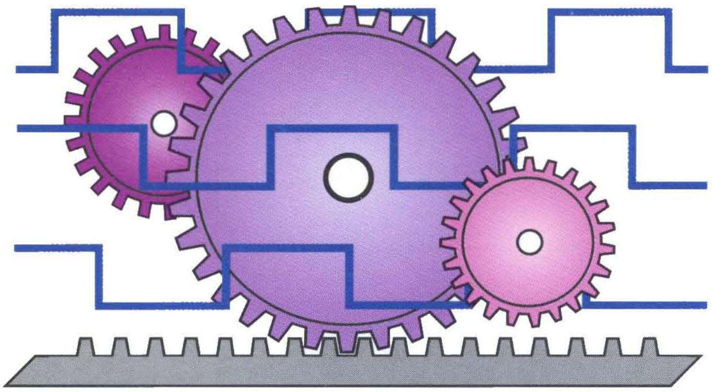
MC68EC040UM/AD

MC68EC040 USER'S MANUAL MOTOROLA



MC68EC040

32-BIT
EMBEDDED
CONTROLLER
USER'S MANUAL

MOTOROLA

Introduction	1
Programming Model	2
Data Organization and Addressing Capabilities	3
Instruction Set Summary	4
Signal Description	5
Access Control Unit	6
Instruction and Data Caches	7
Bus Operation	8
Exception Processing	9
Instruction Execution Timing	10
Electrical Characteristics	11
Ordering Information and Mechanical Data	12
Appendix A	A
Appendix B	B
Index	I

1 Introduction

2 Programming Model

3 Data Organization and Addressing Capabilities

4 Instruction Set Summary

5 Signal Description

6 Access Control Unit

7 Instruction and Data Caches

8 Bus Operation

9 Exception Processing

10 Instruction Execution Timing

11 Electrical Characteristics

12 Ordering Information and Mechanical Data

A Appendix A

B Appendix B

I Index

MOTOROLA

MC68EC040

THIRD-GENERATION 32-BIT EMBEDDED CONTROLLER

PREFACE

The complete documentation package for the MC68EC040 consists of the MC68EC040UM/AD, *MC68EC040 User's Manual*, the MC68040UM/AD, *MC68040 User's Manual*, the *MC68000PM/AD, M68000 Programmer's Reference Manual*, and the MC68040DH/AD, *MC68040 Designer's Handbook*.

The *MC68EC040 User's Manual* describes the items that pertain to the MC68EC040 device which are not contained in the the *MC68040 User's Manual.* Each section will either reference the MC68040 User's Manual or contain the information on the MC68EC040. The *M68000 Programmer's Reference Manual* contains the complete instruction set of all the M68000 Family. The *MC68040 Designer's Handbook* contains detailed timing and electrical specifications and system design guidelines and information.

This user's manual is organized the same as the MC68040 User's Manual with exception of the Glossary and Index. Future plans are to incorporate the MC68EC040 documentation into the MC68040 User's Manual.

Section 1. Introduction
Section 2. Programming Model
Section 3. Data Organization and Addressing Capabilities
Section 4. Instruction Set
Section 5. Signal Description
Section 6. Access Control Unit
Section 7. Instruction and Data Caches
Section 8. Bus Operation
Section 9. Exception Processing
Section 10. Instruction Execution Timing
Section 11. Electrical Characteristics
Section 12. Ordering Information and Mechanical Data
Appendix A. 68000 Family Summary
Appendix B. Software Considerations

THIS PAGE
INTENTIONALY
LEFT
BLANK

TABLE OF CONTENTS

Paragraph Number — Title — Page Number

Section 1
Introduction

1.1 Features 1-1
1.2 MC68EC040 Overview 1-2
1.3 MC68EC040 Extensions To The M68000 Family 1-3
1.4 Programming Model 1-4
1.5 Data Types And Addressing Modes 1-6
1.6 Instruction Set Overview 1-8
1.7 Instruction and Data Caches 1-10

Section 2
Programming Model

Section 3
Data Organization and Addressing Capabilities

Section 4
Instruction Set Summary

Section 5
Signal Description

5.4 Bus Transfer Control Signals 5-4
5.7 Processor Control Signals 5-4
5.7.1 Cache Disable ($\overline{\text{CDIS}}$) 5-4
5.7.3 Reset In ($\overline{\text{RSTI}}$) 5-4
5.7.4 Reset Out ($\overline{\text{RSTO}}$) 5-5
5.8 Interrupt Control Signals 5-5
5.8.1 Interrupt Priority Level ($\overline{\text{IPL2-IPL0}}$) 5-5
5.8.2 Interrupt Pending Status ($\overline{\text{IPEND}}$) 5-5
5.8.3 Autovector ($\overline{\text{AVEC}}$) 5-5
5.12 Signal Summary 5-5

TABLE OF CONTENTS(Continued)

Paragraph Number	Title	Page Number
	Section 6 Access Control Unit	
6.1	Effect of $\overline{\text{RSTI}}$ on the ACU	6-1
6.2	Access Control Registers	6-1
6.3	Registers	6-2
6.3.1	Access Control Registers	6-2
	Section 7 Instruction and Data Caches	
	Section 8 Bus Operation	
8.9	Special Modes of Operation	8-2
8.10	Reset Operation	8-2
	Section 9 Exception Processing	
9.1	Exception Processing Sequence	9-1
9.2	Stack Frames	9-3
9.3	Integer Unit Exceptions	9-7
9.3.1	Reset Exception	9-7
9.3.2	Access Error Exception	9-9
9.3.3	Address Error Exception	9-10
9.3.4	Instruction Trap Exception	9-10
9.3.5	Illegal Instruction and Unimplemented Instruction Exception	9-11
9.3.6	Unimplemented Floating-Point Instruction Exception	9-11
9.3.7	Privilege Violation Exception	9-12
9.3.8	Trace Exception	9-12
9.3.9	Format Error Exception	9-14
9.3.10	Interrupt Exceptions	9-14
9.3.11	Breakpoint Instruction Exception	9-19
9.4	Exception Priorities	9-19
9.5	Return From Exceptions	9-21
9.6	Access Fault Recovery	9-23
9.6.1	Access Error Stack Frame	9-23
9.6.1.1	Effective Address	9-24

TABLE OF CONTENTS(Concluded)

Paragraph Number	Title	Page Number
	Section 9	
	Exception Processing	
9.6.1.2	Special Status Word	9-24
9.6.1.3	Writeback Status	9-26
9.6.1.4	Fault Address	9-26
9.6.1.5	Writeback Data	9-26
9.6.2	Address Errors	9-27
9.6.3	Returning from Access Errors	9-27
9.7	Floating-Point Exceptions	9-27
9.7.1	Unimplemented Floating-Point Instructions	9-27
	Section 10	
	Instruction Execution Timing	
	Section 11	
	Electrical Characteristics	
11.1	Maximum Ratings	11-1
11.2	Thermal Characteristics	11-1
11.3	DC Eclectrical Specifications	11-2
11.4	Clock AC Timing Specifications	11-3
11.5	Output AC Timing Specifications	11-4
11.6	Input AC Timing Specifications	11-5
	Section 12	
	Ordering Information and Mechanical Data	
12.1	Ordering Information	12-1
12.2	Pin Assignments	12-2
12.3	Mechanical Data	12-3
	Appendix A	
	M68000 Family Summary	
	Appendix B	
	Software Considerations	

LIST OF FIGURES

Figure Number	Title	Page Number
1-1	MC68EC040 Block Diagram	1-2
1-2	Programming Model	1-5
5-1	Functional Signal Groups	5-2
6-1	Access Control Register Format	6-3
8-38	Initial Power-On Reset Timing	8-3
8-39	Normal Reset Timing	8-4
9-1	Reset Operation Flowchart	9-8
9-2	Interrupt Pending	9-15
9-3	Interrupt Recognition	9-17
9-4	Assertion of $\overline{\text{IPEND}}$	9-18
9-5	RTE Instruction for Throwaway Four-Word Frame	9-22
9-6	Access Error Stack Frame	9-24
11-1	Clock Input Timing Diagram	11-3
11-2	Read/Write Timing	11-6
11-3	Bus Arbitration Timing	11-7
11-4	Snoop Hit Timing	11-8
11-5	Snoop Miss Timing	11-9
11-6	Other Signal Timing	11-10

LIST OF TABLES

Table Number	Title	Page Number
1-1	Data Types	1-5
1-2	Addressing Modes	1-6
1-3	Instruction Set Summary	1-8
5-1	Signal Index	5-3
5-10	Signal Summary	5-6
9-1	Exception Vector Assignments	9-3
9-2	Exception Stack Frames	9-5
9-3	Privileged Instructions	9-12
9-4	Tracing Control	9-13
9-5	Interrupt Levels and Mask Values	9-15
9-6	Exception Priority Groups	9-20
9-7	Writeback Data Alignment	9-27

THIS PAGE
INTENTIONALY
LEFT
BLANK

SECTION 1 INTRODUCTION

The MC68EC040 is Motorola's third generation of M68000-compatible, high-performance, 32-bit microprocessors. The MC68EC040 is a embedded controller employing a highly integrated architecture to provide very high performance in a monolithic HCMOS device. The MC68EC040 integrates an MC68040-compatible integer unit, an access control unit (ACU), and independent 4K-byte instruction and data caches. A high degree of an instruction execution parallelism is achieved through the use of a multistage instruction pipeline, multiple internal buses, and a full internal Harvard architecture, including the separate caches for both instruction and data accesses. Cache functionality is enhanced by the inclusion of on-chip bus snooping logic to directly support cache coherency in multi-master applications.

The MC68EC040 is user object-code compatible with previous members of the M68000 Family and is specifically optimized to reduce the execution time of compiler-generated code. The MC68EC040 is pin compatible with the MC68040. The MC68EC040 is implemented in Motorola's latest HCMOS technology, providing an ideal balance between speed, power, and physical device size.

Figure 1-1 provides a simplified block diagram of the MC68EC040. Each ACU consists of an access control register (ACR), a main cache, and a snoop controller. External memory requests from each cache are prioritized by the bus controller, which executes bus transfers on the external bus.

1.1 FEATURES

The main features of the MC68EC040 include:

- MC68040-Compatible Integer Execution Unit
- 4K-Byte Instruction Cache and 4K-Byte Data Cache Accessible Simultaneously
- 32-Bit, Nonmultiplexed External Address and Data Buses with Synchronous Bursting Interface
- User Object-Code Compatibility with All Earlier M68000 Microprocessors
- Concurrent integer unit, ACU, and Bus Controller Operation Maximizes Throughput
- Low Latency Bus Accesses for Reduced Cache-Miss Penalty
- Multimaster/Multiprocessor Support via Bus Snooping
- 4-Gigabyte Direct Addressing Range

The large on-chip instruction and data caches result in improved system performance and increased functionality while reducing the external memory latency and the dependency on fast memory designs.

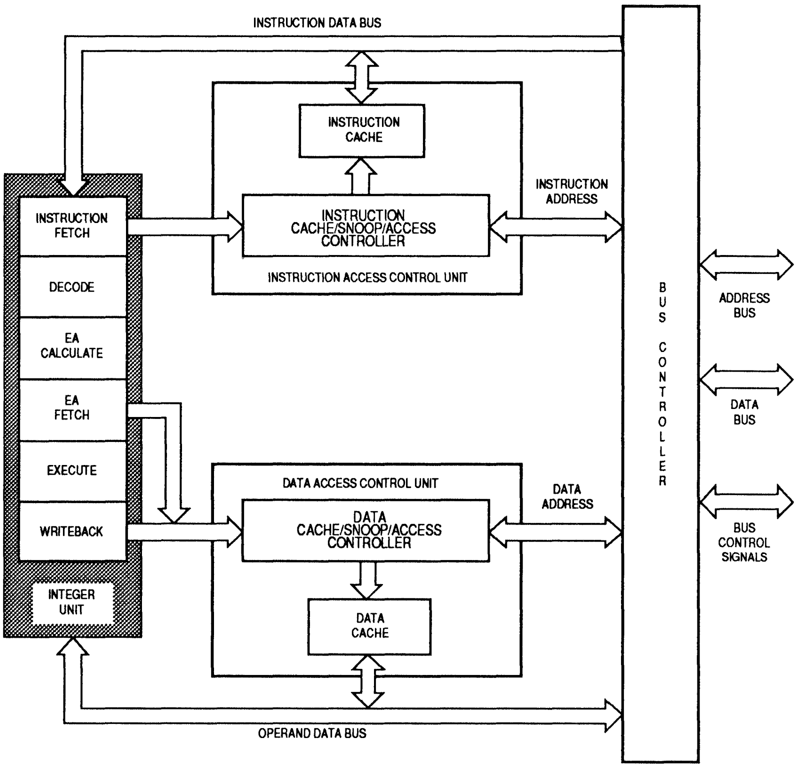


Figure 1-1. MC68EC040 Block Diagram

1.2 MC68EC040 OVERVIEW

The MC68EC040 is an enhanced, 32-bit, HCMOS embedded controller that combines the high-performance integer processing unit of the MC68040 microprocessor with independent 4K-byte data and instruction caches. The MC68EC040 maintains the 32-bit registers available with the entire M68000 Family as well as the 32-bit address and data paths, a rich instruction set, and versatile addressing modes. Instruction execution proceeds in parallel with accesses to the internal caches, ACU and bus controller activity. Additionally, the integer unit is optimized for high-level language environments. The full addressing range of the MC68EC040 is 4 Gbytes (4,294,967,296 bytes); however, most MC68EC040 systems implement a much smaller physical memory.

The high degree of parallelism in the instruction execution unit is achieved through a multi-stage pipeline, the bus controller, and separate instruction and data caches. The multi-stage pipeline will operate on up to six instructions concurrent with bus controller operations, ACU, and separate instruction and data cache accesses. Multiple internal buses, separate instruction and data caches, and a sophisticated bus controller decouple the high-performance MC68EC040 integer unit from the external bus. The copyback cache mode provides even higher performance by reducing the external bus activity for data writes. The MC68EC040 will burst fill four 32-bit long words into the instruction and data caches, when needed by the integer unit, to increase the decoupling of the controller from the external bus. The sophisticated bus controller can delay writes when the external bus is needed to fetch instruction or data which is not present in the internal caches. By reducing the external bus utilization, the MC68EC040 can sustain a much higher performance without an expensive external memory system. The decoupling of the external bus allows the external memory system to be slower and much less expensive without a significant impact on the MC68EC040 performance.

