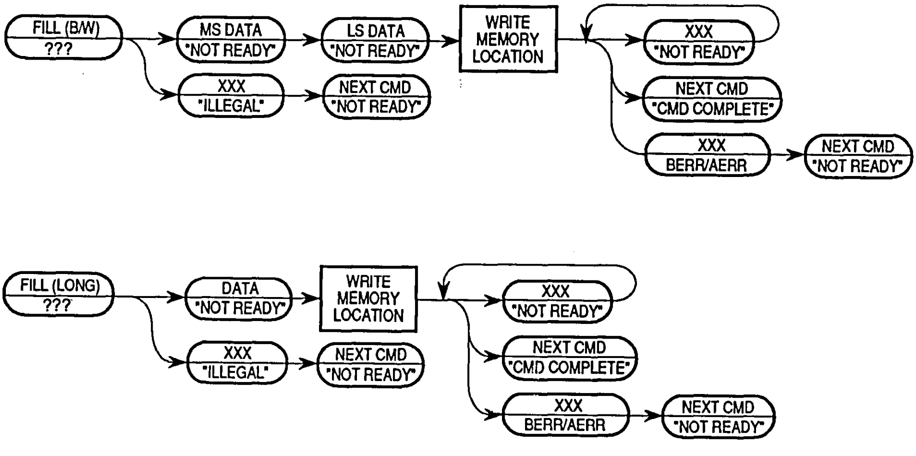
> The FILL command does not check to see that a valid address is present in the temporary register. Therefore, FILL is a valid command only when preceded by another FILL or by a WRITE command; otherwise, the results are undefined.

The size field is examined each time a FILL command is given, allowing the operand size to be altered dynamically.

Command Format:

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

Command Sequence:



Operand Data:

A single operand is data to be written to the memory location. Byte data is transmitted as a 16-bit word, justified in the least significant byte; 16- and 32-bit operands are transmitted as 16 and 32 bits, respectively.

Result Data:

Status is returned as in the WRITE command: $0FFFF for a successful operation and $10001 for a bus or address error during write.

5.7.2.8.12 Resume Execution (GO). The pipeline is flushed and refilled before normal instruction execution is resumed. Prefetching begins at the current PC and current privilege level. If either the PC or SR is altered during BDM, the updated value of these registers is used when prefetching commences.

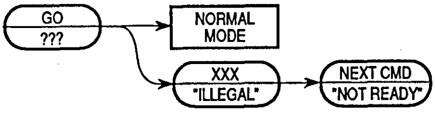
NOTE

A bus error or address error on the first instruction prefetch from the new PC allows BDM to exit and to be trapped as a normal mode exception. The stacked value of the current PC may or may not be valid in this case, depending on the state of the machine prior to entering BDM. In the case of an address error, the PC does not reflect the true return PC. Instead, the stacked fault address is the (odd) return PC.

Command Format:

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

Command Sequence:



Operand Data:

None

Result Data:

None

5.7.2.8.13 Call User Code (CALL). This instruction provides a convenient way to patch user code. The current PC is stacked at the location pointed to by the current SP (SP selected by S-bit latched when BDM entered). The stacked PC serves as a return address to be restored by the return from subroutine (RTS), which terminates the patch routine. The 32-bit operand data is then loaded into the PC. The pipeline is flushed and refilled from the location pointed to by the new PC. BDM is exited, and instruction execution is initiated.

As an example, consider the following code segment that is supposed to output a character to an asynchronous communications interface adaptor. Note the missing check of the transmit data register empty (TDRE) flag.

```
CHKSTAT   MOVE.B   ACIAS,D0      Move ACIA status to D0
          BEQ.B    CHKSTAT       Loop till condition true
          MOVE.B   DATA,ACIAD    Output data
          •
          •
          •
MISSING   ANDI.B   #2,D0         Check for TDRE
          RTS                    Return to in-line code
```

BDM and the CALL command can be used to insert the missing code by observing the following sequence:

1. Breakpoint user program at CHKSTAT;
2. Enter BDM;
3. Execute CALL command to MISSING; Exit BDM;
4. Execute MISSING code; and
5. Return to user program.

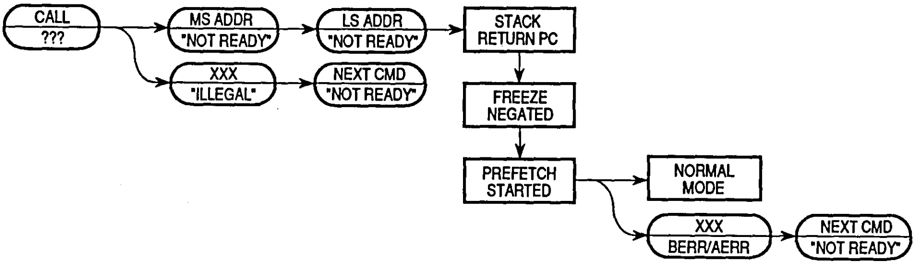
NOTE

Bus errors or address errors that occur during stacking of the return address cause the CPU32 to return an error status via the serial interface and to remain in BDM. A bus error or address error on the first instruction prefetch from the new PC allows BDM to exit and to be trapped as a normal mode exception. The stacked value of the current PC may or may not be valid in this case, depending on the state of the machine prior to entering BDM. In the case of an address error, the return PC does not reflect the true return PC. Instead, the stacked fault address is the (odd) return PC.

Command Format:

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

Command Sequence:



Operand Data:

The 32-bit operand data is the starting location of the patch routine, which is the initial PC upon exiting BDM.

Result Data:

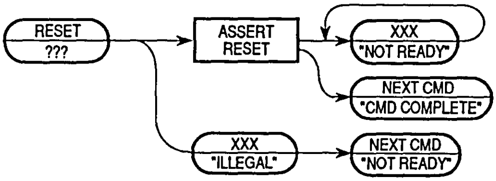
None

5.7.2.8.14 Reset Peripherals (RST). RST asserts $\overline{\text{RESET}}$ for 512 clock cycles. The CPU32 is not reset by this command. This command is synonymous with the CPU32 RESET instruction.

Command Format:

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

Command Sequence:



Operand Data:
None

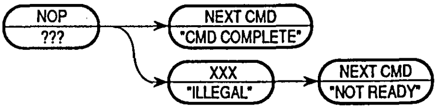
Result Data:
The "command complete" response ($0FFFF) is loaded into the serial shifter after the negation of RESET.

5.7.2.8.15 No Operation (NOP). NOP performs no operation and may be used as a null command where required.

Command Format:

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

Command Sequence:



Operand Data:
None

Result Data:
The "command complete" response ($0FFFF) is returned during the next shift operation.

5.7.2.8.16 Future Commands. Unassigned command opcodes are reserved by Motorola for future expansion. All unused formats within any revision level will perform a NOP and return the ILLEGAL command.

5.7.3 Deterministic Opcode Tracking

The CPU32 utilizes deterministic opcode tracking to trace program execution. Two new signals, IPIPE and IFETCH , provide all the information required to analyze the operation of the instruction pipeline.

5.7.3.1 INSTRUCTION FETCH (IFETCH). IFETCH indicates which bus cycles are accessing data to fill the instruction pipeline. IFETCH is pulse-width modulated to multiplex two indications on a single pin. Asserted for a single clock cycle, IFETCH indicates that the data from the current bus cycle is routed to the instruction pipeline. IFETCH held low for two clock cycles indicates that the instruction pipeline has been flushed. The operand of the bus cycle is used to begin filling the empty pipeline. Both user and supervisor mode fetches are signaled by IFETCH.

Proper tracking of bus cycles via the IFETCH signal on a fast bus requires a simple state machine. On a two-clock bus, IFETCH may signal a pipeline flush with associated prefetch and a consecutive prefetch. That is, IFETCH remains asserted for three clocks, two clocks indicating the flush/fetch and a third clock signaling the second fetch. These two operations are easily discerned if the tracking logic samples IFETCH on the two rising edges of CLKOUT, which follow the address strobe (data strobe during show cycles) falling edge. Three-clock and slower bus cycles allow time for negation of the signal between consecutive indications and do not experience this operation.

5.7.3.2 INSTRUCTION PIPE (IPIPE).

IPIPE signals the advances of the internal instruction pipeline (see Figure 5-36). The pipeline can be modeled as a three-stage FIFO in which data can be used out of both the second and third stages. The instruction register B (IRB) stage, which provides for initial decoding of the opcode and decoding of any extension words, is a source for immediate data. On the other hand, the IRC stage supplies residual decoding of the opcode during instruction execution. Assertion of IPIPE for a single clock cycle indicates the use of data out of the second stage (IRB). Regardless of the presence of valid data in the initial stage (IR), the contents of IRB are invalidated. If the IR stage contains valid data, the data is copied into IRB (IR ➧ IRB), and the IRB stage is revalidated.

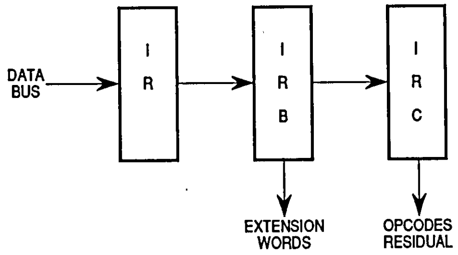


Figure 5-36. Functional Model of Instruction Pipeline

Assertion of $\overline{\text{IPIPE}}$ for two clock cycles indicates the start of a new instruction and subsequent replacement of data in the final stage (IRC). This action causes a full advance of the pipeline: IRB ➧ IRC and IR ➧ IRB. IR is refilled during the next instruction fetch bus cycle. Data loaded into IR propagates through empty pipeline stages automatically, which implies that an accurate model of pipeline operation should include valid bits for the IR and IRB stages. Advancing the pipeline, either explicitly via $\overline{\text{IPIPE}}$ or implicitly by negated valid bits, should set the valid bit of the stage being loaded and negate the valid bit of the register supplying the data.

Because instruction execution is not timed to bus activity, $\overline{\text{IPIPE}}$ is synchronized with the system clock and not the bus. Figure 5-37 illustrates the timing in relation to the system clock. $\overline{\text{IPIPE}}$ should be sampled on the falling edge of the clock. The assertion of $\overline{\text{IPIPE}}$ for a single cycle after deassertion for one or more cycles indicates a use of the data in IRB (advance of IR into IRB). Assertion for two clock cycles indicates that a new instruction has started and both the IR ➧ IRB and IRB ➧ IRC transfers have occurred. Loading IRC always indicates that a new instruction is beginning execution. The opcode is the word loaded into IRC by the transfer.

In some cases, instructions using immediate addressing initiate the start of an instruction and a second pipeline advance. That is, the $\overline{\text{IPIPE}}$ signal is not to be negated between the two indications, which implies the need for a state machine to track the state of $\overline{\text{IPIPE}}$. The state machine can be resynchronized during periods of inactivity on the signal.

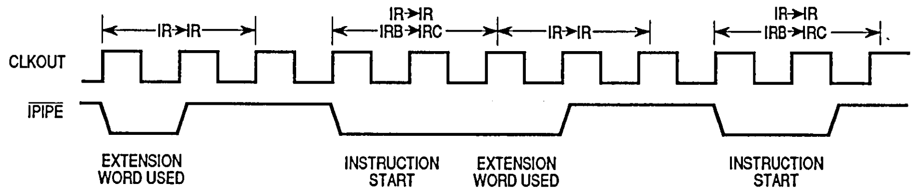


Figure 5-37. Instruction Pipeline Timing Diagram

5.7.3.3 OPCODE TRACKING DURING LOOP MODE. $\overline{\text{IPIPE}}$ and $\overline{\text{IFETCH}}$ continue to work normally during loop mode. $\overline{\text{IFETCH}}$ indicates all instruction fetches up through the point that data begins recirculating within the instruction pipeline. $\overline{\text{IPIPE}}$ continues to signal the start of instructions and the use of extension words even though data is being recirculated internally. $\overline{\text{IFETCH}}$ returns to normal operation with the first fetch after exiting loop mode.

5.8 INSTRUCTION EXECUTION TIMING

This section, which describes the instruction execution timing of the CPU32 using external clock cycles, provides accurate execution and operation timing guidelines but not exact timings for every possible circumstance. This approach is used since exact execution time for an instruction or operation is highly dependent on concurrency of independently scheduled resources, memory speeds, and other variables. The timing numbers presented in this section allow the assembly language programmer or compiler writer to predict the performance of the CPU32. Additionally, the timings for exception processing are included so that designers of multitasking or real-time systems can predict task-switch overhead, maximum interrupt latency, and similar timing parameters. Instruction timings are given in clock cycles to eliminate clock frequency dependencies.

5.8.1 Resource Scheduling

Some of the variability in instruction execution timings results from the overlap of resource utilization. The processor can be viewed as consisting of several independently scheduled resources. Since little of the resource scheduling is directly related to instruction boundaries, it is impossible to make accurate estimates of the time required to execute a particular instruction without knowing the complete context within which the instruction is executing. The position of these resources within the CPU32 is shown in Figure 5-38.

5.8.1.1 MICROSEQUENCER. The microsequencer is either executing microinstructions or awaiting completion of accesses necessary to continue executing microcode. The microsequencer controls the bus controller, instruction execution, and internal processor operations such as calculation of effective address and setting of condition codes. The microsequencer initiates instruction word prefetches after a change of flow and controls the validation of instruction words in the instruction pipeline.

5.8.1.2 INSTRUCTION PIPELINE. The CPU32 contains a two-word instruction pipeline where instruction opcodes are decoded. As shown in Figure 5-38, instruction words (instruction operation words and all extension words) enter the pipeline at stage B. To reach stage C, an instruction word must have been completely decoded. Each of the pipeline stages has a status bit that reflects whether or not the word in that stage was loaded with data from a bus cycle which terminated abnormally. Stages of the pipeline are filled from an initial request by the microsequencer and are subsequently filled by the prefetch controller as they are emptied.

The instruction pipeline contains an additional stage which serves as a buffer. Prefetches completing on the bus before stage B of the instruction pipeline have been emptied are temporarily stored in this buffer.

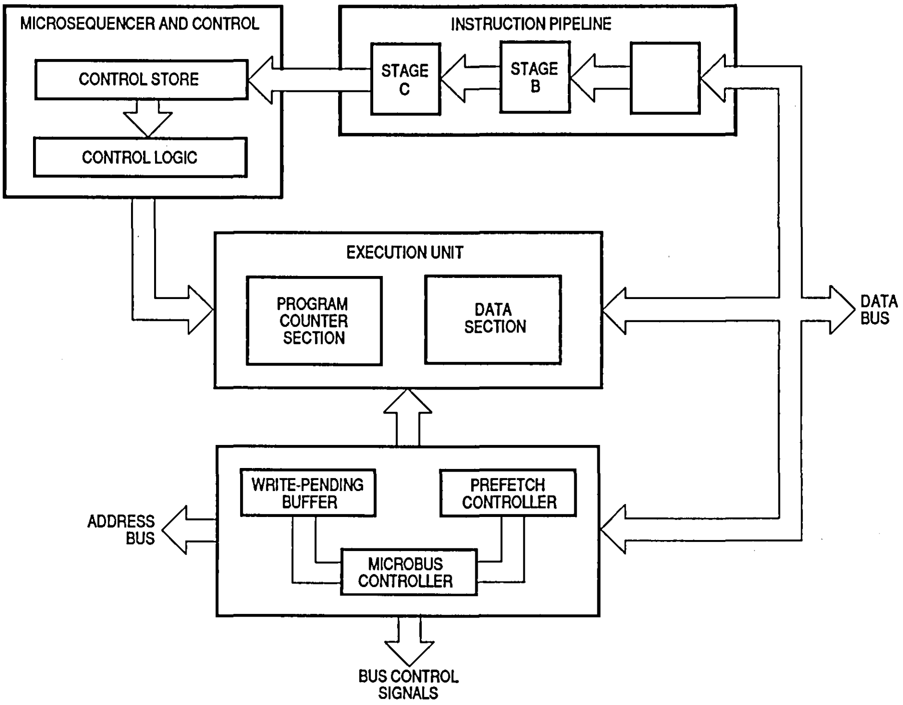


Figure 5-38. Block Diagram of Independent Resources

5.8.1.3 BUS CONTROLLER RESOURCES. The bus controller and microsequencer can operate concurrently. The bus controller can perform a read or write or schedule a prefetch while the microsequencer controls an effective address calculation or sets the condition codes. The microsequencer may also request a bus cycle that the bus controller cannot perform immediately. In this case, the bus cycle is queued, and the bus controller runs the cycle when the current cycle is complete.

The bus controller consists of the instruction prefetch controller, the write-pending buffer, and the microbus controller. These three resources transact all reads, writes, and instruction prefetches required for instruction execution.

5.8.1.3.1 Prefetch Controller. The instruction prefetch controller receives an initial request from the microsequencer to initiate prefetching at a given address. Subsequent prefetches are requested by the prefetch controller whenever a pipeline stage is invalidated, either through completion of an instruction or use of extension words. The prefetch occurs as soon as the bus is free of operand accesses already requested by the microsequencer. Additional state information permits the controller to inhibit prefetch requests when a change in instruction flow (e.g., JMP) is anticipated.

For the typical program, a change of flow can be expected in approximately 10 to 25 percent of the instructions executed. Each time this happens, the instruction pipeline must be flushed and refilled from the new instruction stream. If priority were given to instruction prefetches rather than to operand accesses, it is likely that many instruction words would be flushed and never used, meanwhile delaying needed operand cycles. To maximize the available bus bandwidth, the CPU32 will schedule a prefetch only when the next instruction is not a change-of-flow instruction and when room exists in the pipeline for the prefetch.

5.8.1.3.2 Write-Pending Buffer. The CPU32 incorporates a single-operand write-pending buffer, allowing the microsequencer to continue execution after the request for a write cycle is queued in the bus controller. The time occupied by the write at the end of an instruction can utilize the next instruction's head cycle time, thus reducing overall execution time. Interlocks prevent the microsequencer from overwriting this buffer.

5.8.1.3.3 Microbus Controller. The microbus controller performs the bus cycles issued to the bus controller by the microsequencer. Operand accesses always take priority over instruction prefetches. Word and byte operands are accessed in a single CPU-initiated bus cycle, although the external bus interface may be required to initiate a second cycle in the case of a word operand to a byte-sized external port. Long operands are accessed in two bus cycles, the most significant word first.

The goal of the bus controller is to maximize the useful bandwidth of the bus by not starting prefetches when those prefetches will be discarded due to a change of flow. The instruction pipeline is capable of recognizing certain instructions like branches and RTS and informs the bus controller that no more prefetches are required.

5.8.1.4 INSTRUCTION EXECUTION OVERLAP. Overlap is the time, measured in clock cycles, that an instruction executes concurrently with the previous instruction. As illustrated in Figure 5-39, portions of instructions A and B execute simultaneously. The overlapped portion of instruction B is absorbed in the execution time of A. Similarly, the overlap time between instructions B and C reduces the overall execution time of the two instructions. Each instruction contributes to the total overlap time. The portion of the time at the end of the execution time of instruction A that can overlap at the beginning of instruction B is called the tail of instruction A. The portion of time at the beginning of instruction B that can overlap the end of instruction A is called the head of instruction B. The total overlap time between instructions A and B consists of the lesser of the tail of A and the head of B.

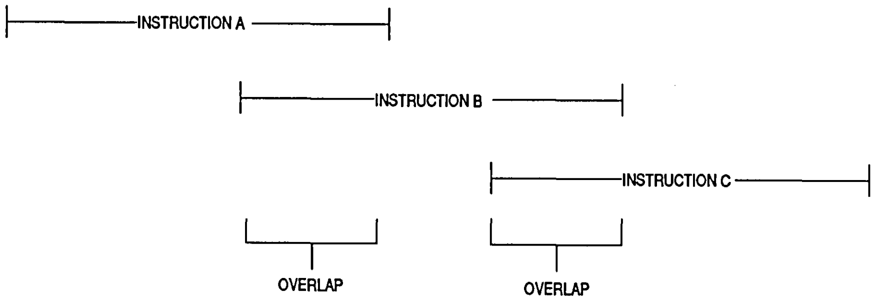


Figure 5-39. Simultaneous Instruction Execution

The execution time attributed to instructions A, B, and C after considering the overlap is illustrated in Figure 5-40. The overlap time is attributed to the execution time of the completing instruction. The following equation shows the method for calculating the overlap time:

$$\text{Overlap} = \text{in } (\text{Tail}_{N}, \text{Head}_{N+1})$$

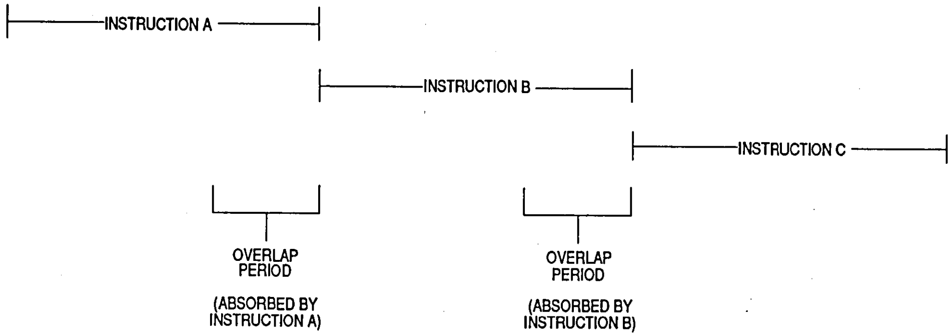


Figure 5-40. Attributed Instruction Times

5

5.8.1.5 EFFECTS OF WAIT STATES. The CPU32 is capable of accessing on-chip memory and peripherals with an access time of two clocks. While it is possible to get two-clock external accesses when the bus is operated in a synchronous mode, the typical external memory speed is three clocks or more.

All instruction times given in the following timing tables assume that both instruction fetches and operand cycles are to the two-clock memory and are for word access only (unless explicitly mentioned otherwise). Any time a long access is made, the time for the additional bus cycle(s) must be added to the overall execution time. Wait states due to slow external memory must be added into that memory for each bus cycle.

A typical application will have a mixture of bus speeds: e.g., program executing from an off-chip ROM, accesses to on-chip peripherals, storage of variables in a slower off-chip RAM, and external peripherals with speeds ranging from moderate to very slow. To arrive at an accurate instruction time calculation, each bus access must be individually considered. Many instructions have a head cycle count, which can overlap the cycles of an operand fetch to slower memory started by the previous instruction. For such cases, the increase in access time has no effect on the total execution time of that pair of instructions.

When tracing the execution time of instructions by monitoring the external bus, note that the order of operand accesses is always the same for a particular instruction sequence, and, provided the bus speed is identical across those sequences, the interleaving of instruction prefetches with operands is also identical.

5.8.1.6 INSTRUCTION EXECUTION TIME CALCULATION. The overall execution time for an instruction may depend on the overlap with the previous and following instructions. Therefore, to calculate instruction time estimations, the entire code sequence must be analyzed as a whole. To derive the actual instruction execution times for an instruction sequence, the instruction times listed in the tables must be adjusted to account for the overlap with previous and subsequent instructions.

The formula for this calculation is as follows:

$$C1 - \min(T1, H2) + C2 - \min(T2, H3) + C3 - \min(T3, H4) + \ldots$$

where:

C_N is the number of cycles listed for instruction N

H_N is the head time for instruction N

T_N is the tail time for instruction N

min (T_N, H_M) is the minimum of parameters T_N and H_M

The number of cycles for the instruction (C_N above) can also be composed of one or two effective address calculations in addition to the raw number in the cycles column. In these cases, overall instruction time is calculated as if it were multiple instructions according to the following equation:

$$\langle CEA \rangle - \min(T_{EA}, H_{OP}) + C_{OP}$$

where:

⟨CEA⟩ is the instruction's effective address time

C_{OP} is the instruction's operation time

H_{OP} is the instruction operation's head time

T_{EA} is the effective address' tail time

min (T_N, H_M) is the minimum of parameters T_N and H_M

The overall head for the instruction is the head for the effective address, and the overall tail for the instruction is the tail for the operation. Therefore, the actual equation of the execution time becomes:

$$C_{OP}1 - \min(T_{OP}1, H_{EA}2) + \langle CEA2 \rangle - \min$$
$$(T_{EA}2, H_{OP}2) + C_{OP}2 - \min(T_{OP}2, H_{EA}3) + \ldots$$

Every instruction must prefetch to replace itself in the instruction pipe. Usually, these prefetches occur during or after the instruction. A prefetch is permitted to begin in the first clock of any indexed effective addressing mode. Additionally, a prefetch for an instruction is permitted to begin two clocks before the end of an instruction, provided the bus is not being used. If the bus is being used, then the prefetch will occur at the next available time when the bus would otherwise be idle.

5.8.1.7 EFFECTS OF NEGATIVE TAILS. When the CPU32 changes instruction flow, the instruction decode pipeline must begin refilling before instruction execution can resume. Refilling forces a two-clock idle period at the end of the change-of-flow instruction, which can be used to prefetch an additional word on the new instruction path. Because of the stipulation that each instruction must prefetch to replace itself, the concept of negative tails has been introduced to account for these free clocks on the bus.

On a two-clock bus, it is not necessary to adjust the instruction timings to account for the potential extra prefetch. The cycle time of the microsequencer and bus are matched; therefore, no additional benefit or penalty is obtained.

On slower buses, negative tails are used to compensate for the pipeline delays by increasing the caculated instruction overlap. Normally, increasing the length of prefetch bus cycles directly affects the cycle count and tail values found in the tables. By using the following equations, negative tail values are used to negate the effects of the slower bus.

Note that many instructions listed as having negative tails are change-of-flow instructions and that the bus speed used in the calculation is the new instruction stream.

Use the following equations to apply a negative tail on a bus slower than two clocks. However, the equations are generalized so that they can be used on any speed bus with any tail value.

```
NEW–TAIL = OLD–TAIL + (NEW–CLOCK – 2)
IF ((NEW–CLOCK – 4) >0) then
  NEW–CYCLE = OLD–CYCLE + (NEW–CLOCK – 2) + (NEW–CLOCK  – 4)
ELSE
  NEW–CYCLE = OLD–CYCLE + (NEW–CLOCK  – 2)
```

where:
NEW–TAIL/NEW–CYCLE is the adjusted tail/cycle at the slower speed
OLD–TAIL/OLD–CYCLE is the value listed in the instruction timing tables
NEW–CLOCK is the number of clocks per cycle at the slower speed

5.8.2 Instruction Stream Timing Example

Some programming examples will allow a more detailed examination of these effects. For all examples, the memory access is from external synchronous memory, the bus is idle, and the instruction pipeline is full at the start.

5.8.2.1 TIMING EXAMPLE 1: EXECUTION OVERLAP. The example shown in Figure 5-41 illustrates the overlapping of execution due to the bus controller's ability to execute bus cycles while the sequencer is calculating the next effective address. One clock is saved between each instruction since that is the minimum time of the individual head and tail numbers.

Instructions

```
MOVE.W    A1, (A0)+
ADDQ.W    #1, (A0)
CLR.W     $30 (A1)
```

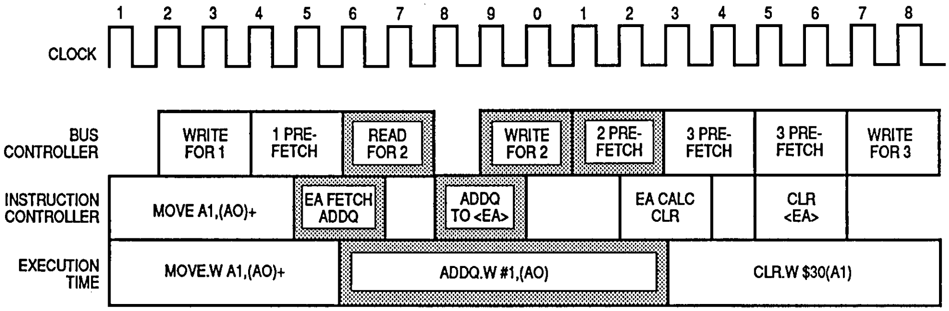


Figure 5-41. Example 1 — Instruction Stream

5.8.2.2 TIMING EXAMPLE 2: BRANCH INSTRUCTIONS.

Example 2 shows what happens when a branch instruction is executed for both the taken and not-taken cases (see Figures 5-42 and 5-43). The instruction stream is for a simple limit check with the variable already in a data register.

Instructions

```
MOVEQ     #7, D1
CMP.L     D1, D0
BLE.B     NEXT
MOVE.L    D1, (A0)
```

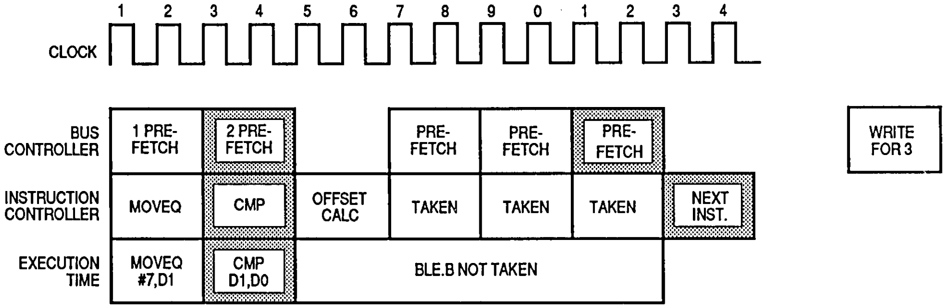


Figure 5-42. Example 2 — Branch Taken

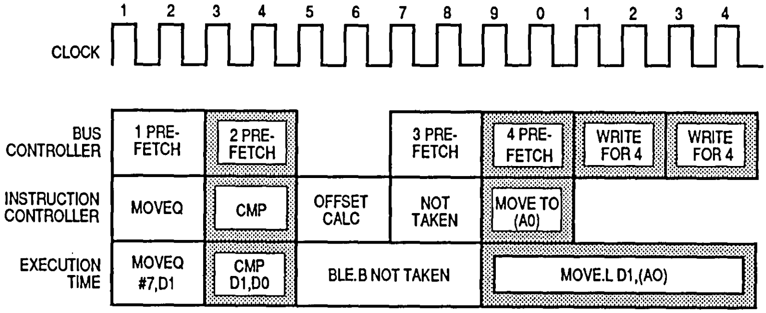


Figure 5-43. Example 2 — Branch Not Taken

5.8.2.3 TIMING EXAMPLE 3: NEGATIVE TAILS. This example (see Figure 5-44) shows how to properly account for the negative tail figures for branches and other change-of-flow instructions. For this example, the bus speed is assumed to be four clocks per access. Instruction three is at the branch destination.

Instructions

```
MOVEQ     #7, D1
BRA.W     FARAWAY
MOVE.L    D1, D0
```

The CPU32 has a two-word instruction pipeline, but, due to internal delays, the minimum time for a branch instruction allows three bus cycles. The negative tail is intended to serve as a reminder that on a fast bus an extra two clocks are available for prefetching a third word but that on a slower bus the third word is not forced to be fetched.

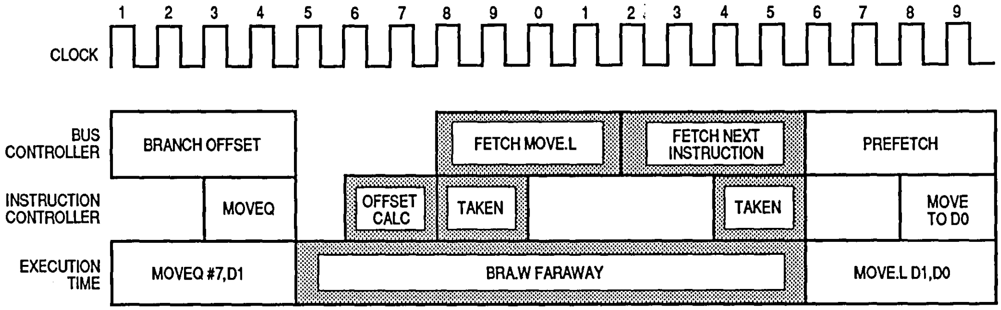


Figure 5-44. Example 3 — Branch Negative Tail

Example 3 actually illustrates three different considerations in calculating the time for an instruction. The branch instruction does not attempt to prefetch beyond the minimum number of words needed for itself; the negative tail allows execution to begin sooner than would be calculated for a three-word pipeline; and there is a one-clock delay caused by the displacement arriving late at the CPU32.

The negative tail only needs to be calculated on changes of flow, but the concept can be generalized to any instruction so that only two words are required to be in the pipeline, but up to three words may be present. When there is an opportunity for the extra prefetch, it is made. A prefetch to replace an instruction can begin ahead of the instruction, resulting in a faster processor.

5.8.3 Instruction Timing Tables

5

The following assumptions apply to the times shown in the tables in this section:

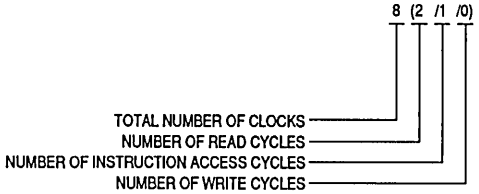
- A 16-bit data bus is used for all memory accesses.
- All memory accesses occur with two-clock bus cycles and no wait states.
- The instruction pipeline is full at the beginning of the instruction and is refilled by the end of the instruction.

Three values are listed for each instruction and addressing mode:

Head This value is the number of cycles at the beginning of the instruction available for the previous instruction's write to complete or for a prefetch to occur.

Tail This value is the number of cycles at the end of the instruction used by the instruction to complete a write.

Cycles This field contains four numbers per entry, three of which are contained in parenthesis. The outer number represents the minimum number of cycles required for the instruction to complete. Within the parenthesis, the numbers represent the number of bus accesses performed by the instruction. The first number inside the parenthesis is the number of operand read accesses performed by the instruction. The second number is the number of instruction fetches performed by the instruction, including all prefetches to keep the instruction and the instruction pipeline filled. The third number is the number of write accesses performed by the instruction.



The total number of bus-activity clocks and internal clocks (not overlapped by bus activity) of the instruction in this example are derived as follows:

(2 reads 2 clocks/read)+
(1 instruction access × 2 clocks/access)+
(0 writes × 2 clocks/write) = 6 clocks of bus activity
8 clocks total − 6 clocks bus activity = 2 internal clocks

One example from the timing tables is the ADD.L (12, A3, D7.W • 4), D2 instruction, with the instructions and data from two-clock memory. The effective addressing mode is listed as head=4, tail=4, cycles=10 (2/1/0). The difference from FEA timing table (see **5.8.3.1 FETCH EFFECTIVE ADDRESS**) is because the table is listed for word accesses, and this example is for a long access. The instruction itself has a head=0, tail=0, and cycles=2(0/1/0) from the arithmetic/logical timing table (see **5.8.3.5 ARITHMETIC/LOGICAL INSTRUCTIONS**). Assuming no trailing write exists from the previous instruction, the execution time is calculated in the following manner.

The effective address calculation requires six clocks, with the replacement fetch for the effective address occurring during this time (leaving a head of four). If there had not been time in the head to perform the prefetch due to a previous trailing write, then time must be allotted in the middle of the instruction or after the tail to do the prefetches. The read of the memory requires two bus cycles at two clocks each. This read time, implied in the tail figure for the effective address, cannot be overlapped with the instruction since the instruction has a head of zero. An additional two clocks are required for the actual ADD, which makes the total 6+4+2=12 clocks. If the bus cycles take more time (i.e., the memory is off-chip), then add the appropriate number of clocks to each memory access.

An example of the overlapped execution possible on the CPU32 is with the instruction sequence MOVE.L D0, (A0) followed by LSL.L #7, D2. The MOVE has a head of zero and a tail of four, since it is a long write. The LSL has a head of four; therefore, the trailing write from the MOVE will overlap the LSL completely. Thus, this two-instruction sequence has a head of zero and a tail of zero and a total execution of eight clocks instead of 12 clocks obtained by adding the individual cycle times.

General observations regarding calculation of execution time are as follows:

- Any time the number of bus cycles is listed as "y," substitute a value of one for byte and word cycles and a value of two for long cycles. For long bus cycles, usually add a value of two to the tail.

- The time calculated for an instruction on a three-clock (or longer) bus is usually longer than the actual execution time. All times shown are for two-clock bus cycles.

- If the previous instruction has a negative tail, then a prefetch for the current instruction may begin during the execution of the previous instruction in advance of the instruction needing the prefetch.

- Certain instructions requiring an immediate extension word (immediate word effective address, absolute word effective address, address register indirect with displacement effective address, conditional branches with word offsets, bit operations, LPSTOP, TBL, MOVEM, MOVEC, MOVES, MOVEP, MUL.L, DIV.L, CHK2, CMP2, and DBcc) are not permitted to begin until the extension word has been in the instruction pipeline for at least one cycle. This does not apply to long offsets or displacements.

5.8.3.1 FETCH EFFECTIVE ADDRESS. The fetch effective address table indicates the number of clock periods needed for the processor to calculate and fetch the specified effective address. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles	Notes
Dn	—	—	0(0/0/0)	—
An	—	—	0(0/0/0)	—
(An)	1	1	3(y/0/0)	1
(An)+	1	1	3(y/0/0)	1
–(An)	2	2	4(y/0/0)	1
(d_{16},An) or (d_{16},PC)	1	3	5(y/1/0)	1,3
(yyy).W	1	3	5(y/1/0)	1
(yyy).L	1	5	7(y/2/0)	1
#⟨data⟩.W	1	1	3(0/1/0)	1
#⟨data⟩.B	1	1	3(0/1/0)	1
#⟨data⟩.L	1	3	5(0/2/0)	1
(d_8,An,Xn.Sz × Sc) or (d_8,PC,Xn.Sz × Sc)	4	2	8(y/1/0)	1,2,3,4
(O) (All Suppressed)	2	2	6(y/1/0)	1,4
(d_{16})	1	3	7(y/2/0)	1,4
(d_{32})	1	5	9(y/3/0)	1,4
(An)	1	1	5(y/1/0)	1,2,4
(Xm.Sz × Sc)	4	2	8(y/1/0)	1,2,4
(An,Xm.Sz × Sc)	4	2	8(y/1/0)	1,2,3,4
(d_{16},An) or (d_{16},PC)	1	3	7(y/2/0)	1,3,4
(d_{32},An) or (d_{32},PC)	1	5	9(y/3/0)	1,3,4
(d_{16},An,Xm) or (d_{16},PC,Xm)	2	2	8(y/2/0)	1,3,4
(d_{32},An,Xm) or (d_{32},PC,Xm)	1	3	9(y/3/0)	1,3,4
(d_{16},An,Xm.Sz × Sc) or (d_{16},PC,Xm.Sz × Sc)	2	2	8(y/2/0)	1,2,3,4
(d_{32},An,Xm.Sz × Sc) or (d_{32},PC,Xm.Sz × Sc)	1	3	9(y/3/0)	1,2,3,4

y = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

NOTES:

1. The read of the effective address and replacement fetches overlap the head of the operation by the amount specified in the tail.
2. Size and scale of the index register do not affect execution time.
3. The program counter may be substituted for the base address register An.
4. When adjusting the prefetch time for slower buses, extra clocks may be subtracted from the head until the head reaches zero, at which time additional clocks must be added to both the tail and cycle counts.

5.8.3.2 CALCULATE EFFECTIVE ADDRESS.

The calculate effective address table indicates the number of clock periods needed for the processor to calculate the specified effective address. The timing is equivalent to fetch effective address except there is no read cycle. The tail and cycle time are reduced by the amount of time the read would occupy. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles	Notes
Dn	—	—	0(0/0/0)	—
An	—	—	0(0/0/0)	—
(An)	1	0	2(0/0/0)	—
(An)+	1	0	2(0/0/0)	—
−(An)	2	0	2(0/0/0)	—
(d$_{16}$,An) or (d$_{16}$,PC)	1	1	3(0/1/0)	1,3
(yyy).W	1	1	3(0/1/0)	1
(yyy).L	1	3	5(0/2/0)	1
(d$_{8}$,An,Xn.Sz × Sc) or (d$_{8}$,PC,Xn.Sz × Sc)	4	0	6(0/1/0)	2,3,4
(O) (All Suppressed)	2	0	4(0/1/0)	4
(d$_{16}$)	1	1	5(0/2/0)	1,4
(d$_{32}$)	1	3	7(0/3/0)	1,4
(An)	1	0	4(0/1/0)	4
(Xm.Sz × Sc)	4	0	6(0/1/0)	2,4
(An,Xm.Sz × Sc)	4	0	6(0/1/0)	2,4
(d$_{16}$,An) or (d$_{16}$,PC)	1	1	5(0/2/0)	1,3,4
(d$_{32}$,An) or (d$_{32}$,PC)	1	3	7(0/3/0)	1,3,4
(d$_{16}$,An,Xm) or (d$_{16}$,PC,Xm)	2	0	6(0/2/0)	3,4
(d$_{32}$,An,Xm) or (d$_{32}$,PC,Xm)	1	1	7(0/3/0)	1,3,4
(d$_{16}$,An,Xm.Sz × Sc) or (d$_{16}$,PC,Xm.Sz × Sc)	2	0	6(0/2/0)	2,3,4
(d$_{32}$,An,Xm.Sz × Sc) or (d$_{32}$,PC,Xm.Sz × Sc)	1	1	7(0/3/0)	1,2,3,4

y = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

NOTES:

1. Replacement fetches overlap the head of the operation by the amount specified in the tail.
2. Size and scale of the index register do not affect execution time.
3. The program counter may be substituted for the base address register An.
4. When adjusting the prefetch time for slower buses, extra clocks may be subtracted from the head until the head reaches zero, at which time additional clocks must be added to both the tail and cycle counts.

5.8.3.3 MOVE INSTRUCTION. The MOVE instruction table indicates the number of clock periods needed for the processor to calculate the destination effective address and to perform the MOVE or MOVEA instruction. For entries with CEA or FEA, refer to the appropriate table to calculate this portion of the instruction timing. The destination effective addresses are divided by their formats (refer to **5.3.4.4 EFFECTIVE ADDRESS ENCODING SUMMARY**). The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

When using this table, begin at the top and move downward. Use the first entry that matches both source and destination addressing modes.

Instruction	Head	Tail	Cycles
MOVE Rn, Rn	0	0	2(0/1/0)
MOVE ⟨FEA⟩, Rn	0	0	2(0/1/0)
MOVE Rn, (Am)	0	2	4(0/1/y)
MOVE Rn, (Am)+	1	1	5(0/1/y)
MOVE Rn, −(Am)	2	2	6(0/1/y)
MOVE Rn, ⟨CEA⟩	1	3	5(0/1/y)
MOVE #, ⟨CEA⟩	2	2	6(0/1/y)*
MOVE ⟨FEA⟩, (An)	2	2	6(0/1/y)
MOVE ⟨FEA⟩, (An)+	2	2	6(0/1/y)
MOVE ⟨FEA⟩, −(An)	2	2	6(0/1/y)
MOVE ⟨CEA⟩, ⟨FEA⟩	2	2	6(0/1/y)

y = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

* = An # fetch effective address time must be added for this instruction (e.g., ⟨FEA⟩+⟨CEA⟩+⟨OPER⟩).