The instruction and data caches operate independently from the rest of the machine, storing information for fast access by the execution units. Each cache resides on its own internal address bus and data bus, allowing simultaneous access to both. The Harvard-style architecture allows a combined internal transfer rate of 200 Mbytes per second at a 25-MHz clock rate when accessing both the data and instruction caches simultaneously. The data cache provides writethrough or copyback write modes that can be configured on a 16-Mbyte to 4-Gbyte block basis by using the access control unit. Cache coherency is maintained by the bus snooping of the bus controller on the MC68EC040.

The MC68EC040 bus controller supports a high-speed, nonmultiplexed, synchronous external bus interface, which allows the following transfer sizes: byte, word (2 bytes), long word (4 bytes), and line (16 bytes). Line accesses are performed using burst transfers for both reads and writes to provide high data transfer rates. The bus controller unit is capable of handling 50 Mbyte per second of long word transfers and burst transfers of 80 Mbytes per second.

1.3 MC68EC040 EXTENSIONS TO THE M68000 FAMILY

The MC68EC040 integer unit pipeline has been expanded to include effective address calculation and operand fetch, with commonly used effective addressing modes further optimized. Conditional branches are optimized for the more common case of the branch taken, and both execution paths of the branch are fetched and decoded to minimize refilling of the instruction pipeline. The user instruction MOVE16 has been added to the instruction set to support efficient 16-byte memory-to-memory data transfers.

The MC68EC040 has an access control unit (ACU) to provide for two data and two instruction memory blocks. The ACU provides write protection and caching modes to complement the caching mechanisms of the MC68EC040. The ACU will allow for peripherals and shared memory to be updated in a serialized and non-cache mode for a memory block.

Separate 4K-byte on-chip instruction and data caches operate independently, and are accessed in parallel with address translation. Each cache resides on a separate internal address bus and data bus, allowing simultaneous access to both. The data cache provides writethrough or copyback write modes that can be configured on a 16 Mbyte block basis. The caches support external bus snooping to maintain cache coherency in multi-master systems.

The MC68EC040 bus controller supports a high-speed, nonmultiplexed synchronous external bus interface. Burst accesses are supported for both reads and writes to provide high data transfer rates to and from the caches. Additional bus signals support bus snooping and external cache tag maintenance.

1.4 PROGRAMMING MODEL

The MC68EC040 registers are partitioned into two levels of privilege: user and supervisor. User programs, executing in the lower-privilege mode, can only use the resources of the user model. System software executing in the supervisor mode has unrestricted access to all processor resources.

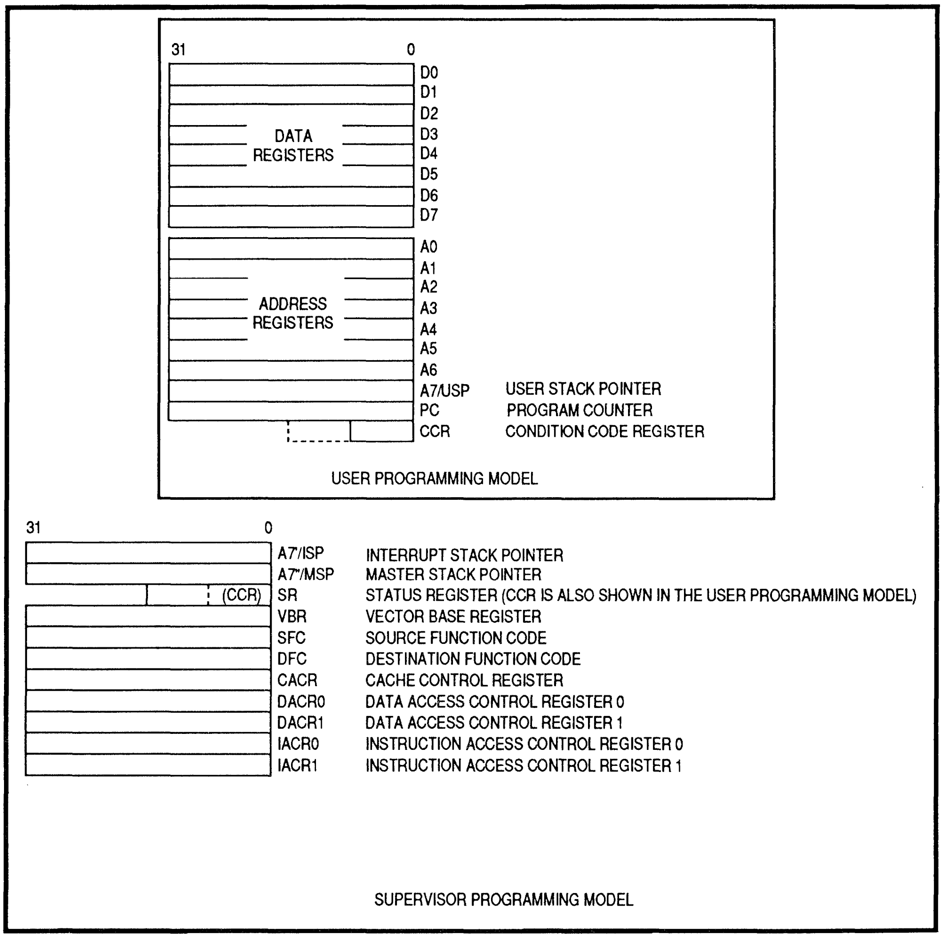


Figure 1-2. Programming Model

The user programming model consists of 16 general-purpose, 32-bit registers and two control registers, and is the same as the user programming model of the M68000 family.

The supervisor programming model is used exclusively by MC68EC040 system programmers to implement operating system functions, and I/O control, and ACU subsystems. This supervisor/user distinction in the M68000 architecture allows all application software to be written to execute in the nonprivileged user mode and migrate to the MC68EC040 from any M68000 platform without modification. Since system software is usually modified by system designers when porting to a new design, the control features are properly placed in the supervisor programming model. For example, the access control registers of the MC68EC040 can only be read or written by the supervisor software. Programming resources of user application programs are unaffected by the existence of the transparent translation registers.

Data registers (D7-D0) contain operands for bit and bit field, byte, word, long-word, and quad-word operations. Address registers (A6-A0) and the stack pointer register (A7) are address registers may be used as software stack pointers or base address registers. Register A7 is also used as an user stack pointer in user mode, and is either the interrupt (A7') or master stack pointer (A7") in supervisor mode. In supervisor mode, the active stack pointer (interrupt or master) is selected based on the setting of the master (M) bit in the status register (SR). In addition, A7-A0 may be used for word and long-word operations. Registers D7-D0 and A7-A0 may be used as index registers.

The program counter (PC) usually contains the address of the instruction being executed by the MC68EC040. 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) contains the condition codes that reflect the results of a previous operation and can be used for conditional instruction execution in a program. The lower byte of the SR is accessible in user mode as the condition code register (CCR). Access to the upper byte of the SR, which contains operation control information, is restricted to the supervisor mode.

As part of exception processing, the vector number of the exception provides an index into the exception vector table. The base address of the exception vector table is stored in the vector base register (VBR).

Alternate source function code (SFC) and destination function code (DFC) registers contain 3-bit function codes, which can be considered extensions of the 32-bit logical address. Function codes are automatically generated by the processor to select address spaces for data and program accesses in the user and supervisor modes. The alternate function code registers are used by certain instructions to explicitly specify the function codes for various operations.

The cache control register (CACR) controls enabling of the on-chip instruction and data caches of the MC68EC040.

There are four access control registers: two for instruction accesses (IACR1-IACR0), and two for data accesses (DACR1-DACR0). These registers allow portions of the address space to be accessed with special access functionality.

1.5 DATA TYPES AND ADDRESSING MODES

The MC68EC040 supports the basic data types shown in Table 1-1. The instruction set supports operations on other data types such as memory addresses.

Table 1-1. Data Types

Operand Data Type	Size	Notes
Bit	1 Bit	—
Bit Field	1-32 Bits	Field of Consecutive Bits
BCD	8 Bits	Packed: 2 Digits/Byte Unpacked: 1 Digit/Byte
Byte Integer	8 Bits	—
Word Integer	16 Bits	—
Long-Word Integer	32 Bits	—
Quad-Word Integer	64 Bits	Any Two Data Registers
16-Byte	128 Bits	Memory-Only, Aligned to 16-Byte Boundary

The MC68EC040 addressing modes are shown in Table 1-2. The register indirect addressing modes support postincrement, predecrement, offset, and indexing, which are particularly useful for handling data structures common to sophisticated applications and high-level languages. The program counter indirect mode also has indexing and offset capabilities; this addressing mode is typically required to support position-independent software. In addition to these addressing modes, the MC68EC040 provides index sizing and scaling features that enhance software performance. Data formats are supported orthogonally by all arithmetic operations and by all appropriate addressing modes.

Table 1-2. Addressing Modes

Addressing Modes	Syntax
Register Direct Addressing	
Data Register Direct	Dn
Address Register Direct	An
Register Indirect	
Address Register Indirect	(An)
Address Register Indirect with Postincrement	(An);pl
Address Register Indirect with Predecrement	-(An)
Address Register Indirect with Displacement	$(d_{16}$,An)
Register Indirect with Index	
Address Register Indirect with Index (8-Bit Displacement)	$(d_8$,An,Xn)
Address Register Indirect with Index (Base Displacement)	(bd,An,Xn)
Memory Indirect	
Memory Indirect Postindexed	([bd,An],Xn,od)
Memory Indirect Preindexed	([bd,An,Xn],od)
Program Counter Indirect with Displacement	(d16,PC)
Program Counter Indirect with Index	
PC Indirect with Index (8-Bit Displacement)	$(d_8$,PC,Xn)
PC Indirect with Index (Base Displacement)	(bd,PC,Xn)
Program Counter Memory Indirect	
PC Memory Indirect Postindexed	([bd,PC],Xn,od)
PC Memory Indirect Preindexed	([bd,PC,Xn],od)
Absolute Data Addressing	
Absolute Short	xxx.W
Absolute Long	xxx.L
Immediate	#<data>

NOTES:

Dn	=	Data Register, D0–D7
An	=	Address Register, A0–A7
d8, d16	=	A twos-complement or sign-extended displacement; added as part of the effective address calculation;size is 8 (d8) or 16 (d16) bits; when omitted, assemblers use a value of zero.
Xn	=	Address or data register used as an index register; form is Xn.SIZE*SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.
bd	=	A twos-complement base displacement; when present, size can be 16 or 32 bits.
od	=	Outer displacement added as part of effective address calculation after any memory indirection; use is optional with a size of 16 or 32 bits.
PC	=	Program Counter
<data>	=	Immediate value of 8, 16, or 32 bits
()	=	Effective Address
[]	=	Used as indirect address to long-word address.

1.6 INSTRUCTION SET OVERVIEW

The instructions provided by the MC68EC040 are listed in Table 1-3. The instruction set has been tailored to support high-level languages and is optimized for those instructions most commonly executed (however, all instructions listed are fully supported). Many instructions operate on bytes, words, and long words, and most instructions can use any of the addressing modes of Table 1-2.

The MC68EC040 instruction set includes MOVE16, a new user instruction which allows high-speed transfers of 16-byte blocks between external devices such as memory-to-memory, or coprocessor-to-memory.

Table 1-3. 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
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BEFXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CINV	Invalidate Cache Lines
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
CPUSH	Push and Invalidate Cache Lines
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

Mnemonic	Description
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVE 16	Move 16 Bytes Block
MOVEA	Move Address
MOVE CCR	Move Condition Code Register
MOVE SR	Move Status Register
MOVE USP	Move User Stack Pointer
MOVEC*	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES*	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	Logical Inclusive OR Immediate
PACK	Pack BCD
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
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

* MC68EC040 additions or alterations to the MC68030 and MC68EC030 instruction set

1.7 INSTRUCTION AND DATA CACHES

Because of the phenomenon of locality of reference, instructions and data that are used in a program have a high probability of being reused within a short time. Additionally, instructions and data operands residing near the instructions and data currently in use also have a high probability of being utilized within a short period. The MC68EC040 takes advantage of these locality characteristics with its two on-chip caches, one for instructions and one for data. Both caches are organized as four-way set-associated with 64 sets of four lines. The processor fills the cache lines using burst mode accesses which transfer the entire line as four long words. This mode of operation not only fills the cache efficiently, but also captures adjacent instruction or data items that are likely to be required in the near future due to locality characteristics of the executing task.

The caches improve the overall performance of the system by reducing the number of bus transfers required by the processor to fetch information from memory and by increasing the bus bandwidth available for other bus masters in the system. To further improve system performance, the data cache in the MC68EC040 supports both copyback and writethrough caching modes for storing write accesses. For writes that hit in copyback pages, the data is used to update the cache line without writing the data to memory immediately. This "dirty" data is copied to memory only when required, either to allow replacement of the cache line by new data, or as a result of explicit flushing of the cache line, resulting in a lower bus bandwidth requirement for the processor. Cache coherency for both caches is maintained by bus snooping logic which allows the MC68EC040 to monitor accesses by an alternate bus master. When an alternate master performs bus transfers, the MC68EC040 can update cache lines which hit during an external write, or source data from dirty data cache lines while inhibiting data from memory during external reads.

SECTION 2
PROGRAMMING MODEL

Refer to **Section 2, Programming Model**, of the MC68040 User's Manual. Ignore all references to the floating-point unit (FPU) in reference to the MC68EC040. The functionality of the transparent translation register have been replaced with the functionality of the access control unit (ACU) registers. All other references to the memory management unit (MMU) are to be ignored.

THIS PAGE
INTENTIONALY
LEFT
BLANK

SECTION 3 DATA ORGANIZATION AND ADDRESSING CAPABILITIES

Refer to **Section 3, Data Organization and Addressing Capabilities**, of the MC68040 User's Manual. Ignore all references to the memory management unit (MMU) and the floating-point unit (FPU) in reference to the MC68EC040.

3

THIS PAGE
INTENTIONALY
LEFT
BLANK

SECTION 4
INSTRUCTION SET SUMMARY

Refer to **Section 4, Instruction Set Summary**, of the MC68040 User's Manual. Ignore all references to the memory management unit (MMU) and the floating-point unit (FPU) in reference to the MC68EC040.

THIS PAGE
INTENTIONALY
LEFT
BLANK

SECTION 5
SIGNAL DESCRIPTION

This section contains brief descriptions of the input and output signals in their functional groups (see Figure 5-1) that apply to the MC68EC040. Refer to **Section 5, Signal Description**, of the MC68040 User's Manual for paragraphs **5.1 ADDRESS BUS (A31–A0)** through **5.3.9 Cache Inhibit Out ($\overline{\text{CIOUT}}$)**, **5.4.1 Transfer Start ($\overline{\text{TS}}$)** through **5.4.6 Transfer Burst Inhibit ($\overline{\text{TBI}}$)**, **5.5 Snoop Control Signals** through **5.6.3 Bus Busy ($\overline{\text{BB}}$)** and **5.9 Status and Clock Signals** through **5.11 Power Supply Connections.** The names, mnemonics, and signal descriptions of the input and output signals for the MC68EC040 are listed in Table 5-1. Guaranteed timing specifications for these signals can be found in MC68EC040/D, *MC68EC040 Technical Summary*, ELECTRICAL CHARACTERISTICS.

NOTE

Assertion and **negation** are used to specify forcing a signal to a particular state. **Assertion** and **assert** refer to a signal that is active or true. **Negation** and **negate** refer to a signal that is inactive or false. These terms are used independently of the voltage level (high or low) that they represent.

5

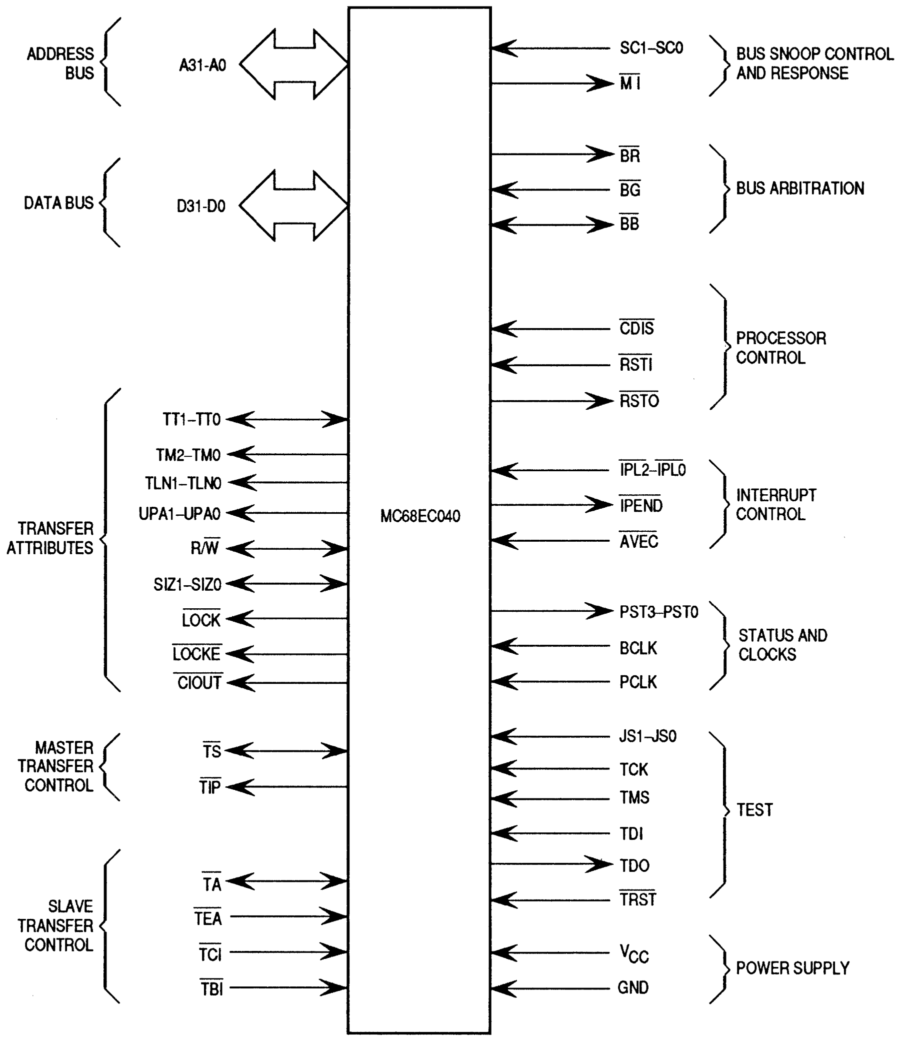


Figure 5-1. Functional Signal Groups

Table 5-1. Signal Index (Sheet 1 of 2)

Signal Name	Mnemonic	Function
Address Bus	A31-A0	32-bit address bus used to address any of 4 Gbytes.
Data Bus	D31-D0	32-bit data bus used to transfer up to 32 bits of data per bus transfer.
Transfer Type	TT1,TT0	Indicates the general transfer type: normal, MOVE16, alternate logical function code, and acknowledge.
Transfer Modifier	TM2,TM0	Indicates supplemental information about the access.
Transfer Line Number	TLN1,TLN0	Indicates which cache line in a set is being pushed or loaded by the current line transfer.
User Programmable Attributes	UPA1,UPA0	User-defined signals, controlled by the corresponding user attribute bits from the access control register.
Read/Write	R/$\overline{W}$	Identifies the transfer as a read or write.
Transfer Size	SIZ1,SIZ0	Indicates the data transfer size. These signals, together with A0 and A1, define the active sections of the data bus.
Bus Lock	$\overline{LOCK}$	Indicates a bus transfer is part of a read-modify-write operation, and that the sequence of transfers should not be interrupted.
Bus Lock End	$\overline{LOCKE}$	Indicates the current transfer is the last in a locked sequence of transfers.
Cache Inhibit Out	$\overline{CIOUT}$	Indicates the processor will not cache the current bus transfer.
Transfer Start	$\overline{TS}$	Indicates the beginning of a bus transfer.
Transfer in Progress	$\overline{TIP}$	Asserted for the duration of a bus transfer.
Transfer Acknowledge	$\overline{TA}$	Asserted to acknowledge a bus transfer.
Transfer Error Acknowledge	$\overline{TEA}$	Indicates an error condition exists for a bus transfer.
Transfer Cache Inhibit	$\overline{TCI}$	Indicates the current bus transfer should not be cached.
Transfer Burst Inhibit	$\overline{TBI}$	Indicates the slave cannot handle a line burst access.
Snoop Control	SC1,SC0	Indicates the snooping operation required during an alternate master access.
Memory Inhibit	$\overline{MI}$	Inhibits memory devices from responding to an alternate master access during snooping operations.
Bus Request	$\overline{BR}$	Asserted by the processor to request bus mastership.
Bus Grant	$\overline{BG}$	Asserted by an arbiter to grant bus mastership to the processor.
Bus Busy	$\overline{BB}$	Asserted by the current bus master to indicate it has assumed ownership of the bus.
Cache Disable	$\overline{CDIS}$	Dynamically disables the internal caches to assist emulator support.
Reset In	$\overline{RSTI}$	Processor reset.
Reset Out	$\overline{RSTO}$	Asserted during execution of a RESET instruction to reset external devices.
Interrupt Priority Level	$\overline{IPL2}$-$\overline{IPL0}$	Provides an encoded interrupt level to the processor.
Interrupt Pending	$\overline{IPEND}$	Indicates an interrupt is pending.

Table 5-1. Signal Index (Sheet 2 of 2)

Signal Name	Mnemonic	Function
Autovector	$\overline{\text{AVEC}}$	Used during an interrupt acknowledge transfer to request internal generation of the vector number.
Processor Status	PST3-PST0	Indicates internal processor status.
Bus Clock	BCLK	Clock input used to derive all bus signal timing.
Processor Clock	PCLK	Clock input used for internal logic timing. The PCLK frequency is exactly 2X the BCLK frequency.
Test Clock	TCK	Clock signal for the IEEE P1149.1 Test Access Port (TAP).
Test Mode Select	TMS	Selects the principle operations of the test-support circuitry.
Test Data Input	TDI	Serial data input for the TAP.
Test Data Output	TDO	Serial data output for the TAP.
Test Reset	$\overline{\text{TRST}}$	Provides an asynchronous reset of the TAP controller.
JTAG Scan	JS1-JS0	Maintains JTAG Scan compatibility with the MC68040.
Power Supply	V_{CC}	Power supply.
Ground	GND	Ground connection.

5.4 BUS TRANSFER CONTROL SIGNALS

The data latch enable (DLE) pin on the MC68040 has been changed to the JTAG scan (JS0) on the MC68EC040 to maintain JTAG Scan compatibility with the MC68040. All other bus transfer control signal functionality is the same on the MC68EC040 as on the MC68040.

5.7 PROCESSOR CONTROL SIGNALS

The following signals control disabling of the caches and support processor and external device initialization. The memory management unit disable ($\overline{\text{MDIS}}$) pin has been removed from the MC68EC040. The pin has been changed to JTAG scan (JS1) to maintain JTAG Scan compatibility with the MC68040.

5.7.1 Cache Disable ($\overline{\text{CDIS}}$)

The cache disable signal dynamically disables the on-chip caches on the next internal cache access boundary. $\overline{\text{CDIS}}$ does not flush the data and instruction caches; entries remain unaltered and become available again after $\overline{\text{CDIS}}$ is negated. Snooping is also unaffected by the assertion of $\overline{\text{CDIS}}$. During a processor reset the $\overline{\text{CDIS}}$ pin will not be used to select the multiplex mode of operation on the MC68EC040. The multiplexed mode of operation has been removed from the MC68EC040.

5.7.3 Reset In ($\overline{\text{RSTI}}$)

This input signal causes the MC68EC040 to enter reset exception processing. The $\overline{\text{RSTI}}$ signal is an asynchronous input that is internally synchronized to the next rising edge of the BCLK signal.

All three-state signals are set to the high-impedance state, and all other outputs are negated when $\overline{\text{RSTI}}$ is recognized. The test pins are not affected by the assertion of $\overline{\text{RSTI}}$.

5.7.4 Reset Out ($\overline{\text{RSTO}}$)

This output is asserted by the MC68EC040 during execution of the RESET instruction to initialize external devices.

5.8 INTERRUPT CONTROL SIGNALS

The following signals control the interrupt functions of the MC68EC040. During a processor reset the levels on the $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ pins will not be used to select the buffer drive. The output buffer drive of the MC68EC040 will be the same as the MC68040 small buffer drive.$\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$

5

5.8.1 Interrupt Priority Level ($\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$)

These input signals provide an indication of an interrupt condition and the encoding of the interrupt level from a peripheral or external prioritizing circuitry. $\overline{\text{IPL2}}$ is the most significant bit of the level number. For example, since the $\overline{\text{IPLn}}$ signals are active low, $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ equal to \$5 corresponds to an interrupt request at interrupt level 2.

5.8.2 Interrupt Pending Status ($\overline{\text{IPEND}}$)

This output signal indicates an interrupt request has been recognized internally and exceeds the current interrupt priority mask in the status register (SR). This output is for use by external devices (other bus masters, for example) to predict processor operation on the following instruction boundaries.

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

This input signal is asserted with $\overline{\text{TA}}$ during an interrupt acknowledge transfer to request internal generation of the vector number. Refer to **Section 8 Bus Operation** for more information on auto vectors.

5.8.4 JTAG Scan (JS1–JS0)

These pins maintains compatibility with the MC68040 JTAG scan patterns. During normal operation these pins must be tied to V_{CC}, GND, or a pullup or pulldown resistor.

5.12 SIGNAL SUMMARY

Table 5-10 provides a summary of the electrical characteristics of the signals discussed in this section.

Table 5-10. Signal Summary

Signal Function	Signal Name	Type	Active	Three-State
Address Bus	A31–A0	Input/Output	High	Yes
Autovector	$\overline{AVEC}$	Input	Low	—
Bus Busy	$\overline{BB}$	Input/Output	Low	Yes
Bus Clock	BCLK	Input	—	—
Bus Grant	$\overline{BG}$	Input	Low	—
Bus Request	$\overline{BR}$	Output	Low	No
Cache Disable	$\overline{CDIS}$	Input	Low	—
Cache Inhibit	$\overline{CIOUT}$	Output	Low	Yes
Data Bus	D31–D0	Input/Output	High	Yes
Ground	GND	Input	—	—
Interrupt Pending	$\overline{IPEND}$	Output	Low	No
Interrupt Priority Level	$\overline{IPL2}$–$\overline{IPL0}$	Input	Low	—
JTAG Scan 0	JS0	Input	—	—
JTAG Scan 1	JS1	Input	—	—
Lock	$\overline{LOCK}$	Output	Low	Yes
Lock End	$\overline{LOCKE}$	Output	Low	Yes
Memory Inhibit	$\overline{MI}$	Output	Low	No
Processor Clock	PCLK	Input	—	—
Processor Status	PST3–PST0	Output	High	No
Read/Write	R/$\overline{W}$	Input/Output	High/Low	Yes
Reset In	$\overline{RSTI}$	Input	Low	—
Reset Out	$\overline{RSTO}$	Output	Low	No
Snoop Control	SC1,SC0	Input	High	—
Transfer Acknowledge	$\overline{TA}$	Input/Output	Low	Yes
Transfer Burst Inhibit	$\overline{TBI}$	Input	Low	—
Transfer Cache Inhibit	$\overline{TCI}$	Input	Low	—
Transfer Error Acknowledge	$\overline{TEA}$	Input	Low	—
Transfer In Progress	$\overline{TIP}$	Output	Low	Yes
Transfer Line Number	TLN1,TLN0	Output	High	Yes
Transfer Modifier	TM2–TM0	Output	High	Yes
Transfer Size	SIZ1,SIZ0	Input/Output	High	Yes
Transfer Start	$\overline{TS}$	Input/Output	Low	Yes
Transfer Type	TT1,TT0	Input/Output	High	Yes
Test Clock	TCK	Input	—	—
Test Data In	TDI	Input	High	—
Test Data Out	TDO	Output	High	Yes
Test Mode Select	TMS	Input	High	—
Test Reset	$\overline{TRST}$	Input	Low	—
User Programmable Attributes	UPA1,UPA0	Output	High	Yes
Power Supply	VCC	Input	—	—

SECTION 6
ACCESS CONTROL UNIT

The MC68EC040 access control unit (ACU) provides a function similar to the mTTRx registers contained in the MC68040. The information provided in this section will describe the four registers contained in the ACU of the MC68EC040. The four registers in the ACU are the data ACU registers (DACR0 and DACR1) and instruction ACU registers (IACR0 and IACR1). The DACRx and IACRx registers can be accessed by using the MOVEC instruction. The ACU allows for marking memory sections with attributes particular to peripherals, shared memory, or other special memory requirements.