NOTE:

For instructions not explicitly listed, use the (MOVE ⟨CEA⟩, ⟨FEA⟩) entry. The source effective address is calculated by the calculate effective address table, and the destination effective address is calculated by the fetch effective address table, even though the bus cycle is for the source effective address.

5.8.3.4 SPECIAL-PURPOSE MOVE INSTRUCTION. The special-purpose MOVE instruction table indicates the number of clock periods needed for the processor to fetch, calculate, and perform the special-purpose MOVE operation on the control registers or specified effective address. Footnotes indicate when to account for the appropriate effective address times. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
EXG	Rn, Rm	2	0	4(0/1/0)
MOVEC	Cr, Rn	10	0	14(0/2/0)
MOVEC	Rn, Cr	12	0	14–16(0/1/0)
MOVE	CCR, Dn	2	0	4(0/1/0)
MOVE	CCR, ⟨CEA⟩	0	2	4(0/1/1)
MOVE	Dn, CCR	2	0	4(0/1/0)
MOVE	⟨FEA⟩, CCR	0	0	4(0/1/0)
MOVE	SR, Dn	2	0	4(0/1/0)
MOVE	SR, ⟨CEA⟩	0	2	4(0/1/1)
MOVE	Dn, SR	4	−2	10(0/3/0)
MOVE	⟨FEA⟩, SR	0	−2	10(0/3/0)
MOVEM.W	⟨CEA⟩, RL	1	0	8+n•4(n+1, 2, 0)*
MOVEM.W	RL, ⟨CEA⟩	1	0	8+n•4(0, 2, n)*
MOVEM.L	⟨CEA⟩, RL	1	0	12+n•4(2n+2, 2, 0)
MOVEM.L	RL, ⟨CEA⟩	1	2	10+n•4(0, 2, 2n)
MOVEP.W	Dn, (d_{16}, An)	2	0	10(0/2/2)
MOVEP.W	(d_{16}, An), Dn	1	2	11(2/2/0)
MOVEP.L	Dn, (d_{16}, An)	2	0	14(0/2/4)
MOVEP.L	(d_{16}, An), Dn	1	2	19(4/2/0)
MOVES (Save)	⟨CEA⟩, Rn	1	1	3(0/1/0)
MOVES (Op)	⟨CEA⟩, Rn	7	1	11(y/1/0)
MOVES (Save)	Rn, ⟨CEA⟩	1	1	3(0/1/0)
MOVES (Op)	Rn, ⟨CEA⟩	9	2	12(0/1/y)
MOVE	USP, An	0	0	2(0/1/0)
MOVE	An, USP	0	0	2(0/1/0)
SWAP	Dn	4	0	6(0/1/0)

y = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.
* = Each bus cycle may take up to four clocks without increasing total execution time.
Cr = Control registers USP, VBR, SFC, and DFC
n = Number of registers to transfer
RL = Register list
< = Maximum time is indicated; certain data or mode combinations may execute faster.

NOTE: The MOVES instruction involves a save step which other instructions do not have. To calculate total the instruction time, calculate the save, the effective address, and the operation execution times, and combine in the order listed, using the equations given in **5.8.1.6 INSTRUCTION EXECUTION TIME CALCULATION.**

5.8.3.5 ARITHMETIC/LOGICAL INSTRUCTIONS. The arithmetical/logical instruction table indicates the number of clock periods needed for the processor to perform the specified arithmetical/logical instruction using the specified addressing mode. Footnotes indicate when to account for the appropriate effective address times. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
ADD(A)	Rn, Rm	0	0	2(0/1/0)
ADD(A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
ADD	Dn, ⟨FEA⟩	0	3	5(0/1/y)
AND	Dn, Dm	0	0	2(0/1/0)
AND	⟨FEA⟩, Dn	0	0	2(0/1/0)
AND	Dn, ⟨FEA⟩	0	3	5(0/1/y)
EOR	Dn, Dm	0	0	2(0/1/0)
EOR	Dn, ⟨FEA⟩	0	3	5(0/1/y)
OR	Dn, Dm	0	0	2(0/1/0)
OR	⟨FEA⟩, Dn	0	0	2(0/1/0)
OR	Dn, ⟨FEA⟩	0	3	5(0/1/y)
SUB (A)	Rn, Rm	0	0	2(0/1/0)
SUB (A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
SUB	Dn, ⟨FEA⟩	0	3	5(0/1/y)
CMP(A)	Rn, Rm	0	0	2(0/1/0)
CMP(A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
CMP2 (Save)	⟨FEA⟩, Rn	1	1	3(0/1/0)
CMP2 (Op)	⟨FEA⟩, Rn	2	0	18(y/1/0)
MULsu.W	⟨FEA⟩, Dn	0	0	26(0/1/0)
MULsu.L (Save)	⟨FEA⟩, Dn	1	1	3(0/1/0)
MULsu.L (Op)	⟨FEA⟩, Dn	2	0	50(0/0/0)
DIVU.W	⟨FEA⟩, Dn	0	0	32(0/1/0)
DIVS.W	⟨FEA⟩, Dn	0	0	42(0/1/0)
DIVU.W	⟨FEA⟩, Dn	0	0	32(0/1/0)
DIVS.W	⟨FEA⟩, Dn	0	0	42(0/1/0)
DIVU.L (Save)	⟨FEA⟩, Dn	1	1	3(0/1/0)
DIVU.L (Op)	⟨FEA⟩, Dn	2	0	46(0/0/0)
DIVS.L (Save)	⟨FEA⟩, Dn	1	1	3(0/1/0)
DIVS.L (Op)	⟨FEA⟩, Dn	2	0	62(0/0/0)
TBLsu	Dn:Dm, Dp	26	0	28-30(0/2/0)
TBLsu (Save)	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBLsu (Op)	⟨CEA⟩, Dn	6	0	33-35(2y/1/0)
TBLNs	Dn:Dm, Dp	30	0	30-34(0/2/0)
TBLNs (Save)	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBLNs (Op)	⟨CEA⟩, Dn	6	0	35-39(2y/1/0)
TBLNu	Dn:Dm, Dp	30	0	34-40(0/2/0)
TBLNu (Save)	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBLNu (Op)	⟨CEA⟩, Dn	6	0	29-45(2y/1/0)

y = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

< = Maximum time is indicated; certain data or mode combinations will execute faster.

su = The execution time is identical for signed or unsigned operands.

NOTE: The CMP2, MUL.L, DIV.L, TBL, and TBLN instructions involve a save step which other instructions do not have. To calculate the total instruction time, calculate the save, the effective address, and the operation execution times, and combine in the order listed, using the equations given in **5.8.1.6 INSTRUCTION EXECUTION TIME CALCULATION**.

5.8.3.6 IMMEDIATE ARITHMETIC/LOGICAL INSTRUCTIONS. The immediate arithmetical/logical instruction table indicates the number of clock periods needed for the processor to fetch the source immediate data value and to perform the specified arithmetical/logical instruction using the specified addressing mode. Footnotes indicate when to account for the appropriate fetch effective or fetch immediate effective address times. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
MOVEQ	#, Dn	0	0	2(0/1/0)
ADDQ	#, Rn	0	0	2(0/1/0)
ADDQ	#, ⟨FEA⟩	0	3	5(0/1/y)
SUBQ	#, Rn	0	0	2(0/1/0)
SUBQ	#, ⟨FEA⟩	0	3	5(0/1/y)
ADDI	#, Rn	0	0	2(0/1/0)*
ADDI	#, ⟨FEA⟩	0	3	5(0/1/y)*
ANDI	#, Rn	0	0	2(0/1/0)*
ANDI	#, ⟨FEA⟩	0	3	5(0/1/y)*
EORI	#, Rn	0	0	2(0/1/0)*
EORI	#, ⟨FEA⟩	0	3	5(0/1/y)*
ORI	#, Rn	0	0	2(0/1/0)*
ORI	#, ⟨FEA⟩	0	3	5(0/1/y)*
SUBI	#, Rn	0	0	2(0/1/0)*
SUBI	#, ⟨FEA⟩	0	3	5(0/1/y)*
CMPI	#, Rn	0	0	2(0/1/0)*
CMPI	#, ⟨FEA⟩	0	3	5(0/1/y)*

y = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

* = An # fetch effective address time must be added for this instruction (e.g., ⟨FEA⟩ + ⟨FEA⟩ + ⟨OPER⟩).

5.8.3.7 BINARY-CODED DECIMAL AND EXTENDED INSTRUCTIONS. The binary-coded decimal and extended instruction table indicates the number of clock periods needed for the processor to perform the specified operation using the specified addressing mode. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
ABCD	Dn, Dm	2	0	4(0/1/0)
ABCD	−(An), −(Am)	2	2	12(2/1/1)
SBCD	Dn, Dm	2	0	4(0/1/0)
SBCD	−(An), −(Am)	2	2	12(2/1/1)
ADDX	Dn, Dm	0	0	2(0/1/0)
ADDX	−(An), −(Am)	2	2	10(2/1/1)
SUBX	Dn, Dm	0	0	2(0/1/0)
SUBX	−(An), −(Am)	2	2	10(2/1/1)
CMPM	(An)+, (Am)+	1	0	8(2/1/0)

5.8.3.8 SINGLE OPERAND INSTRUCTIONS. The single operand instruction table indicates the number of clock periods needed for the processor to perform the specified operation using the specified addressing mode. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
CLR	Dn	0	0	2(0/1/0)
CLR	⟨CEA⟩	0	2	4(0/1/y)
NEG	Dn	0	0	2(0/1/0)
NEG	⟨FEA⟩	0	3	5(0/1/y)
NEGX	Dn	0	0	2(0/1/0)
NEGX	⟨FEA⟩	0	3	5(0/1/y)
NOT	Dn	0	0	2(0/1/0)
NOT	⟨FEA⟩	0	3	5(0/1/y)
EXT	Dn	0	0	2(0/1/0)
NBCD	Dn	2	0	4(0/1/0)
NBCD	⟨FEA⟩	0	2	6(0/1/1)
Scc	Dn	2	0	4(0/1/0)
Scc	⟨CEA⟩	2	2	6(0/1/1)
TAS	Dn	4	0	6(0/1/0)
TAS	⟨CEA⟩	1	0	10(0/1/1)
TST	⟨FEA⟩	0	0	2(0/1/0)

y = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

5.8.3.9 SHIFT/ROTATE INSTRUCTIONS. The shift/rotate instruction table indicates the number of clock periods needed for the processor to perform the specified operation on the given addressing mode. Footnotes indicate when to account for the appropriate effective address times. The number of bits shifted does not affect the execution time, unless noted. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles	Notes
LSd	Dn, Dm	−2	0	(0/1/0)	1
LSd	#, Dm	4	0	6(0/1/0)	—
LSd	〈FEA〉	0	2	6(0/1/1)	—
ASd	Dn, Dm	−2	0	(0/1/0)	1
ASd	#, Dm	4	0	6(0/1/0)	—
ASd	〈FEA〉	0	2	6(0/1/1)	—
ROd	Dn, Dm	−2	0	(0/1/0)	1
ROd	#, Dm	4	0	6(0/1/0)	—
ROd	〈FEA〉	0	2	6(0/1/1)	—
ROXd	Dn, Dm	−2	0	(0/1/0)	2
ROXd	#, Dm	−2	0	(0/1/0)	3
ROXd	〈FEA〉	0	2	6(0/1/1)	—

d = Direction (left or right).

NOTES:

1. Execution time is calculated by this formula: $\max(3+(n/4)+\mathrm{mod}\,(n, 4)+\mathrm{mod}(((n/4\,)+\mathrm{mod}(n, 4)+1,2)\,,6)$ or by the following table.
2. Execution time is calculated by this formula (count $\leqslant$ 63): $\max(3+n+\mathrm{mod}(n+1, 2), 6)$.
3. Execution time is calculated by this formula (count $\leqslant$ 8): $\max(2+n+\mathrm{mod}(n, 2), 6)$.

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

5.8.3.10 BIT MANIPULATION INSTRUCTIONS. The bit manipulation instruction table indicates the number of clock periods needed for the processor to perform the specified operation on the given addressing mode. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles
BCHG #, Dn	2	0	6(0/2/0)*
BCHG Dn, Dm	4	0	6(0/1/0)
BCHG #, ⟨FEA⟩	1	2	8(0/2/1)*
BCHG Dn, ⟨FEA⟩	2	2	8(0/1/1)
BCLR #, Dn	2	0	6(0/2/0)*
BCLR Dn, Dm	4	0	6(0/1/0)
BCLR #, ⟨FEA⟩	1	2	8(0/2/1)*
BCLR Dn, ⟨FEA⟩	2	2	8(0/1/1)
BSET #, Dn	2	0	6(0/2/0)*
BSET Dn, Dm	4	0	6(0/1/0)
BSET #, ⟨FEA⟩	1	2	8(0/2/1)*
BSET Dn, ⟨FEA⟩	2	2	8(0/1/1)
BTST #, Dn	2	0	4(0/2/0)*
BTST Dn, Dm	2	0	4(0/1/0)
BTST #, ⟨FEA⟩	1	0	4(0/2/0)*
BTST Dn, ⟨FEA⟩	2	0	4(0/1/0)

* = An # fetch effective address time must be added for this instruction (e.g., ⟨FEA⟩+⟨FEA⟩+⟨OPER⟩).

5.8.3.11 CONDITIONAL BRANCH INSTRUCTIONS. The conditional branch instruction table indicates the number of clock periods needed for the processor to perform the specified branch on the given branch size, with complete execution times given. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles
Bcc (taken)	2	−2	8(0/2/0)
Bcc.B (not taken)	2	0	4(0/1/0)
Bcc.W (not taken)	0	0	4(0/2/0)
Bcc.L (not taken)	0	0	6(0/3/0)
DBcc (T, not taken)	1	1	4(0/2/0)
DBcc (F, −1, not taken)	2	0	6(0/2/0)
DBcc (F, not −1, taken)	6	−2	10(0/2/0)
DBcc (T, not taken)	4	0	6(0/1/0)*
DBcc (F, −1, not taken)	6	0	8(0/1/0)*
DBcc (F, not −1, taken)	6	0	10(0/0/0)*

* = In loop mode.

5.8.3.12 CONTROL INSTRUCTIONS. The control instruction table indicates the number of clock periods needed for the processor to perform the specified operation on the given addressing mode. Footnotes indicate when to account for the appropriate effective address times. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles
ANDI #, SR	0	−2	12(0/2/0)
EORI #, SR	0	−2	12(0/2/0)
ORI #, SR	0	−2	12(0/2/0)
ANDI #, CCR	2	0	6(0/2/0)
EORI #, CCR	2	0	6(0/2/0)
ORI #, CCR	2	0	6(0/2/0)
BSR.B	3	−2	13(0/2/2)
BSR.W	3	−2	13(0/2/2)
BSR.L	1	−2	13(0/2/2)
CHK ⟨FEA⟩, Dn (no ex)	2	0	8(0/1/0)
CHK ⟨FEA⟩, Dn (ex)	2	−2	42(2/2/6)
CHK2 (Save)⟨FEA⟩, Dn (no ex)	1	1	3(0/1/0)
CHK2 (Op) ⟨FEA⟩, Dn (ex)	2	0	18(y/0/0)
CHK2 (Save)⟨FEA⟩, Dn (ex)	1	1	3(0/1/0)
CHK2 (Op) ⟨FEA⟩, Dn (ex)	2	−2	52(y+2/1/6)
JMP ⟨CEA⟩	0	−2	6(0/2/0)
JSR ⟨CEA⟩	3	−2	13(0/2/2)
LEA ⟨CEA⟩, An	0	0	2(0/1/0)
LINK.W An, #	2	0	10(0/2/2)
LINK.L An, #	0	0	10(0/3/2)
NOP	0	0	2(0/1/0)
PEA ⟨CEA⟩	0	0	8(0/1/2)
RTD #	1	−2	12(2/2/0)
RTR	1	−2	14(3/2/0)
RTS	1	−2	12(2/2/0)
UNLK An	1	0	9(2/1/0)

y = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

NOTE: The CHK2 instruction involves a save step which other instructions do not have. To calculate the total instruction time, calculate the save, the effective address, and the operation execution times, and combine in the order listed, using the equations given in **5.8.1.6 INSTRUCTION EXECUTION TIME CALCULATION.**

5.8.3.13 EXCEPTION-RELATED INSTRUCTIONS AND OPERATIONS. The exception-related instructions and operations table indicates the number of clock periods needed for the processor to perform the specified exception-related actions. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles
BKPT (Acknowledged)	0	0	14(1/0/0)
BKPT (Bus Error)	0	−2	35(3/2/4)
Breakpoint (Acknowledged)	0	0	10(1/0/0)
Breakpoint (Bus Error)	0	−2	42(3/2/6)
Interrupt	0	−2	30(3/2/4)*
RESET	0	0	518(0/1/0)
STOP	2	0	12(0/1/0)
LPSTOP	3	−2	25(0/3/1)
Divide-by-Zero	0	−2	36(2/2/6)
Trace	0	−2	36(2/2/6)
TRAP #	4	−2	29(2/2/4)
ILLEGAL	0	−2	25(2/2/4)
A-line	0	−2	25(2/2/4)
F-line (First Word Illegal)	0	−2	25(2/2/4)
F-line (Second Word Illegal) ea = Rn	1	−2	25(2/3/4)
F-line (Second Word Illegal) ea ≠ Rn (Save)	1	1	3(0/1/0)
F-line (Second Word Illegal) ea ≠ Rn (Op)	4	−2	29(2/2/4)
Privileged	0	−2	25(2/2/4)
TRAPcc (trap)	2	−2	38(2/2/6)
TRAPcc (no trap)	2	0	4(0/1/0)
TRAPcc.W (trap)	2	−2	38(2/2/6)
TRAPcc.W (no trap)	0	0	4(0/2/0)
TRAPcc.L (trap)	0	−2	38(2/2/6)
TRAPcc.L (no trap)	0	0	6(0/3/0)
TRAPV (trap)	2	−2	38(2/2/6)
TRAPV (no trap)	2	0	4(0/1/0)

* = Minimum interrupt acknowledge cycle time is assumed to be three clocks.

NOTE: The F-line (second word illegal) operation involves a save step which other operations do not have. To calculate, total the operation time, the save, the effective address, and the operation execution times, and combine in the order listed, using the equations given in **5.8.1.6 INSTRUCTION EXECUTION TIME CALCULATION.**

5.8.3.14 SAVE AND RESTORE OPERATIONS. The save and restore operations table indicates the number of clock periods needed for the processor to perform the specified state save or return from exception. Complete execution times and stack length are given. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles
BERR on Instruction	0	−2	<58(2/2/12)
BERR on Exception	0	−2	48(2/2/12)
RTE (Four-Word Frame)	1	−2	24(4/2/0)
RTE (Six-Word Frame)	1	−2	26(4/2/0)
RTE (BERR on Instruction)	1	−2	50(12/2/z)
RTE (BERR on Four-Word Frame)	1	−2	66(10/2/4)
RTE (BERR on Six-Word Frame)	1	−2	70(12/2/6)

< = Maximum time is indicated; certain data or mode combinations will execute faster.

z = If a bus error occurred during a write cycle, the cycle is rerun by the RTE.

SECTION 6
DMA CONTROLLER MODULE

The direct memory access (DMA) controller module provides low-latency transfer capability to an external peripheral or for memory-to-memory data transfer. The DMA module, shown in Figure 6-1, provides two channels that allow byte, word, or long-word operand transfers. These single- or dual-address transfers can be to devices that are either on-chip or off-chip. The DMA contains the following features:

- Two, Independent, Fully Programmable DMA Channels
- Single Address Transfers with 32-Bit Address and 32-Bit Data Capability
- Dual-Address Transfers with 32-Bit Address and 16-Bit Data Capability
- Two 32-Bit Transfer Counters
- Four 32-Bit Address Pointers That Can Increment or Remain Constant
- Operand Packing and Unpacking for Dual-Address Transfers
- Supports all Bus-Termination Modes
- Provides Two-Clock-Cycle Internal Module Access
- Provides Full DMA Handshake for Cycle Steal and Burst Transfers

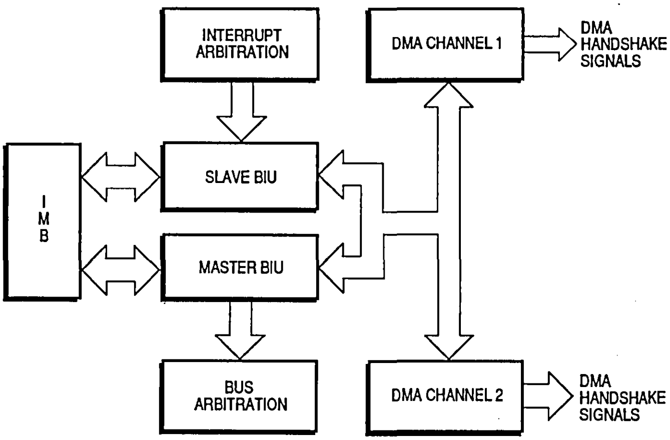


Figure 6-1. DMA Block Diagram

6.1 DMA MODULE SIGNALS

The following signals are used by the DMA module to provide handshake control for either a source or destination external device.

6.1.1 DMA Request ($\overline{\text{DREQ2}}$, $\overline{\text{DREQ1}}$)

These inputs from a peripheral start the DMA process. The assertion level can be either active-low or falling edge.

6.1.2 DMA Acknowledge ($\overline{\text{DACK2}}$, $\overline{\text{DACK1}}$)

These outputs to a peripheral are asserted during accesses, after a DMA is in progress.

6.1.3 DMA Done ($\overline{\text{DONE2}}$, $\overline{\text{DONE1}}$)

These active-low bidirectional signals indicate that the last transfer is being performed.

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

Assertion of $\overline{\text{RESET}}$ aborts any bus cycle in progress and sets all control bits back to their reset state, preventing the DMA channel from recognizing any further DMA requests. In addition, any pending requests are lost.

6.2 OPERATION

The following paragraphs describe the programmable channel functions available with the DMA module, the manner in which data transfer operations are executed, and behavior during exception conditions. This description applies to both channels.

Any channel operation adheres to the following basic sequence:

1. The processor initializes the channel registers.
2. Channel Startup — The STR bit is set in the channel control register, and the first operand transfer request is recognized (either internally or externally generated).

3. Transfer Data Block — After a channel is started, it transfers one operand in response to each request until an entire data block is transferred.

4. Channel Termination — The channel can terminate by normal completion or from an error.

6.2.1 Channel Initialization

Before starting a block transfer operation, the processor must initialize the channel registers with information describing the data block, request generation method, etc. This initialization is accomplished by moving the appropriate values into the registers using the MOVE instruction.

The source address register (SAR) is loaded with the source (read) address. If the transfer is from a peripheral device to memory, this is the location of the peripheral data register. If the transfer is from memory to a peripheral device or memory to memory, the source address is the starting address of the data block. This address may be any byte address. In the single-address write mode of operation, this register is not used.

6

The destination address register (DAR) should contain the destination (write) address. If the transfer is from a peripheral device to memory or memory to memory, the DAR is loaded with the starting address of the data block to be written. If the transfer is from memory to a peripheral device, the DAR is loaded with the address of the peripheral data register. This address may be any byte address. In the single-address read mode of operation, this register is not used.

The manner in which the SAR and DAR change after each cycle depends upon the values in the source size control (SSIZE) and destination size control (DSIZE) fields in the channel control register (CCR), the starting address in the SAR and DAR, and the source address pointer increment (SAPI) and destination address pointer increment (DAPI) bits in the CCR. If programmed to increment, the increment value is 1, 2, or 4 for byte, word, or long-word operands, respectively. If the address register is programmed to remain unchanged (no count), the register is not changed after the operand transfer. The SAR and DAR increment if a bus error terminates the transfer. Therefore, either the SAR or the DAR contains the the next address after the one that caused the bus error.

The byte transfer counter (BTC) must be loaded with the number of byte transfers that are to occur. This register is decremented by 1, 2, or 4 at the end of each transfer. The function code register (FCR) must be loaded with the source and destination function codes. Although these function codes may not be used in the address decode for the memory or peripheral, they are provided if needed.

6.2.2 Channel Startup

After initialization, the channel is started by writing a one to the start (STR) bit in the CCR. If the channel is programmed for internal request, the channel immediately requests the bus and starts transferring data. If the channel is programmed for external request, an external request ($\overline{\text{DREQx}}$) signal must be asserted before the channel requests the bus.

6.2.2.1 EXTERNAL REQUEST. To control the transfer of operands to or from memory in an orderly manner, a peripheral device uses the $\overline{\text{DREQx}}$ pins on the DMA module to request service. If a memory-to-memory transfer is desired or a peripheral with no explicit request signal is used, the DMA module can be programmed to internally generate requests. The request generation method used for the channel is programmed in the request generation field (REQ) in the CCR. If external requests are chosen, the generation of the request from the source or destination is specified by the external control option (ECO) bit of the CCR. The external requests can be for either dual- or single-address transfers. Care must be taken in setting the size of the source and destination ports, due to the control that the size settings have over the SAR, DAR, and BTC. The BTC is decremented by the larger size (SSIZE or DSIZE) contained in the CCR, and multiple cycles are run to complete the entire transfer. Therefore, multiple cycles can result from a single external request.

6.2.2.2 EXTERNAL REQUEST WITH OTHER MODULES. The DMA controller can be externally connected to the serial module and used in conjunction with the serial module to send or receive data. The DMA takes the place of a separate service routine for accessing or storing data that is sent or received by the serial module. Using the DMA also lowers the CPU32 overhead required to handle the data transferred by the serial module. Figure 6-2 shows the external connections required for using the DMA with the serial module.

For serial receive, the DMA reads data from the serial receive data register (when the serial module has filled the buffer on input) and writes data to memory. For serial transmit, the DMA reads data from memory and writes data to the serial transmit data register. Only dual-address mode can be used with the serial module.

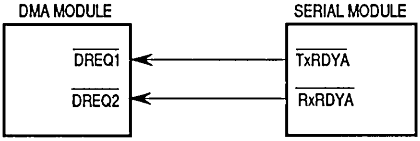

DMA MODULE
SERIAL MODULE
$\overline{\text{DREQ1}}$
$\overline{\text{DREQ2}}$
$\overline{\text{TxRDYA}}$
$\overline{\text{RxRDYA}}$

Figure 6-2. DMA External Connections to Serial Module

The timer modules can be used with the DMA in a similar manner. By connecting TOUT to $\overline{\text{DREQx}}$, the timer can request a DMA transfer.

6.2.2.3 EXTERNAL BURST. For external devices that require very high data transfer rates, the burst request mode allows the DMA channel to use all bus bandwidth under control of the external device. In burst mode, the $\overline{\text{DREQx}}$ input to the DMA is level sensitive and is sampled during the current bus cycle to determine when a valid request is asserted by the device for the next bus cycle. The device requests service by asserting $\overline{\text{DREQx}}$ and leaving it asserted. In response, the DMA arbitrates for the system bus and performs an operand transfer. During each access to the device, the DMA asserts DMA acknowledge ($\overline{\text{DACKx}}$) to indicate to the device that a request is being serviced. If $\overline{\text{DREQx}}$ is asserted when the DMA completes the requested cycle a setup time before $\overline{\text{DACKx}}$, then a valid request for another operand transfer is recognized, and the DMA services that request immediately. If $\overline{\text{DREQx}}$ is negated before $\overline{\text{DACKx}}$ is negated, a new request is not recognized, and the DMA channel releases ownership of the bus.

6.2.2.4 EXTERNAL CYCLE STEAL. For external devices that generate a pulsed signal for each operand to be transferred, the cycle steal request mode uses $\overline{\text{DREQx}}$ as a falling-edge-sensitive input. The DMA channel responds to cycle steal requests the same as all other requests; however, if subsequent $\overline{\text{DREQx}}$ pulses are generated before $\overline{\text{DACKx}}$ is asserted in response to each request, they are ignored. If $\overline{\text{DREQx}}$ is asserted after the DMA channel asserts $\overline{\text{DACKx}}$ for the previous request but before $\overline{\text{DACKx}}$ is negated, then the new request is serviced before bus ownership is released. If a new request is not generated by the time $\overline{\text{DACKx}}$ is negated, the bus is released.

6.2.3 DMA Transfer Operation

The two-channel DMA module supports dual- and single-address transfers. The dual-address operand transfer consists of a source operand read and a destination operand write. Each single-address operand transfer consists of one external bus cycle, which allows either a read or a write cycle to occur.

6.2.3.1 DUAL-ADDRESS MODE. The two DMA channels can each be programmed to operate in a dual-address transfer mode. In this mode, the operand is read from the source address specified in the SAR and placed in the data holding register (DHR). The operand read may take up to four bus cycles to complete because of differences in operand sizes of the source and destination. The operand is then written to the address specified in the DAR. This transfer may also be up to four bus cycles long. In this manner, various combinations of peripheral, memory, and operand sizes may be used.

The dual-address transfers can be started by either the internal request mode or by an external device using the $\overline{\text{DREQx}}$ input signal. When the external device uses $\overline{\text{DREQx}}$, the channel can be programmed to operate in either the cycle steal or burst transfer modes. See **6.2.2.3 EXTERNAL BURST** and **6.2.2.4. EXTERNAL CYCLE STEAL** for information about these modes.

6.2.3.1.1 Dual-Address Source Read. During this type of DMA cycle, the SAR drives the address bus, the FCR drives the source function codes, and the CCR drives the size control. Data is read from the memory or peripheral and placed in the DHR when the bus cycle is terminated. When the complete operand has been read, the SAR is incremented by 1, 2, or 4, depending on the address and size information specified by the SAPI and SSIZE bits of the CCR. See **6.4.6 Source Address Registers** for more information.

6.2.3.1.2 Dual-Address Destination Write. During this type of DMA cycle, the data in the DHR is written to the device or memory selected by the address in the DAR, the destination function codes in the FCR, and the size in the CCR. The same options exist for operand size and alignment as in the dual-address source read. When the complete operand is written, the DAR is incremented by 1, 2, or 4, according to the DAPI and DSIZE bits of the CCR, and the BTC is decremented by the number of bytes transferred. If the BTC is equal to zero and there were no errors, the DONE bit in the CSR and the $\overline{\text{DONEx}}$ signal for the DMA handshake are asserted. See **6.4.7 Destination Address Registers (DARs)** and **6.4.8 Byte Transfer Counters (BTCs)** for more information.

6.2.3.2 SINGLE-ADDRESS MODE. Each DMA channel can be independently programmed to provide single-address transfers. Only external request can be used to start a transfer when the single-address mode is selected. The ECO bit in the CCR controls whether a source read or a destination write cycle occurs on the data bus. If the ECO bit is set, the external handshake signals are used with the source operand and a single address source read occurs. If the ECO bit is cleared, the external handshake signals are used with the destination operand, and a single address destination write takes place.

If external 32-bit devices and a 32-bit bus are used with the MC68340, the DMA can control 32-bit transfers between devices that use the 32-bit bus, in single-address mode only. If both byte and word devices are used on an external bus, then an external multiplexer must be used to correctly transfer data. The SIZx and A0 signals can be used to control this external multiplexer.

6.2.3.2.1 Single-Address Source Read. During the single-address source read cycle, the device or memory selected by the address specified in the SAR, the source function codes in the FCR, and the size in the CCR provide the data and control signals on the data bus. This bus cycle operates like a normal read bus cycle. The destination device is controlled by the DMA handshake signals ($\overline{\mathrm{DREQx}}$, $\overline{\mathrm{DACKx}}$, and $\overline{\mathrm{DONEx}}$). The assertion of $\overline{\mathrm{DACKx}}$ provides the write control to the destination device. For more details about the DMA handshake signals, see **6.1 DMA MODULE SIGNALS.**

6.2.3.2.2 Single-Address Destination Write. During the single-address destination write cycle, the source device is controlled by the DMA handshake signals ($\overline{\mathrm{DREQx}}$, $\overline{\mathrm{DACKx}}$, and $\overline{\mathrm{DONE}}$). When the source device requests service from the DMA channel, the assertion of $\overline{\mathrm{DACKx}}$ by the DMA channel allows the source device to drive data onto the data bus. The data is written to the device or to memory selected by the address specified in the DAR, the destination function codes in the FCR, and the size in the CCR. The data bus is placed in a high-impedance state for this write cycle. For more details about the DMA handshake signals, see **6.1 DMA MODULE SIGNALS.**

6.2.4 Interrupt Operation

The processor can determine the status of a DMA operation by reading the channel status register (CSR), or the DMA channel can interrupt the processor to inform it of certain events.

Interrupts can be generated by error termination of a bus cycle or by normal channel completion. Specifically, if the interrupt enable for error bit INTE in the CCR is set and BES, BED, or CONF is set, the IRQ bit is set. In this case, clearing the INTE, BES, BED, or CONF bits causes the IRQ bit to be cleared. If the INTN bit in the CCR is set and the DONE bit is set, the IRQ bit is set. In this case, clearing the INTN or the DONE bit causes the IRQ bit to be cleared. If the INTB bit in the CCR is set and the BRKP bit in the CSR is set, the IRQ bit is set. Clearing INTB or BRKP clears IRQ.

6.2.5 Bus Arbitration

The DMA controller uses the M68000 bus arbitration protocol to request bus mastership for DMA transfers. Each channel arbitrates for the bus independently, and priority is fixed by hardware in the bus interface unit.

The DMA module transfers are unique in one respect; FC3 can be asserted during the source operand bus cycle and remain asserted until the end of the destination operand bus cycle. The source DMA bus cycle has timing identical to a read bus cycle. The destination DMA bus cycle has timing identical to a write bus cycle.

To guarantee that the DMA does not use all available system bus bandwidth during a transfer, internal requests can specify the amount of bus bandwidth allocated to the DMA. This allocation is done by programming the REQ bits in the CCR to internal programmable rate and the BB bits in the CCR to the percentage of bandwidth desired. When the STR bit in the CCR is set, the DMA channel arbitrates for the bus and begins to transfer data when it becomes bus master. If no exception occurs, the DMA uses the percentage of bus bandwidth, which is programmed into the control register.

6.2.6 Fast-Termination Option

The fast-termination option, using the system integration module (SIM) chip-select logic, can be employed to give a fast bus access of two clock cycles rather than the standard three-cycle access time. The fast-termination option is described in **SECTION 4 SYSTEM INTEGRATION MODULE** and in **SECTION 3 BUS OPERATION.**

If the fast-termination option is used with external request burst mode, an extra DMA cycle results on every burst transfer. Normally, $\overline{\text{DREQx}}$ is negated when $\overline{\text{DACKx}}$ is returned. In the burst mode with fast termination selected, a new cycle starts even if $\overline{\text{DREQx}}$ is negated simultaneously with $\overline{\text{DACKx}}$ assertion.

6.3 PROGRAMMING SEQUENCE

To begin a data transfer operation, move the appropriate values into the registers using the MOVE instruction and the following sequence:

a. Write a zero to the STR bit in CCR.

b. Load the SAR, DAR, and BTC with the source and destination addresses and the byte count.

c. Write the CCR with parameters describing the desired operation of the channel: operand size, count mode, request method, etc. If the STR bit is set at this time, the channel is started.

d. Write to the peripheral device control registers as necessary to enable the device to generate requests and begin operation.

6.3.1 Channel Startup

Once the channel has been initialized, it is started by writing a one to the STR bit in the CCR. If the channel is programmed for internal request, this causes the channel to request the bus and start transferring data. If the channel is programmed for external request, $\overline{\text{DREQx}}$ must be asserted before the channel requests the bus. The $\overline{\text{DREQx}}$ input is ignored until the channel is started since the channel does not recognize transfer requests until it is active.

If any fields in the CCR are modified while the channel is active, that change is effective immediately. To avoid any problems with changing the setup for the DMA channel, a zero should be written to the STR bit in the CCR to halt the DMA channel at the end of the current bus cycle.

6.3.2 Data Transfers

Each operand transfer requires from one to five bus cycles to complete. Once a bus request is recognized and the operand transfer begins, both the source read and/or destination write occur before a new bus request may be honored, even if the new bus request is of higher priority.

6.3.2.1 DUAL-ADDRESS TRANSFERS. Each operand transfer in the dual-address mode requires from two to five bus cycles in response to each operand transfer request. If the source and destination operands are the same size, two cycles will transfer the complete operand. If the source and destination operands are different sizes, the number of cycles will vary. If the source is a long-word and the destination is a byte, there would be one bus cycle for the read and four bus cycles for the write. Once the DMA channel has started a dual-address operand transfer, it must complete that transfer before releasing ownership of the bus or servicing a request for another channel of equal or higher priority, unless one of the bus cycles is terminated with a bus error during the transfer.

6.3.2.2 SINGLE-ADDRESS TRANSFERS. When a request is recognized internally, the DMA channel arbitrates for the bus and executes one bus cycle in response to each request. Since the operand size must be equal to the device port size for single-address transfers and only one bus cycle is run for each request, the number of normally terminated bus cycles executed during a transfer operation is always equal to the value programmed into the corresponding size field of the CCR. The sequencing of the address bus follows the programming of the CCR and address register (SAR or DAR) for the channel.

6.3.2.3 CHANNEL TERMINATION. The channel operation can be terminated for several reasons: the BTC is decremented to zero, a peripheral device asserts $\overline{\text{DONEx}}$ during an operand transfer, the STR bit is cleared in the CCR, a bus cycle is terminated with a bus error, or a reset occurs.

6.4 REGISTER DESCRIPTION

Figure 6-3 is a programmer's model (register map) of all registers in the DMA. Each channel has an independent set of registers that are located at the base address added to the address specified in the columns for channel 1 and channel 2. The base address is specified in **SECTION 4 SYSTEM INTEGRATION MODULE**. The column titled FC (function code) indicates whether a register is restricted to supervisor access (S) or is programmable to exist in either supervisor or user space (S/U).

Unimplemented memory locations return logic zero when accessed. All registers support both byte and word transfers.

ADDRESS		FC		
CH1	CH2		15 8	7 0
780	7A0	S	MODULE CONFIGURATION REGISTER (MCR)	
782	7A2	S	RESERVED	
784	7A4	S	INTERRUPT REGISTER	
786	7A6	S/U	RESERVED	
788	7A8	S/U	CHANNEL CONTROL REGISTER	
78A	7AA	S/U	CHANNEL STATUS REGISTER	FUNCTION CODE REGISTER
78C	7AC	S/U	SOURCE ADDRESS REGISTER MSBs	
78E	7AE	S/U	SOURCE ADDRESS REGISTER LSBs	
790	7B0	S/U	DESTINATION ADDRESS REGISTER MSBs	
792	7B2	S/U	DESTINATION ADDRESS REGISTER LSBs	
794	7B4	S/U	BYTE TRANSFER COUNTER MSBs	
796	7B6	S/U	BYTE TRANSFER COUNTER LSBs	
798	7B8	S/U	RESERVED	
79A	7BA	S/U	RESERVED	
79C	7BC	S/U	RESERVED	
79E	7BE	S/U	RESERVED	

Figure 6-3. Programmer's Model

6.4.1 Module Configuration Registers (MCRs)

The two 16-bit MCRs are always readable and writable in the supervisor mode, although writing is discouraged unless the module is disabled.

MCR1, MCR2 $780, $7A0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STP	FRZ1	FRZ0	SE	0	ISM			SUPV	MAID			IARB			
RESET:															
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Supervisor/User

STP — Stop Bit

1 = The system clock stops within the module except for the intermodule bus (IMB) bus interface unit (BIU). The BIU continues to operate to allow the CPU32 access to the module's MCR. All other register accesses cause a bus error without changing the data. The system clock stops on the low phase of the clock and remains stopped until STP is cleared by the CPU32 or until reset.

0 = The system clock operates normally within the module.

NOTE

The DMA module uses only one set of FRZ bit for both channels. A read or write to either MCR accesses the same control bit.

FRZ1,FRZ0 — Freeze

These bits determine the action to be taken when the FREEZE signal is asserted. The DMA module negates $\overline{BR}$ and keeps it negated until FREEZE is negated or reset. Table 6-1 shows the definition for these two bits.

Table 6-1. FRZ Encoding

FRZ1	FRZ0	Action
0	0	Ignore FREEZE
0	1	Reserved
1	0	Freeze on Boundary
1	1	Reserved

NOTE: The boundary is defined as any bus cycle by the DMA module.

NOTE

The DMA module uses only one set of FRZ bits for both channels. A read or write to either MCR accesses the same control bit.

SE — Single-Address Enable

1 = Selects single-address mode of operation for the channel; external data bus is driven during DMA transfer.

0 = External data bus remains in a high-impedance state during the DMA transfer.

ISM2–ISM0 — Interrupt Service Mask

These bits contain the interrupt service mask. When the interrupt service level on the IMB is greater than the interrupt service mask, the DMA vacates the bus and negates $\overline{BR}$ until the interrupt service level is less than or equal to the interrupt service mask.

SUPV — Supervisor/User

This bit affects all registers that are defined as supervisor/user and determines whether the registers reside in supervisor data space or can be accessed from both supervisor and user programs.

1 = Supervisor-only access; FC2 must be logic one during access.

0 = Unrestricted access; FC2 is ignored during access.

MAID — Master Arbitration ID

These bits establish bus arbitration priority level among modules that have the capability of becoming bus master. In the MC68340, only the SIM and the DMA can be bus masters. Zero is the lowest priority and seven is the highest priority.

IARB — Interrupt Arbitration ID

This field is used to arbitrate for the IMB in the event two or more modules simultaneously generate an interrupt of the same priority level . This field is set to $0 after reset, which prevents this module from arbitrating during the interrupt acknowledge cycle. If no arbitration occurs during the interrupt acknowledge cycle, the spurious interrupt vector is generated and the interrupt is discarded. Initialization software must set this field to a nonzero value.

NOTE

The DMA module uses only one set of IARB bits for both channels. A read or write to either MCR accesses the same control bits.

6.4.2 Interrupt Registers (INTRs)

The INTRs are accessible only in supervisor space.

INTR1, INTR2 $784, $7A4

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	INTL			INTV							
RESET:																
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Supervisor Only

INTL — Interrupt Level

The interrupt level field contains the priority level of the interrupt for its associated channel.

INTV — Interrupt Vector

This field is used by the DMA channel to arbitrate for the bus during interrupt acknowledge bus cycles. During an interrupt acknowledge cycle, the module with the highest priority pending interrupt places the content of this field on the data bus.

6.4.3 Channel Control Registers (CCRs)

The CCRs are accessible in either supervisor or user space.