6.1 EFFECT OF $\overline{\text{RSTI}}$ ON THE ACU

When the MC68EC040 is reset by the assertion of the reset input ($\overline{\text{RSTI}}$) signal, the E bits of the IACRx, and DACRx registers are cleared, disabling address access control.

6.2 ACCESS CONTROL REGISTERS

Four independent access control registers (DACR0 and DACR1 in the data ACU, IACR0 and IACR1 in the instruction ACU) optionally define four blocks of address space. The blocks of addresses defined by the mACRx registers include at least 16 Mbytes of address space; the four blocks can overlap, or they can be separate.

The following description of the address comparison assumes that the mACRx registers are enabled; however, each mACRx register can be independently disabled. A disabled mACRx register is completely ignored.

When an ACU receives an address, the privilege mode and the eight high-order bits of the address are compared to the block of addresses defined by the two mACRx registers for the ACU. The address space block for each mACRx register is defined by an S field, base address field, and address mask field. The S field allows matching either user or supervisor accesses or both accesses. When a bit in the address mask field is set, the corresponding bit of the base address is ignored in the address comparison and privilege mode. Setting successively higher order bits in the address mask increases the size of the access controlled block.

The address for the current bus cycle and an mACRx register address match when the privilege mode and address bits (not including masked bits) are equal. Each mACRx register can specify write protection for the block. When write protection is enabled for a block, write or read-modify-write accesses to the block are aborted and an access error exception occurs.

By appropriately configuring a access control register, flexible mappings can be specified (refer to **6.3.1 Access Control Registers** for field identification). For example, to control access the user address space, the S field is set to $0, and the ADDRESS MASK is set to $FF in both an IACRx and DACRx register. To control access to supervisor accesses of addresses $00000000-$0FFFFFFF with write protection, the BASE ADDRESS field is set to $0x, the ADDRESS MASK is set to $0F, the W bit is set to one, and the S field is set to $1. The inclusion of independent ACR registers in both the instruction and data ACUs provides an exception to the merged instruction and data address space, allowing different access control for instruction and operand accesses. Also, since the instruction memory unit is only used for instruction prefetches, different instruction and data ACR registers can cause PC relative operand fetches to be translated differently from instruction prefetches.

Each mACRx register can specify the caching mode for addresses in its block. The four caching modes are cachable/writethrough, cachable/copyback, noncachable, and noncachable/serialized. The write through and copyback caching modes force write accesses to either update the cache and write through to memory or to only update the cache, respectively. The noncachable mode forces matching entries in the cache to be pushed and invalidated and performs an external access with the cache inhibit out signal ($\overline{\text{CIOUT}}$) is asserted to signal to external caches that the access should not be cached. The noncachable/serialized mode forces reads and writes within the block to occur in sequence to support I/O devices. Refer to **SECTION 7 INSTRUCTION AND DATA CACHES** of the MC68040 User's Manual for detailed information on caching modes.

Two user page attribute bits (U1 and U0) in each mACRx register are driven on the user page attribute (UPA1 and UPA0) signals if an external bus cycle results from an access translated by the mACRx register. These bits can be programmed by the user to support extended addressing, bus snooping, or other applications.

If either of the mACRx registers match during an access to a memory unit (either instruction or data), the access is completed with the access control unit. If both registers match, the mACR0 status bits are used for the access.

6.3 REGISTERS

The registers of the ACUs described here are part of the supervisor programming model for the MC68EC040.

The four registers that control and provide status information for access control of memory accesses in the MC68EC040 are the four independent access control registers (IACR0, IACR1, DACR0, and DACR1). These registers can be directly accessed only by programs that execute in supervisor mode.

6.3.1 Access Control Registers

The data access control registers (DACR0 and DACR1) and instruction access control registers (IACR0 and IACR1) are 32-bit registers that define blocks of address space that are access control. Addresses in a access control block are used as addresses with two user-defined attributes, cacheability control, and optional write protection. The minimum size block that can be defined by a access control (ACR) register is 16 Mbytes of address space. The ACRs can specify blocks that overlap. ACR0 is used if both ACRs in a memory unit match. The assertion of RSTI clears the E bits of the DACRx and IACRx registers.

The ACRs can be used to specify a block of memory as serialized and cache inhibited as a peripheral section. The ACRs can specify a block of memory as writethrough and cacheable as a shared memory block in a multi-processing application. The ACRs can be configured to support many embedded control applications while maintaining cache integrity. The format of the ACRs is shown in Figure 6-1.

31							24	23							16
	ADDRESS BASE								ADDRESS MASK						
E	S		0	0	0	U1	U0	0	CM		0	0	W	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 6-1. Access Control Register Format

The fields of the access control registers are as follows:

ADDRESS BASE

This 8-bit field is compared with address bits A31-A24. Addresses that match in this comparison (and are otherwise eligible) are access control.

ADDRESS MASK

Since this 8-bit field contains a mask for the ADDRESS BASE field, setting a bit in this field causes the corresponding bit in the ADDRESS BASE field to be ignored. Blocks of memory larger than 16 Mbytes may be obtained by setting some of the address mask bits to ones. The low-order bits of this field are normally set to define contiguous blocks larger than 16 Mbytes, although this is not required.

E—Enable

This bit enables and disables transparent translation of the block defined by this register:

0 = Access control disabled
1 = Access control enabled

S—Supervisor/User Mode

This field specifies the way FC2 is used in matching an address:

00 =	Match only if FC2 is 0 (user mode access)
01 =	Match only if FC2 is 1 (supervisor mode access)
1x =	Ignore FC2 when matching

U1, U0—User Page Attributes

These bits are user defined and are not interpreted by the MC68EC040. U0 and U1 are echoed to the UPA0 and UPA1 signals, respectively, if an external bus transfer results from the access.

CM—Cache Mode

This field selects the cache mode and access serialization for a page as follows:

00 = Cachable, Writethrough—If the CM field indicates writethrough, then the access is considered cachable. A read access to a writethrough page is read from the cache if matching data is found; otherwise, the data is read from memory and used to update the cache. Write accesses always write through to memory and update matching cache lines.

01 = Cachable, Copyback—If the CM field indicates copyback, then the access is considered cachable. A read access to a copyback page reads from the cache if matching data is found; otherwise, the data is read from memory and used to update the cache. Write accesses that hit in the cache update the cache line and set the corresponding dirty status bits without an external bus access. If a write misses in the cache, the needed cache line is read from memory and updated in the cache.

10 = Cache Inhibited, Serialized

11 = Cache Inhibited, Not Serialized—If the CM field of a matching address indicates cache inhibited, the cache is bypassed, and an external bus transfer is performed. The data associated with the access is not cached internally, and the $\overline{\text{CIOUT}}$ signal is asserted during the bus transfer to indicate to external caches that the access should not be cached. If the data is already resident in an internal cache, then this data is pushed from the cache if dirty or invalidated if clean.

If the CM field indicates serialized, then the the sequence of read and write accesses to the page is guaranteed to match the sequence expected due to instruction ordering. Without serialization, the integer unit pipeline architecture can allow read accesses to occur before completion of a writeback for a prior instruction. Serialization also forces the operand read accesses for an instruction to occur only once by preventing the instruction from being interrupted after the operand fetch. Otherwise, the instruction is aborted, and the operand is accessed again when the instruction is later restarted. These guarantees apply only when the CM field is set to serialized and accesses are aligned.

Detailed information on caching modes is available in **7.2 Caching Modes**, and information on serialization, in **Section 8 Bus Operations**.

W—Write Protect

This bit indicates if the transparent block is write protected. If set, write and read-modify-write accesses are aborted as if the resident (R) bit in a table descriptor were clear.

0 =	Read and write accesses permitted
1 =	Write accesses not permitted

THIS PAGE
INTENTIONALY
LEFT
BLANK

SECTION 7
INSTRUCTION AND DATA CACHES

Refer to **Section 7, Instruction and Data Caches**, of the MC68040 User's Manual. Ignore all references to the memory management unit (MMU) and the floating-point unit (FPU) in reference to the MC68EC040.

7

THIS PAGE
INTENTIONALY
LEFT
BLANK

SECTION 8
BUS OPERATION

Refer to **Section 8, Bus Operation**, of the MC68040 User's Manual for paragraphs **8.1 Bus Characteristics** thru **8.8.4 Snoop Write Cycle Intervention**. Ignore all references to the memory management unit (MMU) and the floating-point unit (FPU) in reference to the MC68EC040. Paragraph **8.9 Special Modes of Operation** does not apply to the MC68EC040 and should be ignored. Paragraph **8.10 Reset Operation** of the MC68040 User's Manual has been modified and is included in the section.

The MC68EC040 bus interface supports synchronous data transfers between the processor and other devices in the system. This section provides a functional description of the bus, the signals that control the bus, and the bus cycles provided for data transfer operations. Operation of the bus is defined for transfers initiated by the processor as a bus master, and for transfers initiated by an alternate master that are snooped by the processor as a slave device. Descriptions of the error and halt conditions, bus arbitration, and the reset operation are also included.

Access requests by the processor and other potential bus masters in the system are arbitrated by an external arbiter that prioritizes the requests and determines which device is granted access to the bus. When the MC68EC040 is the bus master, it uses the bus to access instructions and data from memory which are not contained in its internal caches, and to write data to memory. Additional bus transfers are used to acknowledge interrupts and breakpoints.

Bus accesses by another bus master which has been granted control of the bus are monitored (snooped) by the processor when it is not the bus master to allow the processor to intervene in the access if required. Control inputs to the processor allow external logic to specify the required snoop operation to perform for each bus transfer by an alternate master. The processor allows memory to respond if no external action is required; otherwise, memory is inhibited and the processor responds to the access as a slave, supplying modified data from its data cache or writing data to an already modified cache line (for alternate master reads and writes, respectively). The snooping mechanism is optimized to support cache coherency in multi-master applications in which the MC68EC040 is the only caching master.

The 32-bit data bus supports byte, word, long-word, and line (16-byte) bus cycles using a handshaked transfer sequence. Line transfers are normally performed using an efficient burst transfer which provides an initial address and time-multiplexes the data bus to transfer four long words of information to or from the slave device. Slave devices which do not support bursting can burst-inhibit a line transfer, forcing the bus master to complete the access using three additional long word bus cycles. All bus input and output signals are synchronous to the rising edge of the bus clock (BCLK) signal.

The MC68EC040 architecture supports byte, word, and long-word integer operands these operands can be located in memory on any byte boundary. Misaligned accesses to the caches are supported with multiplex and alignment logic; misaligned memory accesses are completed by breaking up the access into a sequence of aligned byte or word bus transfers. The user should be aware that operand misalignment causes the MC68EC040 to perform multiple bus cycles for the operand transfer, and therefore, processor performance is optimized if operands are aligned to their natural boundaries (for example, long-word operands should be on long-word boundaries). Instruction words and their associated extension words must be aligned on word boundaries.

8.9 SPECIAL MODES OF OPERATION

This part of the MC68040 User's Manual does not apply to the MC68EC040. The MC68EC040 does not sample the $\overline{IPL2}$–$\overline{IPL0}$, $\overline{CDIS}$, JS0 (DLE on the MC68040), or JSI ($\overline{MDIS}$ on the MC68040) pins on the rising edge of $\overline{RSTI}$.

8.10 RESET OPERATION

The reset input signal ($\overline{RSTI}$) is asserted by an external device to reset the processor. When power is applied to the system, external circuitry should assert $\overline{RSTI}$ for a minimum of ten BCLK cycles after V_{CC} is within tolerance. Figure 8-38 is a timing diagram of the power-on reset operation, showing the relationships between VCC, $\overline{RSTI}$, and bus signals. The BCLK and PCLK clock signals are required to be stable by the time V_{CC} reaches the minimum operating specification. $\overline{RSTI}$ is internally synchronized for two BCLKS before being used, and must therefore meet the specified setup and hold times to BCLK (specifications #51 and #52) only if recognition by a specific BCLK rising edge is required.

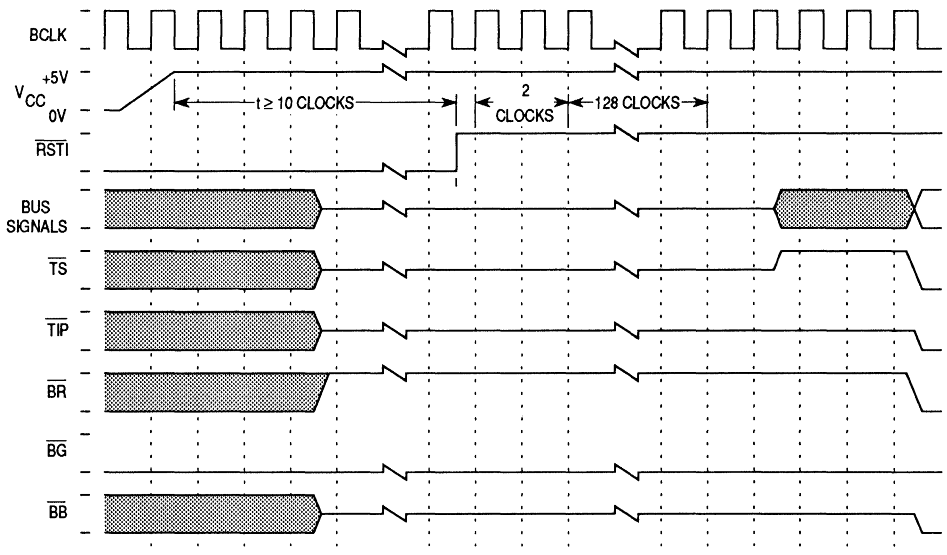


Figure 8-38. Initial Power-On Reset Timing

Once RSTI negates, the processor is internally held in reset for another 128 clock cycles. During the reset period, all three-statable signals three-state, and non-three-statable signals are driven to their inactive state. Once the internal reset signal negates, all bus signals remain in a high-impedance state until the processor is granted the bus. After this, the first bus cycle for reset exception processing begins. In Figure 8-38 the processor assumes implicit ownership of the bus before the first bus cycle begins.

The levels on the CDIS, JS1 (MDIS on the MC68040), and IPL2–IPL0 signals when RSTI negates are not sampled. The special modes of operation (**SECTION 8.9** in the MC68040UM/AD) discussed are not valid on the MC68EC040.

For processor resets, after the initial power-on reset, RSTI should be asserted for at least ten clock periods. Figure 8-39 shows timing associated with a reset when the processor is executing bus cycles. Note that BB and TIP (and TA driven during a snooped access) are asserted before transitioning to a three-state level.

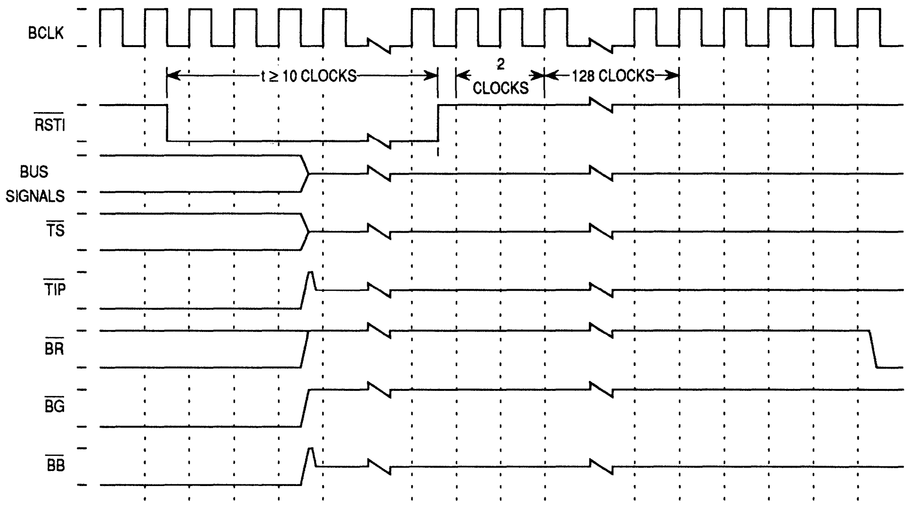


Figure 8-39. Normal Reset Timing

Processor reset causes any bus cycle in progress to terminate as if $\overline{TA}$ or $\overline{TEA}$ had been asserted. Also, the processor initializes registers appropriately for a reset exception. Refer to **SECTION 9 EXCEPTION PROCESSING** reset exceptions.

When a RESET instruction is executed, the processor drives the reset out ($\overline{RSTO}$) signal for 512 BCLK cycles. In this case, the processor resets the external devices of the system, and the internal registers of the processor are unaffected. The external devices connected to the $\overline{RSTO}$ signal are reset at the completion of the reset instruction. An $\overline{RSTI}$ signal that is asserted to the processor during execution of a reset instruction immediately resets the processor and causes the $\overline{RSTO}$ signal to negate. $\overline{RSTO}$ can be logically and'ed with the external signal driving $\overline{RSTI}$ to derive a system reset signal that is asserted for both an external processor reset and execution of a RESET instruction.

SECTION 9 EXCEPTION PROCESSING

This section replaces all of Section 9 of the MC68040 User's Manual.

Exception processing is defined as the activities performed by the processor in preparing to execute a handler routine for any condition that causes an exception. In particular, exception processing does not include execution of the handler routine itself. This section describes the processing for each type of exception, exception priorities, the return from an exception, and bus fault recovery. This section also describes the formats of the exception stack frames.

Processor access errors for a MC68EC040 can be generated by a bus error. Processor accesses of either data items or the instruction stream can result in bus errors. Bus error exceptions must be corrected to complete execution of the current context.

9.1 EXCEPTION PROCESSING SEQUENCE

The MC68EC040 uses a restart exception processing model to minimize interrupt and instruction latency, and to reduce the size of the stack frame (compared to the frame required for a continuation model). Exceptions are recognized at each instruction boundary in the execute stage of the integer pipeline, and force later instructions which have not yet reached the execute stage to be aborted. Instructions which can not be interrupted, such as those that generate locked bus transfers or access serialized pages, are allowed to complete before exception processing begins.

Exception processing occurs in four functional steps. However, all individual bus cycles associated with exception processing (vector acquisition, stacking, etc.) are not guaranteed to occur in the order in which they are described in this section.

In the first step of exception processing, the processor makes an internal copy of the status register (SR). Then the processor sets the S bit in the SR, changing to the supervisor mode. Next, the processor inhibits tracing of the exception handler by clearing the trace enable (T1 and T0) bits. For the reset and interrupt exceptions, the processor also updates the interrupt priority mask.

In the second step, the processor determines the vector number of the exception. For interrupts, the processor performs an interrupt acknowledge cycle to obtain the vector number. For all other exceptions, internal logic provides the vector number. This vector number is used in the last step to calculate the address of the exception vector. Throughout this section, vector numbers are given in decimal notation.

For all exceptions other than reset, the third step is to save the current processor context. The processor creates an exception stack frame on the active supervisor stack and fills it with context information appropriate for the type of exception. Other information may also be stacked, depending on which exception is being processed and the state of the processor prior to the exception. If the exception is an interrupt and the master/interrupt (M) bit of the SR is set, the processor clears the M bit in the SR, and builds a second stack frame on the interrupt stack.

The last step initiates execution of the exception handler. The processor multiplies the vector number by four to determine the exception vector offset. It adds the offset to the value stored in the vector base register (VBR) to obtain the memory address of the exception vector. Next, the processor loads the program counter (and the interrupt stack pointer (ISP) for the reset exception) from the exception vector table entry. After prefetching the first four longwords to fill the instruction pipe, the processor resumes normal processing at the address in the program counter.

All exception vectors are located in supervisor address space, and are accessed using data references. 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.

The MC68EC040 supports a 1024-byte vector table containing 256 exception vectors (See Table 9-1). The first 64 vectors are defined by Motorola and 192 vectors are reserved for interrupt vectors defined by the user. However, external devices may use vectors reserved for internal purposes at the discretion of the system designer.

Table 9-1. Exception Vector Assignments

Vector Number(s)	Vector Offset (Hex)	Assignment
0	000	Reset Initial Interrupt Stack Pointer
1	004	Reset Initial Program Counter
2	008	Access Fault
3	00C	Address Error
4	010	Illegal Instruction
5	014	Integer Divide by Zero
6	018	CHK, CHK2 Instruction
7	01C	TRAPcc, TRAPV Instructions (TRAPcc Used by MC68040, 68030 and 68020)
8	020	Privilege Violation
9	024	Trace
10	028	Line 1010 Emulator (Unimplemented A-Line Opcode)
11	02C	Line 1111 Emulator (Unimplemented F-Line Opcode)
12	030	(Unassigned, Reserved)
13	034	Defined for MC68020 and MC68030, not used by MC68040 and MC68EC040
14	038	Format Error
15	03C	Uninitialized Interrupt
16–23	040–05C	(Unassigned, Reserved)
24	060	Spurious Interrupt
25	064	Level 1 Interrupt Autovector
26	068	Level 2 Interrupt Autovector
27	06C	Level 3 Interrupt Autovector
28	070	Level 4 Interrupt Autovector
29	074	Level 5 Interrupt Autovector
30	078	Level 6 Interrupt Autovector
31	07C	Level 7 Interrupt Autovector
32–47	080–0BC	TRAP #0–15 Instruction Vectors
48 – 55	0C0–0DC	Reserved Floating-Point by the MC68040, not used by MC68EC040
56	0E0	Defined for MC68030 and MC68851, not used by MC68040 and MC68EC040
57	0E4	Defined for MC68851, not used by MC68040 and MC68EC040
58	0E8	Defined for MC68851, not used by MC68040 and MC68EC040
59–63	0EC–0FC	(Unassigned, Reserved)
64–255	100–3FC	User Defined Vectors (192)

9.2 STACK FRAMES

When the processor executes an RTE instruction, it examines the stack frame on top of the active supervisor stack to determine if it is a valid frame and what type of context restoration it requires. The MC68EC040 provides five different stack frames for exception processing and allows for a MC68040 specific stack frame. The set of frames include the four and six word stack frames, four word throwaway stack frame, access error stack frame, and a new eight word unimplemented floating point stack frame. The stack frame that can be generated by the MC68040 and be processed by the MC68EC040 is the floating-point post-instruction stack frame. Table 9-2 summarizes the stack frames.

When the MC68EC040 writes or reads a stack frame, it uses long word operand transfers wherever possible. Using a long-word-aligned stack pointer greatly enhances exception processing performance. The processor does not necessarily read or write the stack frame data in sequential order. The system software should not depend on a particular exception generating a particular stack frame. For compatibility with future devices, the software should be able to handle any type of stack frame for any type of exception. The floating-point post-instruction stack frame will not be generated by the MC68EC040. The MC68040 can not accept the 8 word unimplemented stack frame. The MC68EC040 can handle all MC68040 stack frame formats.

Table 9-2. Exception Stack Frames (Sheet 1 of 2)

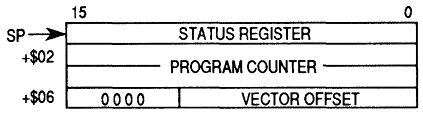
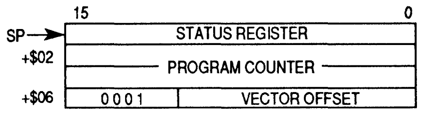
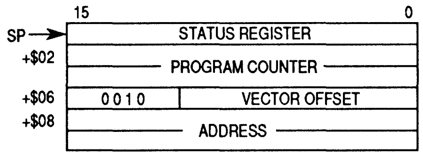
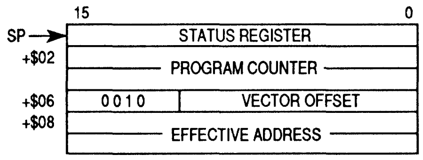
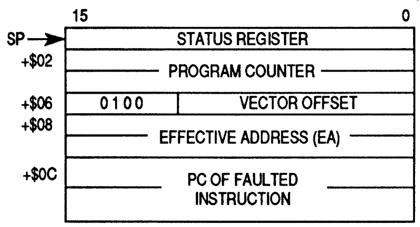
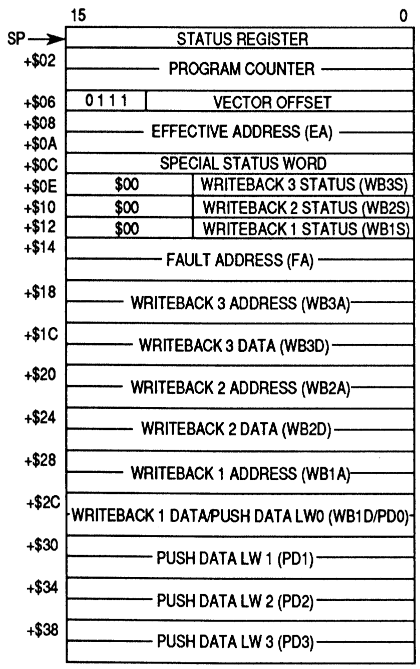
Stack Frame	Exception Types (Stacked PC Points to)	
 15 0 SP +$02 +$06 STATUS REGISTER PROGRAM COUNTER 0000 VECTOR OFFSET FOUR WORD STACK FRAME - FORMAT $0	• Interrupt • Format Error • TRAP #N • Illegal Instruction • A-Line Instruction • Privilege Instruction	(Next Instruction) RTE Instruction (Next Instruction) (Illegal Instruction) (A-Line Instruction) (First word of instruction causing Privilege Violation)
 15 0 SP +$02 +$06 STATUS REGISTER PROGRAM COUNTER 0001 VECTOR OFFSET THROWAWAY FOUR WORD STACK FRAME - FORMAT $1	• Created on Interrupt Stack during interrupt exception processing when transition from master state to Interrupt state occurs	(Next Instruction—Same on master stack)
 15 0 SP +$02 +$06 +$08 STATUS REGISTER PROGRAM COUNTER 0010 VECTOR OFFSET ADDRESS SIX WORD STACK FRAME - FORMAT $2	• CHK • CHK2 • TRAP • TRAPV • Trace • Zero Divide • Address Error	(Next Instruction) ADDRESS is the address of the instruction that caused the exception (Next Instruction) ADDRESS is the calculated effective address for the FP Instruction (Instruction that caused the address error) ADDRESS is the reference address
MC68040 ONLY		
 15 0 SP +$02 +$06 +$08 STATUS REGISTER PROGRAM COUNTER 0010 VECTOR OFFSET EFFECTIVE ADDRESS FLOATING - POINT POST - INSTRUCTION STACK FRAME - FORMAT $3	MC68EC040 can accept this stack frame format, but will not generate a floating-point post instruction stack frame $3.	