CCR1, CCR2 $788, $7A8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTB	INTN	INTE	ECO	SAPI	DAPI	SSIZE		DSIZE		REQ		BB		S/D	STR
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	0

U = Unaffected by reset Supervisor/User

INTB — Interrupt Breakpoint

Setting the breakpoint bit sets the BRKP bit in the CSR. The logic AND of INTB and BRKP generates an interrupt request.

1 = Enables $\overline{\text{IRQx}}$ when breakpoint is recognized and this channel is bus master

0 = Not enabled

INTN — Interrupt Normal

1 = Enables $\overline{\text{IRQx}}$ when this channel finishes a transfer without an error condition (DONE)

0 = Not enabled

INTE — Interrupt Error

1 = Enables $\overline{\text{IRQx}}$ when this channel encounters an error on source read (BES), destination write (BED), or configuration for channel setup (CONF)

0 = Not enabled

ECO — External Control Option

Dual-Address Mode — this bit defines which device generates requests.

1 = If request generation is programmed to be external (REQ = 1X), the source device generates the request, and the control signals ($\overline{\text{DREQx}}$, $\overline{\text{DACKx}}$, and $\overline{\text{DONEx}}$) are part of the source (read) portion of the transfer.

0 = If REQ is programmed to be external, the destination device generates the request, and the control signals ($\overline{\text{DREQx}}$, $\overline{\text{DACKx}}$, and $\overline{\text{DONEx}}$) are part of the destination (write) portion of the transfer.

Single-Address Mode — this bit defines the direction of transfer.

1 = If REQ is programmed to be external, the requesting device receives the data (read from memory), and the control signals ($\overline{\text{DREQx}}$, $\overline{\text{DACKx}}$, and $\overline{\text{DONEx}}$) are used by the requesting device to write data during the source (read) portion of the transfer.

0 = If REQ is programmed to be external, the requesting device provides the data (write to memory), and the control signals ($\overline{DREQx}$, $\overline{DACKx}$, and $\overline{DONEx}$) are used by the requesting device to provide data during the destination (write) portion of the transfer.

SAPI — Source Address Pointer Increment

1 = The SAR is incremented by 1, 2, or 4 after each transfer, according to the source size.

0 = The SAR is not incremented during operand transfer. The address that is written into the SAR under program control is used for the complete data transfer.

DAPI — Destination Address Pointer Increment

1 = The DAR is incremented by 1, 2, or 4 after each transfer, according to the source size.

0 = The DAR is not incremented during operand transfer. The address that is written into the DAR under program control is used for the complete data transfer.

SSIZE — Source Size Control

These bits control the size of the source read bus cycle that the DMA channel is running. Table 6-2 defines these bits.

Table 6-2. SSIZE Encoding

Bit 9	Bit 8	Definition
0	0	Long Word
0	1	Byte
1	0	Word
1	1	Not Used

DSIZE — Destination Size Control

These bits control the size of the destination write bus cycle that the DMA channel is running. Table 6-3 defines these bits.

Table 6-3. DSIZE Encoding

Bit 7	Bit 6	Definition
0	0	Long Word
0	1	Byte
1	0	Word
1	1	Not Used

REQ — Request Generation

These bits control the mode of operation the DMA channel uses to make an operand transfer request. Table 6-4 defines these bits.

Table 6-4. REQ Encoding

Bit 5	Bit 4	Definition
0	0	Internal Request at Programmable Rate
0	1	Reserved
1	0	External Request Burst Transfer Mode
1	1	External Request Cycle Steal

BB — Bus Bandwidth

These bits control the percentage of the IMB that the DMA channel can use during internal requests only. Table 6-5 defines these bits.

Table 6-5. BB Encoding

Bit 5	Bit 4	Definition
0	0	25%
0	1	50%
1	0	75%
1	1	100%

S/D — Single-/Dual-Address Transfer

1 = The DMA channel runs single-address transfers from peripherals or from memory to peripherals or memory. The destination holding register is not used for these transfers because the data is transferred directly into the destination location.

0 = The DMA channel runs standard dual-address transfers.

STR — Start

Internal Request

1 = Start DMA transfer

0 = No action

External Request

1 = This bit must be set for the DMA bus controller to recognize the first request for DMA by the external device using the $\overline{\text{DREQx}}$ input. This bit is set by writing a logic zero to bit 0 of the CCR.

0 = No action

This bit is cleared by reset, writing a logic zero, the DONE status bit being set, or one of the error status bits (BES, BED, or CONF) being set.

6.4.4 Channel Status Registers (CSRs)

The CSRs are accessible in either supervisor or user space.

CSR1, CSR2 $78A, $7AA

7	6	5	4	3	2	1	0
IRQ	DONE	BES	BED	CONF	BRKP	0	0
RESET:							
0	0	0	0	0	0	0	0

Supervisor/User

IRQ — Interrupt Request

1 = An interrupt condition has occurred.

0 = An interrupt condition has not occurred.

This bit is the logical OR of the DONE, BES, BED, CONF, and BRKP bits and is cleared when they are all cleared. IRQ is positioned to allow conditional testing as a signed binary integer. The state of this bit is not affected by the interrupt enable bits in the CCRs. The STR bit in the CCR cannot be set when this bit is set; all error status bits must be cleared before the STR bit can be set.

DONE — DMA Done

1 = The DMA channel has terminated normally.

0 = The DMA channel has not terminated normally.

This bit is cleared by writing a logic one or by reset. Writing a zero has no effect.

BES — Bus Error on Source

1 = The DMA channel has terminated with a bus error during the read bus cycle.

0 = The DMA channel has not terminated with a bus error during the read bus cycle.

This bit is cleared by writing a logic one or by reset. Writing a zero has no effect.

BED — Bus Error on Destination

1 = The DMA channel has terminated with a bus error during the write bus cycle.

0 = The DMA channel has not terminated with a bus error during the write bus cycle.

This bit is cleared by writing a logic one or by reset. Writing a zero has no effect.

CONF — Configuration Error

A configuration error results when either the SAR or the DAR contains an address that does not match the port size specified in the CCR and the BTC register does not match the larger port size or is zero.

1 = The STR bit is set in the CCR, and a configuration error is present.
0 = If STR is set, no configuration error exists.

This bit is cleared by writing a logic one or by reset. Writing a zero has no effect.

BRKP - Breakpoint

1 = The breakpoint signal was set during a DMA transfer.
0 = The breakpoint signal was not set during a DMA transfer.

This bit is cleared by writing a logic one or by reset. Writing a zero has no effect.

6.4.5 Function Code Registers (FCRs)

The FCRs are accessible in either supervisor or user space.

FCR1, FCR2 $78B, $7AB

7	6	5	4	3	2	1	0
SFC				DFC			
RESET:							
U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor/User

SFC — Source Function Code

The source function code bits are defined in Table 6-6.

DFC — Destination Function Code

The destination function code bits are defined in Table 6-6.

NOTE

FC3 can be set for DMA transfers to distinguish the source or destination space from other data or program space, but is not required to be set. Since the CPU32 currently has only 3-bit SFC and DFC capability, it cannot emulate FC3 = 1 at this time.

Table 6-6. Address Space Encoding

Function Code Bits				Address Spaces
3	2	1	0	
0	0	0	0	Reserved (Motorola)
0	0	0	1	User Data Space
0	0	1	0	User Program Space
0	0	1	1	Reserved (User)
0	1	0	0	Reserved (Motorola)
0	1	0	1	Supervisor Data Space
0	1	1	0	Supervisor Program Space
0	1	1	1	CPU Space
1	x	x	x	DMA Space

6.4.6 Source Address Registers (SARs)

The SARs are accessible in either supervisor or user space.

SAR1, SAR2 $78C, $7AC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor/User

This 32-bit register contains the address of the source operand used by the DMA to access memory or peripheral controller registers. During the DMA read cycle, this register drives the address on the address bus. This register can be programmed to increment or remain constant after each operand transfer.

The register is incremented using unsigned arithmetic and will roll over if overflow occurs. For example, if the register contains $FFFFFFFF and is incremented by 1, it will roll over to $00000000. This register is incremented by 1, 2, or 4, depending on the size of the operand and the memory starting address. If the operand size is byte, then the register is always incremented by 1. If the operand size is word and the starting address is even-word aligned, then the register is incremented by 2; if the operand size is word and the address is odd-byte aligned, then the CONF bit is set in the CSR and no transfer occurs.

If the operand size is long word and the address is long-word aligned, then the register is incremented by 4; if the operand size is long word and the address is odd-word or odd-byte aligned, then the CONF bit is set, and no transfer occurs.

When read, this register always contains the next source address. If a bus error terminates the transfer, this register contains the next source address that would have been run had the error not occurred.

6.4.7 Destination Address Registers (DARs)

The DARs are accessible in either supervisor or user space.

DAR1, DAR2 $790, $7B0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor/User

This 32-bit register contains the address of the destination operand used by the DMA to write to memory or peripheral controller registers. During the DMA write cycle, this register drives the address on the address bus. This register can be programmed to increment or remain constant after each operand transfer.

The register is incremented using unsigned arithmetic and will roll over if overflow occurs. For example, if a register contains $FFFFFFFF and is incremented by 1, it will roll over to $00000000. This register can be incremented by 1, 2, or 4, depending on the size of the operand and the starting address. If the operand size is byte, the register is always incremented by 1. If the operand size is word and the starting address is even-word aligned, the register is incremented by 2; if the operand size is word and the address is odd-byte aligned, the CONF bit is set in the CSR, and no transfer occurs. If the operand size is long word and the address is long-word aligned, the register is incremented by 4; if the operand size is long word and the address is odd-word or odd-byte aligned, then the CONF bit is set, and no transfer occurs.

When read, this register always contains the next destination address. If a bus error terminates the transfer, this register contains the next destination address that would have been run had the error not occurred.

6.4.8 Byte Transfer Counter Registers (BTCs)

The BTCs are accessible in either supervisor or user space.

BTC1, BTC2 $794, $7B4

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
RESET:	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
RESET:	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor/User

This 32-bit register contains the number of bytes left to transfer in a given block. This register is decremented by 1, 2, or 4 for each successful operand transfer from source to destination locations. When the BTC decrements to zero and no error has occurred, the DONE bit is set in the CSR. In the external request mode, the $\overline{\text{DONE}}$ handshake line is also asserted when the BTC is decremented to zero.

If the operand size is byte then the register is always decremented by 1. If the operand size is word and the starting count is even word, the register is decremented by 2. If the operand size is word and the count is odd byte, then the CONF bit is set in the CSR, and no transfer occurs. If the operand size is long word and the count is long word, then the register is incremented by 4; if the operand size is long word and the count is odd word or odd byte, the CONF bit is set, and no transfer occurs. If the STR bit is set with a zero count in the BTC, the CONF bit is set, and the STR bit is cleared.

When read, this register always contains the count for the next access. If a bus error terminates the transfer, this register contains the count for the next access that would have been run had the error not occurred.

6.4.9 Data Holding Register (DHR)

This 32-bit register serves as a buffer register for the data being transferred during dual-address DMA cycles. No address is specified since this register can not be addressed by the programmer. The DHR allows the data to be packed and unpacked by the DMA during the transfer. For example, if the source operand size is byte and the destination operand size is word, then two-byte read cycles occur, followed by a one-word write cycle (see Figure 6-4). The two bytes of data are buffered in the DHR until the word write cycle occurs. The DHR allows for packing and unpacking of operands for the following sizes: bytes to words, bytes to long words, words to long words, words to bytes, long words to bytes, and long words to words.

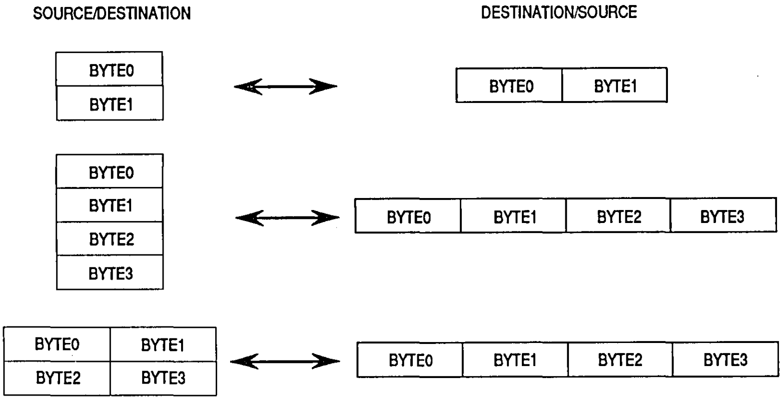


Figure 6-4. Packing and Unpacking of Operands

For normal transfers aligned with the size and address, only two bus cycles are required for each transfer: a read from the source and a write to the destination.

SECTION 7 SERIAL MODULE

The MC68340 serial module is a dual universal asynchronous/synchronous receiver/transmitter that interfaces directly to the CPU32 processor via an intermodule bus (IMB). The serial module, shown in Figure 7-1, consists of the following major functional areas:

- Two Independent Serial Communication Channels (A and B)
- Baud Rate Generator Logic
- Internal Channel Control Logic
- Interrupt Control Logic

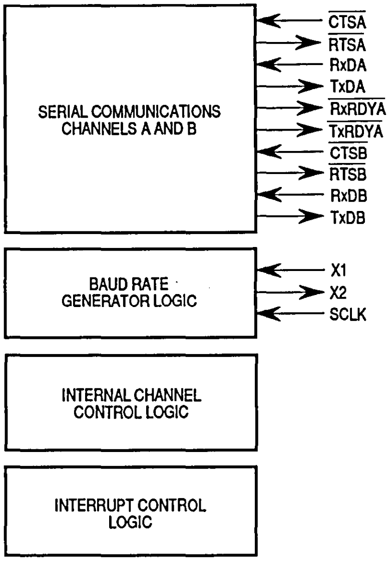

SERIAL COMMUNICATIONS CHANNELS A AND B
CTSA
RTSA
RxDA
TxDA
RxRDYA
TxRDYA
CTSB
RTSB
RxDB
TxDB
BAUD RATE GENERATOR LOGIC
X1
X2
SCLK
INTERNAL CHANNEL CONTROL LOGIC
INTERRUPT CONTROL LOGIC

Figure 7-1. Simplified Block Diagram

7.1 MODULE OVERVIEW

Features of the serial module are as follows:

- Two, Independent, Full-Duplex Asynchronous/Synchronous Receiver/Transmitter Channels
- Maximum Data Transfer:
  - 1× – 3 Mbps
  - 16× – 188 kbps
- Quadruple-Buffered Receiver
- Double-Buffered Transmitter
- Independently Programmable Baud Rate for Each Receiver and Transmitter Selectable from:
  - 19 Fixed Rates: 50 to 76.8k Baud
  - External 1× Clock or 16× Clock
- Programmable Data Format:
  - Five to Eight Data Bits Plus Parity
  - Odd, Even, No Parity, or Force Parity
  - One, One and One-Half, or Two Stop Bits Programmable in One-Sixteenth Bit Increments
- Programmable Channel Modes:
  - Normal (Full Duplex)
  - Automatic Echo
  - Local Loopback
  - Remote Loopback
- Automatic Wakeup Mode for Multidrop Applications
- Seven Maskable Interrupt Conditions
- Parity, Framing, and Overrun Error Detection
- False-Start Bit Detection
- Line-Break Detection and Generation
- Detection of Breaks Originating in the Middle of a Character
- Start/End Break Interrupt/Status
- On-Chip Crystal Oscillator
- TTL Compatibility

7.1.1 Serial Communication Channels A and B

Each communication channel provides a full-duplex asynchronous/synchronous receiver and transmitter using an operating frequency independently selected from a baud rate generator or an external clock input. The transmitter accepts parallel data from the IMB, converts it to a serial bit stream, inserts the appropriate start, stop, and optional parity bits, then outputs a composite serial data stream on the channel transmitter serial data output (TxDx). Refer to **7.3.2.1 TRANSMITTER** for additional information.

The receiver accepts serial data on the channel receiver serial data input (RxDx), converts it to parallel format, checks for a start bit, stop bit, parity (if any), or break condition, and transfers the assembled character onto the IMB during read operations. Refer to **7.3.2.2 RECEIVER** for additional information.

7.1.2 Baud Rate Generator Logic

The crystal oscillator operates directly from a 3.6864-MHz crystal connected across the X1 input and the X2 output or from an external clock of the same frequency connected to X1. The clock serves as the basic timing reference for the baud rate generator and other internal circuits.

The baud rate generator operates from the oscillator or external TTL clock input and is capable of generating 19 commonly used data communication baud rates ranging from 50 to 76.8k by producing internal clock outputs at 16 times the actual baud rate. Refer to **7.2 INTERFACE SIGNAL DESCRIPTIONS** and **7.3.1 Baud Rate Generator** for additional information.

The external clock input (SCLK), which bypasses the baud rate generator, is a synchronous mode of operation when used with a divide-by-1 clock and an asynchronous mode when used with divide-by-16 clock. The external clock input allows the user to use SCLK as the only clock source for the serial module if baud rates are not required.

7.1.3 Internal Channel Control Logic

The serial module receives operation commands from the host and, in turn, issues appropriate operation signals to the internal serial module control logic. This mechanism allows the registers within the module to be accessed and various commands to be performed. Refer to **7.4 REGISTER DESCRIPTION AND PROGRAMMING** for additional information.

7.1.4 Interrupt Control Logic

Seven interrupt request ($\overline{\text{IRQ7}}$–$\overline{\text{IRQ1}}$) outputs are provided to notify the CPU32 that an interrupt has occurred. These interrupts are described in **7.4 REGISTER DESCRIPTION AND PROGRAMMING**. The interrupt status register (ISR) is read by the CPU32 to determine all currently active interrupt conditions. The interrupt enable register (IER) is programmable to mask any events that can cause an interrupt.

7.1.5 Comparison of Serial Module to MC68681

The serial module is code compatible with the MC68681 with some modifications. The following paragraphs describe the differences.

The programming model is slightly altered. The supervisor/user block in the MC68340 closely follows the MC68681. The supervisor-only block has the following changes:

- The interrupt vector register is moved from supervisor/user to supervisor only at a new address.
- MR2A and MR2B are moved from a hidden address location to a location at the bottom of the programming model.
- The timer/counter is eliminated as well as all associated command and status registers.
- Only certain output port pins are available.
- The XTAL_RDY bit in the ISR should be polled until it is cleared to prevent an unstable frequency from being applied to the baud rate generator. The following code is an example:

```
if (XTAL_RDY= =0)
    begin
      write CSR
    end
  else
    begin
      wait
      jump loop
    end
```

7.2 INTERFACE SIGNAL DESCRIPTIONS

The following paragraphs contain a brief description of the serial module signals. Figure 7-2 shows both the external and internal signal groups.

NOTE

The terms assertion and negation are used throughout this section to avoid confusion when dealing with a mixture of active-low and active-high signals. The term assert or assertion indicates that a signal is active or true independent of the level represented by a high or low voltage. The term negate or negation indicates that a signal is inactive or false.

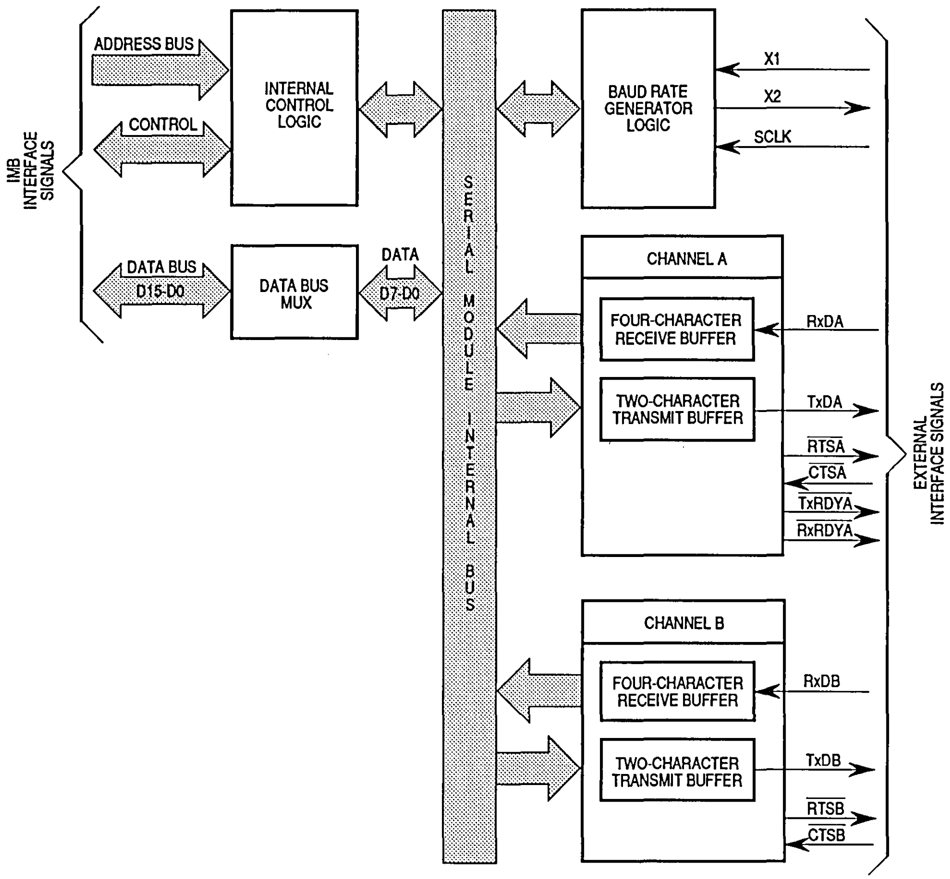

IMB INTERFACE SIGNALS
ADDRESS BUS
CONTROL
DATA BUS
D15-D0
INTERNAL CONTROL LOGIC
DATA BUS MUX
DATA
D7-D0
SERIAL MODULE INTERNAL BUS
BAUD RATE GENERATOR LOGIC
X1
X2
SCLK
CHANNEL A
FOUR-CHARACTER RECEIVE BUFFER
TWO-CHARACTER TRANSMIT BUFFER
RxDA
TxDA
RTSA
CTSA
TxRDYA
RxRDYA
CHANNEL B
FOUR-CHARACTER RECEIVE BUFFER
TWO-CHARACTER TRANSMIT BUFFER
RxDB
TxDB
RTSB
CTSB
EXTERNAL INTERFACE SIGNALS

Figure 7-2. External and Internal Interface Signals

7.2.1 Crystal Input or External Clock (X1)

This input is one of two connections to a crystal or a single connection to an external clock. A crystal or an external clock signal, at 3.6864 MHz, must be supplied when using the baud rate generator. If a crystal is used, a capacitor of approximately 10 pF should be connected from this signal to ground. If this input is not used, it must be connected to V_{CC} or GND. Refer to **SECTION 10 APPLICATIONS** for an example of a clock driver circuit.

7.2.2 Crystal Output (X2)

This output is the additional connection to a crystal. If a crystal is used, a capacitor of approximately 5 pF should be connected from this signal to ground. If an external TTL-level clock is used on X1 or SCLK, the baud rate generator is bypassed and the X2 output must be left open. Refer to **SECTION 10 APPLICATIONS** for an example of a clock driver circuit.

7.2.3 External Input (SCLK)

This input can be used as the clock input for channel A and/or channel B and is programmable in the clock-select registers (CSR). When used as the receiver clock, received data is sampled on the rising edge of the clock. When used as the transmitter clock, data is output on the falling edge of the clock. If this input is not used, it must be connected to V_{CC} or GND.

7.2.4 Channel A Transmitter Serial Data Output (TxDA)

This signal is the transmitter serial data output for channel A. The output is held high ('mark' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on this signal on the falling edge of the programmed clock source, with the least significant bit transmitted first.

7.2.5 Channel A Receiver Serial Data Input (RxDA)

This signal is the receiver serial data input for channel A. Data received on this signal is sampled on the rising edge of the programmed clock source, with the least significant bit received first.

7.2.6 Channel B Transmitter Serial Data Output (TxDB)

This signal is the transmitter serial data output for channel B. The output is held high ('mark' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on this signal at the falling edge of the programmed clock source, with the least significant bit transmitted first.

7.2.7 Channel B Receiver Serial Data Input (RxDB)

This signal is the receiver serial data input for channel B. Data on this signal is sampled on the rising edge of the programmed clock source, with the least significant bit received first.

7.2.8 Channel A Request To Send ($\overline{\mathrm{RTSA}}$)

This active-low output signal is programmable as the channel A request to send or as a dedicated parallel output.

7.2.8.1 $\overline{\mathrm{RTSA}}$. When used for this function, this signal can be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the clear-to-send (CTS) input of a receiver, this signal can be used to control serial data flow.

7.2.8.2 OP0. When used for this function, this output reflects the complement of the value of bit 0 in the output port data register (OP).

7.2.9 Channel B Request To Send ($\overline{\mathrm{RTSB}}$)

This active-low output signal is programmable as the channel B request to send or as a dedicated parallel output.

7.2.9.1 $\overline{\mathrm{RTSB}}$. When used for this function, this signal can be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the clear-to-send (CTS) input of a port, this signal can be used to control serial data flow.

7.2.9.2 OP1. When used for this function, this output reflects the complement of the value of bit 1 in the OP.

7.2.10 Channel A Clear To Send ($\overline{\text{CTSA}}$)

This active-low input is the channel A clear to send.

7.2.11 Channel B Clear To Send ($\overline{\text{CTSB}}$)

This active-low input is the channel B clear to send.

7.2.12 Channel A Transmitter Ready ($\overline{\text{TxRDYA}}$)

This active-low output signal is programmable as the channel A transmitter ready or as a dedicated parallel output.

7.2.12.1 $\overline{\textbf{TxRDYA}}$. When used for this function, this signal reflects the status of bit 2 of the channel A status register (SRA). This signal can be used to control parallel data flow by acting as an interrupt to indicate when the transmitter contains a character.

7.2.12.2 OP6. When used for this function, this output reflects the complement of the value of bit 6 in the OP.

7.2.13 Channel A Receiver Ready ($\overline{\text{RxRDYA}}$)

This active-low output signal is programmable as the channel A receiver ready, channel A FIFO full indicator, or a dedicated parallel output.

7.2.13.1 $\overline{\textbf{RxRDYA}}$. When used for this function, this signal reflects the status of ISR bit 1. This signal can be used to control parallel data flow by acting as an interrupt to indicate when the receiver contains a character.

7.2.13.2 $\overline{\textbf{FFULLA}}$. When used for this function, this signal reflects the status of ISR bit 1. This signal can be used to control parallel data flow by acting as an interrupt to indicate when the receiver FIFO is full.

7.2.13.3 OP4. When used for this function, this output reflects the complement of the value of bit 4 in the OP.

7.3 OPERATION

The following paragraphs describe the operation of the baud rate generator, transmitter and receiver, and other functional operating modes of the serial module.

7.3.1 Baud Rate Generator

The baud rate generator consists of a crystal oscillator, baud rate generator, and clock selectors (see Figure 7-3). The crystal oscillator operates directly from a 3.6864-MHz crystal or from an external clock of the same frequency. The SCLK input bypasses the baud rate generator and provides a synchronous mode of operation when used with a divide-by-1 clock and an asynchronous mode when used with the divide-by-16 clock. The clock is selected by programming the CSR for each channel.

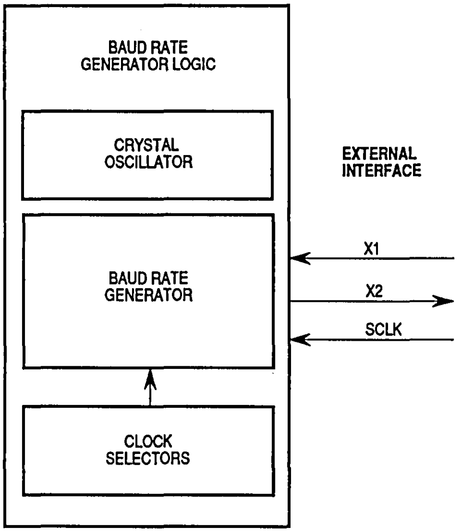


Figure 7-3. Baud Rate Generator Simplified Functional Diagram

7.3.2 Transmitter and Receiver Operating Modes

The simplified functional block diagram of the transmitter and receiver, including command and operating registers, is shown in Figure 7-4. The paragraphs that follow contain descriptions for both these functions in reference to this diagram. For detailed register information, refer to **7.4.1 REGISTER DESCRIPTION AND PROGRAMMING**.

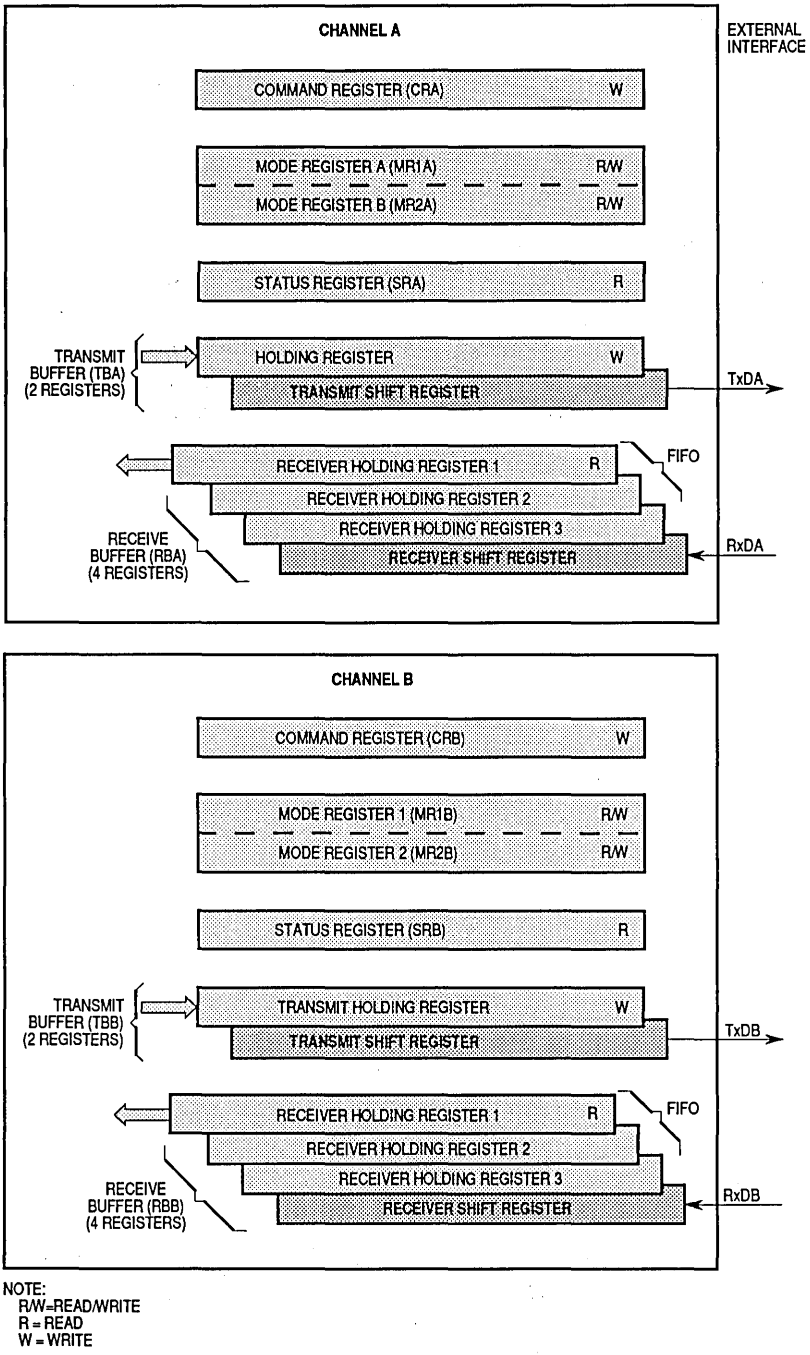

CHANNEL A
EXTERNAL INTERFACE
COMMAND REGISTER (CRA) W
MODE REGISTER A (MR1A) R/W
MODE REGISTER B (MR2A) R/W
STATUS REGISTER (SRA) R
TRANSMIT BUFFER (TBA) (2 REGISTERS)
HOLDING REGISTER W
TRANSMIT SHIFT REGISTER
TxDA
RECEIVER HOLDING REGISTER 1 R
FIFO
RECEIVER HOLDING REGISTER 2
RECEIVER HOLDING REGISTER 3
RECEIVE BUFFER (RBA) (4 REGISTERS)
RECEIVER SHIFT REGISTER
RxDA
CHANNEL B
COMMAND REGISTER (CRB) W
MODE REGISTER 1 (MR1B) R/W
MODE REGISTER 2 (MR2B) R/W
STATUS REGISTER (SRB) R
TRANSMIT BUFFER (TBB) (2 REGISTERS)
TRANSMIT HOLDING REGISTER W
TRANSMIT SHIFT REGISTER
TxDB
RECEIVER HOLDING REGISTER 1 R
FIFO
RECEIVER HOLDING REGISTER 2
RECEIVER HOLDING REGISTER 3
RECEIVE BUFFER (RBB) (4 REGISTERS)
RECEIVER SHIFT REGISTER
RxDB
NOTE:
R/W=READ/WRITE
R = READ
W = WRITE

Figure 7-4. Transmitter and Receiver Simplified Functional Diagram

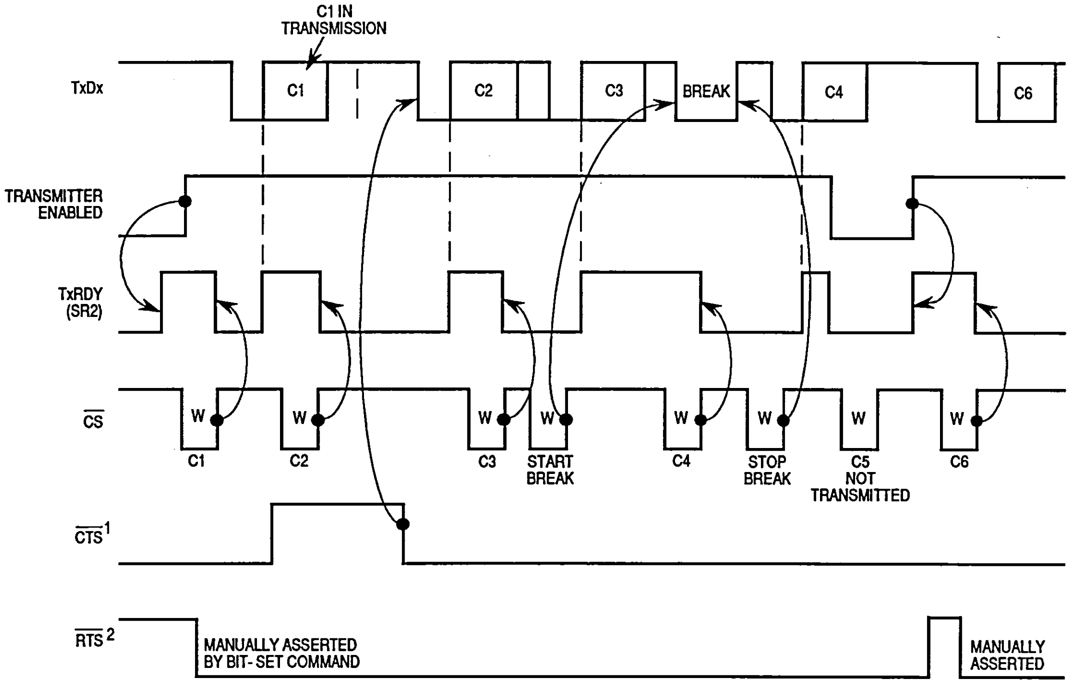
7.3.2.1 TRANSMITTER. The transmitters are enabled through their respective command registers (CR) located within the serial module. The serial module signals the CPU32 when it is ready to accept a character by setting the transmitter-ready bit (TxRDY) in the channel's status register (SR). Functional timing information for the transmitter is shown in Figure 7-5.

The transmitter converts parallel data from the CPU32 to a serial bit stream on TxDx. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The least significant bit is sent first. Data is shifted from the transmitter output on the falling edge of the programmed clock source.

Following transmission of the stop bits, if a new character is not available in the transmitter holding register, the TxDx output remains high ('mark' condition), and the transmitter empty bit (TxEMP) in the SR is set. Transmission resumes and the TxEMP bit is cleared when the CPU32 loads a new character into the transmitter buffer (TB). If a disable command is sent to the transmitter, it continues operating until the character in the transmit shift register, if any, is completely sent out. If the transmitter is reset through a software command, operation ceases immediately (refer to **7.4.1.7 COMMAND REGISTER**). The transmitter is re-enabled through the CR to resume operation after a disable or software reset.

If clear-to-send (CTS) operation is enabled, $\overline{\mathrm{CTSx}}$ must be asserted for the character to be transmitted. If $\overline{\mathrm{CTSx}}$ is negated in the middle of a transmission, the character in the shift register is transmitted, and TxDx remains in the 'mark' state until $\overline{\mathrm{CTSx}}$ is asserted again. If the transmitter is forced to send a continuous low condition by issuing a send break command, the state of $\overline{\mathrm{CTSx}}$ is ignored by the transmitter.

The transmitter can be programmed to automatically negate request-to-send ($\overline{\mathrm{RTS}}$) upon completion of a message transmission. If the transmitter is programmed to operate in this mode, $\overline{\mathrm{RTSx}}$ must be manually asserted before a message is transmitted. In applications in which the transmitter is disabled after transmission is complete and $\overline{\mathrm{RTSx}}$ is appropriately programmed, $\overline{\mathrm{RTSx}}$ is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually re-enabled by reasserting $\overline{\mathrm{RTSx}}$ before the next message is to be sent.



NOTES:

1. TIMING SHOWN FOR MR2(4) = 1
2. TIMING SHOWN FOR MR2(5) = 1
3. C_N = TRANSMIT CHARACTER
4. W = WRITE

Figure 7-5. Transmitter Timing Diagram

7.3.2.2 RECEIVER. The receivers are enabled through their respective CRs located within the serial module. Functional timing information for the receiver is shown in Figure 7-6. The receiver looks for a high-to-low (mark-to-space) transition of the start bit on RxDx. When a transition is detected, the state of RxDx is sampled each 16× clock for seven and one-half clocks (16× clock mode) or at the next rising edge of the bit time clock (1× clock mode). If RxDx is sampled high, the start bit is invalid and the search for the valid start bit begins again. If RxDx is still low, a valid start bit is assumed, and the receiver continues to sample the input at one-bit time intervals, at the theoretical center of the bit, until the proper number of data bits and parity, if any, is assembled and one stop bit is detected. Data on the RxDx input is sampled on the rising edge of the programmed clock source. The least significant bit is received first. The data is then transferred to a receiver holding register, and the receiver-ready bit (RxRDY) in the appropriate SR is set. If the character length is less than eight bits, the most significant unused bits in the holding register are cleared.

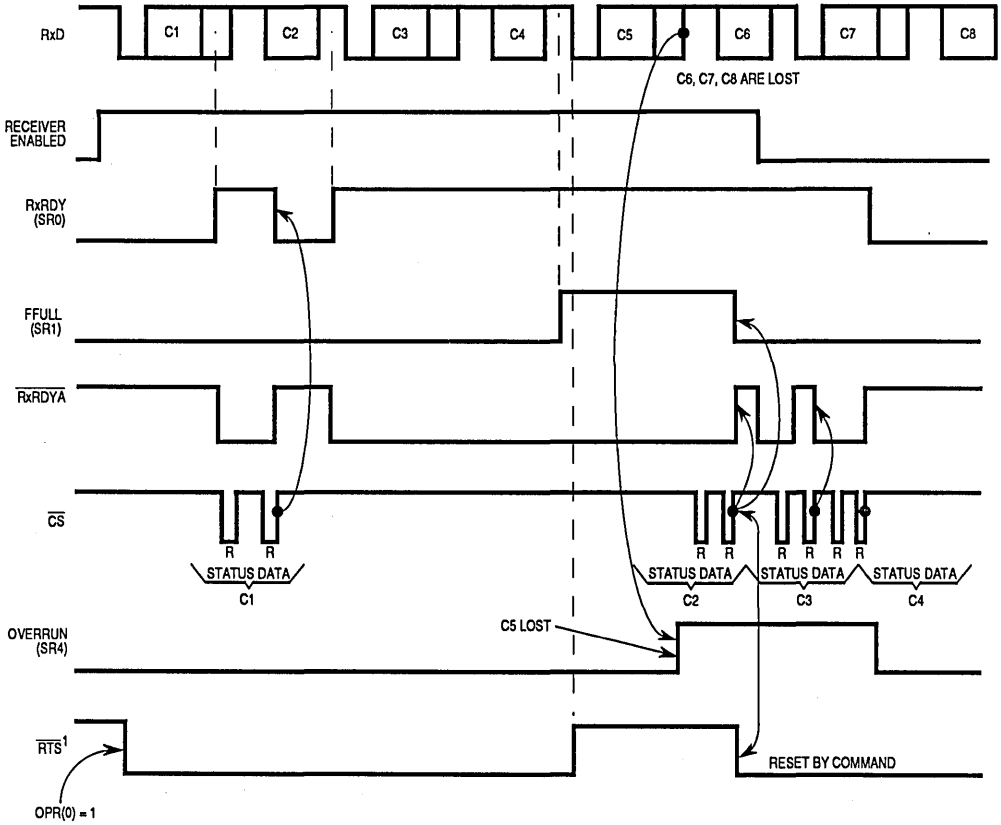
After the stop bit is detected, the receiver immediately looks for the next start bit. However, if a nonzero character is received without a stop bit (framing error) and RxDx remains low for one-half of the bit period after the stop bit is sampled, the receiver operates as if a new start bit is detected. The parity error (PE), framing error (FE), overrun error (OE), and received break (RB) conditions (if any) set error and break flags in the appropriate SR at the received character boundary and are valid only when the RxRDY bit in the SR is set.

If a break condition is detected (RxDx is low for the entire character including the stop bit), a character of all zeros is loaded into the receiver holding register, and the RB and RxRDY bits in the SR are set. The RxDx signal must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character if the break persists through the next character time. When the break begins in the middle of a character, the receiver places the damaged character in the receiver first-in-first-out (FIFO) stack and sets the corresponding error conditions and RxRDY bit in the SR. Then, if the break persists until the next character time, the receiver places an all-zero character into the receiver FIFO and sets the corresponding RB and RxRDY bits in the SR.

7.3.2.3 FIFO STACK. The FIFO stack is used in each channel's receiver buffer logic. The stack consists of three receiver holding registers. The RB consists of the FIFO and a receiver shift register connected to the RxDx (refer to Figure 7-4). Data is assembled in the shift register and loaded into the top empty receiver holding register position of the FIFO. Thus, data flowing from the receiver to the CPU32 is quadruple buffered.

In addition to the data byte, three status bits, PE, FE, and RB, are appended to each data character in the FIFO (OE is not). By programming the error mode control bit in the channel's mode register (MR1), status is provided in character or block modes.