Table 9-2. Exception Stack Frames (Sheet 2 of 2)

Stack Frame	Exception Types (Stacked PC Points to)
 15 0 SP +$02 +$06 +$08 +$0C STATUS REGISTER PROGRAM COUNTER 0100 VECTOR OFFSET EFFECTIVE ADDRESS (EA) PC OF FAULTED INSTRUCTION UNIMPLEMENTED FLOATING-POINT STACK FRAME (8 WORDS) – FORMAT $4	• This stack cannot be generated or read by the MC68040 (EA is the address of the trappped floating-point instruction operand)
 15 0 SP +$02 +$06 +$08 +$0A +$0C +$0E +$10 +$12 +$14 +$18 +$1C +$20 +$24 +$28 +$2C +$30 +$34 +$38 STATUS REGISTER PROGRAM COUNTER 0111 VECTOR OFFSET EFFECTIVE ADDRESS (EA) SPECIAL STATUS WORD $00 WRITEBACK 3 STATUS (WB3S) $00 WRITEBACK 2 STATUS (WB2S) $00 WRITEBACK 1 STATUS (WB1S) FAULT ADDRESS (FA) WRITEBACK 3 ADDRESS (WB3A) WRITEBACK 3 DATA (WB3D) WRITEBACK 2 ADDRESS (WB2A) WRITEBACK 2 DATA (WB2D) WRITEBACK 1 ADDRESS (WB1A) WRITEBACK 1 DATA/PUSH DATA LW0 (WB1D/PD0) PUSH DATA LW 1 (PD1) PUSH DATA LW 2 (PD2) PUSH DATA LW 3 (PD3) ACCESS ERROR STACK FRAME (30 WORDS) - FORMAT $7	(Next Instruction) • Bus Error — TEA* Asserted • Access Control Register Fault — mACUx Write Protected region detected an attempted write

9.3 INTEGER UNIT EXCEPTIONS

The following paragraphs describe the external interrupt exception and the different types of exceptions generated by the MC68EC040 integer unit. The following exceptions are discussed:

- Reset
- Access Fault
- Address Error
- Instruction Trap
- Illegal and Unimplemented Instructions
- Unimplemented Floating Point Instructions
- Privilege Violation
- Trace
- Format Error

9.3.1 Reset Exception

Assertion of the reset in ($\overline{\text{RSTI}}$) input signal causes a reset exception. The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. It aborts any processing in progress when it is recognized, and that processing cannot be recovered. Figure 9-1 is a flowchart of the reset exception, which performs the following operations:

1. Clears both trace bits (T1 and T0) in the SR to disable tracing.
2. Places the processor in the interrupt mode of the supervisor privilege mode by setting the supervisor (S) bit and clearing the master (M) bit in the SR.
3. Sets the processor interrupt priority mask to the highest priority level (level seven).
4. Initializes the VBR to zero ($00000000).
5. Clears the enable bits for the on-chip caches in the cache control register.
6. Clears the enable bits for the four ACRx registers.
7. Generates a vector number to reference the reset exception vector (two long words) at offset zero in the supervisor address space.
8. Loads the first long word of the reset exception vector into the ISP.
9. Loads the second long word of the reset exception vector into the program counter (PC).
10. Prefetches the first four longwords beginning at the memory location pointed to by the PC.

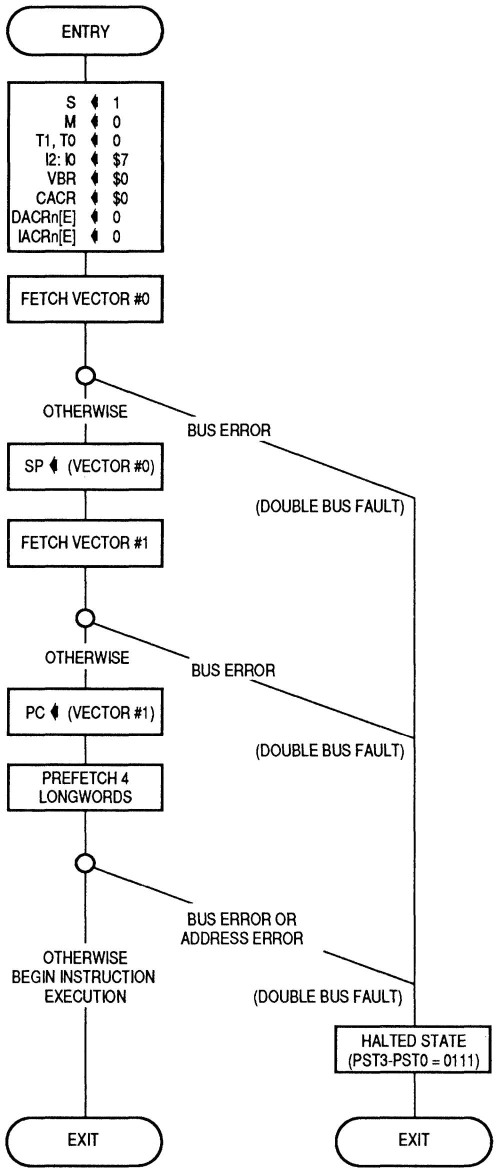


Figure 9-1. Reset Operation Flowchart

After the initial instruction prefetches, program execution begins at the address in the PC. The reset exception does not invalidate entries in the instruction or data caches, nor does it save the value of either the PC or the SR.

Execution of the reset instruction does not cause a reset exception, nor does it affect any internal registers, but it does cause the MC68EC040 to assert the reset out ($\overline{\text{RSTO}}$) signal, resetting all external devices.

9.3.2 Access Error Exception

An access error exception occurs when a data access or instruction prefetch access faults due to an external bus error.

External logic can abort a bus cycle and signal a bus error by asserting the $\overline{\text{TEA}}$ input signal. A bus error on a data write access always results in an access fault exception, and the processor begins exception processing immediately. A bus error on a data read also causes exception processing to begin immediately if the access is a byte, word, or long word access, or if the bus error occurs on the first transfer of a line read. Bus errors on the second, third, or fourth transfers for a data line read cause the transfer to be aborted, but result in a bus error only if the execution unit is specifically requesting the long word being transferred. For example, if a misaligned operand spans the first two long words in the line being read, a bus error on the second transfer causes an exception, but a bus error on the third or last transfer does not (unless the execution unit has generated another operand access which references data in these transfers).

Bus errors which occur during instruction prefetches are deferred until the processor attempts to use the prefetched information. For instance, if a bus error occurs while prefetching other instructions after a change of flow instruction (BRA, JMP, JSR, TRAP#n, etc.), the exception condition is cleared by execution of the new instruction flow. This also applies to the not-taken branch for a conditional branch instruction, even though both sides of the branch are decoded.

The processor begins exception processing for a bus error by making an internal copy of the current SR. The processor then enters the supervisor mode , and clears T1 and T0. The processor generates exception vector number 2, for the bus error vector. It saves the vector offset, PC, and the internal copy of the SR on the stack. The saved PC value is the logical address of the instruction that was executing at the time the fault was detected. This is not necessarily the instruction that initiated the bus cycle, since the processor overlaps execution of instructions. The processor also saves information to allow continuation after a fault for a MOVEM instruction and to support other pending exceptions. The fault address and pending writeback information is saved. The information saved on the stack is sufficient to identify the cause of the bus fault, complete pending writebacks, and recover from the error. Exception handler must complete the pending writebacks. Up to three writebacks can be pending for push errors and data access errors.

If a bus error occurs during the exception processing for a bus error, address error, or reset, or while the processor is loading internal state information from the stack during the execution of an RTE instruction, a double bus fault occurs and the processor enters the halted state as indicated by the PST3-PST0 signals equaling 0111. In this case, the processor does not attempt to alter the current state of memory. Only an external reset can restart a processor halted by a double bus fault.

The supervisor stack has special requirements to ensure that exceptions can be stacked. The stack must be resident with correct protection in the direction of growth, to ensure that exception stacking never has a bus error.

A special case exists for systems that allow arbitration of the processor bus during locked transfer sequences. If a locked transfer occurs and during the write-back a bus error is asserted and an access error exception will be taken. The instruction will be rerun after returning from the access error.

9.3.3 Address Error Exception

An address error exception occurs when the processor attempts to prefetch an instruction from an odd address. (This includes the case of a conditional branch instruction with an odd branch offset, that is not taken.) A prefetch bus cycle is not executed and the processor begins exception processing after the currently executing instructions complete. If another exception is generated by the completion of these instructions, the address error exception is deferred and the new exception is serviced. After exception processing for the address error commences, the sequence is the same as that for bus error exceptions , except that the vector number is 3 and the vector offset in the stack frame refers to the address error vector. A type $2 stack frame is generated that contains the address of the instruction that caused the address error and the address itself (with bit zero of the address cleared). If an address error occurs during the exception processing for a bus error, address error, or reset, a double bus fault occurs.

9.3.4 Instruction Trap Exception

Certain instructions are used to explicitly cause trap exceptions. The TRAP#n instruction always forces an exception, and is useful for implementing system calls in user programs. The TRAPcc, TRAPV, CHK, and CHK2 instructions force exceptions if the user program detects an error, which may be an arithmetic overflow or a subscript value that is out of bounds. The DIVS and DIVU instructions force exceptions if a division operation is attempted with a divisor of zero.

When a trap exception occurs, the processor copies the internally SR , enters the supervisor mode, and clears T1 and T0. The processor generates a vector number according to the instruction being executed. Vector 5 is for DIVx, vector 6 is for CHK and CHK2, and vector 7 is for TRAPcc, and TRAPV instructions. For the TRAP#n instruction, the vector number is 32 plus n. The stack frame saves the trap vector offset, the PC, and the internal copy of the SR on the supervisor stack. The saved value of the PC is the logical address of the instruction following the instruction that caused the trap. For all instruction traps other than TRAP#n, a pointer to the instruction that caused the trap is also saved. Instruction execution resumes at the address in the exception vector after the required instruction prefetches.

9.3.5 Illegal Instruction and Unimplemented Instruction Exceptions

An illegal instruction is an instruction that contains any bit pattern that does not correspond to the bit pattern of a valid MC68EC040 instruction, or a MOVEC instruction with an undefined register specification field in the first extension word. An illegal instruction exception corresponds to vector number 4, and occurs when the processor attempts to execute an illegal instruction.

An illegal instruction exception is also taken after a breakpoint acknowledge bus cycle is terminated, whether by the assertion of the transfer acknowledge ($\overline{TA}$) or the transfer error acknowledge ($\overline{TEA}$) signal.

Instruction word patterns with bits [15:12] equal to $A are referred to as unimplemented instructions with A-line opcodes. When the processor attempts to execute an unimplemented instruction with an A-line opcode, an exception is generated with vector number 10, permitting efficient emulation of unimplemented instructions.

Instructions that have bits [15:12] of the first word equal to $F and do not correspond to legal instructions for the MC68EC040 are treated as unimplemented instructions with F-line opcodes when an execution is attempted. The exception vector number for an unimplemented instruction with an F-line opcode is number 11.

Exception processing for illegal and unimplemented instructions is similar to that for instruction traps. When the processor has identified an illegal or unimplemented instruction, it initiates exception processing instead of attempting to execute the instruction. The processor copies the SR, enters the supervisor mode, and clears the trace bits, disabling further tracing. The processor generates the vector number, either 4, 10, or 11, according to the exception type. The illegal or unimplemented instruction vector offset, current PC, and copy of the SR are saved on the supervisor stack, with the saved value of the PC being the address of the illegal or unimplemented instruction. Instruction execution resumes at the address contained in the exception vector. It is the responsibility of the handling routine to adjust the stacked program counter if the instruction is emulated in software or is to be skipped on return from the handler.

9.3.6 Unimplemented Floating-Point Instruction Exception

Instructions that correspond to legal MC68040 floating point or MC68881/MC68882 instructions but are not implemented in the MC68EC040 are defined as unimplemented floating point instructions. Like other unimplemented instructions, an F-line exception is generated when an unimplemented floating-point instruction is encountered. To aid in emulation of these instructions, the processor partially decodes the instruction to determine the effective address of the memory operand if required and places the EA on the unimplemented exception stack frame before taking the F-line exception.

Exception processing for unimplemented floating-point instructions is slightly different from that for other unimplemented instructions. A longer stack frame is created that also contains the calculated effective address of the memory operand, the PC of the faulted instruction and PC of the next instruction. The stacked PC (SP+$02) points to the next instruction to be executed after the unimplemented floating point instruction.

9.3.7 Privilege Violation Exception

In order to provide system security, the instructions listed in Table 9-3 are privileged. An attempt to execute one of the privileged instructions while at the user mode causes a privilege violation exception.

Table 9-3. Privileged Instructions

ANDI to SR	MOVE to SR
CINV	MOVE USP
CPUSH	ORI TO SR
EORI to SR	PTEST
MOVEC	RESET
MOVES	RTE
MOVE from SR	STOP

Exception processing for privilege violations is similar to that for illegal instructions. When the processor identifies a privilege violation, it begins exception processing before executing the instruction. The processor copies the SR, enters the supervisor mode, and clears T1 and T0. The processor generates vector number 8, saves the privilege violation vector offset and the current PC value, and the internal copy of the SR on the supervisor stack. The saved value of the PC is the logical address of the first word of the instruction that caused the privilege violation. Instruction execution resumes after the required prefetches from the address in the privilege violation exception vector.

9.3.8 Trace Exception

To aid in program development, the M68000 processors include an instruction- by-instruction tracing capability. The MC68EC040 can be programmed to trace all instructions or only instructions that change program flow. In the trace mode, an instruction generates a trace exception after it completes execution, allowing a debugger program to monitor execution of a program.

The T1 and T0 bits in the supervisor portion of the SR control tracing. The state of these bits when an instruction begins execution determines whether the instruction generates a trace exception after the instruction completes. Clearing the T1 bit and setting the T0 bit causes an instruction that forces a change of flow to take a trace exception. Instructions that increment the PC normally do not take the trace exception. Instructions that are traced in this mode include all branches, jumps, instruction traps, and returns. This mode also includes SR manipulations, because the processor must re-prefetch instruction words to fill the pipe again any time an instruction that can modify the SR is executed. Table 9-4 shows the different trace modes.

Table 9-4. Tracing Control

T1	T0	Tracing Function
0	0	No Tracing
0	1	Trace on Change of Flow (BRA, JMP, etc.)
1	0	Trace on Instruction Execution (Any Instruction)
1	1	Undefined, Reserved

In general terms, a trace exception is an extension to the function of any traced instruction. The execution of a traced instruction is not complete until the trace exception processing is completed. If an instruction does not complete due to a bus error or address error exception, trace exception processing is deferred until after the execution of the suspended instruction is resumed and the instruction execution completes. If an interrupt is pending at the completion of an instruction, the trace exception processing occurs before the interrupt exception processing starts. If an instruction forces an exception as part of its normal execution, the forced exception processing occurs before the trace exception is processed.

When the processor is in the trace mode and attempts to execute an illegal or unimplemented instruction, that instruction does not cause a trace exception since it is not executed. This is of particular importance to an instruction emulation routine that performs the instruction function, adjusts the stacked program counter to skip the unimplemented instruction, and returns. Before returning, the trace bits of the SR on the stack should be checked. If tracing is enabled, the trace exception processing should be emulated also, in order for the trace exception handler to account for the emulated instruction.

The exception processing for a trace starts at the end of normal processing for the traced instruction, and before the start of the next instruction. The processor makes an internal copy of the SR, and enters the supervisor mode. It also clears the T0 and T1 bits of the SR, disabling further tracing. The processor supplies vector number 9 for the trace exception, and saves the trace exception vector offset, PC value, and the copy of the SR on the supervisor stack. The saved value of the PC is the logical address of the next instruction to be executed. Instruction execution resumes after the required prefetches from the address in the trace exception vector.

The STOP instruction does not perform its function when it is traced. A STOP instruction that begins execution with T1=1 and T0=0 forces a trace exception after it loads the SR. Upon return from the trace handler routine, execution continues with the instruction following the STOP, and the processor never enters the stopped condition.

9.3.9 Format Error Exception

Just as the processor checks that prefetched instructions are valid, the processor also performs some checks of data values for control operations. The RTE instruction checks the validity of the stack format code.

If any of the checks previously described determine that the format of the data is improper, the instruction generates a format error exception. This exception saves a format $0 stack frame, generates exception vector number 14, and continues execution at the address in the format exception vector. The stacked PC value is the logical address of the instruction that detected the format error.

9.3.10 Interrupt Exceptions

When a peripheral device requires the services of the MC68EC040, or is ready to send information that the processor requires, it may signal the processor to take an interrupt exception. The interrupt exception transfers control to a routine that responds appropriately.

The peripheral device uses the active low interrupt priority level signals ($\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$) to signal an interrupt condition to the processor and to specify the priority of that condition. The three signals encode a value of zero through seven ($\overline{\text{IPL0}}$ is the least-significant bit). High levels on all three signals correspond to no interrupt requested (level 0) and low levels on $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ correspond to interrupt request level 7. Values one through seven specify one of seven levels of interrupts; level seven has the highest priority. External circuitry can chain or otherwise merge signals from devices at each level, allowing an unlimited number of devices to interrupt the processor.

The $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ interrupt signals must maintain the interrupt request level until the MC68EC040 acknowledges the interrupt in order to guarantee that the interrupt is recognized. The MC68EC040 continuously samples the $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ signals on consecutive rising edges of BCLK in order to synchronize and debounce these signals. An interrupt request that is held constant for two consecutive clock periods is considered a valid input. Although the protocol requires that the request remain until the processor runs an interrupt acknowledge cycle for that interrupt value, an interrupt request that is held for as short a period as two clock cycles could be recognized.

The SR of the MC68EC040 contains an interrupt priority mask (I2, I1, I0, bits 10-8). The value in the interrupt mask is the highest priority level that the processor ignores. When an interrupt request has a priority higher than the value in the mask, the processor makes the request an interrupt pending. Figure 9-2 is a flowchart of the procedure for making an interrupt pending.

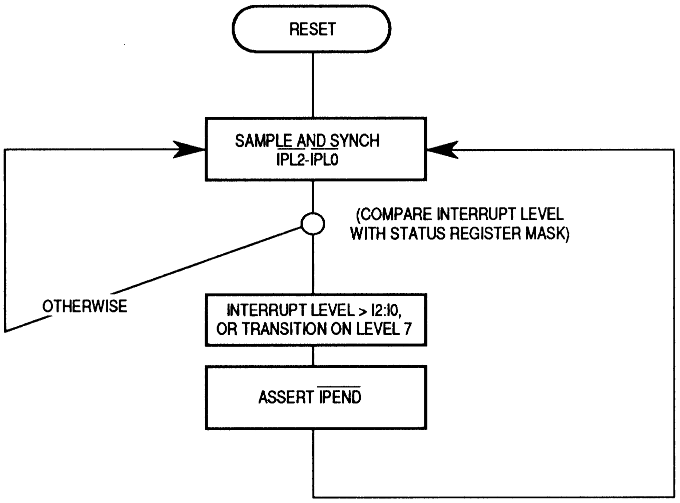

RESET
SAMPLE AND SYNCH IPL2-IPL0
(COMPARE INTERRUPT LEVEL WITH STATUS REGISTER MASK)
OTHERWISE
INTERRUPT LEVEL > I2:I0, OR TRANSITION ON LEVEL 7
ASSERT IPEND

Figure 9-2. Interrupt Pending Procedure

When several devices are connected to the same interrupt level, each device should hold its interrupt priority level constant until its corresponding interrupt acknowledge cycle to ensure that all requests are processed.

Table 9-5 lists the interrupt levels, the states of $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ that define each level, and the mask value that allows an interrupt at each level.

Table 9-5. Interrupt Levels and Mask Values

Requested Interrupt Level	Control Line Status			Interrupt Mask Level Required for Recognition
	$\overline{\text{IPL2}}$	$\overline{\text{IPL1}}$	$\overline{\text{IPL0}}$	
0‡	High	High	High	N/A‡
1	High	High	Low	0
2	High	Low	High	0–1
3	High	Low	Low	0–2
4	Low	High	High	0–3
5	Low	High	Low	0–4
6	Low	Low	High	0–5
7	Low	Low	Low	0–7

‡Indicates that no interrupt is requested.

Priority level seven, the non-maskable interrupt (NMI), is a special case. Level seven interrupts cannot be masked by the interrupt priority mask and they are transition sensitive. The processor recognizes an interrupt request each time the external interrupt request level changes from some lower level to level seven, regardless of the value in the mask. Figure 9-3 shows two examples of interrupt recognitions, one for level six and one for level seven. When the MC68EC040 processes a level 6 interrupt, the SR mask is automatically updated with a value of 6 before entering the handler routine so that subsequent level six interrupts are masked. Provided that no instruction lowers the mask value is executed, the external request can be lowered to level three and then raised back to level six and a second level six interrupt is not processed. However, if the MC68EC040 is handling a level seven interrupt (status register mask set to 7) and the external request is lowered to level three and than raised back to level seven, a second level seven interrupt is processed. The second level seven interrupt is processed because the level seven interrupt is transition sensitive. A level seven interrupt is also generated by a level comparison if the request level and mask level are at seven and the priority mask is then set to a lower level (with the MOVE to SR or RTE instruction, for example). As shown in Figure 9- 3 for level six interrupt request level and mask level, this is the case for all interrupt levels.

LEVEL 6 EXAMPLE

	EXTERNAL $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$		SR MASK (12:10		ACTION	
	100 ($3)		101 ($5)			INITIAL CONDITIONS
IF	101 ($6)	THEN	110 ($6)	AND	LEVEL 6 INTERRUPT	(LEVELCOMPARISON)
IF	100 ($3)	AND STILL	110 ($5)	THEN	NO ACTION	
IF	101 ($6)	AND STILL	110 ($6)	THEN	NO ACTION	
IF	100 ($6)	AND RTE SO THAT	101 ($5)	THEN	LEVEL 6 INTERRUPT	(LEVELCOMPARISON)

LEVEL 7 EXAMPLE

	EXTERNAL $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$		SR MASK (12:10		ACTION	
	100 ($3)		101 ($5)			INITIAL CONDITIONS
IF	001 ($7)	THEN	111 ($7)	AND	LEVEL 7 INTERRUPT	(TRANSITION)
IF	100 ($3)	AND STILL	111 ($7)	THEN	NO ACTION	
IF	000 ($7)	AND STILL	111 ($7)	THEN	LEVEL 7 INTERRUPT	(TRANSITION)
IF	000 ($7)	AND RTE SO THAT	101 ($7)	THEN	LEVEL 7 INTERRUPT	(LEVELCOMPARISON)

Figure 9-3. Interrupt Recognition Examples

Note that a mask value of six and a mask value of seven both inhibit request levels of one through six from being recognized. In addition, neither masks a transition to an interrupt request level of seven. The only difference between mask values of six and seven occurs when the interrupt request level is seven and the mask value is seven. If the mask value is lowered to six, a second level seven interrupt is recognized.

The MC68EC040 asserts $\overline{\text{IPEND}}$ when it makes an interrupt request pending. Figure 9-4 shows the assertion of $\overline{\text{IPEND}}$ relative to the assertion of an interrupt level on the $\overline{\text{IPL}}$ lines. $\overline{\text{IPEND}}$ signals to external devices that an interrupt exception will be taken at an upcoming instruction boundary (following any higher-priority exception).

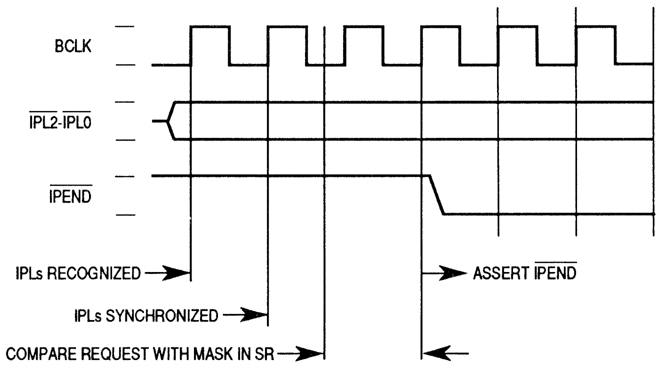


Figure 9-4. Assertion of $\overline{\text{IPEND}}$

When processing an interrupt exception, the processor first makes an internal copy of the SR, sets the mode to supervisor, suppresses tracing, and sets the processor interrupt mask level to the level of the interrupt being serviced. The processor attempts to obtain a vector number from the interrupting device using an interrupt acknowledge bus cycle with the interrupt level number output on the transfer modifier signals. For a device that cannot supply an interrupt vector, the autovector signal ($\overline{\text{AVEC}}$) can be asserted. The MC68EC040 uses an internally generated autovector, which is one of vector numbers 25-31, that corresponds to the interrupt level number. If external logic indicates a bus error during the interrupt acknowledge cycle, the interrupt is considered spurious, and the processor generates the spurious interrupt vector number, 24.

Once the vector number is obtained, the processor saves the exception vector offset, PC value, and the internal copy of the SR on the active supervisor stack. The saved value of the PC is the logical address of the instruction that would have been executed had the interrupt not occurred.

If the M bit of the SR is set, the processor clears the M bit and creates a throwaway exception stack frame on top of the interrupt stack as part of interrupt exception processing. This second frame contains the same PC value and vector offset as the frame created on top of the master stack, but has a format number of 1. The copy of the SR saved on the throwaway frame is exactly the same as that placed on the master stack except that the S bit is set in the version placed on the interrupt stack. (It may or may not be set in the copy saved on the master stack.) The resulting SR (after exception processing) has the S bit set and the M bit cleared.

The processor loads the address in the exception vector into the PC, and normal instruction execution resumes after the required prefetches for the interrupt handler routine.

Most M68000 Family peripherals use programmable interrupt vector numbers as part of the interrupt request/acknowledge mechanism of the system. If this vector number is not initialized after reset and the peripheral must acknowledge an interrupt request, the peripheral usually returns the vector number for the uninitialized interrupt vector, 15.

9.3.11 Breakpoint Instruction Exception

In order to use the MC68EC040 in a hardware emulator, it must provide a means of inserting breakpoints in the emulator code, and of performing appropriate operations at each breakpoint. For the MC68000 and MC68008, this can be done by inserting an illegal instruction at the breakpoint and detecting the illegal instruction exception from its vector location. However, since the VBR on the MC68010, MC68020, MC68EC020, MC68030, MC68EC030, MC68040, MC68LC040, and MC68EC040 allows arbitrary relocation of exception vectors, the exception address cannot reliably identify a breakpoint. The MC68020, MC68EC020, MC68030, MC68EC030, MC68040, MC68LC040, and MC68EC040 processors provide a breakpoint capability with a set of breakpoint instructions, $4848-$484F, for eight unique breakpoints.

When the MC68EC040 executes a breakpoint instruction, it performs a breakpoint acknowledge cycle (read cycle) with an acknowledge transfer type and transfer modifier value of $0. Refer to **SECTION 8 BUS OPERATION** for a description of the breakpoint acknowledge cycle. After external hardware terminates the bus cycle with either $\overline{\text{TA}}$ or $\overline{\text{TEA}}$, the processor performs illegal instruction exception processing.

9.4 EXCEPTION PRIORITIES

When several exceptions occur simultaneously, they are processed according to a fixed priority. Table 9-6 lists the exceptions, grouped by characteristics. Each group has a priority, from 0 through 7, with 0 as the highest priority.

Table 9-6. Exception Priority Groups

Group/ Priority	Exception and Relative Priority	Characteristics
0	Reset	Aborts all processing (instruction or exception) and does not save old context.
1	Data Access Error (ACU Fault or Bus Error)	Aborts current instructions.
2	Floating-Point Pre-Instruction	Exception processing begins before the floating-point instruction is executed. MC68040 only
3	BKPT #n, CHK, CHK2, Divide by Zero, RTE, TRAP #n, TRAPV	Exception processing is part of instruction execution.
	Illegal Instruction, Unimplemented Line A and Line F, Privilege Violation	Exception processing begins before instruction is executed.
	Unimplemented Floating-Point Instruction	Exception processing begins before instruction is executed.
4	Floating-Point Post-Instruction	MC68040 only.
5	Address Error	Reported after all previous instructions and associated exceptions complete.
6	Trace	Exception processing begins when current instruction or previous exception processing is completed.
7	Instruction Access Error (Bus Error)	Reported after all previous instructions and associated exceptions complete.
8	Interrupt	Exception processing begins when current instruction or previous exception processing is completed.