NOTES:
1. Timing shown for MR1(7) = 1
2. Timing shown for OPCR(4) = 1 and MR1(6) = 0
3. R = Read
4. C_N = Received Character

Figure 7-6. Receiver Timing Diagram

The RxRDY bit in the SR is set whenever one or more characters are available to be read by the CPU32. A read of the RB produces an output of data from the top of the FIFO stack. After the read cycle, the data at the top of the FIFO stack and its associated status bits are 'popped', and new data can be added at the bottom of the stack by the receiver shift register. The FIFO-full status bit (FFULL) is set if all three stack positions are filled with data. Either the RxRDY or FFULL bit can be selected to cause an interrupt.

In the character mode, status provided in the SR is given on a character-by-character basis and thus applies only to the character at the top of the FIFO. In the block mode, the status provided in the SR is the logical OR of all characters coming to the top of the FIFO stack since the last reset error command. A continuous logical OR function of the corresponding status bits is produced in the SR as each character reaches the top of the FIFO stack. The block mode is useful in applications where the software overhead of checking each character's error cannot be tolerated. In this mode, entire messages are received, and only one data integrity check is performed at the end of the message. This mode allows a data-reception speed advantage, but does have a disadvantage since each character is not individually checked for error conditions by software. If an error occurs within the message, the error is not recognized until the final check is performed, and no indication exists as to which character in the message is at fault.

In either mode, reading the SR does not affect the FIFO. The FIFO is 'popped' only when the RB is read. The SR should be read prior to reading the RB. If all three of the FIFO's receiver holding registers are full when a new character is received, the new character is held in the receiver shift register until a FIFO position is available. If an additional character is received during this state, the contents of the FIFO are not affected. However, the character previously in the receiver shift register is lost, and the OE bit in the SR is set when the receiver detects the start bit of the new overrunning character.

To support control-flow capability, the receiver can be programmed to automatically negate and assert $\overline{\text{RTSx}}$. When in this mode, $\overline{\text{RTSx}}$ is automatically negated by the receiver when data is detected and the FIFO stack is full. When a FIFO position becomes available, $\overline{\text{RTSx}}$ is asserted by the receiver. Using this mode of operation, overrun errors are prevented by connecting the $\overline{\text{RTSx}}$ to the $\overline{\text{CTSx}}$ input of the transmitting device. $\overline{\text{RTSx}}$ must be manually asserted the first time in this mode.

If the FIFO stack contains characters and the receiver is disabled, the characters in the FIFO can still be read by the CPU32. If the receiver is reset, the FIFO stack and all of the receiver status bits, the corresponding output ports, and the interrupt request are reset. No additional characters are received until the receiver is re-enabled.

7.3.3 Loop Modes

Each serial module channel can be configured to operate in various loop modes as shown in Figure 7-7. These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs with further information available in **7.4 REGISTER DESCRIPTION AND PROGRAMMING.**

The channel should be disabled when switching between modes. The selected mode is activated immediately upon mode selection, regardless of whether a character is being received or transmitted.

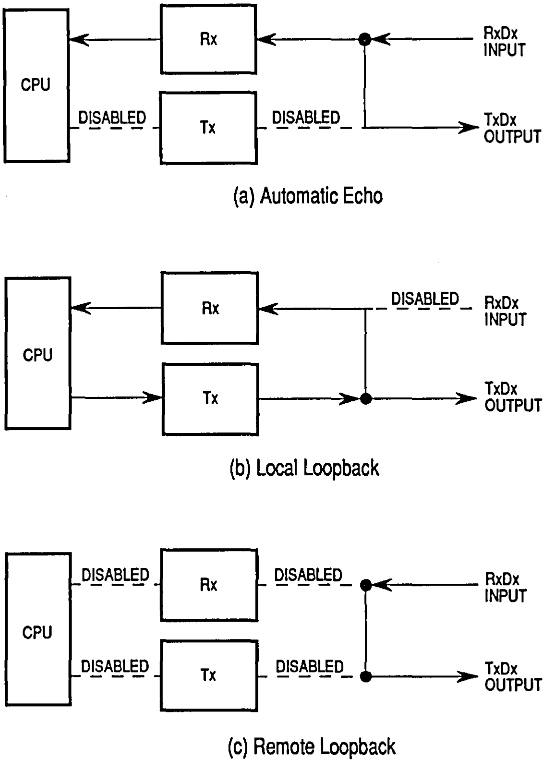
7.3.3.1 AUTOMATIC ECHO MODE. In this mode, the channel automatically retransmits the received data on a bit-by-bit basis. The local CPU32-to-receiver communication continues normally, but the CPU32-to-transmitter link is disabled. While in this mode, received data is clocked on the receiver clock and retransmitted on TxDx. The receiver must be enabled, but the transmitter need not be enabled.

Since the transmitter is not active, the SR TxRDY and TxEMP bits are inactive, and data is transmitted as it is received. Received parity is checked, but not recalculated for transmission. Character framing is also checked, but stop bits are transmitted as received. A received break is echoed as received until the next valid start bit is detected.

7.3.3.2 LOCAL LOOPBACK MODE. In this mode, TxDx is internally connected to RxDx. This mode is useful for testing the operation of a local serial module channel by sending data to the transmitter and checking data assembled by the receiver. In this manner, correct channel operations can be assured. Also, both transmitter and CPU32-to-receiver communications continue normally in this mode. While in this mode, the RxDx input data is ignored, the TxDx is held marking, and the receiver is clocked by the transmitter clock. The transmitter must be enabled, but the receiver need not be enabled.

7.3.3.3 REMOTE LOOPBACK MODE. In this mode, the channel automatically transmits received data on the TxDx output on a bit-by-bit basis. The local CPU32-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote channel. While in this mode, the receiver clock is used for the transmitter.

Since the receiver is not active, received data can not be read by the CPU32, and the error status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are transmitted as received. A received break is echoed as received until the next valid start bit is detected.



(a) Automatic Echo

(b) Local Loopback

(c) Remote Loopback

Figure 7-7. Loop Modes Functional Diagram

7.3.4 Multidrop Mode

A channel can be programmed to operate in a wakeup mode for multidrop or multiprocessor applications. Functional timing information for the multidrop mode is shown in Figure 7-8. The mode is selected by setting bits 3 and 4 in mode register 1 (MR1). This mode of operation allows the master station to be connected to several slave stations (maximum of 256). In this mode, the master transmits an address character followed by a block of data characters targeted for one of the slave stations. The slave stations have their channel receivers disabled. However, they continuously monitor the data stream sent out by the master station. When an address character is sent by the master, the slave receiver channel notifies its respective CPU by setting the RxRDY bit in the SR and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the master station. Slave stations not addressed continue to monitor the data stream for the next address character. Data fields in the data stream are separated by an address character. After a slave receives a block of data, the slave station's CPU disables the receiver and initiates the process again.

A transmitted character from the master station consists of a start bit, a programmed number of data bits, an address/data (A/D) bit flag, and a programmed number of stop bits. The A/D bit identifies the type of character being transmitted to the slave station. The character is interpreted as an address character if the A/D bit is set or as a data character if the A/D bit is cleared. The polarity of the A/D bit is selected by programming bit 2 of the MR1. The MR1 should be programmed before enabling the transmitter and loading the corresponding data bits into the transmitter buffer (TB).

In multidrop mode, the receiver continuously monitors the received data stream regardless of whether it is enabled or disabled. If the receiver is disabled, it sets the RxRDY bit and loads the character into the receiver holding register FIFO stack provided the received A/D bit is a one (address tag). The character is discarded if the received A/D bit is a zero (data tag). If the receiver is enabled, all received characters are transferred to the CPU32 via the receiver holding register stack during read operations.

In either case, the data bits are loaded into the data portion of the stack while the A/D bit is loaded into the status portion of the stack normally used for a parity error (SR bit 5). Framing error, overrun error, and break-detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may still contain error detection and correction information. One way to provide error detection, if 8-bit characters are not required, is to use software to calculate parity and append it to the unused 5-, 6-, or 7-bit character.

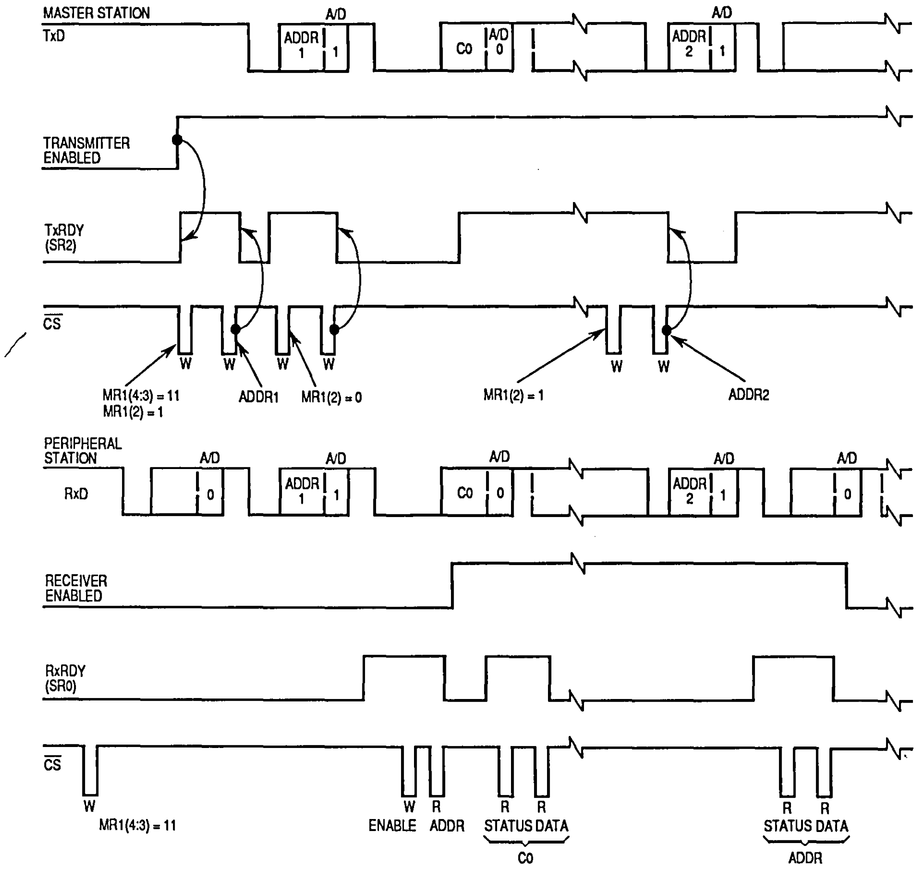


Figure 7-8. Multidrop Mode Timing Diagram

7.3.5 Bus Operation

This section describes the operation of the IMB during read, write, and interrupt acknowledge cycles to the serial module.

7.3.5.1 READ CYCLES. The serial module is accessed by the CPU32 with no wait states. The serial module responds to byte, word, and long-word reads, although only eight bits of valid data are returned. Reserved registers return logic zero during reads.

7.3.5.2 WRITE CYCLES. The serial module is accessed by the CPU32 with no wait states. The serial module responds to byte, word, and long-word writes, but care must be taken to properly align and size write operations as A0 determines the alignment of the data. Write cycles to read-only registers and reserved registers complete in a normal manner without exception processing; however, the data is ignored.

7.3.5.3 INTERRUPT ACKNOWLEDGE CYCLES. The serial module is capable of arbitrating for interrupt servicing and supplying the interrupt vector when it has successfully won arbitration. The vector number must be provided if interrupt servicing is necessary; thus, the interrupt vector register (IVR) must be initialized. If the IVR is not initialized, a spurious interrupt exception will be taken if interrupts are generated.

7.4 REGISTER DESCRIPTION AND PROGRAMMING

This section contains a detailed description of each register and its specific function as well as examples of basic serial module programming.

7.4.1 Register Description

The operation of the serial module is controlled by writing control words into the appropriate registers. A list of serial module registers and their associated addresses are listed in Table 7-1. The command, clock-select, status, and mode registers are duplicated for each channel to provide independent operation and control.

NOTE

The contents of the mode registers (MR1 and MR2), clock-select register (CSR), and the auxiliary control register (ACR) bit 7 should only be changed after the receiver/transmitter is issued a software reset command — i.e., channel operation must be disabled. Care should also be taken if the register contents are changed during receiver/transmitter operations, as undesirable results may be produced.

In the registers discussed in the following pages, the numbers in the upper right-hand corner indicate the offset of the register from the base address specified in the base address register in the SIM. The numbers above the register description represent the bit position in the register. The second line contains the mnemonic for the bit. The values shown below the register diagram are the values of those register bits after reset. A value of U indicates that the bit value is unaffected by reset. The read/write status and the access privilege are shown in the last line.

Table 7-1. Register Addressing and Address-Triggered Commands

ADDRESS	FC	REGISTER READ (R/W=1)	REGISTER WRITE (R/W=0)
700	S[1]	MCR (HIGH BYTE)	MCR (HIGH BYTE)
701	S	MCR (LOW BYTE)	MCR (LOW BYTE)
702	S	DO NOT ACCESS[3]	DO NOT ACCESS[3]
703	S	DO NOT ACCESS[3]	DO NOT ACCESS[3]
704	S	INTERRUPT LEVEL (ILR)	INTERRUPT LEVEL (ILR)
705	S	INTERRUPT VECTOR (IVR)	INTERRUPT VECTOR (IVR)
710	S/U[2]	MODE REGISTER MR1A	MODE REGISTER MR1A
711	S/U	STATUS REGISTER A (SRA)	CLOCK-SELECT REGISTER A (CSRA)
712	S/U	DO NOT ACCESS[3]	COMMAND REGISTER A (CRA)
713	S/U	RECEIVER BUFFER A (RBA)	TRANSMITTER BUFFER A (TBA)
714	S/U	INPUT PORT CHANGE REGISTER (IPCR)	AUXILIARY CONTROL REGISTER (ACR)
715	S/U	INTERRUPT STATUS REGISTER (ISR)	INTERRUPT ENABLE REGISTER (IER)
716	S/U	DO NOT ACCESS[3]	DO NOT ACCESS[3]
717	S/U	DO NOT ACCESS[3]	DO NOT ACCESS[3]
718	S/U	MODE REGISTER MR1B	MODE REGISTER MR1B
719	S/U	STATUS REGISTER B (SRB)	CLOCK SELECT REGISTER B (CSRB)
71A	S/U	DO NOT ACCESS[3]	COMMAND REGISTER B (CRB)
71B	S/U	RECEIVER BUFFER B (RBB)	TRANSMITTER BUFFER B (TBB)
71C	S/U	DO NOT ACCESS[3]	DO NOT ACCESS[3]
71D	S/U	INPUT PORT REGISTER (IP)	OUTPUT PORT CONTROL REGISTER (OPCR)
71E	S/U	DO NOT ACCESS[3]	OUTPUT PORT (OP)[4] BIT SET
71F	S/U	DO NOT ACCESS[3]	OUTPUT PORT (OP)[4] BIT RESET
720	S/U	MODE REGISTER MR2A	MODE REGISTER MR2A
721	S/U	MODE REGISTER MR2B	MODE REGISTER MR2B

NOTES:

1. S — Register permanently defined as supervisor-only access
2. S/U — Register programmable as either supervisor or user access
3. A read or write to these locations currently has no effect.
4. Address-triggered commands

7.4.1.1 MODULE CONFIGURATION REGISTER (MCR).

The MCR controls the serial module configuration. The register can be either read or written when the module is enabled and is in the supervisor state.

MCR $700

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STP	FRZ1	FRZ0	ICCS	0	0	0	0	SUPV	0	0	0	IARB3	IARB2	IARB1	IARB0
RESET:															
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Read/Write Supervisor/User

STP — Stop Mode

1 = Stops all clocks within the serial module (including the crystal and SCLK) except for the clock from the IMB. The clocks are stopped on the low phase of the clock and will remain stopped until this bit is cleared. The clock from the IMB remains active to allow the CPU32 to access the MCR. Accesses to other serial module registers produce a bus error while in stop mode. The serial module should be disabled (in a known state) prior to setting the STP bit; otherwise, unpredictable results may occur.

0 = The serial module operates in normal mode.

FRZ1–FRZ0 — Freeze

These bits determine the action taken when the FREEZE signal is asserted by the IMB. Table 7-2 lists the action taken for each combination of bits.

Table 7-2. FREEZE Control Bits

FRZ1	FRZ0	ACTION
0	0	Ignore FREEZE
0	1	Reserved (FREEZE Ignored)
1	0	Freeze on Character Boundary
1	1	Freeze on Character Boundary

If FREEZE is asserted, channel A and channel B freeze independently of each other. The transmitter and receiver freeze at character boundaries. The transmitter does not freeze in the send break mode. Communications can be lost if the channel is not programmed to support flow control. If the channel is programmed for flow control, the assertion of FREEZE causes the $\overline{\text{RTSx}}$ and $\overline{\text{CTSx}}$ pins to disable at the end of the character boundary. See **SECTION 4 SYSTEM INTEGRATION MODULE** for more information.

ICCS — Input Capture Clock Select

1 = Selects SCLK as the input capture clock for both channels. The data is captured on the $\overline{\text{CTSA}}$ and $\overline{\text{CTSB}}$ pins on the rising edge of the clock.

0 = The crystal clock is the input capture clock for both channels.

SUPV — Supervisor/User

1 = The serial module registers, which are defined as supervisor or user, reside in supervisor data space and are only accessible from supervisor programs.

0 = The serial module registers, which are defined as supervisor or user, reside in user data space and are accessible from either supervisor or user programs.

The value of this bit has no effect on registers permanently defined as supervisor only.

IARB3–IARB0 — Interrupt Arbitration Bits

Each module that generates interrupts has an IARB field. The value of the IARB field allows arbitration during an interrupt acknowledge cycle among modules that simultaneously generate the same interrupt level. No two modules can share the same IARB value. The reset value of IARB is $0, which prevents this module from arbitrating during the interrupt acknowledge cycle. The system software should initialize the IARB field to a value from $F (highest priority) to $1 (lowest priority).

7.4.1.2 INTERRUPT LEVEL REGISTER (ILR). The ILR contains the priority level for the serial module interrupt request. When the serial module is enabled, this register can be read or written to at any time while in supervisor mode.

ILR $704

7	6	5	4	3	2	1	0
0	0	0	0	0	IL2	IL1	IL0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor/User

IL2–IL0 — Interrupt Level Bits

Each module that can generate interrupts has an ILR. The priority level encoded in these bits is sent to the CPU32 on the appropriate $\overline{\text{IRQx}}$ signal. The CPU32 uses this value to determine servicing priority. See **SECTION 5 CPU32** for more information.

7.4.1.3 INTERRUPT VECTOR REGISTER (IVR). The IVR contains the 8-bit vector number of the interrupt.

IVR $705

7	6	5	4	3	2	1	0
IVR7	IVR6	IVR5	IVR4	IVR3	IVR2	IVR1	IVR0
RESET:							
0	0	0	0	1	1	1	1

Read /Write Supervisor Only

IVR7–IVR0 — Interrupt Vector Bits

Each module that can generate interrupts has an IVR. This 8-bit number indicates the offset from the base of the vector table where the address of the exception handler for the specified interrupt is located. The IVR is reset to $0F, which indicates an uninitialized interrupt condition. See **SECTION 5 CPU32** for more information.

7.4.1.4 MODE REGISTER 1 (MR1). MR1 controls some of the serial module configuration. This register can be read or written at any time when the serial module is enabled.

MR1A, MR1B $710, $718

7	6	5	4	3	2	1	0
RxRTS	R/F	ERR	PM1	PM0	PT	B/C1	B/C0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor/User

RxRTS — Receiver Request-to-Send Control

1 = Upon receipt of a valid start bit, $\overline{\text{RTSx}}$ is negated if the channel's FIFO is full. $\overline{\text{RTSx}}$ is reasserted when the FIFO has an empty position available.

0 = $\overline{\text{RTSx}}$ is asserted by setting bit 1 or 0 in the OP and negated by clearing bit 1 or 0 in the OP.

This feature can be used for flow control to prevent overrun in the receiver by using the $\overline{\text{RTSx}}$ output to control the $\overline{\text{CTSx}}$ input of the transmitting device. If both the receiver and transmitter are programmed for $\overline{\text{RTS}}$ control, $\overline{\text{RTS}}$ control will be disabled for both, as this configuration is incorrect. See **7.4.1.17 MODE REGISTER 2** for information on programming the transmitter RTS control.

R/F — Receiver-Ready Select

1 = Bit 5 for channel B and bit 1 for channel A in the ISR reflect the channel FIFO full status. These ISR bits are set when the receiver FIFO is full and are cleared when a position is available in the FIFO.

0 = Bit 5 for channel B and bit 1 for channel A in the ISR reflect the channel receiver-ready status. These ISR bits are set when a character has been received and are cleared when the CPU32 reads the receiver buffer.

ERR–Error Mode

This bit controls the meaning of the three FIFO status bits (FE, PE, and RB) for the channel.

1 = Block mode — The values in the channel SR are the accumulation (i.e., the logical OR) of the status for all characters coming to the top of the FIFO since the last reset error status command for the channel was issued. Refer to **7.4.1.7 COMMAND REGISTER (CR)** for more information on serial module commands.

0 = Character mode — The values in the channel SR reflect the status of the character at the top of the FIFO.

PM1–PM0 — Parity Mode

These bits encode the type of parity used for the channel. The parity bit is added to the transmitted character, and the receiver performs a parity check on incoming data. These bits can alternatively select multidrop mode for the channel. Table 7-3 lists the parity mode and type or the multidrop mode for each combination of the parity mode and the parity type bits.

PT — Parity Type

This bit selects the parity type if parity is programmed by the parity mode bits, and, if multidrop mode is selected, it configures the transmitter for address character transmission or data character transmission.

Table 7-3. Parity Mode and Parity Type Control Bits

PM1	PM0	Parity Mode	PT	Parity Type
0	0	With Parity	0	Even Parity
0	0	With Parity	1	Odd Parity
0	1	Force Parity	0	Low Parity
0	1	Force Parity	1	High Parity
1	0	No Parity	X	No Parity
1	1	Multidrop Mode	0	Data Character
1	1	Multidrop Mode	1	Address Character

B/C1,B/C0 — Bits per Character

These bits select the number of data bits per character to be transmitted. The character length listed in Table 7-4 does not include start, parity, or stop bits.

Table 7-4. Bits/Character Control Bits

B/C1	B/C0	Bits/Character
0	0	Five Bits
0	1	Six Bits
1	0	Seven Bits
1	1	Eight Bits

7.4.1.5 STATUS REGISTER (SR). The SR indicates the status of the characters in the FIFO and the status of the channel transmitter and receiver. This register can only be read when the serial module is enabled.

SRA, SRB $711, $719

	7	6	5	4	3	2	1	0
	RB	FE	PE	CE	TxEMP	TxRDY	FFULL	RxRDY
RESET:	0	0	0	0	0	0	0	0

Read Only Supervisor/User

RB — Received Break

1 = An all-zero character of the programmed length has been received without a stop bit. The RB bit is only valid when the RxRDY bit is set. Only a single FIFO position is occupied when a break is received. Further entries to the FIFO are inhibited until the channel RxDx returns to the high state for at least one-half bit time, which is equal to two successive edges of the internal or external 1 × clock or 16 successive edges of the external 16 × clock.

The received break circuit detects breaks that originate in the middle of a received character. However, if a break begins in the middle of a character, it must persist until the end of the next detected character time.

0 = No break has been received.

FE — Framing Error

1 = A stop bit was not detected when the corresponding data character in the FIFO was received. The stop-bit check is made in the middle of the first stop-bit position. The bit is valid only when the RxRDY bit is set.

0 = No framing error has occurred.

PE — Parity Error

1 = When the with parity or force parity mode is programmed (MR1), the corresponding character in the FIFO was received with incorrect parity. When the multidrop mode is programmed, this bit stores the received A/D bit. This bit is valid only when the RxRDY bit is set.

0 = No parity error has occurred.

OE — Overrun Error

1 = One or more characters in the received data stream have been lost. This bit is set upon receipt of a new character when the FIFO is full and a character is already in the shift register waiting for an empty FIFO position. When this occurs, the character in the receiver shift register and its break detect, parity error, and framing error status, if any, are lost. This bit is cleared by the reset error status command in the CR.

0 = No overrun has occurred.

TxEMP — Transmitter Empty

1 = The channel transmitter has underrun (both the transmitter holding register and transmitter shift registers are empty). This bit is set after transmission of the last stop bit of a character if there are no characters in the transmitter holding register awaiting transmission. This bit is cleared when the transmitter holding register is loaded by the CPU32 or when the transmitter is disabled.

0 = The TB is not empty.

TxRDY — Transmitter Ready

1 = The transmitter holding register is empty and ready to be loaded with a character. This bit is set when the character is transferred to the transmitter shift register. This bit is also set when the transmitter is first enabled. Characters loaded into the transmitter holding register while the transmitter is disabled are not transmitted and are lost.

0 = The transmitter holding register was loaded by the CPU32. This bit is also cleared when the transmitter is disabled.

FFULL — FIFO Full

1 = A character was transferred from the receiver shift register to the receiver FIFO and the transfer caused the FIFO to become full (all three FIFO holding register positions are occupied).

0 = The CPU32 has read the RB and one or more FIFO positions are available. Note that if there is a character in the receiver shift register because the FIFO is full, this character will be moved into the FIFO when a position is available, and the FIFO will remain full.

RxRDY — Receiver Ready

1 = A character has been received and is waiting in the FIFO to be read by the CPU32. This bit is set when a character is transferred from the receiver shift register to the FIFO.

0 = The CPU32 has read the RB, and no characters remain in the FIFO after this read.

7.4.1.6 CLOCK-SELECT REGISTER (CSR). The CSR selects the baud rate clock for the channel receiver and transmitter. This register can only be written.

NOTE

This register should only be written after the external crystal is stable (XTAL_RDY bit (bit 3) of the ISR is zero).

CSRA, CSRB $711, $719

	7	6	5	4	3	2	1	0
	RCS3	RCS2	RCS1	RCS0	TCS3	TCS2	TCS1	TCS0
RESET:	0	0	0	0	0	0	0	0

Write Only Supervisor/User

RCS3–RCS0 — Receiver Clock Select

These bits select the baud rate clock for the channel receiver from a set of baud rates listed in Table 7-5. The baud rate set selected depends upon ACR bit 7. Set 1 is selected if ACR bit 7 = 0, and set 2 is selected if ACR bit 7 = 1. The receiver clock is always 16 times the baud rate shown in this list, except when SCLK is used.

Table 7-5. Receiver Clock Select

RCS3	RCS2	RCS1	RCS0	Set 1	Set 2
0	0	0	0	50	75
0	0	0	1	110	110
0	0	1	0	134.5	134.5
0	0	1	1	200	150
0	1	0	0	300	300
0	1	0	1	600	600
0	1	1	0	1200	1200
0	1	1	1	1050	2000
1	0	0	0	2400	2400
1	0	0	1	4800	4800
1	0	1	0	7200	1800
1	0	1	1	9600	9600
1	1	0	0	38.4k	19.2k
1	1	0	1	76.8k	38.4k
1	1	1	0	SCLK/16	SCLK/16
1	1	1	1	SCLK/1	SCLK/1

TCS3–TCS0 — Transmitter Clock Select

These bits select the baud rate clock for the channel transmitter from a set of baud rates listed in Table 7-6. The baud rate set selected depends upon ACR bit 7. Set 1 is selected if ACR bit 7 = 0, and set 2 is selected if ACR bit 7 = 1. The receiver clock is always 16 times the baud rate shown in this list, except when SCLK is used.

Table 7-6. Transmitter Clock Select

TCS3	TCS2	TCS1	TCS0	Set 1	Set 2
0	0	0	0	50	75
0	0	0	1	110	110
0	0	1	0	134.5	134.5
0	0	1	1	200	150
0	1	0	0	300	300
0	1	0	1	600	600
0	1	1	0	1200	1200
0	1	1	1	1050	2000
1	0	0	0	2400	2400
1	0	0	1	4800	4800
1	0	1	0	7200	1800
1	0	1	1	9600	9600
1	1	0	0	38.4k	19.2k
1	1	0	1	76.8k	38.4k
1	1	1	0	SCLK/16	SCLK/16
1	1	1	1	SCLK/1	SCLK/1

7.4.1.7 COMMAND REGISTER (CR). CR is used to supply commands to the channel. Multiple commands can be specified in a single write to the CR if the commands are not conflicting — e.g., enable transmitter and reset transmitter commands cannot be specified in a single command. This register can only be written when the serial module is enabled.

CRA, CRB $712, $71A

7	6	5	4	3	2	1	0
MISC3	MISC2	MISC1	MISC0	TC1	TC0	RC1	RC0
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

MISC3–MISC0 — Miscellaneous Commands

These bits select a single command as listed in Table 7-7.

Table 7-7. Miscellaneous Command Control Bits

MISC3	MISC2	MISC1	MISC0	Command
0	0	0	0	No Command
0	0	0	1	No Command
0	0	1	0	Reset Receiver
0	0	1	1	Reset Transmitter
0	1	0	0	Reset Error Status
0	1	0	1	Reset Break-Change Interrupt
0	1	1	0	Start Break
0	1	1	1	Stop Break
1	0	0	0	Assert $\overline{\text{RTS}}$
1	0	0	1	Negate $\overline{\text{RTS}}$
1	0	1	0	No Command
1	0	1	1	No Command
1	1	0	0	No Command
1	1	0	1	No Command
1	1	1	0	No Command
1	1	1	1	No Command

Reset Receiver — The reset receiver command resets the channel receiver. The receiver is immediately disabled, the SR RxRDY and FFULL bits (bits 0 and 1) are cleared, and the receiver FIFO pointer is reinitialized. All other registers are unaltered. This command should be used in lieu of the receiver disable command whenever the receiver configuration is changed because it places the receiver in a known state.

Reset Transmitter — The reset transmitter command resets the channel transmitter. The transmitter is immediately disabled, and the SR TxRDY and TxEMP bits (bits 2 and 3) are cleared. All other registers are unaltered. This command should be used in lieu of the transmitter disable command whenever the transmitter configuration is changed because it places the transmitter in a known state.

Reset Error Status — The reset error status command clears the channel's SR RB, PE, FE, and OE (bits 7–4). This command is also used in the block mode to clear all error bits after a data block is received.

Reset Break-Change Interrupt — The reset break-change interrupt command clears the break-detect change bit (bit 2 of the ISR).

Start Break — The start break command forces the channel's TxDx low. If the transmitter is empty, the start of the break conditions can be delayed up to one bit time. If the transmitter is active, the break begins when transmission of the character is complete. If a character is in the transmitter shift register, the start of the break is delayed until the character is transmitted. If the transmitter holding register has a character, that character is transmitted after the break. The transmitter must be enabled for this command to be accepted. The state of the $\overline{\mathrm{CTSx}}$ input is ignored for this command.

Stop Break — The stop break command causes the channel's TxDx to go high (mark) within two bit times. Characters stored in the TB, if any, are transmitted.

Assert $\overline{\mathrm{RTS}}$ — The assert $\overline{\mathrm{RTS}}$ command forces the channel's $\overline{\mathrm{RTSx}}$ output low.

Negate $\overline{\mathrm{RTS}}$ — The negate $\overline{\mathrm{RTS}}$ command forces the channel's $\overline{\mathrm{RTSx}}$ output high.

TC1–TC0 — Transmitter Commands
These bits select a single command as listed in Table 7-8.

Table 7-8. Transmitter Command Bits

TC1	TC0	Command
0	0	No Action Taken
0	1	Transmitter Enable
1	0	Transmitter Disable
1	1	Do Not Use

No Action Taken — The no action taken command causes the transmitter to stay in its current mode. If the transmitter is enabled, it remains enabled; if disabled, it remains disabled.

Transmitter Enable — The transmitter enable command enables operation of the channel's transmitter. The TxRDY and TxEMP bits (bits 2 and 3 of the SR) are also set. If the transmitter is already enabled, this command has no effect.

Transmitter Disable — The transmitter disable command terminates transmitter operation and clears the TxRDY and TxEMP bits (bits 2 and 3 of the

SR). However, if a character is being transmitted when the transmitter is disabled, the transmission of the character is completed before the transmitter becomes inactive. If the transmitter is already disabled, this command has no effect.

Do Not Use — Do not use this bit combination as the result is indeterminate.

RC1–RC0 — Receiver Commands
These bits select a single command as listed in Table 7-9.

Table 7-9. Receiver Command Bits

RC1	RC0	Command
0	0	No Action Taken
0	1	Receiver Enable
1	0	Receiver Disable
1	1	Do Not Use

No Action Taken — The no action taken command causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.

Receiver Enable — The receiver enable command enables operation of the channel's receiver. If the serial module is not in multidrop mode, this command also forces the receiver into the search-for-start-bit state. If the receiver is already enabled, this command has no effect.

Receiver Disable — The receiver disable command disables the receiver immediately. Any character being received is lost. The command has no effect on the receiver status bits or any other control register. If the serial module is programmed to operate in the local loopback mode or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, this command has no effect.

Do Not Use — Do not use this bit combination because the result is indeterminate.

7.4.1.8 RECEIVER BUFFER (RB). The RB contains three receiver holding registers and a serial shift register. The channel's RxDx pin is connected to the serial shift register. The holding registers act as a FIFO. The CPU32 reads from the top of the stack while the receiver shifts and updates from the bottom of the stack when the shift register has been filled (see Figure 7-4). This register can only be read when the serial module is enabled.

RBA, RBB $713, $71B

7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
RESET:							
0	0	0	0	0	0	0	0

Read Only Supervisor/User

RB7–RB0 — These bits contain the character in the RB.

7.4.1.9 TRANSMITTER BUFFER (TB). The TB consists of two registers, the transmitter holding register and the transmitter shift register (See Figure 7-4). The holding register accepts characters from the bus master if the TxRDY bit in the channel SR is set. A write to the TB clears the TxRDY bit, inhibiting any more characters until the shift register is ready to accept more data. The shift register, when empty, checks to see if the holding register has a valid character to be sent (TxRDY bit cleared). If there is a valid character, the shift register loads the character and reasserts the TxRDY bit in the channel's SR. Writes to the TB when the channel's SR TxRDY bit is clear and when the transmitter is disabled have no effect on the TB. This register can only be written when the serial module is enabled.

TBA, TBB $713, $71B

7	6	5	4	3	2	1	0
TB7	TB6	TB5	TB4	TB3	TB2	TB1	TB0
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

TB7–TB0 — These bits contain the character in the TB.

7.4.1.10 INPUT PORT CHANGE REGISTER (IPCR). The IPCR shows the current state and the change of state for the $\overline{\text{CTSA}}$ and $\overline{\text{CTSB}}$ pins. This register can only be read when the serial module is enabled.

IPCR $714

7	6	5	4	3	2	1	0
0	0	COSB	COSA	0	0	CTSB	CTSA
RESET:							
0	0	0	0	0	0	U	U

Read Only Supervisor/User

COSB, COSA — Change of State

1 = A change of state, a high-to-low or low-to-high transition, lasting longer than 25–50 µs when using a crystal as the sampling clock, or longer than one period when using SCLK, has occurred at the corresponding $\overline{\text{CTSx}}$ input (MCR bit 12 controls selection of sampling clock). When these bits are set, the ACR can be programmed to generate an interrupt to the CPU32.

0 = The CPU32 has read the IPCR. A read of the IPCR also clears bit 7 of the ISR.

CTSB, CTSA — Current State

1 = The current state of the respective $\overline{\text{CTSx}}$ input is negated.

0 = The current state of the respective $\overline{\text{CTSx}}$ input is asserted.

The information contained in these bits is latched and reflects the state of the input pins at the time that the IPCR is read.

7.4.1.11 AUXILIARY CONTROL REGISTER (ACR). The ACR selects which baud rate is used and controls the handshake of the transmitter/receiver. This register can only be written when the serial module is enabled.

ACR $714

7	6	5	4	3	2	1	0
BRG	0	0	0	0	0	IECB	IECA
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

BRG — Baud Rate Generator Set Select

1 = Set 2 of the available baud rates is selected.

0 = Set 1 of the available baud rates is selected. Refer to **7.4.1.6 CLOCK SELECT REGISTER (CSR)** for more information on the baud rates.

IECB, IECA — Input Enable Control

1 = ISR bit 7 will be set and an interrupt output will be generated when the corresponding bit in the IPCR (COSB or COSA) is set by an external transition on the channel's $\overline{\text{CTSx}}$ input (if bit 7 of the interrupt enable register (IER) is set to enable interrupts).

0 = Setting the corresponding bit in the IPCR has no effect on ISR bit 7.

7.4.1.12 INTERRUPT STATUS REGISTER (ISR). The ISR provides status for all potential interrupt sources. The contents of this register are masked by the IER. If a flag in the ISR is set and the corresponding bit in IER is also set, the $\overline{\text{IRQx}}$ output is asserted. If the corresponding bit in the IER is cleared, the state of the bit in the ISR has no effect on the output. This register can only be read when the serial module is enabled.

NOTE

The IER does not mask reading of the ISR. True status is provided regardless of the contents of IER. The contents of ISR are cleared when the serial module is reset.

ISR $715

7	6	5	4	3	2	1	0
COS	DBB	RxRDYB	TxRDYB	XTAL_RDY	DBA	RxRDYA	TxRDYA
RESET:							
0	0	0	0	0	0	0	0

Read Only Supervisor/User

COS — Change of State

1 = A change of state has occurred at one of the $\overline{\text{CTSx}}$ inputs, and has been selected to cause an interrupt by programming bit 1 and/or bit 0 of the ACR.

0 = The CPU32 has read the IPCR.

DBB — Delta Break B

1 = The channel B receiver has detected the beginning or end of a received break.

0 = The CPU32 has issued a channel B reset break change interrupt command. Refer to **7.4.1.7 COMMAND REGISTER (CR)** for more information on this command.

RxRDYB — Channel B Receiver Ready or FIFO Full

The function of this bit is programmed by MR1B bit 6.

1 = If programmed as receiver ready, a character has been received in channel B and is waiting in the RB FIFO. If programmed as FIFO full, a character has been transferred from the receiver shift register to the FIFO, and the transfer has caused the channel B FIFO to become full (all three positions are occupied).

0 = If programmed as receiver ready, the CPU32 has read the RB. After this read, if more characters are still in the FIFO, the bit is set again after the FIFO is "popped." If programmed as FIFO full, the CPU32 has read the RB. If a character is waiting in the receiver shift register because the FIFO is full, the bit will be set again when the waiting character is loaded into the FIFO.

TxRDYB — Channel B Transmitter Ready

This bit is the duplication of the TxRDY bit in SRB.

1 = The transmitter holding register is empty and ready to be loaded with a character. This bit is set when the character is transferred to the transmitter shift register. This bit is also set when the transmitter is first enabled. Characters loaded into the transmitter holding register while the transmitter is disabled are not transmitted.

0 = The transmitter holding register was loaded by the CPU32. This bit is also cleared when the transmitter is disabled.

XTAL_RDY — Serial Clocks Running

This bit is always read as a zero when the serial clock is running. Note that this bit cannot be enabled to generate an interrupt.

1 = This bit is set at reset. See MC68340/D, *MC68340 Technical Summary*, for more information on the timing of reset and power-down.

0 = This bit is cleared after the baud rate generator is stable. The CSR should not be accessed until this bit is zero.

DBA — Delta Break A

1 = The channel A receiver has detected the beginning or end of a received break.

0 = The CPU32 has issued a channel A reset break change interrupt command. Refer to **7.4.1.7 COMMAND REGISTER (CR)** for more information on this command.

RxRDYA — Channel A Receiver Ready or FIFO Full

The function of this bit is programmed by MR1A bit 6.

1 = If programmed as receiver ready, a character has been received in channel A and is waiting in the RB FIFO. If programmed as FIFO full, a character has been transferred from the receiver shift register to the FIFO, and the transfer has caused the channel A FIFO to become full (all three positions are occupied).

0 = If programmed as receiver ready, the CPU32 has read the RB. After this read, if more characters are still in the FIFO, the bit is set again after the FIFO is "popped." If programmed as FIFO full, the CPU32 has read the RB. If a character is waiting in the receiver shift register because the FIFO is full, the bit will be set again when the waiting character is loaded into the FIFO.

TxRDYA — Channel A Transmitter Ready

This bit is the duplication of the TxRDY bit in SRA.

1 = The transmitter holding register is empty and ready to be loaded with a character. This bit is set when the character is transferred to the transmitter shift register. This bit is also set when the transmitter is first enabled. Characters loaded into the transmitter holding register while the transmitter is disabled are not transmitted.

0 = The transmitter holding register was loaded by the CPU32. This bit is also cleared when the transmitter is disabled.

7.4.1.13 INTERRUPT ENABLE REGISTER (IER). The IER selects the corresponding bits in the ISR that cause an interrupt output ($\overline{\text{IRQx}}$). If one of the bits in the ISR is set and the corresponding bit in the IER is also set, the $\overline{\text{IRQx}}$ output is asserted. If the corresponding bit in the IER is zero, the state of the bit in the ISR has no effect on the $\overline{\text{IRQx}}$ output. The IER does not mask the reading of the ISR. Note that the XTAL_RDY bit cannot be enabled to generate an interrupt. This register is equivalent to the interrupt mask register (IMR) in the MC68681. This register can only be written when the serial module is enabled.

IER $715

	7	6	5	4	3	2	1	0
	COS	DBB	RxRDYB	TxRDYB	0	DBA	RxRDYA	TxRDYA
RESET:	0	0	0	0	0	0	0	0

Write Only Supervisor/User

COS — Change of State
1 = Enable interrupt
0 = Disable interrupt

DBB — Delta Break B
1 = Enable interrupt
0 = Disable interrupt

RxRDYB — Channel B Receiver Ready or FIFO full
1 = Enable interrupt
0 = Disable interrupt

TxRDYB — Channel B Transmitter Ready
1 = Enable interrupt
0 = Disable interrupt

DBA — Delta Break A
1 = Enable interrupt
0 = Disable interrupt

RxRDYA — Channel A Receiver Ready or FIFO full
1 = Enable interrupt
0 = Disable interrupt

TxRDYA — Channel A Transmitter Ready
1 = Enable interrupt
0 = Disable interrupt

7.4.1.14 INPUT PORT (IP). The IP register enables the $\overline{\text{CTSx}}$ inputs. This register can only be read.

IP $71D

	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	CTSB	CTSA
RESET:	0	0	0	0	0	0	U	U

Read Only Supervisor/User

CTSB, CTSA — Current State

1 = The current state of the respective $\overline{\text{CTSx}}$ input is negated.

0 = The current state of the respective $\overline{\text{CTSx}}$ input is asserted.

The information contained in these bits is latched and reflects the state of the input pins at the time that the IP is read.

NOTE

These bits have the same function and value of the IPCR bits 1 and 0.

7.4.1.15 OUTPUT PORT CONTROL REGISTER (OPCR). The OPCR individually configures each bit of the 8-bit parallel OP for general-purpose use or as an auxiliary function serving the communication channels. This register can only be written.

OPCR $71D

	7	6	5	4	3	2	1	0
	OP7 TxRDYB	OP6 TxRDYA	OP5 RxRDYB	OP4 RxRDYA	OP3	OP2	OP1 RTSB	OP0 RTSA
RESET:	1	1	1	1	1	1	1	1

Write Only Supervisor/User

NOTE

OP bits 7, 5, 3, and 2 are not pinned out on the MC68340; thus changing bits 7, 5, 3, and 2 of this register has no effect.

OP6 — Output Port 6/$\overline{\text{TxRDYA}}$

1 = The $\overline{\text{TxRDYA}}$ pin functions as the transmitter-ready signal for channel A. The signal reflects the complement of the value of bit 2 of the channel A SR; thus, $\overline{\text{TxRDYA}}$ is a logic zero when the transmitter is ready.

0 = The $\overline{\text{TxRDYA}}$ pin functions as a dedicated output. The signal reflects the complement of the value of bit 6 of the OP.

OP4 — Output Port 4/RxRDYA

1 = The $\overline{\text{RxRDYA}}$ pin functions as the receiver ready or FIFO-full signal for channel A (depending on the value of bit 6 of MR1A). The signal reflects the complement of the value of ISR bit 1; thus, $\overline{\text{RxRDYA}}$ is a logic zero when the receiver is ready.

0 = The $\overline{\text{RxRDYA}}$ pin functions as a dedicated output. The signal reflects the complement of the value of bit 4 of the OP.

OP1 — Output Port 1/$\overline{\mathrm{RTSB}}$

1 = The $\overline{\mathrm{RTSB}}$ pin functions as the ready-to-send signal for channel B. The signal is asserted and negated according to the configuration programmed by MR1B bit 7 for the receiver and MR2B bit 5 for the transmitter.

0 = The $\overline{\mathrm{RTSB}}$ pin functions as a dedicated output. The signal reflects the complement of the value of bit 1 of the OP.

OP0 — Output Port 0/$\overline{\mathrm{RTSA}}$

1 = The $\overline{\mathrm{RTSA}}$ pin functions as the ready-to-send signal for channel A. The signal is asserted and negated according to the configuration programmed by MR1A bit 7 for the receiver and MR2A bit 5 for the transmitter.

0 = The $\overline{\mathrm{RTSA}}$ pin functions as a dedicated output. The signal reflects the complement of the value of bit 0 of the OP.

7.4.1.16 OUTPUT PORT DATA REGISTER (OP).

The OP contains the complements of the logic levels that are driven if any of the $\overline{\mathrm{TxRDYA}}$, $\overline{\mathrm{RxRDYA}}$, $\overline{\mathrm{RTSB}}$, and $\overline{\mathrm{RTSA}}$ outputs are configured as parallel outputs in the OPCR. The bits in this register are set by performing a bit set command (writing to offset $71E) and are cleared by performing a bit reset command (writing to offset $71F). This register can only be written when the serial module is enabled.

Bit Set

OP $71E

7	6	5	4	3	2	1	0
$\overline{\mathrm{OP7}}$	$\overline{\mathrm{OP6}}$	$\overline{\mathrm{OP5}}$	$\overline{\mathrm{OP4}}$	$\overline{\mathrm{OP3}}$	$\overline{\mathrm{OP2}}$	$\overline{\mathrm{OP1}}$	$\overline{\mathrm{OP0}}$
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

NOTE

OP bits 7, 5, 3, and 2 are not pinned out on the MC68340; thus, changing these bits has no effect.

$\overline{\mathrm{OP6}}$, $\overline{\mathrm{OP4}}$, $\overline{\mathrm{OP1}}$, $\overline{\mathrm{OP0}}$ — Output Port Parallel Outputs

1 = These bits can be set by performing a bit set command (writing a one to the bit position(s) at this address).

0 = These bits are not affected by writing a zero to this address.

Bit Reset

OP $71F

7	6	5	4	3	2	1	0
$\overline{OP7}$	$\overline{OP6}$	$\overline{OP5}$	$\overline{OP4}$	$\overline{OP3}$	$\overline{OP2}$	$\overline{OP1}$	$\overline{OP0}$
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

NOTE

OP bits 7, 5, 3, and 2 are not pinned out on the MC68340; thus, changing these bits has no effect.

$\overline{OP6}$, $\overline{OP4}$, $\overline{OP1}$, $\overline{OP0}$ — Output Port Parallel Outputs

1 = These bits can be cleared by performing a bit reset command (writing a one to the bit position(s) at this address).

0 = These bits are not affected by writing a zero to this address.

7.4.1.17 MODE REGISTER 2 (MR2). MR2 controls some of the serial module configuration. This register can be read or written at any time the serial module is enabled.

MR2A, MR2B $720, $721

7	6	5	4	3	2	1	0
CM1	CM0	TxRTS	TxCTS	SB3	SB2	SB1	SB0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor/User

CM1–CM0 — Channel Mode

These bits select a channel mode as listed in Table 7-10. See **7.3.3 Loop Modes** for more information on the individual modes.

Table 7-10. Channel Mode Bits

CM1	CM0	Mode
0	0	Normal
0	1	Automatic Echo
1	0	Local Loopback
1	1	Remote Loopback

TxRTS — Transmitter Ready to Send

This bit controls the negation of the $\overline{\text{RTSA}}$ or $\overline{\text{RTSB}}$ signals. The output is normally asserted by setting OP0 or OP1 and negated by clearing OP0 or OP1 (see **7.4.1.14 OUTPUT PORT DATA REGISTER (OP)**).

1 = In applications where the transmitter is disabled after transmission is complete, setting this bit causes the particular OP bit to be cleared automatically one bit time after the characters, if any, in the channel transmit shift register and the transmitter holding register are completely transmitted, including the programmed number of stop bits. This feature is used to automatically terminate transmission of a message. If both the receiver and the transmitter in the same channel are programmed for RTS control, RTS control is disabled for both since this is an incorrect configuration.

0 = Clearing this bit has no effect on the transmitter $\overline{\text{RTSx}}$.

TxCTS — Clear to Send

1 = The transmitter checks the state of the $\overline{\text{CTSx}}$ input each time it is ready to send a character. If $\overline{\text{CTSx}}$ is asserted, the character is transmitted. If $\overline{\text{CTSx}}$ is negated, the channel TxDx remains in the high state, and the transmission is delayed until $\overline{\text{CTSx}}$ is asserted. Changes in $\overline{\text{CTSx}}$ while a character is being transmitted do not affect transmission of that character. If both TxCTS and TxRTS are enabled, TxCTS controls the operation of the transmitter.

0 = The $\overline{\text{CTSx}}$ has no effect on the transmitter.

SB3–SB0 — Stop-Bit Length Control

These bits select the length of the stop bit appended to the transmitted character as listed in Table 7-11. Stop-bit lengths of nine-sixteenth to two bits, in increments of one-sixteenth bit, are programmable for character lengths of six, seven, and eight bits. For a character length of five bits, one and one-sixteenth to two bits are programmable in increments of one-sixteenth bit. In all cases, the receiver only checks for a high condition at the center of the first stop-bit position — i.e., one bit time after the last data bit or after the parity bit, if parity is enabled.

If an external 1 × clock is used for the transmitter, MR2 bit 3 = 0 selects one stop bit, and MR2 bit 3 = 1 selects two stop bits for transmission.

Table 7-11. Stop-Bit Length Control Bits

SB3	SB2	SB1	SB0	Length 6–8 Bits	Length 5 Bits
0	0	0	0	0.563	1.063
0	0	0	1	0.625	1.125
0	0	1	0	0.688	1.188
0	0	1	1	0.750	1.250
0	1	0	0	0.813	1.313
0	1	0	1	0.875	1.375
0	1	1	0	0.938	1.438
0	1	1	1	1.000	1.500
1	0	0	0	1.563	1.563
1	0	0	1	1.625	1.625
1	0	1	0	1.688	1.688
1	0	1	1	1.750	1.750
1	1	0	0	1.813	1.813
1	1	0	1	1.875	1.875
1	1	1	0	1.938	1.938
1	1	1	1	2.000	2.000

7.4.2 PROGRAMMING

The basic interface software required for operation of the serial module is shown in Figure 7-9. The routines are divided into three categories:

- Serial Module Initialization
- I/O Driver
- Interrupt Handling

7.4.2.1 SERIAL MODULE INITIALIZATION. The serial module initialization routines consist of SINIT and CHCHK. SINIT is called at system initialization time to check channel A and channel B operation. Before SINIT is called, the calling routine allocates two words on the system stack. Upon return to the calling routine, SINIT passes information on the system stack to reflect the status of the channels. If SINIT finds no errors in either channel A or channel B, the respective receivers and transmitters are enabled. The CHCHK routine performs the actual channel checks as called from the SINIT routine. When called, SINIT places the specified channel in the local loopback mode and checks for the following errors:

- Transmitter Never Ready
- Receiver Never Ready
- Parity Error
- Incorrect Character Received

7.4.2.2 I/O DRIVER EXAMPLE. The I/O driver routines consist of INCH, OUTCH, and POUTCH. INCH is the terminal input character routine and gets a character from the channel A receiver and places it in the lower byte of register D0. OUTCH is used to send the character in the lower byte of register D0 to the channel A transmitter. POUTCH sends the character in the lower byte of D0 to the channel B transmitter.

7.4.2.3 INTERRUPT HANDLING. The interrupt handling routine consists of SIRQ which is executed after the serial module generates an interrupt caused by a channel A change in break (beginning of a break). SIRQ then clears the interrupt source, waits for the next change-in-break interrupt (end of break), clears the interrupt source again, then returns from exception processing to the system monitor.

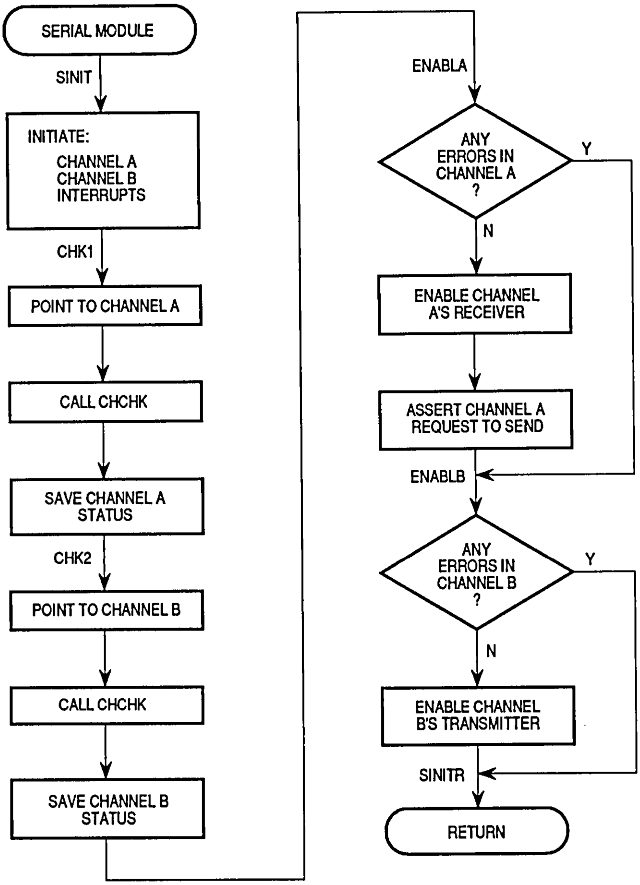


Figure 7-9. Serial Module Programming Flowchart (1 of 5)

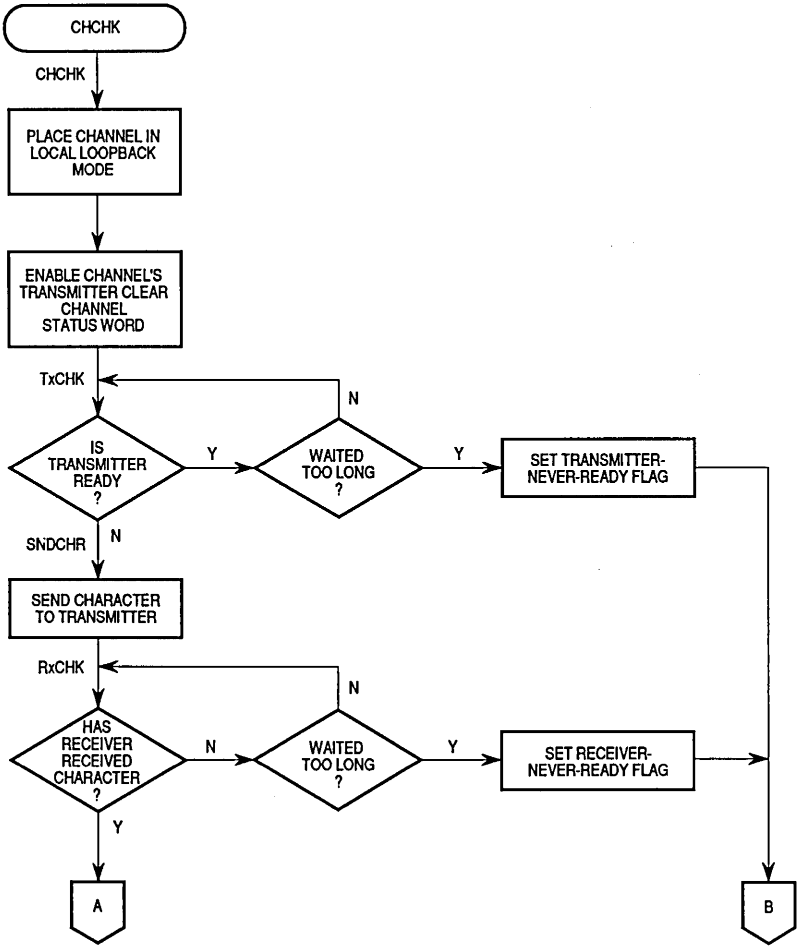

CHCHK
CHCHK
PLACE CHANNEL IN LOCAL LOOPBACK MODE
ENABLE CHANNEL'S TRANSMITTER CLEAR CHANNEL STATUS WORD
TxCHK
N
IS TRANSMITTER READY ?
Y
WAITED TOO LONG ?
Y
SET TRANSMITTER-NEVER-READY FLAG
SNDCHR
N
SEND CHARACTER TO TRANSMITTER
RxCHK
N
HAS RECEIVER RECEIVED CHARACTER ?
N
WAITED TOO LONG ?
Y
SET RECEIVER-NEVER-READY FLAG
Y
A
B

Figure 7-9. Serial Module Programming Flowchart (2 of 5)

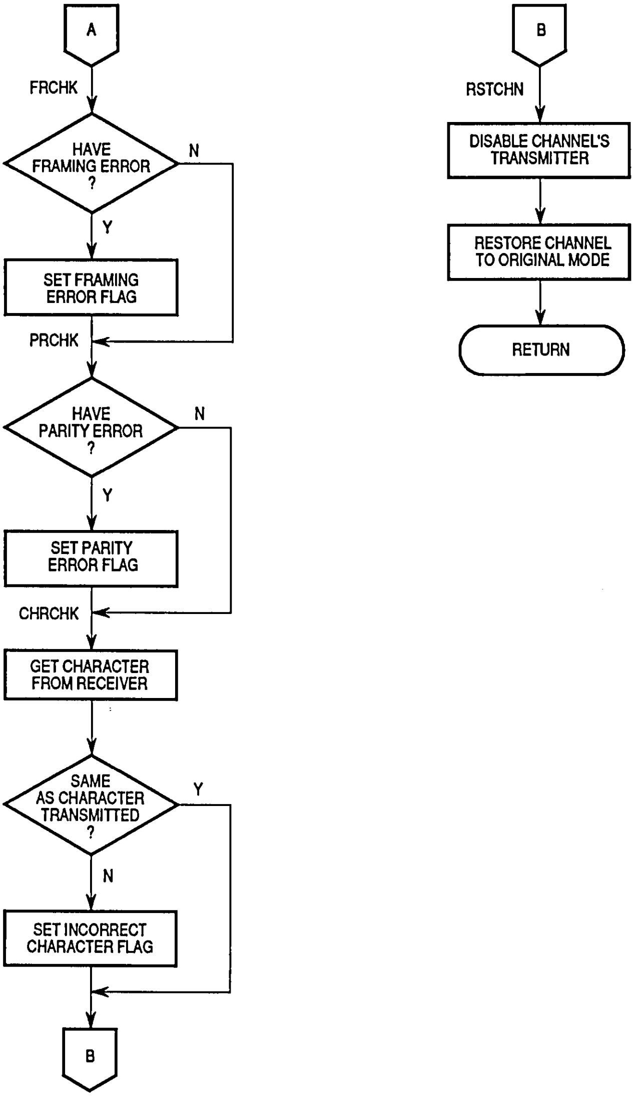

A
FRCHK
HAVE FRAMING ERROR ?
N
Y
SET FRAMING ERROR FLAG
PRCHK
HAVE PARITY ERROR ?
N
Y
SET PARITY ERROR FLAG
CHRCHK
GET CHARACTER FROM RECEIVER
SAME AS CHARACTER TRANSMITTED ?
Y
N
SET INCORRECT CHARACTER FLAG
B
B
RSTCHN
DISABLE CHANNEL'S TRANSMITTER
RESTORE CHANNEL TO ORIGINAL MODE
RETURN

Figure 7-9. Serial Module Programming Flowchart (3 of 5)

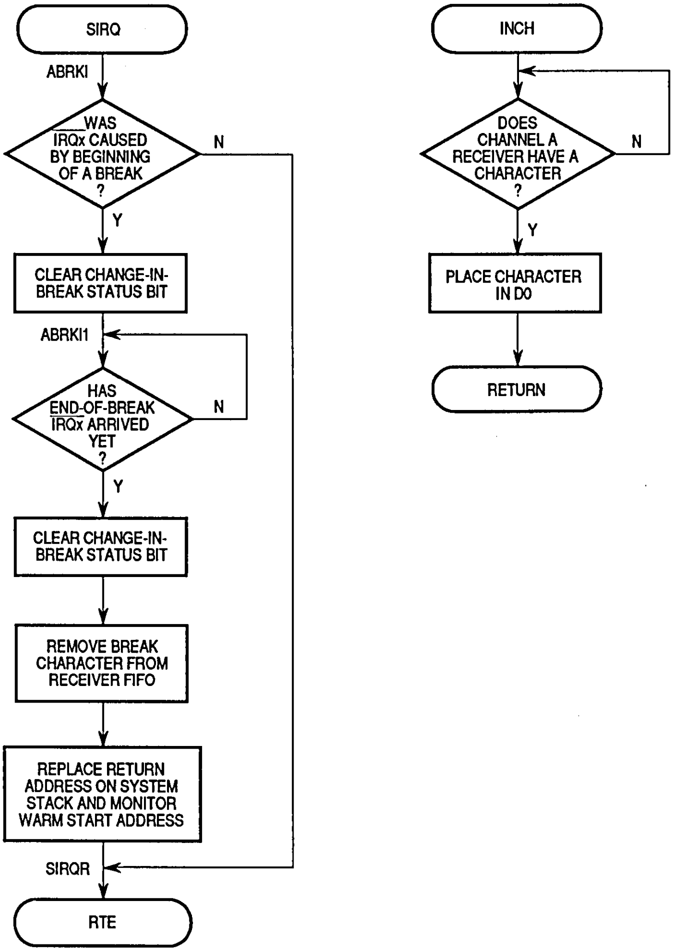

SIRQ
ABRKI
WAS IRQx CAUSED BY BEGINNING OF A BREAK ?
N
Y
CLEAR CHANGE-IN-BREAK STATUS BIT
ABRKI1
HAS END-OF-BREAK IRQx ARRIVED YET ?
N
Y
CLEAR CHANGE-IN-BREAK STATUS BIT
REMOVE BREAK CHARACTER FROM RECEIVER FIFO
REPLACE RETURN ADDRESS ON SYSTEM STACK AND MONITOR WARM START ADDRESS
SIRQR
RTE
INCH
DOES CHANNEL A RECEIVER HAVE A CHARACTER ?
N
Y
PLACE CHARACTER IN D0
RETURN

Figure 7-9. Serial Module Programming Flowchart (4 of 5)

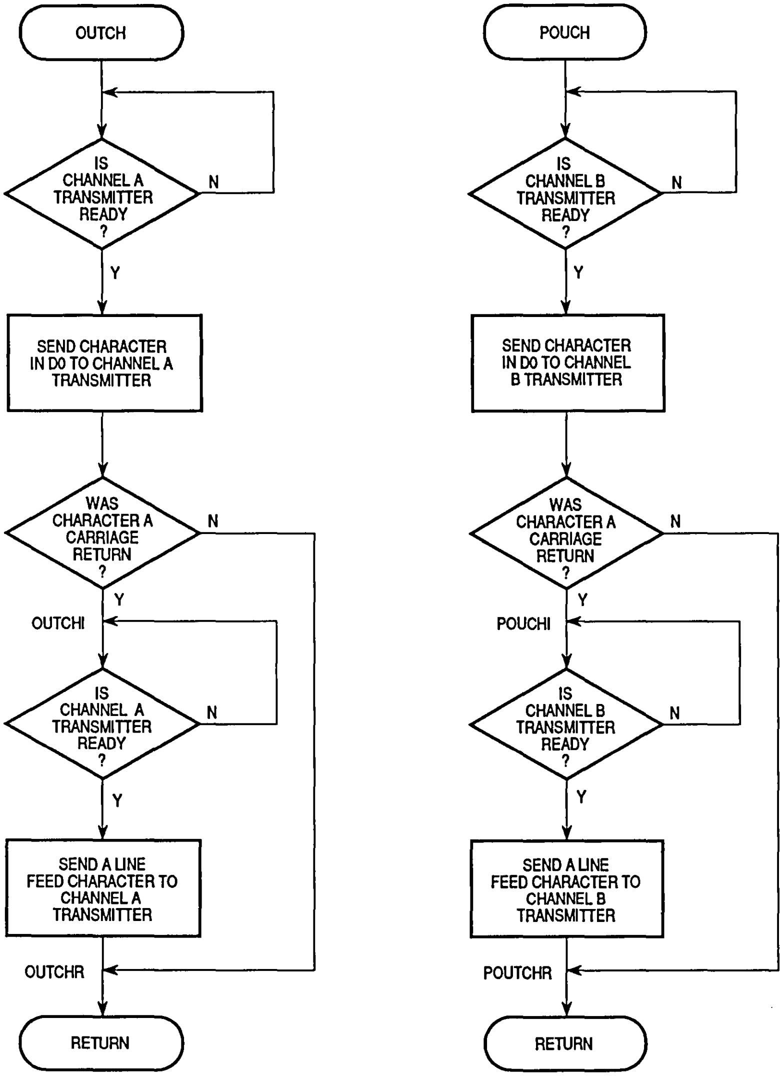

OUTCH
IS CHANNEL A TRANSMITTER READY ?
N
Y
SEND CHARACTER IN D0 TO CHANNEL A TRANSMITTER
WAS CHARACTER A CARRIAGE RETURN ?
N
Y
OUTCHI
IS CHANNEL A TRANSMITTER READY ?
N
Y
SEND A LINE FEED CHARACTER TO CHANNEL A TRANSMITTER
OUTCHR
RETURN
POUCH
IS CHANNEL B TRANSMITTER READY ?
N
Y
SEND CHARACTER IN D0 TO CHANNEL B TRANSMITTER
WAS CHARACTER A CARRIAGE RETURN ?
N
Y
POUCHI
IS CHANNEL B TRANSMITTER READY ?
N
Y
SEND A LINE FEED CHARACTER TO CHANNEL B TRANSMITTER
POUTCHR
RETURN

Figure 7-9. Serial Module Programming Flowchart (5 of 5)

7

SECTION 8
TIMER MODULES

Each MC68340 timer module contains a counter/timer (timer 1 and timer 2) as shown in Figure 8-1. Each timer interfaces directly to the CPU32 via an intermodule bus (IMB). Each timer consists of the following major areas:

- A General-Purpose Counter/Timer
- Internal Control Logic
- Interrupt Control Logic

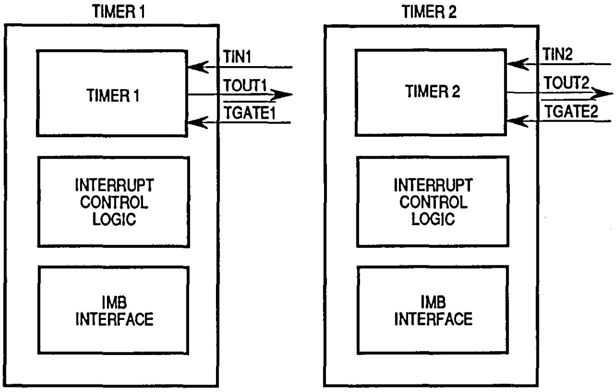


Figure 8-1. Simplified Block Diagram

8.1 MODULE OVERVIEW

Each timer module consists of the following functional features:

- Versatile General-Purpose Timer
- 8-Bit Prescaler/16-Bit Counter
- Timers Can Be Externally Cascaded for a Maximum Count Width of 48 Bits

- Programmable Timer Modes:
  - Event Counting
  - Period and Pulse-Width Measurement
  - Input Capture
  - Output Compare
  - Waveform Generation
  - Pulse Generation
- Seven Maskable Interrupt Conditions Based on Programmable Events

8.1.1 Timer and Counter Functions

The term "timer" is used to reference either timer 1 or timer 2, since the two are functionally equivalent.

The timer can perform virtually any application traditionally assigned to timers and counters. The timer can be used to generate timed events that are independent of the timing errors to which real-time programmed microprocessors are susceptible — for example, those of dynamic memory refreshing, DMA cycle steals, and interrupt servicing.

The timer has several functional areas: an 8-bit countdown prescaler, a 16-bit downcounter, timeout logic, compare logic, and clock selection logic. Figure 8-2 shows a block diagram of the timer module.

8.1.1.1 PRESCALER AND COUNTER. The counter can be driven directly by the selected clock or the prescaler output. Both the counter and prescaler are updated on the falling edge of the clock. During reset, the prescaler is set to $FF, and the counter is set to $0000. The counter is loaded with a programmed value on the first falling edge of the counter clock after the timer is enabled and again when a timeout occurs (counter reaches $0000). The prescaler and counter can be used as one 24-bit counter by enabling the prescaler and selecting the divide-by-256 prescaler output. Refer to **8.4 REGISTER DESCRIPTION** for additional information on how to program the timer.

8.1.1.2 TIMEOUT DETECTION. Timeout is achieved when all 16 stages of the counter transition to zero, a counter value of $0000. Timeout is a defined counter event which triggers specific actions depending upon the programmed mode of operation. Refer to **8.3 OPERATING MODES** for descriptions of the individual modes.

8.1.1.3 COMPARATOR. The comparator block compares the value in the 16-bit compare register (COM) with the output of the 16-bit counter. When an exact match is detected, bits in the status register (SR) are set to indicate this condition. When in the input capture/output compare mode, a match is a defined counter event that can affect the output of the timer (TOUT). Refer to **8.3.1 Input Capture/Output Compare** for additional information on this mode.

8.1.1.4 CLOCK SELECTION LOGIC. The clock selection logic consists of two multiplexers that select the clocks applied to the prescaler and counter. The first multiplexer (labeled clock logic in Figure 8-2) selects between the clock input to the timer (TIN) or one-half the frequency of the system clock (CLKOUT). This output of the first multiplexer (called selected clock) is applied to both the 8-bit prescaler and the second multiplexer. The second multiplexer selects the clock for the 16-bit counter, which is either the selected clock or the 8-bit prescaler output.

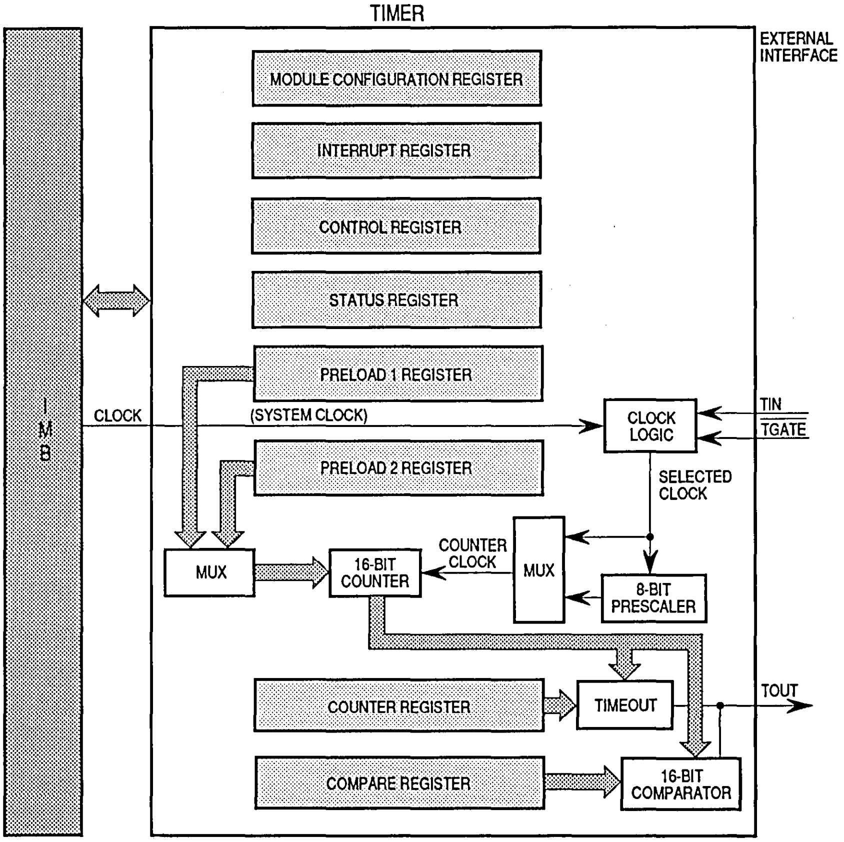


Figure 8-2. Timer Functional Diagram

8.1.2 Internal Control Logic

The timer receives operation commands on the IMB and, in turn, issues appropriate operation signals to the internal timer control logic. This mechanism allows the timer registers to be accessed and programmed. Refer to **8.4 REGISTER DESCRIPTION** for additional information.

8.1.3 Interrupt Control Logic

Each timer has seven interrupt request outputs ($\overline{\text{IRQ7}}$–$\overline{\text{IRQ1}}$) provided to notify the CPU32 that an interrupt has occurred. The interrupts are described in **8.4 REGISTER DESCRIPTION.** SR bits indicate all currently active interrupt conditions. The interrupt enable bits (IE) in the control register (CR) are programmable to mask any events that may cause an interrupt.

8.2 SIGNAL DEFINITIONS

This section contains a brief description of the timer interface signals (see Figure 8-3).

NOTE

The terms assertion and negation are used throughout this section to avoid confusion when dealing with a mixture of active-low and active-high signals. The term assert or assertion indicates that a signal is active or true independent of the level represented by a high or low voltage. The term negate or negation indicates that a signal is inactive or false.

8.2.1 Timer Input (TIN)

This input can be programmed to be the clock that causes events to occur in the counter and prescaler. TIN is internally synchronized to the system clock to guarantee that a valid TIN level is recognized. Additionally, the high and low levels of TIN must each be stable for at least one system clock period plus the sum of the setup and hold times for TIN. Refer to MC68340/D, *MC68340 Technical Summary*, for additional information.

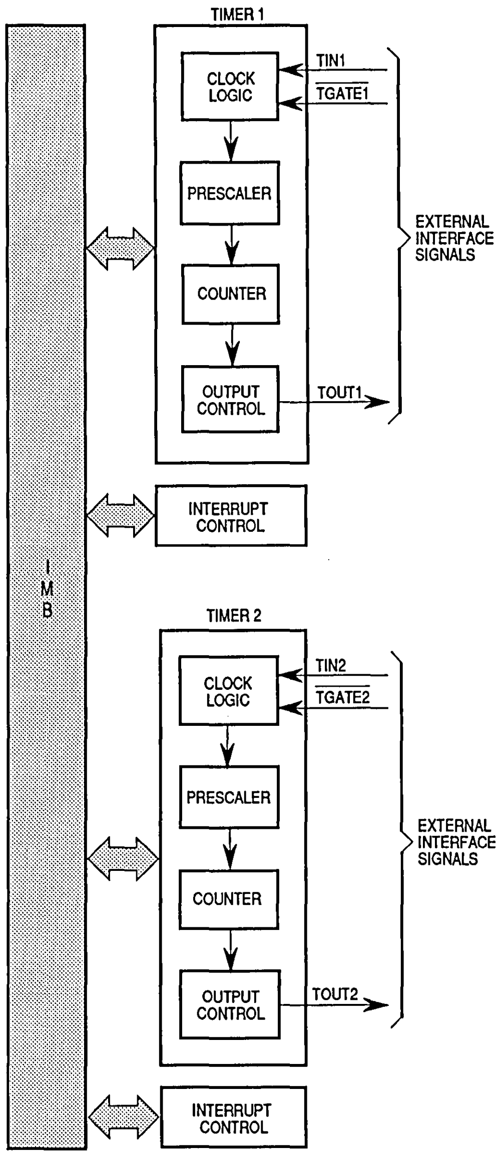


Figure 8-3. External and Internal Interface Signals

8.2.2 Timer Gate ($\overline{\text{TGATE}}$)

This active-low input can be programmed to enable and disable the counter and prescaler. $\overline{\text{TGATE}}$ may also be programmed to be a simple input. For more information on the modes of operation, refer to **8.3 OPERATING MODES**. To guarantee that the timer recognizes a valid level on $\overline{\text{TGATE}}$, the signal is synchronized with the system clock. Additionally, the high and low levels of this input must each be stable for at least one system clock period plus the sum of the setup and hold times for $\overline{\text{TGATE}}$. Refer to MC68340/D, *MC68340 Technical Summary*, for additional information.

8.2.3 Timer Output (TOUT)

This output drives the various output waveforms generated by the timer. The initial level and transitions can be programmed by the output control (OC) bits in the CR.

8.3 OPERATING MODES

The following paragraphs contain a detailed description of each timer operation mode and of the IMB operation during accesses to the timer. Changing the mode of operation should only be attempted when the timer is in reset (the software reset (SWR) bit in the CR is cleared). Changing modes while the timer is running may produce unpredictable results.

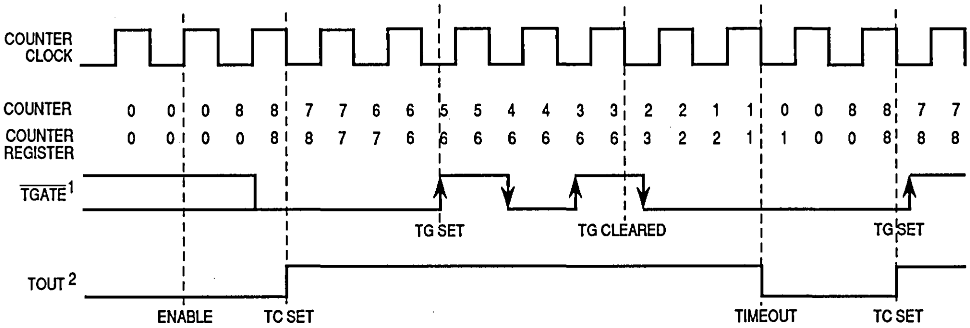
8

8.3.1 Input Capture/Output Compare

This mode has the capability of capturing a counter value by holding the value in the counter register (CNTR). Additionally, this mode can provide compare information via TOUT to indicate when the counter has reached the compare value. This mode can be used for square-wave generation, pulse-width modulation, or periodic interrupt generation. This mode can be selected by programming the operation mode bits (MODE) in the CR to 000.

The timer is enabled when the counter prescaler enable (CPE) and SWR bits in the CR are set. Once enabled, the counter enable (ON) bit in the SR is set, and the next falling edge of the counter clock causes the counter to be loaded with the value in the preload 1 register (PREL1).

The $\overline{\text{TGATE}}$ signal functions differently in this mode than it does in the other modes. $\overline{\text{TGATE}}$ does not enable or disable the counter/prescaler input clock; instead, it is used to disable shadowing. Normally, the counter is decremented on the falling edge of the counter clock, and the CNTR is updated on the next rising edge of the system clock; thus, the CNTR shadows the actual value of the counter. The timer gate interrupt (TG) bit in the SR must be cleared for shadowing to occur. $\overline{\text{TGATE}}$ is used to set the TG bit and disable shadowing. If the timing gate is enabled (TGE bit of the CR is set), the TG bit is set by the rising edge of $\overline{\text{TGATE}}$. Shadowing is disabled until the TG bit is cleared by writing a one to its location in the SR. See Figure 8-4 for a depiction of this mode. If the timing gate is disabled (TGE bit is cleared), $\overline{\text{TGATE}}$ has no effect on the operation of the timer; thus the input capture function is inoperative. At all times, the $\overline{\text{TGATE}}$ level bit (TGL) in the SR reflects the level of the $\overline{\text{TGATE}}$ signal.



Operation Mode Bits in Control Register = 000
Preload 1 Register = 8
Compare Register = 7
TGE Bit of Status Register = 1
1: TG Bit in Status Register Initially = 0
2: Output Control Bits in Control Register = 10

Figure 8-4. Input Capture/Output Compare Mode

Because it is not affected by $\overline{\text{TGATE}}$, the counter continues to decrement on the falling edge of the counter clock and load from the PREL1 at timeout, regardless of the value of $\overline{\text{TGATE}}$.

When the counter counts down to the value contained in the COM, this condition is reflected by setting the timer compare (TC) and compare (COM) bits in the SR. TOUT responds as selected by the OC bits in the CR. The output level (OUT) bit in the SR reflects the value on TOUT. Shadowing does not affect this operation.

If the counter counts down to $0000, a timeout is detected, causing the SR timeout interrupt (TO) bit to be set and the SR COM bit to be cleared. On the next falling edge of the counter clock after the timeout is detected, the value in PREL1 is again loaded into the counter. TOUT responds as selected by the CR OC bits.

A square-wave generator can be implemented by programming the CR OC bits to toggle mode. The value in the COM should be one-half the value in PREL1 to cause an event to happen twice in the countdown.

This mode can be used as a pulse-width modulator by programming the CR OC bits to zero mode or one mode. The value in the PREL1 specifies the frequency, and the COM determines the pulse width. The pulse widths can be changed by writing a new value to the COM.

Periodic interrupt generation can be accomplished by enabling the TO, TC, and/or TG bits in the SR to generate interrupts by programming the IE bits of the CR. When enabled, the programmed $\overline{\text{IRQx}}$ signal is asserted whenever the specified bits are set.

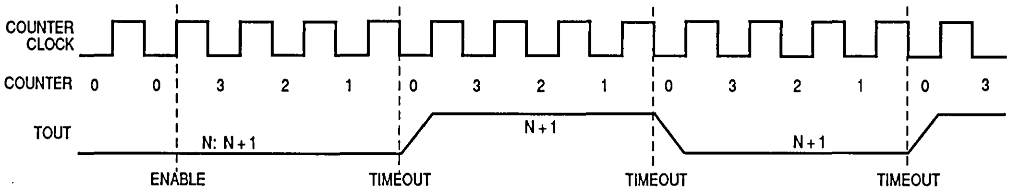
TOUT signal transitions can be controlled by writing new values into the COM. Caution must be exercised when accessing the COM. If it were to be accessed simultaneously by the compare logic and by a write, the old compare value may actually get compared to the counter value.

8.3.2 Square-Wave Generator

This mode can be used for generating both square-wave output and periodic interrupts. The square wave is generated by counting down from the value in the PREL1 to timeout (counter value of $0000). TOUT changes state on each timeout as programmed. This mode can be selected by programming the CR MODE bits to 001.

The timer is enabled by setting the SWR and CPE bits in the CR and, if $\overline{\text{TGATE}}$ is programmed to control enabling and disabling the counter (TGE bit set in the CR), then asserting $\overline{\text{TGATE}}$. When the timer is enabled, the ON bit in the SR is set. On the next falling edge of the counter clock, the counter is loaded with the value stored in the PREL1 (N). With each successive falling edge of the counter clock, the counter decrements. The time between enabling the timer and the first timeout can range from N to N+1 periods. When $\overline{\text{TGATE}}$ is used to enable the timer, the enabling of the timer is asynchronous; however, if timing is carefully considered, the time to the first timeout can be known. For additional details on timing, see MC68340/D, *MC68340 Technical Summary*.

TOUT behaves as a square wave when the OC bits of the CR are programmed for toggle mode. A timeout occurs every N+1 periods (allowing for the zero cycle), resulting in a change of state on TOUT (see Figure 8-5). The SR OUT bit reflects the level of TOUT. If this mode is used to generate periodic interrupts, TOUT may be enabled if a square wave is also desired.



Operation Mode Bits in Control Register = 001
Preload 1 Register = N = 3
Output Control Bits in Control Register = 01

Figure 8-5. Square-Wave Generator Mode

If $\overline{\text{TGATE}}$ is negated when it is enabled to control the timer (TGE=1), the prescaler and counter are disabled. Additionally, the SR TG bit is set, indicating that $\overline{\text{TGATE}}$ was negated. The SR ON bit is cleared, indicating that the timer is disabled. If $\overline{\text{TGATE}}$ is reasserted, the timer is re-enabled and begins counting from the value attained when $\overline{\text{TGATE}}$ was negated. The ON bit is set again.

If $\overline{\text{TGATE}}$ is disabled (TGE=0), $\overline{\text{TGATE}}$ has no effect on the operation of the timer. In this case, the counter begins counting on the falling edge of the counter clock immediately after the SWR and CPE bits in the CR are set. The TG bit of the SR cannot be set. At all times, TGL in the SR reflects the level of $\overline{\text{TGATE}}$.

If the counter counts down to the value stored in the COM register, then the COM and TC bits in the SR are set. The counter continues counting down to timeout. At this time, the SR TO bit is set, and the SR COM bit is cleared. The next falling edge of the counter clock after timeout causes the value in PREL1 to be loaded back into the counter, and the counter begins counting down from this value.

The period of the square-wave generator can be changed dynamically by writing a new value into the PREL1. Caution must be used because, if PREL1 is accessed simultaneously by the counting logic and a CPU32 write, the old PREL1 value may actually get loaded into the counter at timeout.

Periodic interrupt generation can be accomplished by enabling the TO, TC, and/or TG bits in the SR to generate interrupts by programming the CR IE bits. When enabled, the programmed $\overline{\text{IRQx}}$ signal is asserted whenever the specified bits are set.

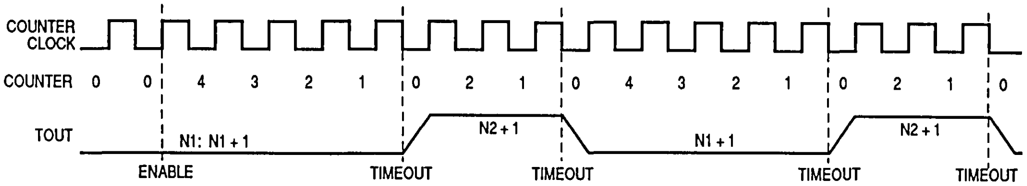
8.3.3 Variable Duty-Cycle Square-Wave Generator

In this mode, both the PREL1 and PREL2 registers are used to generate a square wave with virtually any duty cycle. The square wave is generated by counting down from the value in the PREL1 to timeout (count value $0000), then loading that value from PREL2 and again counting down to timeout. When this second timeout occurs, the value from PREL1 is loaded into the counter, and the cycle repeats. TOUT can be programmed to change state with every timeout, thus generating a variable duty-cycle square wave. This mode can be selected by programming the MODE bits in the CR to 010.

The timer is enabled by setting both the SWR and CPE bits in the CR and, if $\overline{\text{TGATE}}$ is enabled (CR TGE bit is set), then asserting $\overline{\text{TGATE}}$. When the timer is enabled, the ON bit in the SR is set. On the next falling edge of the counter clock, the counter is loaded with the value stored in the PREL1 register (N1). With each successive falling edge of the counter clock, the counter decrements. The time between enabling the timer and the first timeout can range from N1 to N1+1 periods. When $\overline{\text{TGATE}}$ is used to enable the timer, the enabling of the timer is asynchronous; however, if timing is carefully considered, the time to the first timeout can be known. For additional details on timing, see the MC68340/D, *MC68340 Technical Summary*.

If the counter counts down to the value stored in the COM register, the COM and timer compare interrupt (TC) bits in the SR are set. The counter continues counting down to timeout. At this time, the TO bit in the SR is set, and the COM bit is cleared. The next falling edge of the counter clock after timeout causes the value in PREL2 (N2) to be loaded into the counter, and the counter begins counting down from this value. Each successive timeout causes the counter to be loaded alternately with the values from PREL1 and PREL2.

TOUT behaves as a variable duty-cycle square wave when the CR OC bits are programmed for toggle mode. The second timeout occurs after N2+1 periods (allowing for the zero cycle), resulting in a change of state on TOUT. The third timeout occurs after N1+1 periods, resulting in a change of state on TOUT, and so on (see Figure 8-6). The OUT bit in the SR reflects the level of TOUT.



Operation Mode Bits in Control Register = 010
Preload 1 Register = N1 = 4
Preload 2 Register = N2 = 2
Output Control Bits in Control Register = 01

Figure 8-6. Variable Duty-Cycle Square-Wave Generator Mode

If TGATE is negated when it is enabled (TGE = 1), the prescaler and counter are disabled. Additionally, the TG bit of the SR is set, indicating that TGATE was negated. The ON bit of the SR is cleared, indicating that the timer is disabled. If TGATE is reasserted, the timer is re-enabled and begins counting from the value attained when TGATE was negated. The ON bit is set again.

If TGATE is not enabled (TGE = 0), TGATE has no effect on the operation of the timer. In this case, the counter would begin counting on the falling edge of the counter clock immediately after the SWR and CPE bits in the CR are set. The SR TG bit cannot be set. At all times, the TGL bit in the SR reflects the level of TGATE.

The duty cycle of the waveform generated on TOUT can be dynamically changed by writing new values into PREL1 and/or PREL2. If PREL1 or PREL2 is being accessed simultaneously by the counter logic and a CPU32 write, the old preload value may actually get loaded into the counter at timeout. If at timeout, the counting logic was accessing PREL2 and the CPU32 was writing to PREL1 (or visa versa), there would be no unexpected results.

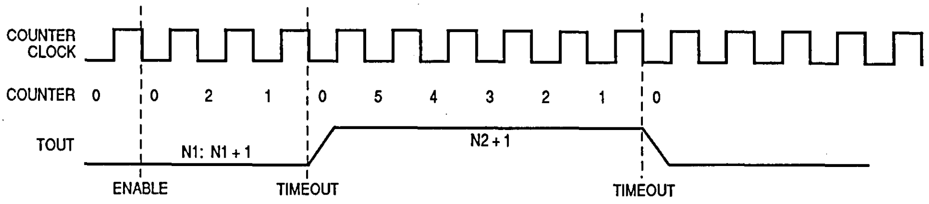
8.3.4 Variable-Width Single-Shot Pulse Generator

This mode is used to produce a one-time pulse that has a delay controlled by the value stored in PREL1 and a duration controlled by the value stored in PREL2. With TOUT programmed to change state, this sequence creates a single pulse of variable width. This mode can be selected by programming the CR MODE bits to 011.

The timer is enabled by setting both the SWR and CPE bits in the CR and, if $\overline{\text{TGATE}}$ is enabled (TGE bit in the CR is set), then asserting $\overline{\text{TGATE}}$. When the timer is enabled, the ON bit in the SR is set. On the next falling edge of the counter clock, the counter is loaded with the value stored in the PREL1 register (N1). With each successive falling edge of the counter clock, the counter decrements. The time between enabling the timer and the first timeout can range from N1 to N1+1 periods. When $\overline{\text{TGATE}}$ is used to enable the counter, the enabling of the timer is asynchronous; however, if timing is carefully considered, the time to the first timeout can be known. For additional details on timing, see MC68340/D, *MC68340 Technical Summary*.

If the counter counts down to the value stored in the COM, the COM and TC bits in the SR are set. The counter continues counting down to timeout. At this time, the SR TO bit is set and the SR COM bit is cleared. The next falling edge of the counter clock after timeout causes the value in PREL2 (N2) to be loaded into the counter, and the counter begins counting down from this value. After the second timeout, the selected clock is held high, disabling the prescaler and counter. Additionally, the SR ON and COM bits are cleared.

TOUT behaves as a variable-width pulse when the OC bits of the CR are programmed for toggle mode. TOUT is a logic zero between the time that the timer is enabled and the first timeout. When this event occurs, TOUT transitions to a logic one. The second timeout occurs after N2+1 periods (allowing for the zero cycle), resulting in TOUT returning to a logic zero (see Figure 8-7). The OUT bit in the SR reflects the level of TOUT.



Operation Mode Bits in Control Register = 011
Preload 1 Register = N1 = 2
Preload 2 Register = N2 = 5
Output Control bits in Control Register = 01

Figure 8-7. Variable-Width Single-Shot Pulse Generator Mode

If $\overline{\text{TGATE}}$ is negated when it is enabled (TGE = 1), the prescaler and counter are disabled. Additionally, the SR TG bit is set, indicating that $\overline{\text{TGATE}}$ was negated. The SR ON bit is cleared, indicating that the timer is disabled. If $\overline{\text{TGATE}}$ is reasserted, the timer is re-enabled and begins counting from the value attained when $\overline{\text{TGATE}}$ was negated. The ON bit is set again. If $\overline{\text{TGATE}}$ is not enabled (TGE = 0), $\overline{\text{TGATE}}$ has no effect on the operation of the timer. In this case, the counter would begin counting on the falling edge of the counter clock immediately after the SWR and CPE bits in the CR are set. The SR TG bit cannot be set. At all times, the TGL bit in the SR reflects the level of $\overline{\text{TGATE}}$.

The width of the pulse generated on TOUT (the value in PREL2) can be changed while the counter is counting down from the value in PREL1. Caution must be used because, if PREL2 is accessed simultaneously by the counting logic and a CPU32 write, the old PREL2 value may actually get loaded into the counter at timeout.

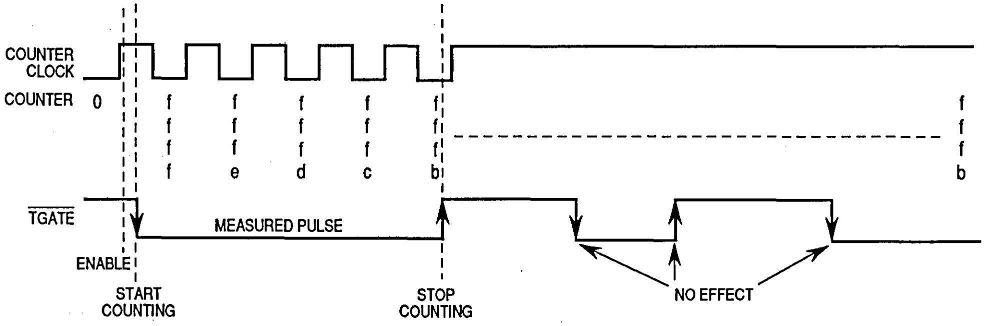
8.3.5 Pulse-Width Measurement

This mode is used to count the clock cycles during a particular event (see Figure 8-8). The event is defined by the assertion and negation of $\overline{\text{TGATE}}$. When $\overline{\text{TGATE}}$ is asserted, the counter begins counting down from \$FFFF. When $\overline{\text{TGATE}}$ is negated, the counter stops counting and holds the value at which it stopped. Further assertions and negations of $\overline{\text{TGATE}}$ have no effect on the counter. This mode can be selected by programming the CR MODE bits to 100.

The timer is enabled by setting the SWR, CPE, and TGE bits in the CR. $\overline{\text{TGATE}}$ assertion starts the counter. When the timer is enabled, the SR ON bit is set. On the next falling edge of the counter clock, the counter is loaded with the value \$FFFF. With each successive falling edge of the counter clock, the counter decrements. PREL1 and PREL2 are not used in this mode.

When $\overline{\text{TGATE}}$ is negated, the SR TG bit is set, the ON bit is negated, and the prescaler and counter are disabled. Subsequent transitions on $\overline{\text{TGATE}}$ do not re-enable the counter. The TGL bit in the SR reflects the level of $\overline{\text{TGATE}}$ at all times.

If the counter counts down to the value stored in the COM register, the COM and TC bits in the SR are set. If the counter counts down to \$0000, a timeout is detected. This sets the SR TO, and the clears the COM bit. At timeout, the next falling edge of the counter clock causes the counter to reload with \$FFFF. TOUT transitions at timeout or is disabled as programmed by the OC bits of the CR. The OUT bit in the SR reflects the level on TOUT.



Operation Mode Bits in Control Register = 100
TGE Bit of Control Register = 1

Figure 8-8. Pulse-Width Measurement Mode

To determine the number of cycles counted, the value in the CNTR must be read, inverted, and incremented by 1 (the first count is $FFFF which, in effect, includes a count of zero). The counter counts in a true 2^{16} fashion. For measuring pulses of even greater duration, the value in the prescaler output bits in the SR are readable and can be thought of as an extension of the least significant bits in the CNTR.

NOTE

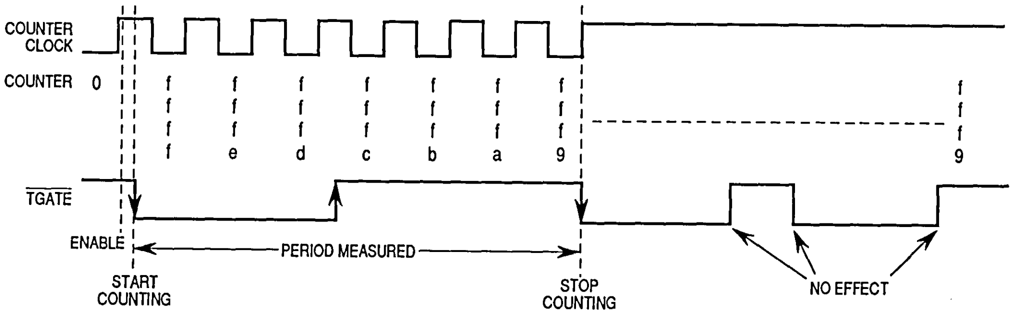
Once the timer has been enabled, do not clear the SR TG bit until the pulse has been measured and $\overline{\text{TGATE}}$ has been negated.

8.3.6 Period Measurement

This mode is used to count the period of a particular event. The event is defined by the assertion, negation, and subsequent reassertion of $\overline{\text{TGATE}}$. When $\overline{\text{TGATE}}$ is asserted, the counter begins counting down from $FFFF. The negation of $\overline{\text{TGATE}}$ has no effect on the counter. When $\overline{\text{TGATE}}$ is reasserted, the counter stops counting and holds the value at which it stopped. Further assertions and negations of $\overline{\text{TGATE}}$ have no effect on the counter. This mode can be selected by programming the CR MODE bits to 101.

The timer is enabled by setting the SWR, CPE, and the TGE bits in the CR. The assertion of $\overline{\text{TGATE}}$ starts the counter. When the timer is enabled, the SR ON bit is set. On the next falling edge of the counter clock, the counter is loaded with the value of $FFFF. With each successive falling edge of the counter clock, the counter decrements. The PREL1 and PREL2 registers are not used in this mode.

The first negation of TGATE is ignored, but on the second assertion of TGATE, the SR TG bit is set, the SR ON bit is negated, and the prescaler and counter are disabled. Subsequent transitions on TGATE do not re-enable the counter. See Figure 8-9 for a depiction of this mode. The TGL in the SR reflects the level of TGATE at all times.



Operation Mode Bits in Control Register = 101
TGE Bit of Control Register = 1

Figure 8-9. Period Measurement Mode

If the counter counts down to the value stored in the COM register, the COM and TC bits in the SR are set. If the counter counts down to $0000, a timeout is detected. This sets the SR TO bit, and clears the SR COM bit. At timeout, the next falling edge of the counter clock reloads the counter with $FFFF. TOUT transitions at timeout or is disabled as programmed by the OC bits of the CR, and the OUT bit in the SR reflects the level on TOUT.

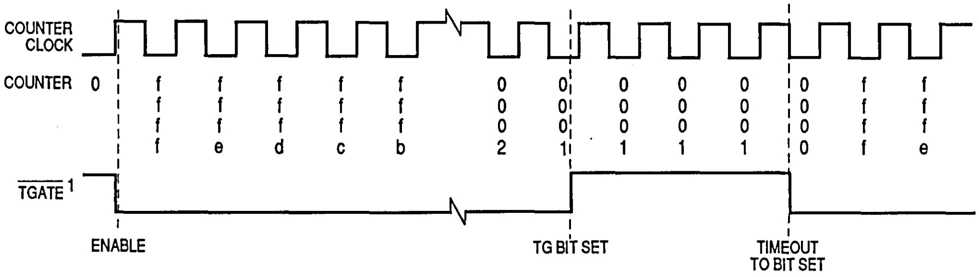
To determine the number of cycles counted, the value in the CNTR must be read, inverted, and incremented by 1 (the first count is $FFFF which, in effect, includes a count of zero). The counter counts in a true 2^{16} fashion. For measuring pulses of even greater duration, the value in the PO bits in the SR are readable and can be thought of as an extension of the least significant bits in the CNTR.

NOTE

Once the timer has been enabled, do not clear the SR TG bit until the pulse has been measured and TGATE has been negated.

8.3.7 Event Count

This mode is used to count events by interpreting the falling edges the counter clock as events (see Figure 8-10). These events may be external or internal to the chip — for example, counting the number of system clock cycles required to execute a sequence of instructions. As another example, by connecting AS to TIN, the number of bus cycles to complete a sequence of instructions could be counted. This mode can be selected by programming the CR MODE bits to 110.



Operation Mode Bits in Control Register = 110
1: TGE Bit of the Control Register Set

Figure 8-10. Event Count Mode

The timer is enabled by setting the SWR and CPE bits in the CR and, if TGATE is enabled (TGE bit of the CR is set), then asserting TGATE. When the timer is enabled, the SR ON bit is set. On the next falling edge of the counter clock, the counter is loaded with the value of $FFFF. With each successive falling edge of the counter clock, the counter decrements. The PREL1 and PREL2 registers are not used in this mode.

If TGATE is not enabled (CR TGE bit is cleared), then TGATE does not start or stop the timer or affect the TG bit of the SR. In this case, the counter would begin counting on the falling edge of the counter clock immediately after the SWR and CPE bits in the CR are set.

If TGATE is enabled (CR TGE bit is set), then the assertion of TGATE starts the counter. The negation of TGATE disables the counter, sets the SR TG bit, and clears the ON bit in the SR. If TGATE is reasserted, the timer resumes counting from where it was stopped, and the ON bit is set again. Further assertions and negations of TGATE have the same effect. The TGL bit in the SR reflects the level of TGATE at all times.

If the counter counts down to the value stored in the COM register, the COM and TC bits in the SR are set. If the counter counts down to $0000, a timeout is detected. This event sets the TO in the SR and clears the COM bit. At timeout, the next falling edge of the counter clock reloads the counter with $FFFF. TOUT transitions at timeout or is disabled as programmed by the CR OC bits. The SR OUT bit reflects the level on TOUT.

To determine the number of cycles counted, the value in the CNTR must be read, inverted, and incremented by 1 (the first count is $FFFF which, in effect, includes a count of zero). The counter counts in a true 2^{16} fashion. For measuring pulses of even greater duration, the value in the PO bits in the SR are readable, and can be thought of as an extension of the least significant bits in the CNTR.

8.3.8 Timer Bypass

In this mode, the counter and prescaler cannot be enabled. However TGATE and TOUT can be used for I/O. This mode can be selected by programming the CR MODE bits to 111.

TGATE can be used as a simple input port when the CR is configured as follows:

CR

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SWR	IE2	IE1	IE0	TGE	PSE	CPE	CLK	POT2	POT1	POT0	MODE2	MODE1	MODE0	OC1	OC0
TGATE AS A SIMPLE INPUT:	X	X	0	X	X	X	1	X	X	X	X	1	1	1	X	X

X = Don't care

When TGATE is asserted, the SR ON bit is set. When TGATE is negated, the ON bit is cleared. The value of the TGL bit in the SR reflects the level of TGATE. TGATE can also be used as an input port that generates interrupts on a low-to-high transition of TGATE when the CR is configured as follows:

CR

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SWR	IE2	IE1	IE0	TGE	PSE	CPE	CLK	POT2	POT1	POT0	MODE2	MODE1	MODE0	OC1	OC0
TGATE AS A INPUT/INTERRUPT:	X	X	1	X	1	X	1	X	X	X	X	1	1	1	X	X

When TGATE is negated, the SR TG bit is set, and the programmed IRQx signal is asserted to the CPU32. The TG bit can only be cleared by writing a one to this bit position. The value of the SR TGL bit reflects the level of TGATE.

Additionally, TOUT can be used as a simple output port when the CR is configured as follows:

CR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWR	IE2	IE1	IE0	TGE	PSE	CPE	CLK	POT2	POT1	POT0	MODE2	MODE1	MODE0	OC1	OC0

$\overline{\text{TGATE}}$ AS A SIMPLE OUTPUT

0	X	X	X	X	X	1	X	X	X	X	1	1	1	OC1	OC0

SWR must be a zero to change the value of TOUT. Changing the value of the CR OC bits determines the level of TOUT as shown in Table 8-1.

Table 8-1. OC Encoding

OC1	OC0	TOUT
0	0	Hi-Z
0	1	0
1	0	0
1	1	1

A read of the SR while in this mode always shows the TO, TC, and COM bits cleared, and the PO bits as $FF. The SR OUT bit always indicates the level on the TOUT pin.

8.3.9 Bus Operation

The following paragraphs describe the operation of the IMB during read, write, and interrupt acknowledge cycles to the timer.

8.3.9.1 READ CYCLES. The timer is accessed with no wait states. The timer responds to byte, word, and long-word reads, and 16 bits of valid data are returned. Read cycles from reserved registers return logic zero.

8.3.9.2 WRITE CYCLES. The timer is accessed with no wait states. The timer responds to byte, word, and long-word writes. Write cycles to read-only registers and bits as well as reserved registers complete in a normal manner without exception processing; however, the data is ignored.

8.3.9.3 INTERRUPT ACKNOWLEDGE CYCLES. The timer is capable of arbitrating for interrupt servicing and supplying the interrupt vector when it has successfully won arbitration. The vector number must be provided if interrupt servicing is necessary; thus, the interrupt register (IR) must be initialized. If the IR is not initialized, a spurious interrupt exception will be taken if interrupt servicing is necessary.

8.4 REGISTER DESCRIPTION

The following paragraphs contain a detailed description of each register and its specific function. The operation of the timer is controlled by writing control words into the appropriate registers. Timer registers and their associated addresses are listed in Figure 8-11. For more information about a particular register, refer to the individual register description. The ADDR column indicates the offset of the register from the base address of the timer. An FC column designation of S indicates that register access is restricted to supervisor only. A designation of S/U indicates that access is governed by the SUPV bit in the module configuration register (MCR).

TIMER 1	TIMER 2	FC	15 — 0
$600	$640	S	MODULE CONFIGURATION REGISTER (MCR)
$602	$642	S	RESERVED
$604	$644	S	INTERRUPT REGISTER (IR)
$606	$646	S/U	CONTROL REGISTER (CR)
$608	$648	S/U	STATUS/PRESCALER REGISTER (SR)
$60A	$64A	S/U	COUNTER REGISTER (CNTR)
$60C	$64C	S/U	PRELOAD 1 REGISTER (PREL1)
$60E	$60E	S/U	PRELOAD 2 REGISTER (PREL2)
$610	$650	S/U	COMPARE REGISTER (COM)
$612–$63F	$652–$67F	S/U	RESERVED

Figure 8-11. Programming Model

In the registers discussed in the following paragraphs, the numbers in the upper right-hand corner indicate the offset of the register from the base address specified by the base address register in the SIM. The first number is the offset for timer 1; the second number is the offset for timer 2. The numbers on the top line of the register represent the bit position in the register. The second line contains the mnemonic for the bit. The value of these bits after reset is shown below the register. The access privilege is shown in the lower right-hand corner.

8.4.1 Module Configuration Register (MCR)

The MCR controls the timer configuration. The register can be either read or written when in supervisor state.

MCR $600, $640

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	STP	FRZ1	FRZ0	0	0	0	0	0	SUPV	0	0	0	IARB3	IARB2	IARB1	IARB0
RESET:	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Supervisor Only

STP — Stop Mode

1 = Stops all clocks within the timer except for the clock from the IMB. The clocks are stopped on the low phase of the clock and will remain stopped until this bit is cleared. The clock from the IMB remains active to allow the CPU32 to access the MCR. Accesses to other timer registers produce a bus error while in stop mode. The timer should be disabled (in a known state) prior to setting the STP bit; otherwise, unpredictable results may occur.

0 = The timer operates in normal mode.

FRZ1,FRZ0 — Freeze

These bits determine the action taken when the FREEZE signal is asserted on the IMB. Table 8-2 lists the action taken for each bit combination.

Table 8-2. FRZ Encoding

FRZ1	FRZ0	ACTION
0	0	Ignore FREEZE
0	1	Reserved (FREEZE ignored)
1	0	Execution Freeze
1	1	Execution Freeze

SUPV — Supervisor/User

1 = The timer registers defined as supervisor/user reside in supervisor data space and are only accessible from supervisor programs.

0 = The timer registers defined as supervisor/user reside in user data space and are accessible from either supervisor or user programs.

The value of this bit has no effect on registers permanently defined as supervisor-only access.

IARB3–IARB0 — Interrupt Arbitration Bits

Each module that generates interrupts has an IARB field. The value of the IARB field allows arbitration during an interrupt acknowledge (IACK) cycle among modules that simultaneously generate the same interrupt level. No two modules can share the same IARB value. (Timer 1 and timer 2 should be programmed with different values if both are used.) The reset value of IARB is $0, which prevents this module from arbitrating during the IACK cycle. The system software should initialize the IARB field to a value from $F (highest priority) to $1 (lowest priority).

8.4.2 Interrupt Register (IR)

The IR contains the priority level for the timer interrupt request and the 8-bit vector number of the interrupt. The register can be read or written to at any time while in supervisor mode when the timer is enabled (STP bit in the MCR = 0).

IR $604, $644

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	IL2	IL1	IL0	IVR7	IVR6	IVR5	IVR4	IVR3	IVR2	IVR1	IVR0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Supervisor Only

IL2–IL0 — Interrupt Level Bits

Each module that can generate interrupts has an interrupt level field. The priority level encoded in these bits is sent to the CPU32 on the appropriate $\overline{\text{IRQx}}$ signal. The CPU32 uses this value to determine servicing priority. See **SECTION 5 CPU32** for more information.

IVR7–IVR0 — Interrupt Vector Bits

Each module that can generate interrupts has an interrupt vector register (IVR). This 8-bit number indicates the offset from the base of the vector table where the address of the exception handler for the specified interrupt is located. The IVR is reset to $0F, which indicates an uninitialized interrupt condition. See **SECTION 5 CPU32** for more information.

8.4.3 Control Register (CR)

The CR controls the operation of the timer. The register can be always read or written when the timer is enabled (STP bit in the MCR = 0). Writing to the CR is discouraged when the timer is in enabled (SWR bit = 1); however it is allowed so that the programmer can have total control of timer operation.

NOTE

Changing the mode of operation while the timer is running may produce unpredictable results.

CR $606, $646

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SWR	IE2	IE1	IE0	TGE	PSE	CPE	CLK	POT2	POT1	POT0	MODE2	MODE1	MODE0	OC1	OC0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Supervisor/User

SWR — Software Reset

1 = Removes the software reset.

0 = A software reset is performed by first clearing this bit and then clearing the TC, TG, and TO bits in the SR. The prescaler is loaded with $FF, the counter is set to $0000, and the SR COM bit is cleared. When this bit is zero, the timer is disabled.

IE2–IE0 — Interrupt Enable

These bits determine which sources of interrupts, TO, TG, and TC, are enabled to generate an interrupt request to the CPU32. Table 8-3 lists which interrupts are enabled for all bit combinations.

Table 8-3. IE Encoding

IE2	IE1	IE0	Enabled Interrupts
0	0	0	Polling Mode (No Interrupts Enabled)
0	0	1	TC Enabled
0	1	0	TG Enabled
0	1	1	TG and TC Enabled
1	0	0	TO Enabled
1	0	1	TO and TC Enabled
1	1	0	TO and TG Enabled
1	1	1	TO, TG, and TC Enabled

TGE — Timing Gate Enable

1 = The $\overline{\text{TGATE}}$ signal is enabled to control the enabling and disabling of the prescaler and counter, except in the input capture/output compare mode (see **8.3.1 Input Capture/Output Compare**).

0 = The $\overline{\text{TGATE}}$ signal has no effect on timer operation.

PSE — Prescaler Select Enable

This bit selects which clock is used for the counter clock.

1 = The counter is decremented by the prescaler output tap as selected by the POT field in the CR.

0 = The counter is decremented by the selected clock.

The prescaler continues to decrement regardless of how PSE is set.

CPE — Counter Prescaler Enable

1 = The selected clock is enabled. If the TGE bit is set, then $\overline{\text{TGATE}}$ must also be asserted (except in the input capture/output compare mode).

0 = The selected clock is held high, halting the prescaler and counter.

CLK — Clock

1 = The selected clock is taken from the TIN input.

0 = The selected clock is one-half the system clock's frequency.

The TOUT of one timer can be fed externally into the TIN input of the other timer, resulting in a 32-bit counter if the prescalers are not used and a 48-bit counter if they are used.

POT2–POT0 — Prescaler Output Tap

If PSE is set, these bits encode which of the prescaler's output taps act as the counter clock. A division of the selected clock is applied to the counter as listed in Table 8-4.

Table 8-4. POT Encoding

POT2	POT1	POT0	Division of Selected Clock
0	0	1	Divide by 2
0	1	0	Divide by 4
0	1	1	Divide by 8
1	0	0	Divide by 16
1	0	1	Divide by 32
1	1	0	Divide by 64
1	1	1	Divide by 128
0	0	0	Divide by 256

MODE2–MODE0 — Operation Mode

These bits select one of the eight modes of operation for the timer as listed in Table 8-5. Refer to **8.3 OPERATING MODES** for more information on the individual modes.

Table 8-5. MODE Encoding

MODE2	MODE1	MODE0	OPERATION MODE
0	0	0	Input Capture/Output Compare
0	0	1	Square-Wave Generator
0	1	0	Variable Duty-Cycle Square-Wave Generator
0	1	1	Variable-Width Single-Shot Pulse Generator
1	0	0	Pulse-Width Measurement
1	0	1	Period Measurement
1	1	0	Event Count
1	1	1	Timer Bypass (Simple Test Mode)

OC1–OC0 — Output Control

These bits select the conditions under which TOUT changes (see Table 8-6). These bits may have a different effect when in the input capture/output compare mode. Caution should be used when modifying the OC bits near timer events.

Table 8-6. OC Encoding

OC1	OC0	TOUT MODE
0	0	Disabled
0	1	Toggle Mode
1	0	Zero Mode
1	1	One Mode

Disabled — TOUT is disabled and three-stated.

Toggle Mode — If the timer is disabled (SWR=0) when this encoding is programmed, TOUT is immediately set to zero. If the timer is enabled (SWR=1), timeout events (counter reaches $0000) toggle TOUT. In the input capture/output compare mode, TOUT is immediately set to zero if the timer is disabled (SWR=0). If the timer is enabled (SWR=1), timer compare events toggle TOUT. (Timer compare events occur when the counter reaches the value stored in the COM.)

Zero Mode — If the timer is disabled (SWR=0) when this encoding is programmed, TOUT is immediately set to zero. If the timer is enabled (SWR=1), TOUT will be set to zero at the next timeout. In the input capture/output compare mode, TOUT is immediately set to zero if the timer is disabled

(SWR=0). If the timer is enabled (SWR=1), TOUT will be set to zero at timeouts and set to one at timer compare events. If the COM is $0000, TOUT will be set to zero at the timeout/timer compare event.

One Mode — If the timer is disabled (SWR=0) when this encoding is programmed, TOUT is immediately set to one. If the timer is enabled (SWR=1), TOUT will be set to one at the next timeout. In the input capture/output compare mode, TOUT is immediately set to one if the timer is disabled (SWR=0). If the timer is enabled (SWR=1), TOUT will be set to one at timeouts and set to zero at timer compare events. If the COM is $0000, TOUT will be set to one at the timeout/timer compare event.

8.4.4 Status Register (SR)

The SR contains timer status information as well as the state of the prescaler. This register is updated on the rising edge of the system clock when a read of its location is not in progress, allowing the most current information to be contained in this register. The register can be read, and the TO, TG, and TC bits can be written when the timer is enabled
(MCR STP bit=0).

SR $608, $648

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ	TO	TG	TC	TGL	ON	OUT	COM	P07	P06	P05	P04	P03	P02	P01	P00
RESET ($\overline{\text{TGATE}}$ NEGATED):															
0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1
RESET ($\overline{\text{TGATE}}$ ASSERTED):															
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Supervisor/User

Interrupt Request

The positioning of this bit in the most significant location in this register allows it it be conditionally tested as if it were a signed binary integer.

1=An interrupt condition has occurred. This bit is the logical OR of the enabled TO, TG, and TC interrupt bits.

0=The bit(s) that caused the interrupt condition has been cleared. If an $\overline{\text{IRQx}}$ signal has been asserted, it is negated when this bit is cleared.

TO — Timeout Interrupt

1 = The counter has transitioned from $0001 to $0000, and the counter has rolled over. This bit does not affect the programmed $\overline{\text{IRQx}}$ signal if the IE2 bit in the CR is cleared.

0 = This bit is cleared by the timer whenever the $\overline{\text{RESET}}$ signal is asserted on the IMB, regardless of the mode of operation. This bit may also be cleared by writing a one to it. Writing a zero to this bit does not alter its contents. This bit is not affected by disabling the timer (SWR cleared).

TG — Timer Gate Interrupt

1 = This bit is set whenever the CR TGE bit is set and the $\overline{\text{TGATE}}$ signal transitions in the manner to which the particular mode of operation responds. Refer to **8.3 OPERATING MODES** for more details. This bit does not affect the programmed $\overline{\text{IRQx}}$ signal if the IE1 bit in the CR is cleared.

0 = This bit is cleared by the timer whenever the $\overline{\text{RESET}}$ signal is asserted on the IMB, regardless of the mode of operation. This bit may also be cleared by writing a one to it. Writing a zero to this bit does not alter its contents. This bit is not affected by disabling the timer (SWR cleared).

TC — Timer Compare Interrupt

1 = This bit is set when the counter transitions (off a clock/event falling edge) to the value in the COM. This bit does not affect the programmed $\overline{\text{IRQx}}$ signal if the IE0 bit in the CR is cleared.

0 = This bit is cleared by the timer whenever the $\overline{\text{RESET}}$ signal is asserted on the IMB, regardless of the mode of operation. This bit may also be cleared by writing a one to it. Writing a zero to this bit does not alter its contents. This bit is not affected by disabling the timer (SWR cleared).

TGL — $\overline{\text{TGATE}}$ Level

1 = The $\overline{\text{TGATE}}$ signal is negated.

0 = The $\overline{\text{TGATE}}$ signal is asserted.

ON — Counter Enabled

1 = This bit is set whenever the SWR and CPE bits are set in the CR. If the CR TGE bit is set, $\overline{\text{TGATE}}$ must also be asserted (except in the input capture/output compare mode), since this signal then controls the enabling and disabling of the counter. If all these conditions are met, the counter is enabled and begins counting down.

0 = The counter is not enabled and does not begin counting down.

OUT — Output Level

1 = TOUT is a logic one.

0 = TOUT is a logic zero, or the pin is three-stated.

COM — Compare Bit

This bit is used to indicate when the counter output value is at or between the value in the COM and $0000 (timeout), inclusive.

1 = This bit is set when the counter output equals the value in the COM.

0 = This bit is cleared when a timeout occurs, the COM register is accessed (read or write), the timer is reset with the SWR bit, or the $\overline{\text{RESET}}$ signal is asserted on the IMB. This bit is cleared regardless of the state of the TC bit.

This bit can be used to indicate when a write to the PREL1 or PREL2 registers will not cause a problem during a counter reload at timeout. To ensure that the write to the PREL register is recognized at timeout, the latency between the read of the COM bit and the write to the PREL register must be taken into account.

PO7–PO0 — Prescaler Output

These bits show the levels on each of the eight output taps of the prescaler. These values are updated every time that the system clock goes high and a read cycle of this byte in the SR is not in progress.

8.4.5 Counter Register (CNTR)

The CNTR reflects the value of the counter. This value can be reliably read at any time since it is updated on every rising edge of the system clock (except in the input capture/output compare mode) when a read of the register is not in progress. This read-only register can be read when the timer is enabled (STP bit of the MCR =0).

CNTR $60A, $64A

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Supervisor/User

All 24 bits of the prescaler and the counter may be obtained by one long-word read at the address of the SR, since the CNTR is contiguous to it. Any changes in the prescaler value due to the two cycles necessary to perform a long-word read should be considered. If this latency presents a problem, the $\overline{\text{TGATE}}$ signal may be used to disable the decrement function while the reads are occurring.

8.4.6 Preload 1 Register (PREL1)

The PREL1 stores a value that is loaded into the counter in some modes of operation. This value is loaded into the counter on the first falling edge of the counter clock after the counter is enabled. This register can be be read and written when the timer is enabled (STP bit of the MCR=0). However, a write to this register must be completed before timeout for the new value to be reliably loaded into the counter.

PREL1 $60C, $64C

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR1-15	PR1-14	PR1-13	PR1-12	PR1-11	PR1-10	PR1-9	PR1-8	PR1-7	PR1-6	PR1-5	PR1-4	PR1-3	PR1-2	PR1-1	PR1-0
RESET:															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Supervisor/User

For some modes of operation, this register is also used to reload the counter one falling clock edge after a timeout occurs. Refer to **8.3 OPERATING MODES** for more information on the individual modes.

8.4.7 Preload 2 Register (PREL2)

PREL2 is used in addition to PREL1 in the variable duty-cycle square-wave generator and variable-width single-shot pulse generator modes. When in either of these modes, the value in PREL1 is loaded into the counter on the first falling edge of the counter clock after the counter is enabled. After timeout, the value in PREL2 is loaded into the counter. This register can be be read and written when the timer is enabled (STP bit in the MCR=0). However, a write to this register must be completed before timeout for the new value to be reliably loaded into the counter.

PREL2 $60E, $64E

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR2-15	PR2-14	PR2-13	PR2-12	PR2-11	PR2-10	PR2-9	PR2-8	PR2-7	PR2-6	PR2-5	PR2-4	PR2-3	PR2-2	PR2-1	PR2-0
RESET:															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Supervisor/User

8.4.8 Compare Register (COM)

The COM can be used in any mode. When the 16-bit counter reaches the value in the COM, the TC and COM bits in the SR are set. In the input capture/output compare mode, a compare event can be programmed to set, clear, or toggle TOUT. The register can be be read and written when the timer is enabled (STP bit in the MCR=0).

COM $610, $650

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COM15	COM14	COM13	COM12	COM11	COM10	COM9	COM8	COM7	COM6	COM5	COM4	COM3	COM2	COM1	COM0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Supervisor/User

The COM can be used to produce an interrupt when the SR TC bit has been enabled to produce an interrupt and the counter counts down to a preselected value. The COM can also be used to indicate that the timer is approaching timeout.

Caution must be exercised when accessing the COM. If it were to be accessed simultaneously by the compare logic and by a write, the old compare value may get compared to the counter value.

8

SECTION 9
IEEE 1149.1 TEST ACCESS PORT

The MC68340 includes dedicated user-accessible test logic that is fully compatible with the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to development of this proposed standard under the sponsorship of the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The MC68340 implementation supports circuit-board test strategies based on this standard.

The test logic includes a test access port (TAP) consisting of four dedicated signal pins, a 16-state controller, and two test data registers. A boundary scan register links all device signal pins into a single shift register. The test logic, implemented utilizing static logic design, is independent of the device system logic. The MC68340 implementation provides the following capabilities:

a. Perform boundary scan operations to test circuit-board electrical continuity

b. Bypass the MC68340 for a given circuit-board test by effectively reducing the boundary scan register to a single cell

c. Sample the MC68340 system pins during operation and transparently shift out the result in the boundary scan register

d. Disable the output drive to pins during circuit-board testing

NOTE

Certain precautions must be observed to ensure that the IEEE 1149.1 test logic does not interfere with nontest operation. See **9.4 NON-IEEE 1149.1 OPERATION** for details.

9.1 OVERVIEW

This section, which includes aspects of the IEEE 1149.1 implementation that are specific to the MC68340, is intended to be used with the supporting IEEE 1149.1 document. The discussion includes those items required by the standard to be defined and, in certain cases, provides additional information specific to the MC68340 implementation. For internal details and applications of the standard, refer to the IEEE 1149.1 document.

An overview of the MC68340 implementation of IEEE 1149.1 is shown in Figure 9-1. The MC68340 implementation includes a 3-bit instruction register and two test registers, a 1-bit bypass register and a 133-bit boundary scan register. This implementation includes a dedicated TAP consisting of the following signals:

TCK — a test clock input to synchronize the test logic

TMS — a test mode select input (with an internal pullup resistor) that is sampled on the rising edge of TCK to sequence the test controller's state machine

TDI — a test data input (with an internal pullup resistor) that is sampled on the rising edge of TCK

TDO — a three-stateable test data output that is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK.

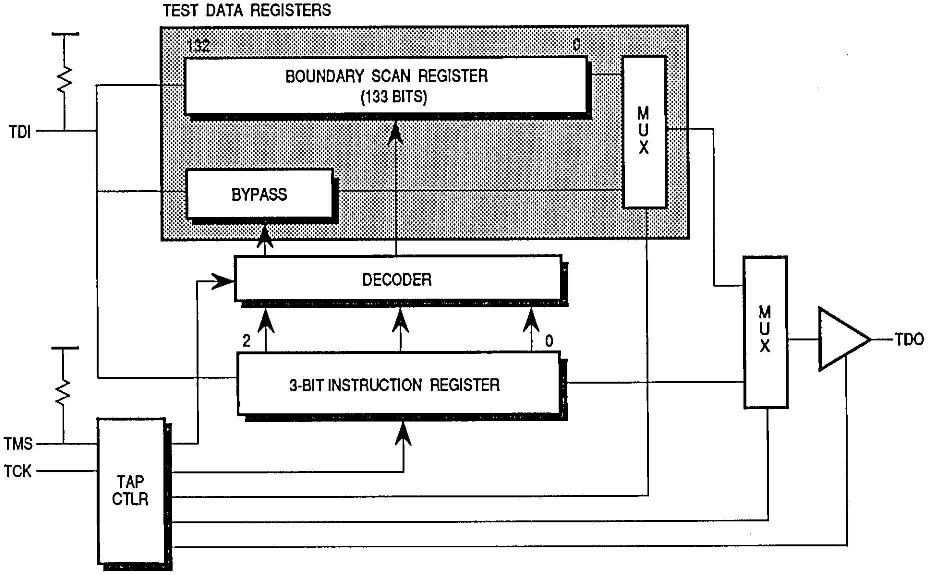


Figure 9-1. Test Logic Block Diagram

9.2 INSTRUCTION REGISTER

The MC68340 IEEE 1149.1 implementation includes the three mandatory public instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) but does not support any of the optional public instructions defined by IEEE 1149.1. One additional public instruction (HI-Z) provides the capability for disabling all device output drivers. The MC68340 includes a 3-bit instruction register without parity consisting of a shift register with three parallel outputs. Data is transferred from the shift register to the parallel outputs during the update-IR controller state. The three bits are used to decode the four unique instructions shown in Table 9-1.

The parallel output of the instruction register is reset to all ones in the test-logic-reset controller state. Note that this preset state is equivalent to the BYPASS instruction.

Table 9-1. Instructions

Code			Instruction
B2	B1	B0	
0	0	0	EXTEST
0	0	1	SAMPLE/PRELOAD
X	1	X	BYPASS
1	0	0	HI-Z
1	0	1	BYPASS

During the capture-IR controller state, the parallel inputs to the instruction shift register are loaded with the 2-bit binary value into the two least significant bits and the loss-of-crystal (LOC) status signal into bit 2. The parallel outputs, however, remain unchanged by this action since an update-IR signal is required to modify them.

The LOC status bit of the instruction register indicates whether an internal clock is detected when operating with a crystal clock source. The LOC bit is clear when a clock is detected and set when it is not. The LOC bit is always clear when an external clock is used. The LOC bit can be used to detect faulty connectivity when a crystal is used to clock the device.

9.2.1 EXTEST (000)

The external test (EXTEST) instruction selects the 133-bit boundary scan register, including cells for all device signal and clock pins and associated control signals. The XTAL, X2, and XFC pins are associated with analog signals and are not included in the boundary scan register. EXTEST also asserts internal reset for the MC68340 system logic to force a predictable benign internal state while performing external boundary scan operations.

By using the TAP, the register is capable of a) scanning user-defined values into the output buffers, b) capturing values presented to input pins, c) controlling the direction of bidirectional pins, and d) controlling the output drive of three-stateable output pins.

All MC68340 bidirectional pins, except the open-drain I/O pins ($\overline{\text{DONE1}}$, $\overline{\text{DONE2}}$, $\overline{\text{HALT}}$, and $\overline{\text{RESET}}$), have a single register bit in the boundary scan register for pin data. All bidirectional pins, except $\overline{\text{HALT}}$ and $\overline{\text{RESET}}$, have an associated control bit in the boundary scan register. $\overline{\text{DONE1}}$ and $\overline{\text{DONE2}}$ have two control bits in the boundary scan register: one for on (logic zero), and one for off (high impedance). To ensure proper operation, these open-drain pins require external pullups. Twenty-four bits in the boundary scan register define the output enable signal for associated groups of bidirectional and three-stateable pins.

Boundary scan bit definitions are shown in Table 9-2. The first column in Table 9-2 defines the bit's ordinal position in the boundary scan register. The shift register cell nearest TDO (i.e., first to be shifted out) is defined as bit zero; the last bit to be shifted out is 132.

The second column references one of the five MC68340 cell types depicted in Figures 9-2–9-6, which describe the cell structure for that bit.

The third column lists the pin name for all pin-related cells or defines the name of bidirectional control register bits. The 24 control bits and their bit positions are as follows:

1.	tout2.ctl	(29)	13.	db.ctl	(86)
2.	irq7.ctl	(52)	14.	ab24.ctl	(88)
3.	irq6.ctl	(54)	15.	ab25.ctl	(90)
4.	irq5.ctl	(56)	16.	ab26.ctl	(92)
5.	cs3.ctl	(58)	17.	ab27.ctl	(94)
6.	irq3.ctl	(60)	18.	ab28.ctl	(96)
7.	cs2.ctl	(62)	19.	ab29.ctl	(98)
8.	cs1.ctl	(64)	20.	ab30.ctl	(100)
9.	cs0.ctl	(66)	21.	ab31.ctl	(102)
10.	dma.ctl	(83)	22.	pb0.ctl	(123)
11.	ab.ctl	(84)	23.	ifetch.ctl	(126)
12.	berr.ctl	(85)	24.	tout1.ctl	(131)

The active level of the control bits (i.e., output driver on) is defined by the last digit of the cell type listed for each control bit. For example, the active-high level for tout2.ctl (bit 29) is logic one since the cell type is IO.Ctl1. The active level for irq7.ctl (bit 52) is logic zero since the cell type is IO.Ctl0. IO.Ctl0 (see Figure 9-5) differs from IO.Ctl1 (see Figure 9-4) by an inverter in the output enable path.

The fourth column lists the pin type for convenience where TS-Output indicates a three-stateable output pin, I/O indicates a bidirectional pin, and OD-I/O denotes an open-drain bidirectional pin. An open-drain output pin has two states: off (high impedance) and logic zero.

The last column indicates the associated boundary scan register control bit for bidirectional, three-state, and open-drain output pins.

Bidirectional pins include a single scan cell for data (IO.Cell) as depicted in Figure 9-6. These cells are controlled by one of the two cells shown in Figures 9-4 and 9-5. One or more bidirectional data cells can be serially connected to a control cell as shown in Figure 9-7. Note that, when sampling the bidirectional data cells, the cell data can be interpreted only after examining the IO control cell to determine pin direction.

Table 9-2. Boundary Scan Bit Definitions

BIT #	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
0	IO.Cell	FC3	I/O*	ab.ctl
1	IO.Cell	FC2	I/O*	ab.ctl
2	IO.Cell	FC1	I/O*	ab.ctl
3	IO.Cell	FC0	I/O*	ab.ctl
4	IO.Cell	A23	I/O*	ab.ctl
5	IO.Cell	A22	I/O*	ab.ctl
6	IO.Cell	A21	I/O*	ab.ctl
7	IO.Cell	A20	I/O*	ab.ctl
8	IO.Cell	A19	I/O*	ab.ctl
9	IO.Cell	A18	I/O*	ab.ctl
10	IO.Cell	A17	I/O*	ab.ctl
11	IO.Cell	A16	I/O*	ab.ctl
12	IO.Cell	A15	I/O*	ab.ctl
13	IO.Cell	A14	I/O*	ab.ctl
14	IO.Cell	A13	I/O*	ab.ctl
15	IO.Cell	A12	I/O*	ab.ctl
16	IO.Cell	A11	I/O*	ab.ctl
17	IO.Cell	A10	I/O*	ab.ctl
18	IO.Cell	A9	I/O*	ab.ctl
19	IO.Cell	A8	I/O*	ab.ctl
20	IO.Cell	A7	I/O*	ab.ctl
21	IO.Cell	A6	I/O*	ab.ctl
22	IO.Cell	A5	I/O*	ab.ctl
23	IO.Cell	A4	I/O*	ab.ctl
24	IO.Cell	A3	I/O*	ab.ctl
25	IO.Cell	A2	I/O*	ab.ctl
26	IO.Cell	A1	I/O*	ab.ctl
27	I.Pin	$\overline{\text{TGATE2}}$	Input	—
28	O.Latch	TOUT2	TS-Output	tout2.ctl
29	IO.Ctl1	tout2.ctl	—	—
30	I.Pin	TIN2	Input	—
31	I.Pin	RxD1	Input	—
32	O.Latch	TxD1	Output	—
33	O.Latch	$\overline{\text{RTS1}}$	Output	—
34	I.Pin	$\overline{\text{CTS1}}$	Input	
35	O.Latch	$\overline{\text{RxRDY1}}$	Output	—

Table 9-2. Boundary Scan Bit Definitions (Continued)

BIT #	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
36	O.Latch	$\overline{\mathrm{TxRDY1}}$	Output	—
37	I.Pin	RxD2	Input	
38	O.Latch	TxD2	Output	—
39	O.Latch	$\overline{\mathrm{RTS2}}$	Output	—
40	I.Pin	$\overline{\mathrm{CTS2}}$	Input	—
41	I.Pin	SCLK	Input	—
42	I.Pin	X1	Input	—
43	I.Pin	$\overline{\mathrm{DREQ1}}$	Input	—
44	O.Latch	$\overline{\mathrm{DACK1}}$	TS-Output	dma.ctl
45	O.Latch	$\overline{\mathrm{DONE1}}$	OD-I/O	dma.ctl
46	I.Pin	$\overline{\mathrm{DONE1}}$	OD-I/O	—
47	I.Pin	$\overline{\mathrm{DREQ2}}$	Input	—
48	O.Latch	$\overline{\mathrm{DACK2}}$	—	dma.ctl
49	O.Latch	$\overline{\mathrm{DONE2}}$	OD-I/O	dma.ctl
50	I.Pin	$\overline{\mathrm{DONE2}}$	OD-I/O	—
51	IO.Cell	$\overline{\mathrm{IRQ7}}$	I/O	irq7.ctl
52	IO.Ctl0	irq7.ctl	—	—
53	IO.Cell	$\overline{\mathrm{IRQ6}}$	I/O	irq6.ctl
54	IO.Ctl0	irq6.ctl	—	—
55	IO.Cell	$\overline{\mathrm{IRQ5}}$	I/O	irq5.ctl
56	IO.Ctl0	irq5.ctl	—	—
57	IO.Cell	$\overline{\mathrm{CS3}}$	I/O	cs3.ctl
58	IO.Ctl0	cs3.ctl	—	—
59	IO.Cell	$\overline{\mathrm{IRQ3}}$	I/O	irq3.ctl
60	IO.Ctl0	irq3.ctl	—	—
61	IO.Cell	$\overline{\mathrm{CS2}}$	I/O	cs2.ctl
62	IO.Ctl0	cs2.ctl	—	—
63	IO.Cell	$\overline{\mathrm{CS1}}$	I/O	cs1.ctl
64	IO.Ctl0	cs1.ctl	—	—
65	IO.Cell	$\overline{\mathrm{CS0}}$	I/O	cs0.ctl
66	IO.Ctl0	cs0.ctl	—	—
67	IO.Cell	D0	I/O	db.ctl
68	IO.Cell	D1	I/O	db.ctl
69	IO.Cell	D2	I/O	db.ctl
70	IO.Cell	D3	I/O	db.ctl
71	IO.Cell	D4	I/O	db.ctl
72	IO.Cell	D5	I/O	db.ctl
73	IO.Cell	D6	I/O	db.ctl
74	IO.Cell	D7	I/O	db.ctl
75	IO.Cell	D8	I/O	db.ctl
76	IO.Cell	D9	I/O	db.ctl
77	IO.Cell	D10	I/O	db.ctl
78	IO.Cell	D11	I/O	db.ctl
79	IO.Cell	D12	I/O	db.ctl
80	IO.Cell	D13	I/O	db.ctl
81	IO.Cell	D14	I/O	db.ctl
82	IO.Cell	D15	I/O	db.ctl
83	IO.Ctl1	dma.ctl	—	—
84	IO.Ctl1	ab.ctl	—	—
85	IO.Ctl0	berr.ctl	—	—
86	IO.Ctl1	db.ctl	—	—
87	IO.Cell	A24	I/O	ab24.ctl
88	IO.Ctl0	ab24.ctl	—	—
89	IO.Cell	A25	I/O	ab25.ctl
90	IO.Ctl0	ab25.ctl	—	—
91	IO.Cell	A26	I/O	ab26.ctl
92	IO.Ctl0	ab26.ctl	—	—
93	IO.Cell	A27	I/O	ab27.ctl
94	IO.Ctl0	ab27.ctl	—	—
95	IO.Cell	A28	I/O	ab28.ctl
96	IO.Ctl0	ab28.ctl	—	—
97	IO.Cell	A29	I/O	ab29.ctl
98	IO.Ctl0	ab29.ctl	—	—
99	IO.Cell	A30	I/O	ab30.ctl
100	IO.Ctl0	ab30.ctl	—	—
101	IO.Cell	A31	I/O	ab31.ctl
102	IO.Ctl0	ab31.ctl	—	—
103	IO.Cell	A0	I/O*	ab.ctl
104	IO.Ctl0	$\overline{\mathrm{DSACK0}}$	I/O**	berr.ctl
105	IO.Cell	$\overline{\mathrm{DSACK1}}$	I/O**	berr.ctl
106	IO.Cell	$\overline{\mathrm{RMC}}$	I/O	ab.ctl
107	IO.Cell	R/$\overline{\mathrm{W}}$	I/O*	ab.ctl

Table 9-2. Boundary Scan Bit Definitions (Continued)

BIT #	CELL TYPE	PIN/CELL NAME	PIN TYPE	OUTPUT CTL CELL
108	IO.Cell	SIZ1	I/O*	ab.ctl
109	IO.Cell	SIZ0	I/O*	ab.ctl
110	IO.Cell	$\overline{\text{DS}}$	I/O*	ab.ctl
111	IO.Cell	$\overline{\text{AS}}$	I/O*	ab.ctl
112	I.Pin	$\overline{\text{BGACK}}$	Input	
113	O.Latch	$\overline{\text{BG}}$	Output	
114	I.Pin	$\overline{\text{BR}}$	Input	
115	IO.Cell	$\overline{\text{BERR}}$	I/O**	berr.ctl
116	O.Latch	$\overline{\text{HALT}}$	OD-I/O	—
117	I.Pin	$\overline{\text{HALT}}$	OD-I/O	—
118	O.Latch	$\overline{\text{RESET}}$	OD-I/O	—
119	I.Pin	$\overline{\text{RESET}}$	OD-I/O	—
120	O.Latch	CLKOUT	Output	—
121	I.Pin	EXTAL	Input	—
122	IO.Cell	MODCK	I/O	pb0.ctl
123	IO.Ctl0	pb0.ctl	—	—
124	O.Latch	$\overline{\text{IPIPE}}$	Output	—
125	IO.Cell	$\overline{\text{IFETCH}}$	I/O	ifetch.ctl
126	IO.Ctl0	ifetch.ctl	—	—
127	I.Pin	$\overline{\text{BKPT}}$	Input	—
128	O.Latch	FREEZE	Output	—
129	I.Pin	TIN1	Input	—
130	O.Latch	TOUT1	TS-Output	tout1.ctl
131	IO.Ctl1	tout1.ctl	—	—
132	O.Latch	$\overline{\text{TGATE1}}$	Input	—

NOTES:
The noted pins are implemented differently than defined in the signal definition description:
*Input during Motorola factory test
**Output during Motorola factory test

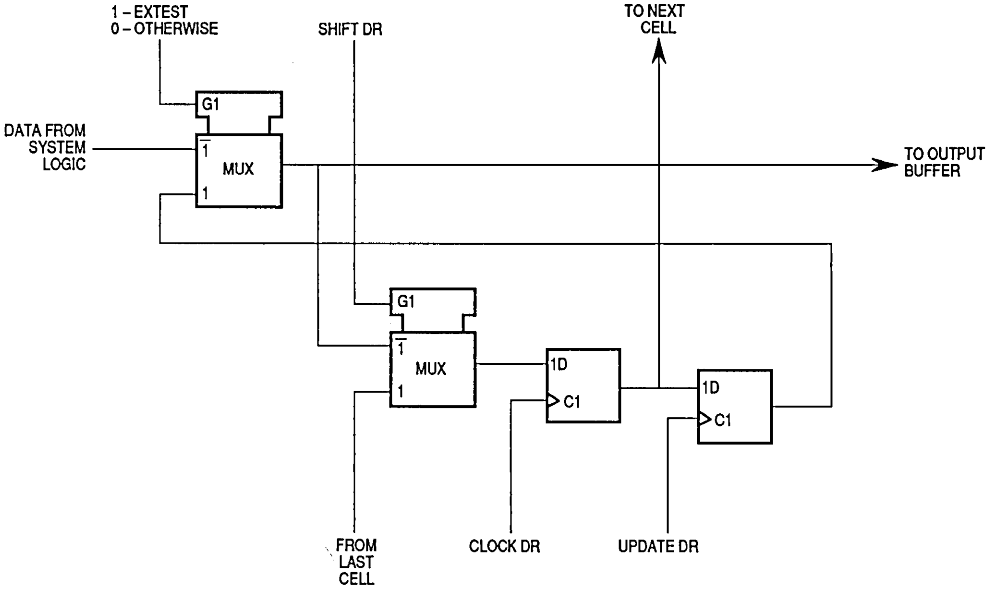


Figure 9-2. Output Latch Cell (O.Latch)

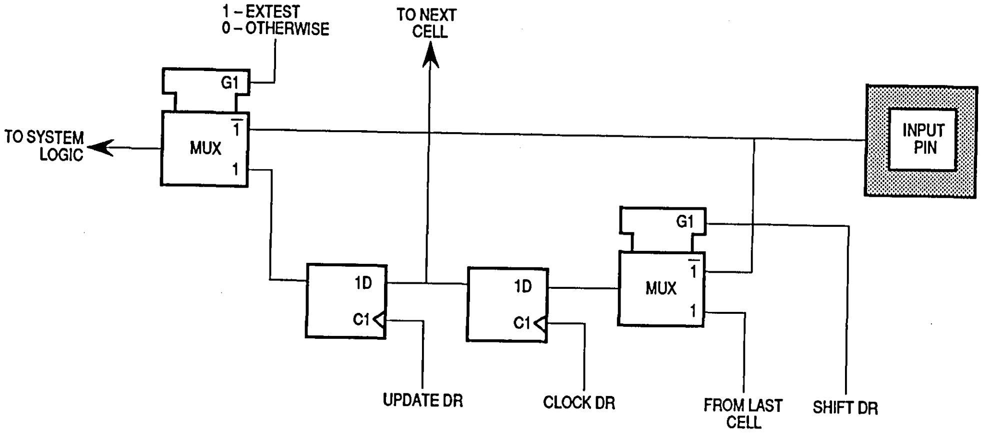


Figure 9-3. Input Pin Cell (I.Pin)

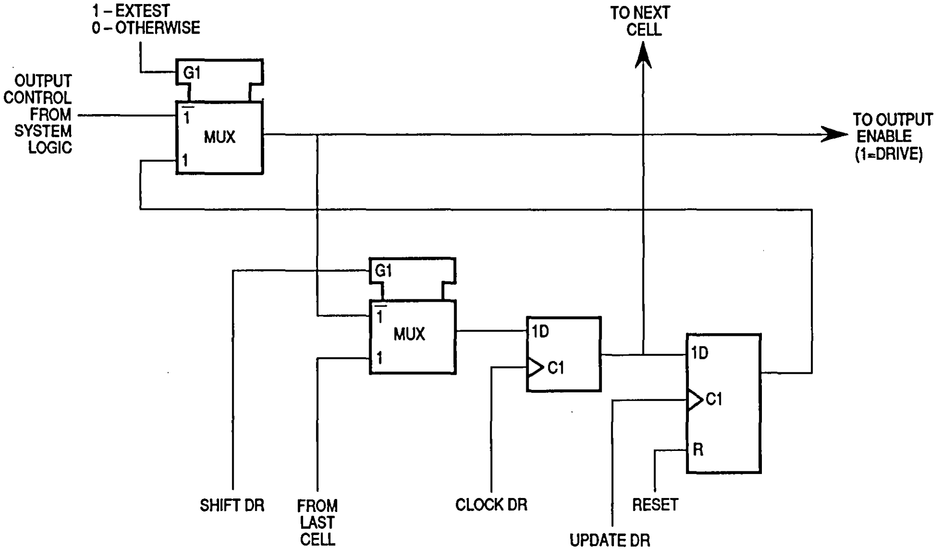


Figure 9-4. Active-High Output Control Cell (IO.Ctl1)

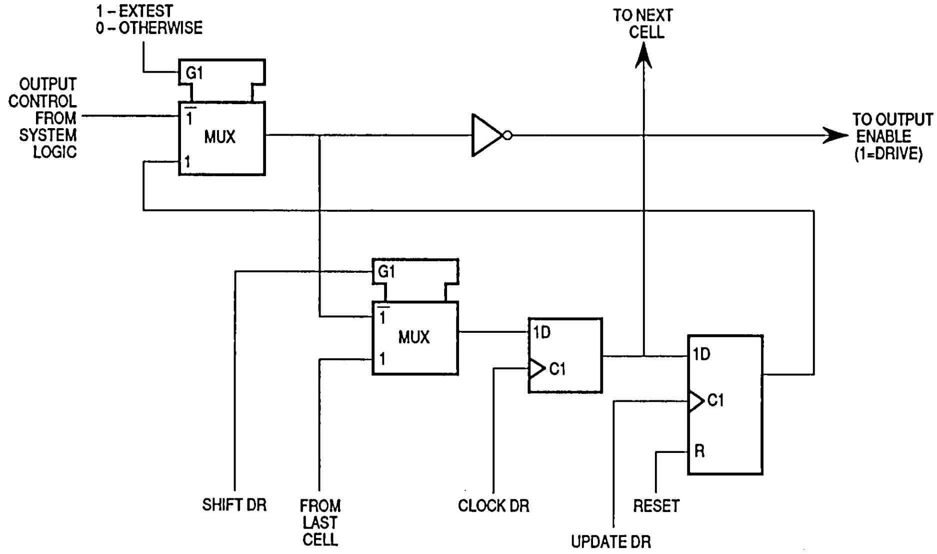


Figure 9-5. Active-Low Output Control Cell (IO.Ctl0)

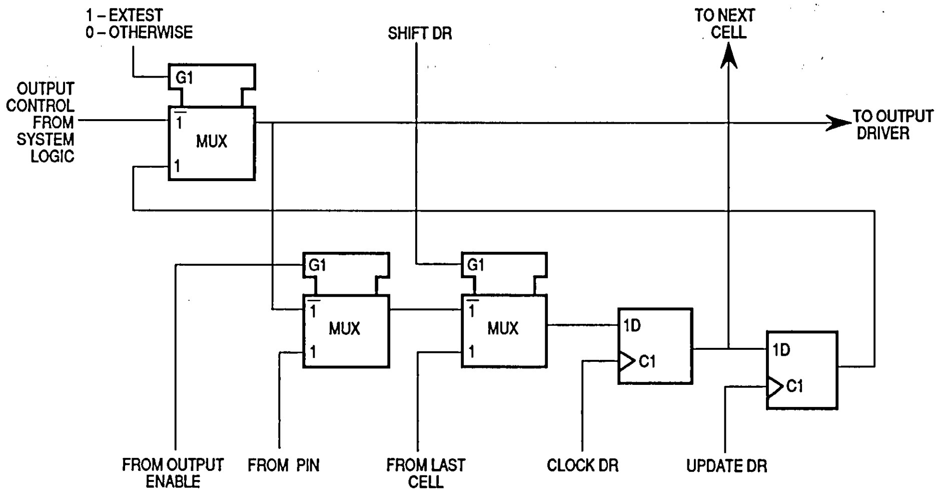
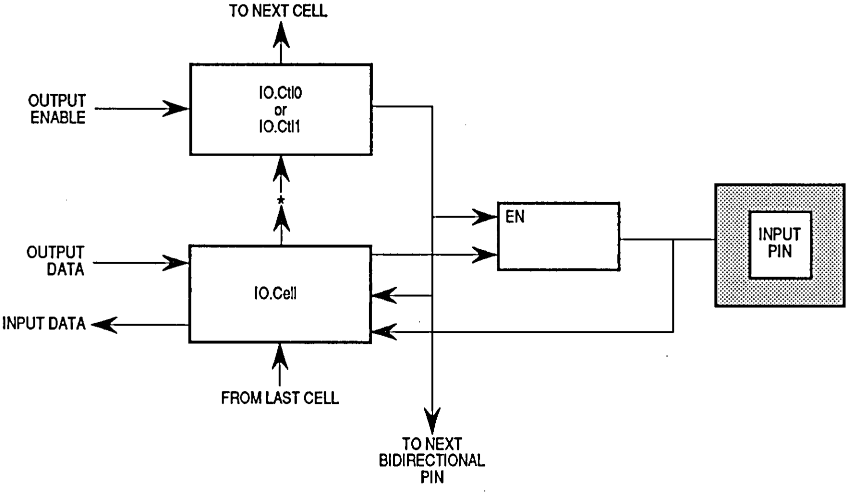


Figure 9-6. Bidirectional Data Cell (IO.Cell)



NOTE:
More than one IO.Cell could be serially connected and controlled by a single IO.Ctlx cell.

Figure 9-7. General Arrangement for Bidirectional Pins

9.2.2 BYPASS (X1X, 101)

The BYPASS instruction selects the single-bit bypass register as shown in Figure 9-8. This creates a shift-register path from TDI to the bypass register and, finally, to TDO, circumventing the 133-bit boundary scan register. This instruction is used to enhance test efficiency when a component other than the MC68340 becomes the device under test.

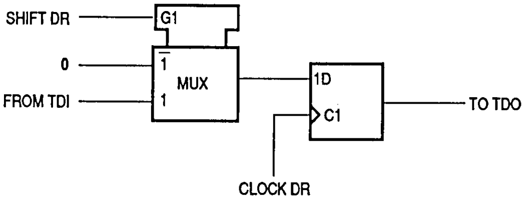


Figure 9-8. Bypass Register

When the bypass register is selected by the current instruction, the shift-register stage is set to a logic zero on the rising edge of TCK in the capture-DR controller state. Therefore, the first bit to be shifted out after selecting the bypass register will always be a logic zero.

9.2.3 SAMPLE/PRELOAD (001)

The SAMPLE/PRELOAD instruction provides two separate functions. First, it provides a means to obtain a snapshot of system data and control signals. The snapshot occurs on the rising edge of TCK in the capture-DR controller state. The data can be observed by shifting it transparently through the boundary scan register.

NOTE

Since there is no internal synchronization between the IEEE 1149.1 clock (TCK) and the system clock (CLKOUT), the user must provide some form of external synchronization to achieve meaningful results.

The second function of SAMPLE/PRELOAD is to initialize the boundary scan register output cells prior to selection of EXTEST. This initialization ensures that known data will appear on the outputs when entering the EXTEST instruction.

9.2.4 HI-Z (100)

The HI-Z instruction is not included in the IEEE 1149.1 standard. It is provided as a manufacturer's optional public instruction to prevent having to backdrive the output pins during circuit-board testing. When HI-Z is invoked, all output drivers, including the two-state drivers, are turned off (i.e., high impedance). The instruction selects the bypass register.

9.3 MC68340 RESTRICTIONS

The control afforded by the output enable signals using the boundary scan register and the EXTEST instruction requires a compatible circuit-board test environment to avoid device-destructive configurations. The user must avoid situations in which the MC68340 output drivers are enabled into actively driven networks.

The MC68340 includes on-chip circuitry to detect the initial application of power to the device. Power-on reset (POR), the output of this circuitry, is used to reset both the system and IEEE 1149.1 logic. The purpose for applying POR to the IEEE 1149.1 circuitry is to avoid the possibility of bus contention during power-on. The time required to complete device power-on is power-supply dependent. However, the IEEE 1149.1 TAP controller remains in the test-logic-reset state while POR is asserted. The TAP controller does not respond to user commands until POR is negated.

The MC68340 features a low-power stop mode, which is invoked using a CPU instruction called LPSTOP. The interaction of the IEEE 1149.1 interface with low-power stop mode is as follows:

1. The TAP controller must be in the test-logic-reset state to either enter or remain in the low-power stop mode. Leaving the TAP controller test-logic-reset state negates the ability to achieve low-power, but does not otherwise affect device functionality.
2. The TCK input is not blocked in low-power stop mode. To consume minimal power, the TCK input should be externally connected to V_{CC} or ground.
3. The TMS and TDI pins include on-chip pullup resistors. In low-power stop mode, these two pins should remain either unconnected or connected to V_{CC} to achieve minimal power consumption.

9.4 NON-IEEE 1149.1 OPERATION

In non-IEEE 1149.1 operation, there are two constraints. First, the TCK input does not include an internal pullup resistor and should not be left unconnected to preclude mid-level inputs. The second constraint is to ensure that the IEEE 1149.1 test logic is kept transparent to the system logic by forcing TAP into the test-logic-reset controller state, using either of two methods. During power-up, POR forces the TAP controller into this state. After power-up is concluded, TMS must be sampled as a logic one for five consecutive TCK rising edges. If TMS either remains unconnected or is connected to V_{CC}, then the TAP controller cannot leave the test-logic-reset state, regardless of the state of TCK.

SECTION 10
APPLICATIONS

This section provides guidelines for using the MC68340. Included in this section are a discussion of the requirements for a minimum system configuration, sample initialization sequences for system startup, and interfacing to memory.

10.1 MINIMUM SYSTEM CONFIGURATION

One of the powerful features of the MC68340 is the small number of external components needed to create an entire system. The information contained in the following paragraphs details a simple high-performance MC68340 system (see Figure 10-1). This system configuration features the following hardware:

- Processor Clock Circuitry
- Reset Circuitry
- SRAM Interface
- ROM Interface
- Serial Interface

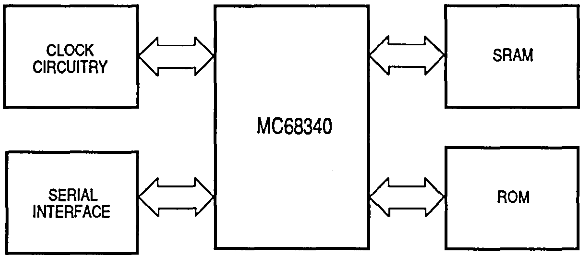


Figure 10-1. Minimum System Configuration Block Diagram

10.1.1 Processor Clock Circuitry

The MC68340 has an on-chip clock synthesizer that can operate from an on-chip phase-locked loop (PLL) and a voltage-controlled oscillator (VCO). The clock synthesizer uses an external crystal connected between the EXTAL and XTAL pins as a reference frequency source. Figure 10-2 shows a typical circuit using an inexpensive 32.768-kHz watch crystal. The 20-M resistor connected between the EXTAL and XTAL pins provides biasing for a faster oscillator startup time. The crystal manufacturer's documentation should be consulted for specific recommendations on external component values.

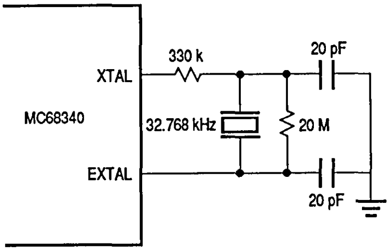
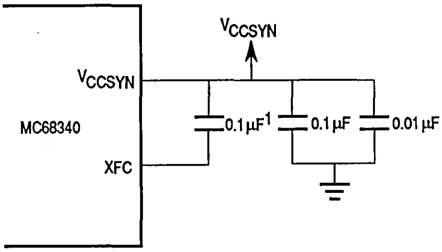


Figure 10-2. Sample Crystal Circuit

A separate power pin (V_{CCSYN}) is used to allow the clock circuits to operate with the rest of the device powered down and to provide increased noise immunity for the clock circuits. The source for V_{CCSYN} should be a quiet power supply, and external bypass capacitors (see Figure 10-3) should be placed as close as possible to the V_{CCSYN} pin to ensure a stable operating frequency.



NOTE 1: Must be a low leakage capacitor.

Figure 10-3. XFC and V_{CCSYN} Capacitor Connections

Additionally, the PLL requires that an external low-leakage filter capacitor, typically in the range of 0.01 to 0.1 μF, be connected between the XFC and V_{CCSYN} pins. Smaller values of the external filter capacitor provide a faster response time for the PLL, and larger values provide greater frequency stability. Figure 10-3 depicts examples of both an external filter capacitor and bypass capacitors for V_{CCSYN}.

10.1.2 Reset Circuitry

Because it is optional, reset circuitry is not shown in Figure 10-1. The MC68340 holds itself in reset after power-up and asserts $\overline{\text{RESET}}$ to the rest of the system. If an external reset pushbutton switch is desired, an external reset circuit is easily constructed by using open-collector cross-coupled NAND gates to debounce the output from the switch.

10.1.3 SRAM Interface

The SRAM interface is very simple when the programmable chip selects are used. External circuitry to decode address information and circuitry to return data and size acknowledge (DSACK) is not required. However, external ICs are required to provide write enables for the high and low bytes of data.

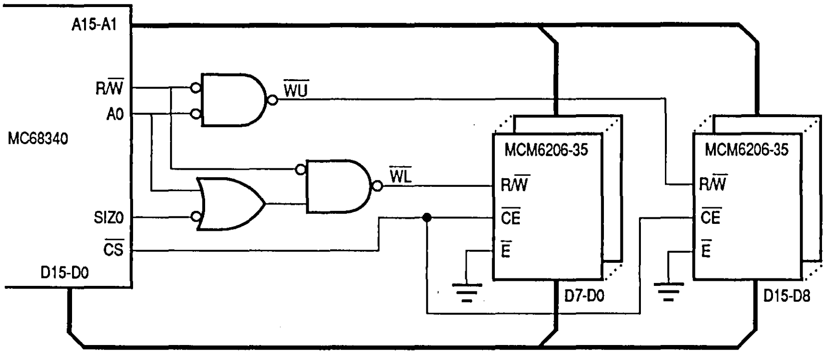


Figure 10-4. SRAM Interface

The SRAM interface shown in Figure 10-4 is a two-clock interface at 16.78-MHz operating frequency. The MCM6206-35 memories provide an access time of 15 ns when the $\overline{E}$ input is low. If buffers are required to reduce signal loading or if slower and less expensive memories are desired, a three-clock cycle can be used. In the circuit shown in Figure 10-4, additional memories can be used provided that the MC68340 specification for load capacitance on the chip-select ($\overline{CS}$) signal is not exceeded. (Address buffers may be needed, however.)

10.1.4 ROM Interface

Using the programmable chip selects creates a very straightforward ROM interface. As shown in Figure 10-5, no external circuitry is needed. Care must be used, however, not to overload the address bus. Address buffers may be required to ensure that the total system input capacitance on the address signals does not exceed the C_L specification.

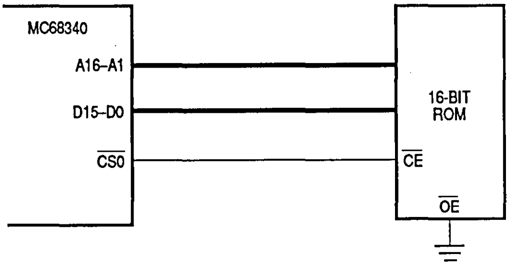


Figure 10-5. EPROM Interface

10.1.5 Serial Interface

The necessary circuitry to create an RS-232 interface with the MC68340 includes an external crystal and an RS-232 receiver/driver (see Figure 10-6). The resistor and capacitor values shown are typical; the crystal manufacturer's documentation should be consulted for specific recommendations on external component values. The circuit shown does not include modem support (ready to send (RTS) and clear to send (CTS) are not shown); however, these signals can be connected to the receiver/driver and to the connector in a similar manner as the connections for TxD and RxD.

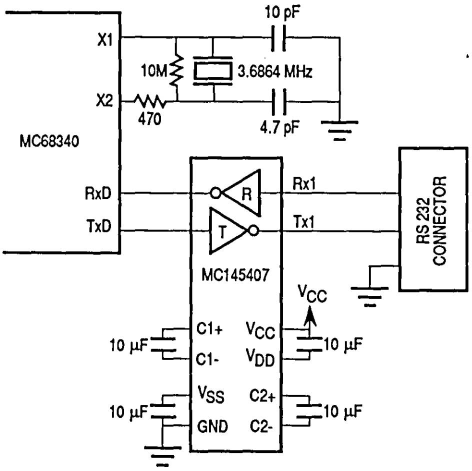


Figure 10-6. Serial Interface

10.2 MC68340 INITIALIZATION SEQUENCE

The following paragraphs discuss a suggested method for initializing the MC68340 after power-up.

10.2.1 Startup

RESET is asserted by the MC68340 during the time in which V_{CC} is ramping up, the VCO is locking onto the frequency, and the MC68340 is going through the reset operation. After RESET is negated, four bus cycles are run, with CS0 being asserted to fetch the 32-bit program counter (PC) and the 32-bit stack pointer (SP) from the boot ROM. Until programmed differently, CS0 is a 16-bit-wide, three-wait-state chip select.

After the PC and the SP are fetched, the following programming steps should be followed:

- Initialize and set the valid bit in the module base address register (CPU space address $03FF00) with the desired base address for the internal modules.
- Initialize and set the valid bits in the necessary chip-select base address and address mask registers. Following this step, other system resources requiring the CSx signals can be accessed. Care must be exercised when changing the address for CS0. The address of the instruction following the MOVE instruction to the CS0 base address register must match the value of the PC at that time.

10.2.2 SIM Module Configuration

The order of the following SIM register initializations is not important; however, time can be saved by initializing the clock synthesizer control register first to quickly increase to the desired processor operating frequency. The module base address register must be initialized prior to any of following steps.

Clock Synthesizer Control Register (SYNCR)

- Set frequency control bits (W, X, Y) to specify frequency.
- Select action taken during loss of crystal (RSTEN bit): activate a system reset or operate in limp mode.
- Select system clock and CLKOUT during LPSTOP (STSIM and STEXT bits).

Module Configuration Register (MCR)

- If using the software watchdog and/or the periodic interrupt timer, select action taken when FREEZE is asserted (FRZ bits).
- Select port B configuration (FIRQ bit). Note that this bit is used in combination with the bits in the port B pin assignment register (PPARB) to program the function of the port B pins.
- Select the access privilege for the supervisor/user registers (SUPV bit).
- Select the interrupt arbitration level for the SIM (IARB bits).

Autovector Register (AVR)

- Select the desired external interrupt levels for internal autovectoring.

System Protection Control Register (SYPCR) (Note that this register can only be written once after reset.)

- Enable the software watchdog, if desired (SWE bit).
- If the watchdog is enabled, select whether a system reset or a level 7 interrupt is desired at timeout (SWRI bit).
- If the watchdog is enabled, select the timeout period (SWT bits).
- Enable the double bus fault monitor, if desired (DBF bit).
- Enable the external bus monitor, if desired (BME bit).
- Select timeout period for bus monitor (BMT bits).

Software Watchdog Interrupt Vector Register (SWIV)

- If using the software watchdog, program the vector number for a software watchdog interrupt.

Periodic Interrupt Timer Register (PITR)

- If using the software watchdog, select whether or not to prescale (SWP bit).
- If using the periodic interrupt timer, select whether or not to prescale (PTP bit).
- Program the count value for the periodic timer, or program a zero value to turn off the periodic timer (PITR bits).

Periodic Interrupt Control Register (PICR)

- If using the periodic timer, program the desired interrupt level for the periodic interrupt timer (PIRQL bits).
- If using the periodic timer, program the vector number for a periodic timer interrupt.

Port A and B Registers

- Program the desired function of the port A signals (PPARA1 and PPARA2 registers).
- Program the desired function of the port B signals (PPARB register).

10.2.3 DMA Module Configuration

The following paragraphs describe DMA module initialization and operation. If the DMA capability of the MC68340 is being used, the initialization steps should be performed during the initialization sequence for the part. The operation steps should be performed to start a DMA transfer.

10.2.3.1 DMA MODULE INITIALIZATION. The following steps can be accomplished in any order when initializing the DMA module. These steps need to be performed for each channel used.

Module Configuration Register (MCR)

- Initialize the stop bit (STP) for normal operation. (Only one STP bit exists for both channels.)
- Select whether to respond to or ignore FREEZE (FRZ bits). (Only one set of FRZ bits exits for both channels.)
- If desired, enable single-address mode (SE bit).
- Program the interrupt service mask to set the level below which interrupts are ignored during a DMA transfer (ISM bits).

- Select the access privilege for the supervisor/user registers (SUPV bit).
- Program the master arbitration ID (MAID) to establish priority on the intermodule bus. Note that the two DMA channels should have distinct IDs if both channels are being used. (If they are programmed the same, channel 1 has priority.)
- Select the interrupt arbitration level for the DMA channel (IARB bits). (Only one set of IARB bits exits for both channels.)

Interrupt Register (INTR)

- Program the interrupt priority level for the channel interrupt (INTL bits).
- Program the vector number for a channel interrupt (INTV bits).

Channel Control Register (CCR)

- If desired, enable the interrupt when breakpoint is recognized and the channel is the bus master (INTB bit).
- If desired, enable the interrupt when done without an error condition (INTN bit).
- If desired, enable the interrupt when the channel encounters an error (INTE bit).
- Select which device generates requests if in dual-address mode, or select the direction of transfer if in single-address mode (ECO bit).

10.2.3.2 DMA MODULE OPERATION (SINGLE-ADDRESS MODE). The following steps are required to begin a DMA transfer in single-address mode.

Channel Control Register (CCR)

- Write a zero to the start bit (STR) to prevent the channel from starting the transfer prematurely.
- Select the amount by which to increment the source address for a read cycle (SAPI bits) or the destination address for a write cycle (DAPI bits).
- Define the transfer size by selecting the destination size for a write cycle (DSIZE) or by selecting the source size for a read cycle (SSIZE bits).
- Select signal address transfer (S/D bit).

Address Register (SAR or DAR)

- Write the source address for a read cycle or the destination address for a write cycle.

Function Code Register (FCR)

- Encode the source function code for a read cycle or the destination function code for a write cycle.

Byte Transfer Counter (BTC)

- Encode the number of bytes to be transferred.

Channel Control Register (CCR)

- Write a one to the start bit (STR) to allow the transfer to begin.

10.2.3.3 DMA MODULE OPERATION (DUAL-ADDRESS MODE). The following steps are required to begin a DMA transfer in dual-address mode.

Channel Control Register (CCR)

- Write a zero to the start bit (STR) to prevent the channel from starting the transfer prematurely.
- Select the amount by which to increment the source and destination addresses (SAPI and DAPI bits).
- Select the source and destination sizes (SSIZE and DSIZE bits).
- Select internal request, external request burst mode, or external request cycle steal (REQ bits).
- If using internal request, select the amount of bus bandwidth used by the DMA (BB bits).
- Select dual-address transfer (S/D bit).

Address Registers (SAR and DAR)

- Write the source and destination addresses.

Function Code Register (FCR)

- Encode the source and destination function codes.

Byte Transfer Counter (BTC)

- Encode the number of bytes to be transferred.

Channel Control Register (CCR)

- Write a one to the start bit (STR) to allow the transfer to begin.

10.2.4 Serial Module Configuration

If the serial capability of the MC68340 is being used, the following steps are required to properly initialize the serial module. Note that the serial module registers can only be accessed by byte operations.

Command Register (CR)

- Reset the receiver and transmitter for each channel.

The following steps program both channels:

Module Configuration Register (MCR)

- Initialize the stop bit (STP) for normal operation.
- Select whether to respond to or ignore FREEZE (FRZ bits).
- Select the input capture clock (ICCS bit).
- Select the access privilege for the supervisor/user registers (SUPV bit).
- Select the interrupt arbitration level for the serial module (IARB bits).

Interrupt Vector Register (IVR)

- Program the vector number for a serial module interrupt.

Interrupt Level Register (ILR)

- Program the interrupt priority level for a serial module interrupt.

Interrupt Enable Register (IER)

- Enable the desired interrupt sources.

Auxiliary Control Register (ACR)

- Select baud rate set (BRG bit).
- Initialize the input enable control (IEC bits).

Output Port Control Register (OPCR)

- Select the function of the output port pins.

Interrupt Status Register (ISR)

- The XTAL–RDY bit should be polled until it is cleared to ensure that an unstable crystal input is not applied to the baud rate generator.

The following steps are channel specific:

Clock Select Register (CSR)

- Select the receiver and transmitter clock.

Mode Register 1 (MR1)

- If desired, program operation of receiver ready-to-send (RxRTS bit).
- Select receiver-ready or FIFO-full notification (R/F bit).
- Select character or block error mode (ERR bit).
- Select parity mode and type (PM and PT bits).
- Select number of bits per character (B/C bits).

Mode Register 2 (MR2)

- Select the mode of channel operation (CM bits).
- If desired, program operation of transmitter ready-to-send (TxRTS bit).
- If desired, program operation of clear-to-send (CTS bit).
- Select stop-bit length (SB bits).

Command Register (CR)

- Enable the receiver and transmitter.

10.2.5 Timer Module Configuration

If the timer capability of the MC68340 is being used, the following steps should be followed to initialize a timer module properly. Note that this sequence must be done for each timer module used.

Command Register (CR)

- Clear the SWR bit to disable the timer.

Status Register (SR)

- Clear the TC, TG, and TO bits to reset the interrupts.

Module Configuration Register (MCR)

- Initialize the stop bit (STP) for normal operation.
- Select whether to respond to or ignore FREEZE (FRZ bits).

- Select the access privilege for the supervisor/user registers (SUPV bit).
- Select the interrupt arbitration level for the serial module (IARB bits).

Interrupt Register (IR)

- Program the interrupt priority level for the channel interrupt (IL bits).
- Program the vector number for a channel interrupt (IVR bits).

Preload Registers (PREL1 and PREL2)

- If required, initialize the preload registers for mode of operation.

Compare Register (COM)

- If desired, initialize the compare register.

The following steps begin operation:

Command Register (CR)

- Set the SWR bit to enable the timer.
- Enable the desired interrupts (IE bits).
- Enable $\overline{\text{TGATE}}$ if required for mode of operation (TGE bit).
- Select the counter clock (PSE bit).
- Enable the selected clock (CPE bit).
- Select the selected clock (CLK bit).
- If the PSE bit is set, select the prescaler output tap (POT bits).
- Select the mode of operation (MODE bits).
- Select the operation of TOUT (OC bits).

10.3 MEMORY INTERFACE INFORMATION

The following paragraphs contain information on using an 8-bit boot ROM, performing access time calculations, calculating frequency-adjusted outputs, and interfacing an 8-bit device to 16-bit memory using single-address mode.

10.3.1 Using a 8-Bit Boot ROM

Upon power-up, the MC68340 uses $\overline{\text{CS0}}$ to begin operation. $\overline{\text{CS0}}$, until otherwise programmed, is a three-wait-state, 16-bit chip select. If an 8-bit ROM is desired, external circuitry can be added to return an 8-bit DSACK in two wait states (see Figure 10-7).

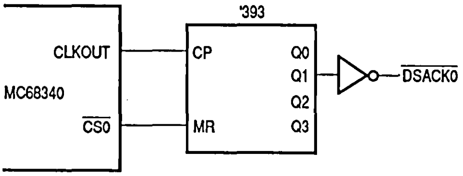


Figure 10-7. External Circuitry for 8-Bit Boot ROM

The '393 is a falling edge triggered counter; thus, CS0 is stable during the time in which it is being clocked. CS0 acts as the asynchronous reset — i.e., when it is asserted, the '393 is allowed to count. The falling edge of S2 provides the first counting edge. Q1 does not transition on this falling edge, but transitions to a logic one on the subsequent edge. DSACK0 is Q1 inverted; thus, on the next falling edge, DSACK0 is seen as asserted, indicating an 8-bit port. When CS0 is negated, Q1 is again held in reset and DSACK0 is negated. The timing diagram in Figure 10-8 illustrates this operation.

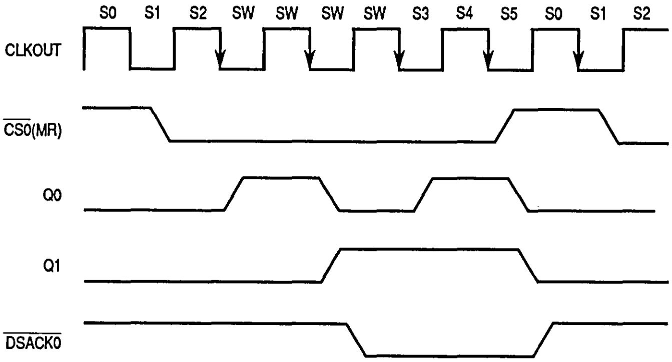


Figure 10-8. 8-bit Boot ROM Timing

10.3.2 Access Time Calculations

The two time paths that are critical in an MC68340 application using the CS signals are shown in Figure 10-9. The first path is the time from adddress valid to when data must be available to the processor; the second path is the time from CS asserted to when data must be available to the processor.

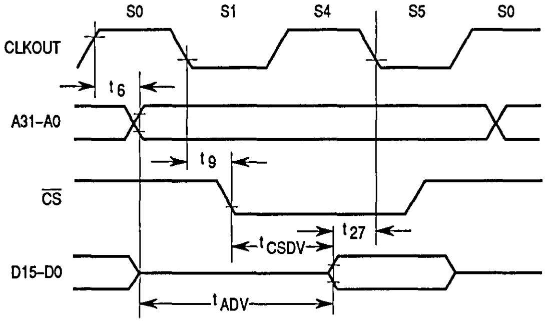


Figure 10-9. Access Time Computation Diagram

As shown in the diagram, an equation for the address access time, t_{ADV}, can be developed as follows:

$$t_{ADV} = T(N - \frac{1}{2}) - t_6 - t_{27}$$

where:

T = system clock period
N = number of clocks per access
t_6 = CLKOUT high to address valid = 30-ns maximum at 16.78 MHz
t_{27} = data-in valid to CLKOUT low setup = 5-ns minimum at 16.78 MHz

An equation for the chip select access time, t_{CSDV}, can be developed as follows:

$$t_{CSDV} = T(N - 1) - t_9 - t_{27}$$

where:

T = system clock period
N = number of clocks per access
t_9 = CLKOUT low to $\overline{CS}$ asserted = 30-ns maximum at 16.78 MHz
t_{27} = data-in valid to CLKOUT low setup = 5-ns minimum at 16.78 MHz

Using these equations, the memory access times at 16.78 MHz are shown in Table 10-1.

Table 10-1. Memory Access Times at 16.78 MHz

	N=2	N=3	N=4	N=5	N=6
t_{ADV}	55 ns	115 ns	175 ns	235 ns	295 ns
t_{CSDV}	25 ns	85 ns	145 ns	205 ns	265 ns

The values can be used to determine how many clock cycles an access will take, given the access time of the memory devices and any delays through buffers or external logic that may be needed.

10.3.3 Calculating Frequency-Adjusted Output

The general relationship between the CLKOUT and most input and output signals is shown in Figure 10-10. Most outputs transition off of a falling edge of CLKOUT, but the same principle applies to those outputs that transition off of a rising edge.

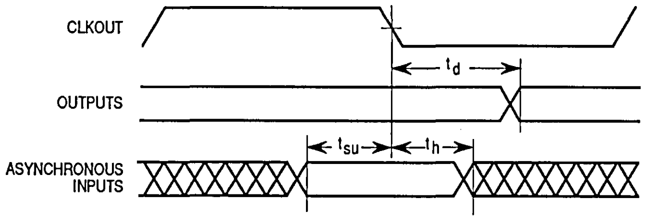


Figure 10-10. Signal Relationships to CLKOUT

For outputs that are referenced to a clock edge, the propagation delay (t_d) does not change as the frequency changes. For instance, specification 6 in the electrical characteristics, shown in MC68340/D, *MC68340 Technical Summary*, shows that address, function code, and size information is valid 3 to 30 ns after the rising edge of S0. This specification does not change even if the device frequency is less than 16.78 MHz. Additionally, the relationship between the asynchronous inputs and the clock edge, as shown in Figure 10-10, does not change as frequency changes.

A second type of specification indicates the minimum amount of time a signal will be asserted. This type of specification is illustrated in Figure 10-11.

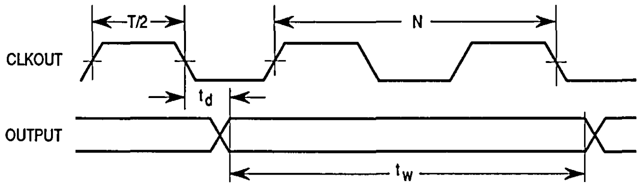


Figure 10-11. Signal Width Specifications

The method for calculating a frequency-adjusted t_W is as follows:

$$t_{W'} = t_W + N\left(\frac{Tf'}{2} - \frac{Tf}{2}\right) + \left(\frac{Tf'}{2} - td\right)$$

where:

$t_{W'}$ = the frequency-adjusted signal width

t_W = the signal width at 16.78 MHz

N = the number of full one-half clock periods in t_W

$\frac{Tf'}{2}$ = one-half the new clock period

$\frac{Tf}{2}$ = one-half the clock period at full speed

t_d = the propagation time from the clock edge

The following calculation uses a 16.78-MHz part, specification 14, $\overline{\text{AS}}$ width asserted, at 12.5 MHz as an example:

t_W = 100 ns

N = 3

$\frac{Tf'}{2} = \frac{80}{2} = 40$ ns

$\frac{Tf}{2} = \frac{60}{2} = 30$ ns

t_d = 30 ns maximum

therefore:

$$t_{W'} = 100 + 3(40 - 30) + (40 - 30) = 140 \text{ ns}$$

The third type of specification used is a skew between two outputs, as shown in Figure 10-12.

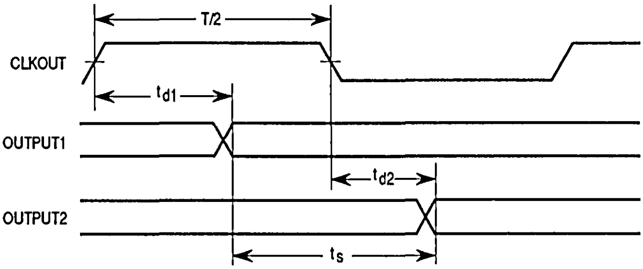


Figure 10-12. Skew between Two Outputs

The method for calculating a frequency-adjusted t_S is as follows:

$$t_S' = t_S + N\left(\frac{Tf'}{2} - \frac{Tf}{2}\right) + \left(\frac{Tf'}{2} - t_{d1}\right)$$

where:

t_S' = the frequency-adjusted skew
t_S = the skew at full speed
N = the number of full one-half clock periods in t_S, if any
$\frac{Tf'}{2}$ = one-half the new clock period
$\frac{Tf}{2}$ = one-half the clock period at full speed
t_{d1} = the propagation time for the first output from the clock edge

The following calculation uses a 16.78-MHz port, specification 21, R/$\overline{\text{W}}$ high to $\overline{\text{AS}}$ asserted, at 8 MHz as an example:

t_S = 15 ns minimum
N = 0
$\frac{Tf'}{2} = \frac{125}{2} = 62.5$ ns
$\frac{Tf}{2} = \frac{60}{2} = 30$ ns
t_{d1} = 30 ns maximum

therefore:

$$t_W' = 15 + 0(62.5 - 30) + (62.5 - 30) = 47.5 \text{ ns minimum}$$

In this manner, new specifications for lower frequencies can be derived for an MC68340.

10.3.4 Interfacing an 8-Bit Device to 16-Bit Memory Using Single-Address DMA Mode

One of the requirements of single-address mode is that the source and destination must be the same port size. However, the MC68340 can perform direct memory accesses in single-address mode between an 8-bit device and 16-bit memory. The port size must be specified as 8 bits, and some external logic is required as shown in Figure 10-13.

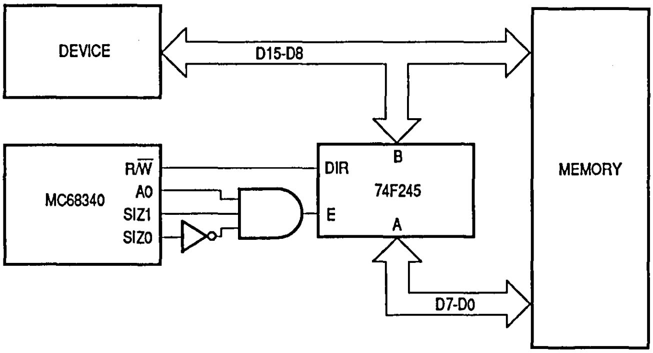


Figure 10-13. Circuitry for Interfacing 8-Bit Device to 16-Bit Memory in Single-Address DMA Mode

During even-byte accesses, the data is transferred directly on D15–D8. However, during odd-byte accesses, the data must be routed on D15–D8 for the 8-bit device and on D7–D0 for the 16-bit memory.

SECTION 11
ELECTRICAL CHARACTERISTICS

This section contains information on the maximum ratings and thermal characteristics of the MC68340. Detailed information on power considerations, DC electrical characteristics, and AC timing specifications is provided in the MC68340/D, *MC68340 Technical Summary*.

11.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	−0.3 to +7.0	V
Input Voltage	V_{in}	−0.3 to +7.0	V
Operating Temperature Range	T_A	0 to 70	°C
Storage Temperature Range	T_{stg}	−55 to +150	°C

The following ratings define a range of conditions in which the device will operate without being damaged. However, sections of the device may not operate normally while being exposed to the electrical extremes. This device contains circuitry to protect against damage due to high static voltages or electrical fields; however, it is advised that normal precautions be taken to avoid application of any voltages higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either GND or V_{CC}).

11.2 THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance — Junction to Case Ceramic 144-Pin QFP Plastic 145-Pin PGA	θ_{JA}	 15* TBD	°C/W
Thermal Resistance — Junction to Ambient Ceramic 144-Pin QFP Plastic 145-Pin PGA	θ_{JA}	 46* 27*	°C/W

*Estimated
TBD — To be determined

11

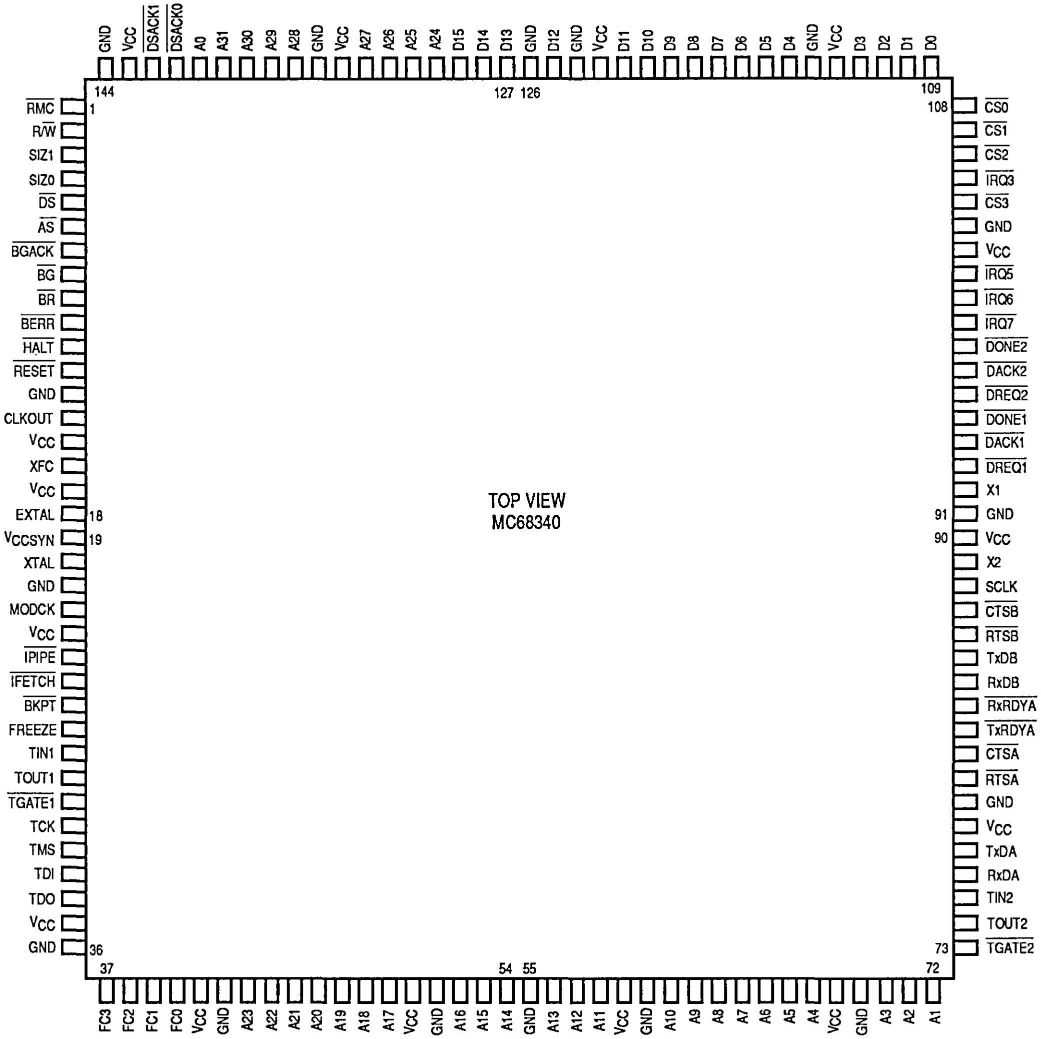
SECTION 12
ORDERING INFORMATION AND MECHANICAL DATA

This section contains the pin assignments and package dimensions of the MC68340. In addition, detailed information is provided to be used as a guide when ordering.

12.1 STANDARD MC68340 ORDERING INFORMATION

Package Type	Frequency (MHz)	Temperature	Order Number
Ceramic Surface Mount FE Suffix	16.78	0°C to +70°C	MC68340FE16
Plastic Pin Grid Array RP Suffix	16.78	0°C to +70°C	MC68340RP16

12.2 PIN ASSIGNMENT — CERAMIC SURFACE MOUNT (FE SUFFIX)



12

The V_{CC} and GND pins are separated into groups to help electrically isolate the different output drivers of the MC68340. These groups are shown in the following table.

Pin Group - FE Suffix	V_{CC}	**GND**
Address Bus, FCx	41, 50, 59, 68, 134	42, 51, 60, 69, 135
Data Bus	113, 123	114, 124
$\overline{\mathrm{RMC}}$, R/$\overline{\mathrm{W}}$, SIZx, $\overline{\mathrm{DS}}$, $\overline{\mathrm{AS}}$, $\overline{\mathrm{BG}}$, $\overline{\mathrm{HALT}}$, $\overline{\mathrm{RESET}}$, CLKOUT, MODCK, $\overline{\mathrm{IPIPE}}$, $\overline{\mathrm{IFETCH}}$, FREEZE, TOUT1, Internal Logic	15, 17, 35, 143	13, 21, 36, 144
$\overline{\mathrm{CSx}}$, $\overline{\mathrm{IRQx}}$, $\overline{\mathrm{DACKx}}$, $\overline{\mathrm{DONEx}}$, $\overline{\mathrm{RTSx}}$, TxDx, $\overline{\mathrm{RxRDYA}}$, $\overline{\mathrm{TxRDYA}}$, TOUT2, Internal Logic	78, 90, 102	79, 91, 103
Oscillator	19	—
Internal Only	23	55, 126

12.3 PIN ASSIGNMENT — PLASTIC PIN GRID ARRAY (RP SUFFIX)

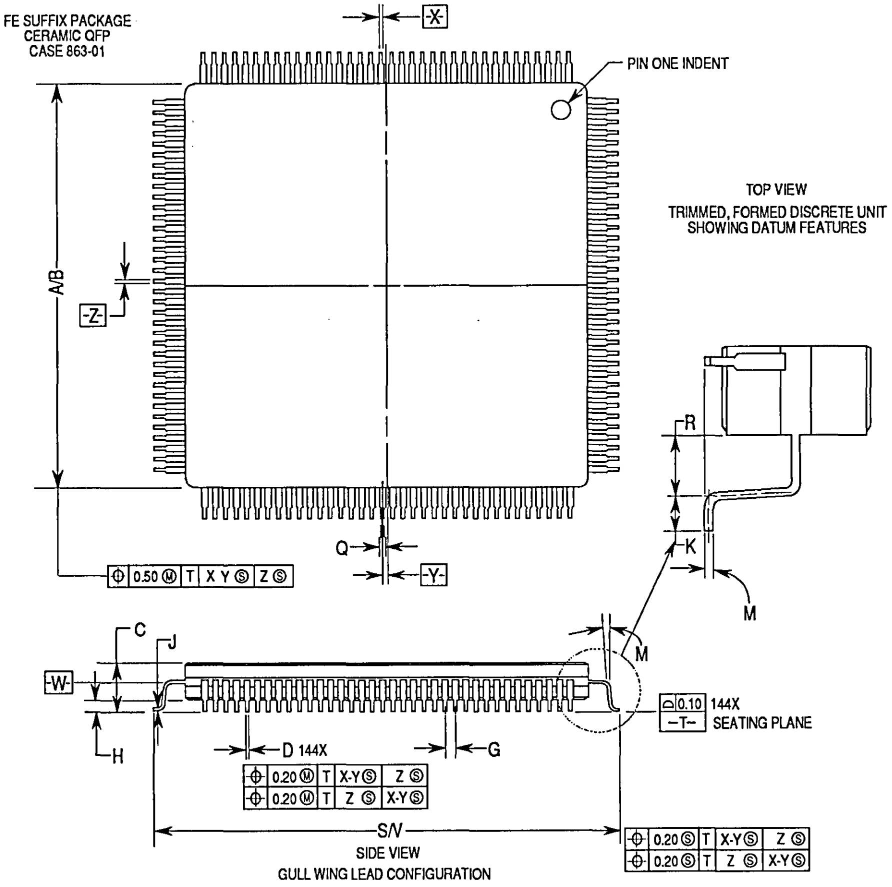
Q	FC1	FC3	TDI	TCK	TIN1	FREEZE	$\overline{\text{IPIPE}}$	MODCK	EXTAL	XFC	$\overline{\text{RESET}}$	$\overline{\text{BERR}}$	$\overline{\text{BR}}$	$\overline{\text{AS}}$	SIZ1
P	A23	FC2	TDO	TMS	TOUT1	$\overline{\text{BKPT}}$	V_{CC}	XTAL	V_{CC}	CLKOUT	$\overline{\text{HALT}}$	$\overline{\text{BGACK}}$	$\overline{\text{DS}}$	R/$\overline{\text{W}}$	$\overline{\text{RMC}}$
N	A22	FC0	GND	V_{CC}	$\overline{\text{TGATE1}}$	$\overline{\text{IFETCH}}$	GND	V_{CCSYN}	V_{CC}	GND	$\overline{\text{BG}}$	SIZ0	GND	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
M	A20	GND	V_{CC}										V_{CC}	A0	A3O
L	A18	A19	A21										A31	A29	A28
K	A16	A17	V_{CC}										GND	V_{CC}	A27
J	A14	A15	GND										A25	A24	A26
H	A12	A13	GND					BOTTOM VIEW					GND	D14	D15
G	A11	GND	V_{CC}										GND	D12	D13
F	A10	A9	A8										V_{CC}	D11	D10
E	A7	A6	A5										D7	D8	D9
D	A4	V_{CC}	GND	NC									GND	D5	D6
C	A3	A2	$\overline{\text{TGATE2}}$	V_{CC}	GND	TxDB	V_{CC}	GND	$\overline{\text{DACK1}}$	$\overline{\text{IRQ7}}$	GND	$\overline{\text{CS2}}$	D1	V_{CC}	D4
B	A1	TOUT2	TxDA	$\overline{\text{RTSA}}$	$\overline{\text{TxRDYA}}$	$\overline{\text{RTSB}}$	X2	X1	$\overline{\text{DONE1}}$	$\overline{\text{DONE2}}$	V_{CC}	$\overline{\text{CS3}}$	$\overline{\text{CS1}}$	D0	D3
A	TIN2	RxDA	$\overline{\text{CTSA}}$	$\overline{\text{RxRDYA}}$	RxDB	$\overline{\text{CTSB}}$	SCLK	$\overline{\text{DREQ1}}$	$\overline{\text{DREQ2}}$	$\overline{\text{DACK2}}$	$\overline{\text{IRQ6}}$	$\overline{\text{IRQ5}}$	$\overline{\text{IRQ3}}$	$\overline{\text{CS0}}$	D2
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The V_{CC} and GND pins are separated into groups to help electrically isolate the different output drivers of the MC68340. These groups are shown in the following table.

Pin Group - RP Suffix	V_{CC}	GND
Address Bus, FCx	D2, G3, K3, K14, M3	D3, G2, J3, K13, M2
Data Bus	C14, F13	D13, G13
$\overline{\text{RMC}}$, R/$\overline{\text{W}}$, SIZx, $\overline{\text{DS}}$, $\overline{\text{AS}}$, $\overline{\text{BG}}$, $\overline{\text{HALT}}$, $\overline{\text{RESET}}$, CLKOUT, MODCK, $\overline{\text{IPIPE}}$, $\overline{\text{IFETCH}}$, FREEZE, TOUT1, Internal Logic	M13, N4, N9, P9	N3, N7, N10, N13
$\overline{\text{CSx}}$, $\overline{\text{IRQx}}$, $\overline{\text{DACKx}}$, $\overline{\text{DONEx}}$, $\overline{\text{RTSx}}$, TxDx, $\overline{\text{RxRDYA}}$, $\overline{\text{TxRDYA}}$, TOUT2, Internal Logic	B11, C4, C7	C5, C8, C11
Oscillator	N8	—
Internal Only	P7	G13, H3

12.4 PACKAGE DIMENSIONS

FE Suffix



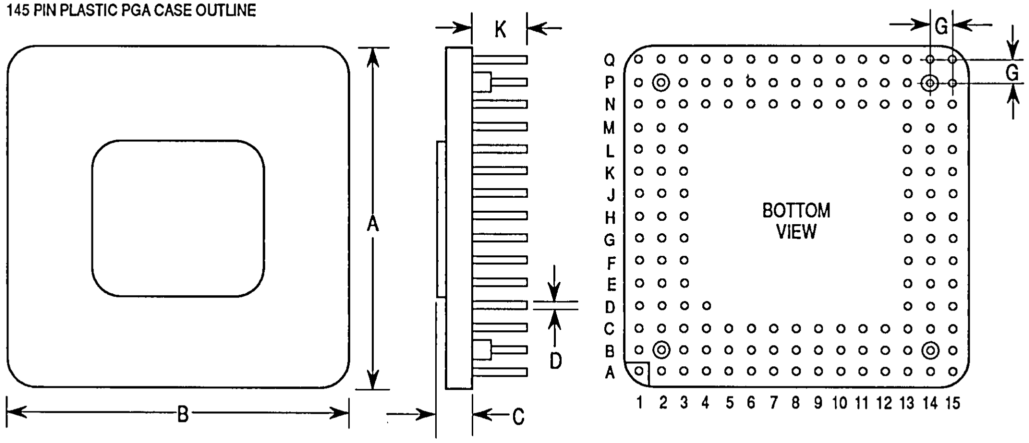
	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	25.84	27.70	1.017	1.091
B	25.84	27.70	1.017	1.091
C	3.55	4.31	0.140	0.170
D	0.22	0.41	0.009	0.016
G	0.65 BSC		0.0256 BSC	
H	0.25	0.88	0.010	0.035
J	0.13	0.25	0.005	0.010
K	0.65	0.95	0.026	0.037
M	0°	8°	0°	8°
Q	0.325 BSC		0.0128 BSC	
R	0.71	—	0.028	—
S	30.95	31.45	1.219	1.238
V	30.95	31.45	1.219	1.238

NOTES:

1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETERS
3. DIM A AND B DEFINE MAXIMUM CERAMIC BODY DIMENSIONS INCLUDING GLASS PROTRUSION AND MISMATCH OF CERAMIC BODY TOP AND BOTTOM.
4. DATUM PLANE -W- IS LOCATED AT THE UNDERSIDE OF LEADS WHERE LEADS EXIT PACKAGE BODY.
5. DATUMS X-Y AND Z TO BE DETERMINED WHERE CENTER LEADS EXIT PACKAGE BODY AT DATUM -W-.
6. DIM S AND V TO BE DETERMINED AT SEATING PLANE, DATUM -T-.
7. DIM A AND B TO BE DETERMINED AT DATUM PLANE -W-.

12.5 PACKAGE DIMENSIONS
RP Suffix

MC68340
145 PIN PLASTIC PGA CASE OUTLINE



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	39.37	39.88	1.550	1.570
B	39.37	39.88	1.550	1.570
C	2.79	3.56	0.110	0.140
D	0.43	0.55	0.017	0.022
G	2.54 BSC		0.100 BSC	
K	4.32	4.95	0.170	0.195

12

INDEX

I

I

I

NOTES

NOTES

NOTES

NOTES

A26290 PRINTED IN USA 9/90 EVANS PRESS EMTR 1148 20,000 MPU YGACAA

Order this document
by MC68340UMAD/AD

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

MC68340

Errata to
MC68340 Integrated Processor User's Manual

January 24, 1991

This errata applies to the MC68340UM/AD, *MC68340 Integrated Processor User's Manual.* The following pages have been amended.

Page:

2-14 In paragraph 2.14.3, the following sentences should be added: "$\overline{\text{DONEx}}$ is an active input in any mode. As an output, $\overline{\text{DONEx}}$ is only active in external request mode. An external pullup resistor is required even if operating only in the internal request mode."

3-31 In Figure 3-14, the flowchart should be labeled "INTERRUPTING DEVICE—MC68340."

4-11 In Figure 4-4, the value of the resistor in series with the crystal should be 330 ohms. Also, the external pin labeled CLKOUT should be labeled EXTCLK.

4-14 The sentence under Internal $\overline{\text{DSACKx}}$ Generation for External Acesses with Programmable Wait States should say, "$\overline{\text{DSACKx}}$ can be generated internally with up to three wait states for a particular device using the DD bits in the address mask register."

In paragraph 4.2.4.2, all references to CS0 should be $\overline{\text{CS0}}$. In the second paragraph after the first sentence, add, "Global chip select does not provide write protection and responds to all function codes."

4-32 Address mask register 2, bit 15 should be AM15; bit 2 should be DD0.

4-34 In paragraph 4.3.5.2, the last sentence should say, "$\overline{\text{IACKx}}$ signals have the same timing as address strobes."

5-128 In paragraph 5.8.1.5, the first sentence should read, "The CPU is capable of accessing on-chip peripherals with an access time of two clocks."

6-2 In paragraph 6.1.3, the following sentences should be added: "$\overline{\text{DONEx}}$ is an active input in any mode. As an output, $\overline{\text{DONEx}}$ is only active in external request mode. An external pullup resistor is required even if operating only in the internal request mode."

6-3 In paragraph 6.2.1, the second sentence should say, "This initialization is accomplished by programming the appropriate registers."

This document contains information on a product under development. Motorola reserves the right to change or discontinue this product without notice.

MOTOROLA

6-12 The first note should say, "The DMA module uses only one STP bit for both channels."

The SE bit description should be:
1 = In single-address mode, the external data bus is driven during DMA transfer.
0 = In single-address mode, the external data bus remains in a high-impedance state during a DMA transfer (used for intermodule DMA).
In dual-address mode, the SE bit has no effect.

6-13 In paragraph 6.4.2, the reset value of bits 7-0 of the interrupt registers should be 00001111.

In the INTV bit description, the first sentence should be deleted.

6-14 In the ECO bit description, add the sentence, "If request generation is programmed to be internal, this bit has no effect."

6-16 In Table 6-5, the BB bits should be labeled bit 3 and bit 2.

Replace the entire STR bit description with the following:

STR — Start

Internal Request
1 = Start DMA transfer. In internal request mode, the transfer starts immediately when STR is set.
0 = The transfer can be stopped by clearing STR.

External Request
1 = Setting this bit enables the DMA to start the transfer when a $\overline{\text{DREQx}}$ input is received from an external device.
0 = The transfer can be stopped by clearing STR.

The bit is cleared by reset, writing a logic zero, the DONE status bit being set, or one of the error status bits (BES, BED, or CONF) being set.

7-8 Paragraph 7.2.12 should say, "This active-low output signal is programmable as the channel A transmitter ready or as a dedicated parallel output, and cannot be masked by the IER."

Paragraph 7.2.12.1 should read, "When used for this function, this signal reflects the complement of the status of bit 2 of the channel A status register (SRA)."

Paragraph 7.2.13 should read, "This active-low output signal is programmable as the channel A receiver ready, channel A FIFO full indicator, or a dedicated parallel output, and cannot be masked by the IER."

The first sentence in paragraph 7.2.13.1 should say, "When used for this function, this signal reflects the complement of the status of bit 0 of the ISR."

The first sentence in paragraph 7.2.13.2 should read, "When used for this function, this signal reflects the complement of the status of bit 1 of the ISR."

7-15 The second sentence in the fourth paragraph should say, "When in this mode, $\overline{\text{RTSx}}$ is automatically negated by the receiver when a valid start bit is detected and the FIFO stack is full."

7-23 The fifth sentence in the last paragraph should be deleted.

7-27 In paragraph 7.4.1.5, bit 4 in the register diagram should be labeled OE.

8-19 In Figure 8-11, the fourth line should be labeled "Command Register (CR)."

Also in Figure 8-11, the address of preload 2 register for timer 2 should be $64E.

8-21 In paragraph 8.4.3, the title should be, "Command Register (CR)."

9-3 In the third paragraph, the first sentence should say, "During the capture-IR controller state, the parallel inputs to the instruction shift register are loaded with the standard 2-bit binary value (01) into the two least significant bits and the loss-of-crystal (LOC) status signal into bit 2."

9-4 In the second paragraph, the third sentence should be deleted.

Also in the second paragraph, the fourth sentence should read, "To ensure proper operation, the open-drain pins require external pullups."

In the list of control bits following the fifth paragraph, bits 10 and 22 should be:
10. done.ctl (83) 22. modck.ctl (123)

The second sentence in the last paragraph should say, "For example, the active-high level for done.ctl (bit 83) is logic one since the cell type is IO.Ctl1."

9-5 In Table 9-2, the following bit numbers require correction:

29	IO.Ctl0	tout2.ctl	—	—
31	I.Pin	$\overline{\text{RxDA}}$	Input	—
32	O.Latch	$\overline{\text{TxDA}}$	Output	—
33	O.Latch	$\overline{\text{RTSA}}$	Output	—
34	I.Pin	$\overline{\text{CTSA}}$	Input	—
35	O.Latch	$\overline{\text{RxRDYA}}$	Output	—
36	O.Latch	$\overline{\text{TxRDYA}}$	Output	—
37	I.Pin	RxDB	Input	—
38	O.Latch	TxDB	Output	—
39	O.Latch	$\overline{\text{RTSB}}$	Output	—
40	I.Pin	$\overline{\text{CTSB}}$	Input	—
44	O.Latch	$\overline{\text{DACK1}}$	Output	—
45	O.Latch	$\overline{\text{DONE1}}$	OD-I/O	done.ctl
48	O.Latch	$\overline{\text{DACK2}}$	Output	—
49	O.Latch	$\overline{\text{DONE2}}$	OD-I/O	done.ctl
83	IO.Ctl1	done.ctl	—	—
104	IO.Cell	$\overline{\text{DSACK0}}$	I/O**	berr.ctl
106	IO.Cell	$\overline{\text{RMC}}$	I/O*	ab.ctl
112	I.Pin	$\overline{\text{BGACK}}$	Input	—
113	O.Latch	$\overline{\text{BG}}$	Output	—
114	I.Pin	$\overline{\text{BR}}$	Input	—
122	IO.Cell	MODCK	I/O	modck.ctl
123	IO.Ctl0	modck.ctl	—	—
131	IO.Ctl0	tout1.ctl	—	—
132	I.Pin	$\overline{\text{TGATE1}}$	Input	—

9-8 In Figure 9-3, the MUX output should be "DATA TO SYSTEM LOGIC."

9-9 In Figure 9-6, the $\overline{1}$ input to the top, left-hand MUX should be "DATA FROM SYSTEM LOGIC."

10-2 In Figure 10-2, the value of the resistor in series with the crystal should be 330 ohms.

10-7 In paragraph 10.2.3, add the following sentence: “The $\overline{\text{DONEx}}$ pin requires an external pullup resistor even if operating only in the internal request mode.”

12-5 The GND pins for internal only should be H13, H3.

Index-8 TxRDYB bit should reference pages 7-38 and 7-40.

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and the Motorola logo are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

Literature Distribution Centers:
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Center; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.
ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

MOTOROLA

A26290-1 PRINTED IN USA 1/91 IMPERIAL LITHO C77282 18,000 MPU YGABAA

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

Order this document
by MC68340UMAD/AD

MC68340

Errata to
MC68340 Integrated Processor User's Manual

January 24, 1991

This errata applies to the MC68340UM/AD, *MC68340 Integrated Processor User's Manual*. The following pages have been amended.

Page:

2-14 In paragraph 2.14.3, the following sentences should be added: "$\overline{\text{DONEx}}$ is an active input in any mode. As an output, $\overline{\text{DONEx}}$ is only active in external request mode. An external pullup resistor is required even if operating only in the internal request mode."

3-31 In Figure 3-14, the flowchart should be labeled "INTERRUPTING DEVICE—MC68340."

4-11 In Figure 4-4, the value of the resistor in series with the crystal should be 330 ohms. Also, the external pin labeled CLKOUT should be labeled EXTCLK.

4-14 The sentence under Internal $\overline{\text{DSACKx}}$ Generation for External Acesses with Programmable Wait States should say, "$\overline{\text{DSACKx}}$ can be generated internally with up to three wait states for a particular device using the DD bits in the address mask register."

In paragraph 4.2.4.2, all references to CS0 should be $\overline{\text{CS0}}$. In the second paragraph after the first sentence, add, "Global chip select does not provide write protection and responds to all function codes."

4-32 Address mask register 2, bit 15 should be AM15; bit 2 should be DD0.

4-34 In paragraph 4.3.5.2, the last sentence should say, "$\overline{\text{IACKx}}$ signals have the same timing as address strobes."

5-128 In paragraph 5.8.1.5, the first sentence should read, "The CPU is capable of accessing on-chip peripherals with an access time of two clocks."

6-2 In paragraph 6.1.3, the following sentences should be added: "$\overline{\text{DONEx}}$ is an active input in any mode. As an output, $\overline{\text{DONEx}}$ is only active in external request mode. An external pullup resistor is required even if operating only in the internal request mode."

6-3 In paragraph 6.2.1, the second sentence should say, "This initialization is accomplished by programming the appropriate registers."

This document contains information on a product under development. Motorola reserves the right to change or discontinue this product without notice.

MOTOROLA

6-12 The first note should say, "The DMA module uses only one STP bit for both channels."

The SE bit description should be:
1 = In single-address mode, the external data bus is driven during DMA transfer.
0 = In single-address mode, the external data bus remains in a high-impedance state during a DMA transfer (used for intermodule DMA).
In dual-address mode, the SE bit has no effect.

6-13 In paragraph 6.4.2, the reset value of bits 7-0 of the interrupt registers should be 00001111.

In the INTV bit description, the first sentence should be deleted.

6-14 In the ECO bit description, add the sentence, "If request generation is programmed to be internal, this bit has no effect."

6-16 In Table 6-5, the BB bits should be labeled bit 3 and bit 2.

Replace the entire STR bit description with the following:

STR — Start

Internal Request
1 = Start DMA transfer. In internal request mode, the transfer starts immediately when STR is set.
0 = The transfer can be stopped by clearing STR.

External Request
1 = Setting this bit enables the DMA to start the transfer when a $\overline{\mathrm{DREQx}}$ input is received from an external device.
0 = The transfer can be stopped by clearing STR.

The bit is cleared by reset, writing a logic zero, the DONE status bit being set, or one of the error status bits (BES, BED, or CONF) being set.

7-8 Paragraph 7.2.12 should say, "This active-low output signal is programmable as the channel A transmitter ready or as a dedicated parallel output, and cannot be masked by the IER."

Paragraph 7.2.12.1 should read, "When used for this function, this signal reflects the complement of the status of bit 2 of the channel A status register (SRA)."

Paragraph 7.2.13 should read, "This active-low output signal is programmable as the channel A receiver ready, channel A FIFO full indicator, or a dedicated parallel output, and cannot be masked by the IER."

The first sentence in paragraph 7.2.13.1 should say, "When used for this function, this signal reflects the complement of the status of bit 0 of the ISR."

The first sentence in paragraph 7.2.13.2 should read, "When used for this function, this signal reflects the complement of the status of bit 1 of the ISR."

7-15 The second sentence in the fourth paragraph should say, "When in this mode, $\overline{\mathrm{RTSx}}$ is automatically negated by the receiver when a valid start bit is detected and the FIFO stack is full."

7-23 The fifth sentence in the last paragraph should be deleted.

7-27 In paragraph 7.4.1.5, bit 4 in the register diagram should be labeled OE.

8-19 In Figure 8-11, the fourth line should be labeled "Command Register (CR)."

Also in Figure 8-11, the address of preload 2 register for timer 2 should be $64E.

8-21 In paragraph 8.4.3, the title should be, "Command Register (CR)."

9-3 In the third paragraph, the first sentence should say, "During the capture-IR controller state, the parallel inputs to the instruction shift register are loaded with the standard 2-bit binary value (01) into the two least significant bits and the loss-of-crystal (LOC) status signal into bit 2."

9-4 In the second paragraph, the third sentence should be deleted.

Also in the second paragraph, the fourth sentence should read, "To ensure proper operation, the open-drain pins require external pullups."

In the list of control bits following the fifth paragraph, bits 10 and 22 should be:
10. done.ctl (83)   22. modck.ctl (123)

The second sentence in the last paragraph should say, "For example, the active-high level for done.ctl (bit 83) is logic one since the cell type is IO.Ctl1."

9-5 In Table 9-2, the following bit numbers require correction:

29	IO.Ctl0	tout2.ctl	—	—
31	I.Pin	$\overline{\text{RxDA}}$	Input	—
32	O.Latch	$\overline{\text{TxDA}}$	Output	—
33	O.Latch	$\overline{\text{RTSA}}$	Output	—
34	I.Pin	$\overline{\text{CTSA}}$	Input	—
35	O.Latch	$\overline{\text{RxRDYA}}$	Output	—
36	O.Latch	$\overline{\text{TxRDYA}}$	Output	—
37	I.Pin	RxDB	Input	—
38	O.Latch	TxDB	Output	—
39	O.Latch	$\overline{\text{RTSB}}$	Output	—
40	I.Pin	$\overline{\text{CTSB}}$	Input	—
44	O.Latch	$\overline{\text{DACK1}}$	Output	—
45	O.Latch	$\overline{\text{DONE1}}$	OD-I/O	done.ctl
48	O.Latch	$\overline{\text{DACK2}}$	Output	—
49	O.Latch	$\overline{\text{DONE2}}$	OD-I/O	done.ctl
83	IO.Ctl1	done.ctl	—	—
104	IO.Cell	$\overline{\text{DSACK0}}$	I/O**	berr.ctl
106	IO.Cell	$\overline{\text{RMC}}$	I/O*	ab.ctl
112	I.Pin	$\overline{\text{BGACK}}$	Input	—
113	O.Latch	$\overline{\text{BG}}$	Output	—
114	I.Pin	$\overline{\text{BR}}$	Input	—
122	IO.Cell	MODCK	I/O	modck.ctl
123	IO.Ctl0	modck.ctl	—	—
131	IO.Ctl0	tout1.ctl	—	—
132	I.Pin	$\overline{\text{TGATE1}}$	Input	—

9-8 In Figure 9-3, the MUX output should be "DATA TO SYSTEM LOGIC."

9-9 In Figure 9-6, the $\overline{1}$ input to the top, left-hand MUX should be "DATA FROM SYSTEM LOGIC."

10-2 In Figure 10-2, the value of the resistor in series with the crystal should be 330 ohms.

10-7 In paragraph 10.2.3, add the following sentence: "The $\overline{\text{DONEx}}$ pin requires an external pullup resistor even if operating only in the internal request mode."

12-5 The GND pins for internal only should be H13, H3.

Index-8 TxRDYB bit should reference pages 7-38 and 7-40.

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and the Motorola logo are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

Literature Distribution Centers:
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Center; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.
ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T.,
Hong Kong.

MOTOROLA

A26290-1 PRINTED IN USA 1/91 IMPERIAL LITHO C77282 18,000 MPU YGABAA