The method used to process exceptions in the MC68EC040 is significantly different from that used in earlier members of the M68000 processor family, due to the restart exception model used. In general, when multiple exceptions are pending, the exception with the highest priority is processed first, and the remaining exceptions are regenerated when the current instruction is restarted. Note that the reset operation clears all other exceptions. Other exceptions to this are noted in the following paragraphs.

As soon as the MC68EC040 has completed exception processing for a condition when an interrupt exception is pending, it begins exception processing for the interrupt exception instead of executing the exception handler for the original exception condition. For example, if simultaneous interrupt and trap exceptions are pending, the exception processing for the trap exception occurs first, followed immediately by exception processing for the interrupt. When the processor resumes normal instruction execution, it is in the interrupt handler, which returns to the trap exception handler.

Exception processing for access error exceptions creates a type $7 stack frame that contains status information that can indicate a pending trace, or unimplemented floating-point instruction exception. The RTE instruction used to return from the access error exception handler checks the status bits for one of these pending exceptions. If one is indicated, the RTE changes the access error stack frame to match the pending exception and fetches the vector for the exception. Instruction execution then resumes in the new exception handler.

Similarly, if a trace exception is pending at the same time a group 3 exception is pending, the trace exception is not reported and the exception handler for the other exception condition must check for the trace condition.

9.5 RETURN FROM EXCEPTIONS

After the processor has completed exception processing for all pending exceptions, the processor resumes normal instruction execution at the address in the vector for the last exception processed. Once the exception handler has completed execution, the processor must return to the system with the system context as it was prior to the exception (if possible). The RTE instruction returns from the handler to the previous system context for any exception.

When the processor executes an RTE instruction, it examines the stack frame on top of the active supervisor stack to determine if it is a valid frame and what type of context restoration it requires. Refer to **9.2 STACK FRAMES** for a description of the stack frame types.

For a normal four word frame (format $0), the processor updates the SR and PC with the data read from the stack, increments the stack pointer by eight, and resumes normal instruction execution.

For the throwaway four word stack (format $1), the processor reads the SR value from the frame, increments the active stack pointer by eight, updates the SR with the value read from the stack, and then begins RTE processing again, as shown in Figure 9-5. The processor reads a new format word from the stack frame on top of the active stack (which may or may not be the same stack used for the previous operation) and performs the proper operations corresponding to that format. In most cases, the throwaway frame is on the interrupt stack and when the SR value is read from the stack, the S and M bits are set. In that case, there is a normal four word frame on the master stack. However, the second frame may be any format (even another throwaway frame) and may reside on any of the three system stacks.

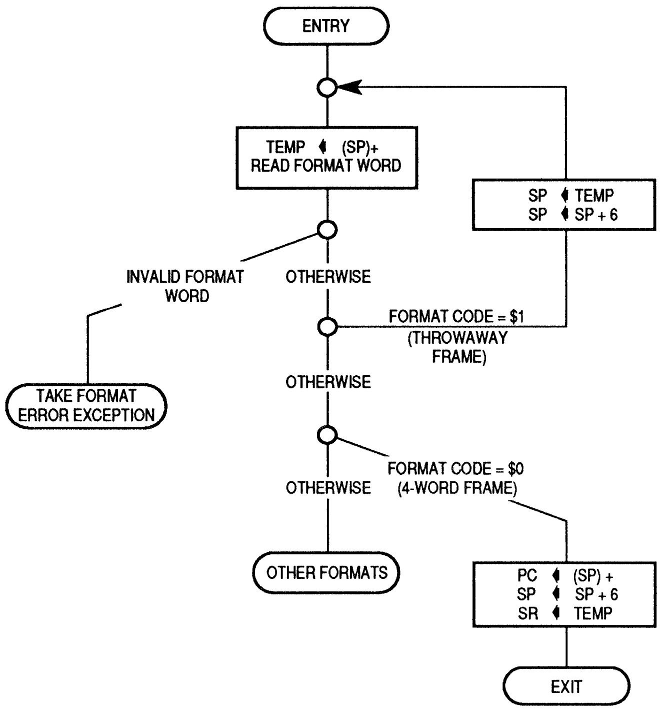


Figure 9-5. RTE Instruction for Throwaway Four-Word Frame

9

For the normal six word stack frame (format $2), the processor restores the SR and PC values from the stack, increments the active supervisor stack pointer by twelve, and resumes normal instruction execution.

For a floating-point post-instruction stack frame (format $3), the processor restores the SR and PC values from the stack, increments the active supervisor stack pointer by twelve, then resumes normal instruction execution. This frame can be processed, but not generated by the MC68EC040. Only the MC68040 can generate a stack frame format $3.

For a unimplemented floating-point instruction stack frame (format $4), the processor restores the SR and PC values from the stack, increments the active supervisor stack pointer by 16, then resumes normal instruction execution.

For the access error stack frame (format $7), the processor restores the SR and PC values from the stack, and checks the 3 continuation status bits in the special status word (SSW) on the stack. If none of the bits are set, the processor increments the active supervisor stack pointer by 30, and resumes normal instruction execution. If the MOVEM continuation bit is set, the processor restores the calculated effective address (EA) from the stack frame, increments the active supervisor stack pointer by 30, and restarts the MOVEM instruction at a point after the EA calculation. All operand accesses for the MOVEM that occurred before the faulted access are repeated. If a continuation bit is set for a pending trace, or unimplemented floating-point instruction the processor restores the calculated EA from the stack frame, increments the active supervisor stack pointer by 30, and immediately begins exception processing for the pending exception. The processor sets only one of the continuation bits when the access error stack frame is created. If multiple bits are set by the access error exception handler, operation of the RTE instruction is undefined.

If the frame format field in the stack frame contains an illegal format code, a format exception occurs. If a format error or access fault exception occurs during the frame validation sequence of the RTE instruction, the processor creates a normal four word or an access fault stack frame below the frame that it was attempting to use. In this way, the faulty stack frame remains intact. The exception handler can examine or repair the faulty frame. In a multiprocessor system, the faulty frame can be left to be used by another processor of a different type when appropriate.

9.6 ACCESS FAULT RECOVERY

Processor accesses of either data items or the instruction stream can result in bus errors. Bus error exceptions must be corrected to complete execution of the current context.

9.6.1 Access Error Stack Frame

A 30-word access error stack frame is created for data and instruction access faults other than instruction address errors. In addition to information about the current processor status and the faulted access, the stack frame also contains pending writebacks that must be completed by the access error exception handler. The access error stack frame is shown in Figure 9-6.

Figure 9-6 describes the fields in the access error stack frame.

		OFFSET
STATUS REGISTER (SR)		$0
PROGRAM COUNTER (PC)		$2
0111	VECTOR OFFSET	$6
EFFECTIVE ADDRESS (EA)		$8
SPECIAL STATUS WORD (SSW)		$C
$00	WRITEBACK 3 STATUS (WB3S)	$E
$00	WRITEBACK 2 STATUS (WB2S)	$10
$00	WRITEBACK 1 STATUS (WB1S)	$12
FAULT ADDRESS		$14
WRITEBACK 3 ADDRESS (WB3A)		$18
WRITEBACK 3 DATA (WB3D)		$1C
WRITEBACK 2 ADDRESS (WB2A)		$20
WRITEBACK 2 DATA (WB2D)		$24
WRITEBACK 1 ADDRESS (WB1A)		$28
WRITEBACK 1 DATA/PUSH DAT LW0 (WB1D/PD0)		$2C
PUSH DATA LW1 (PD1)		$30
PUSH DATA LW1 (PD2)		$34
PUSH DATA LW1 (PD3)		$38

Figure 9-6. Access Error Stack Frame

9.6.1.1 EFFECTIVE ADDRESS. The EA contains address information when one of the continuation flags CM or CT in the SSW is set.

9.6.1.2 SPECIAL STATUS WORD. The SSW is one of several registers saved as part of the access error stack frame. The SSW information indicates whether the fault was caused by an access to the instruction stream, data stream, or both and contains status information for the faulted access.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	CU	CT	CM	MA	ATC	LK	RW	X	SIZE		TT		TM		

The first five fields listed below correspond to the TMn, TTn, SIZn, R/$\overline{W}$, and $\overline{LOCK}$ signals for the faulted access.

TM—Transfer Modifier

TT—Transfer Type

SIZE—Transfer Size

The SIZE field corresponds to the original access size. If a data cache line read results from a read miss, and the line read is bus-errored, the SIZE field in the resulting stack frame indicates the size of the original read generated by the execution unit.

RW—Read/Write

LK—Locked Transfer

CM—Continuation — MOVEM Instruction Execution Pending

Set if a data access is bus errored for a MOVEM. Since the memory location or registers used to calculate the EA may get written over by the MOVEM operation, the MC68EC040 internally saves the EA after calculation. When MOVEM is bus errored, a stack frame is created with CM set, and the EA field contains the calculated EA for the instruction. When the RTE is executed, the MOVEM will restart using the EA on the stack (instead of repeating the EA calculate operation), if the address mode is PC relative (mode=111, register=010,011) or indirect with index (mode=110).

CT—Continuation — Trace Exception Pending

CT is set for an access error with a pending trace exception. All pending accesses are allowed to complete after a trace condition is recognized — if any of these accesses fault, the resulting stack frame has the CT bit set and the EA field contains the address of the instruction being traced. When an RTE is executed with CT set, the MC68EC040 will move the words on the stack at offset $00-$0b from the current SP to offset $30-$3b, adjust the stack pointer by +$30, and change the stack frame format type to $2 before fetching the trace exception vector and jumping directly to trace exception handling. This stack adjustment creates the stack frame which normally would have been created for the trace exception if the pending access had not been bus errored.

CU—Continuation — Unimplemented Floating-Point-Instruction Exception Pending

CU is set for an access error with a pending exception for an unimplemented floating point instruction. Operation is the same as for the CT flag except the RTE fetches the F-line exception vector and the EA field contains the calculated EA determined by the EA field of the unimplemented instruction.

For the case where an unimplemented floating point instruction is traced, the unimplemented exception takes precedence, CU is set, and CT is undefined. The kernel must check for a trace condition using the stacked status register. If true, create required stack frame then jump directly to trace handler.

X—Undefined—Reserved

9.6.1.3 WRITEBACK STATUS. These 8-bit fields contain status information for the three possible writebacks which could be pending after the faulted access. For a data cache line push fault or a MOVE16 write fault, WB1S is zero (invalid).

7	6	5	4	3	2	1	0
V	SIZE		TT		TM		

TM—Transfer Modifier

TT—Transfer Type

SIZE—Transfer Size

V—Valid Write (writeback pending if set)

9.6.1.4 FAULT ADDRESS. The fault address (FA) is the initial address for the access which faulted. The fault address is a physical address only for cache pushes; FA is a logical address for all other cases. For a misaligned access which faults, the FA field contains the address of the first byte of the transfer, regardless of which of the two or three bus transfers for the misaligned access was faulted. For a push fault, the WB1A and FA addresses are the same.

9.6.1.5 WRITEBACK DATA. The writeback data in WB3D and WB2D are register-aligned with byte and word data contained in the least significant byte and word, respectively, of the field. Writeback data in WB1D is memory- aligned, and resides in the byte positions corresponding to the data bus lanes used in writing each byte to memory.

Table 9-7 show this explicitly for each combination of size and A1/A0.

Table 9-7. Writeback Data Alignment

Data Size	Address		Data Alignment	
	A1	A0	WB1D	WB2D, WB3D
Byte	0	0	31:24	7:0
	0	1	23:16	7:0
	1	0	15:8	7:0
	1	1	7:0	7:0
Word	0	0	31:16	15:0
	0	1	23:8	15:0
	1	0	15:0	15:0
	1	1	7:0, 31:24	15:0
Long	0	0	31:0	31:0
	0	1	23:0, 31:24	31:0
	1	0	15:0, 31:16	31:0
	1	1	7:0, 31:8	31:0

NOTE:

For a line transfer fault, the four long words of data PD3-PD0 are already aligned with memory. Bits 31:0 of each field correspond to bits 31:0 of the memory location to be written to, regardless of the value of the address bits A1 and A0 for the writeback address.

9.6.2 Address Errors

For an address error fault, the processor saves a type 2 exception stack frame on the stack. This stack frame contains the PC pointing to the instruction that caused the address error, and the actual address referenced by the instruction. Note that bit zero of the referenced address is cleared on the stack frame. Address error faults must be repaired in software.

9.6.3 Returning from Access Errors

After the bus error exception handler completes all pending operations and executes an RTE to return, the RTE reads only the stack information from offset \$0-\$d in the access error stack frame. For a pending trace exception, unimplemented floating-point instruction exception, the RTE adjusts the stack to match the pending exception and immediately begins exception processing, without requiring the exception to re-occur.

9.7 FLOATING-POINT EXCEPTIONS

9.7.1 Unimplemented Floating-Point Instructions

Floating-point instructions that are supported by the MC68040 and/or MC68881/MC68882 floating-point coprocessors, but are not directly supported by the MC68EC040 in hardware, are defined as unimplemented floating-point instructions. These instructions trap as an F-line exception and must be emulated in software by the F-line exception handler to maintain user object code compatibility.

All MC68040 and/or MC68881/MC68882 floating-point instructions trap as floating-point unimplemented instructions on the MC68EC040. Illegal F-line instructions trap as illegal instructions. The floating point unimplemented instruction stack frame used by the MC68EC040 is not recognized by the MC68040.

Unimplemented floating-point instructions generate an exception that causes a stack frame with a new format code to be generated. The new stack frame contains the SR, PCfor the next instruction, vector offset, effective address, and the PC of faulted instruction. The effective address field contains the calculated effective address of the operand for the faulted floating-point instruction using the addressing mode in which the effective address is calculated. For immediate and register direct addressing modes, this field is $0. PC field SP+$02 contains the address of the next instruction to be executed by the MC68EC040. This value will be restored during RTE execution. The vector offset format number ($4) will be used for this eight-word stack. Note that an MC68040 cannot correctly handle a stack format $4. The PC of the faulted instruction contains a long-word PC of the floating-point instruction that caused the trap to occur. The information is provided so that the iexception handler can determine what floating-point instruction caused the trap to occur is available for software emulation of floating-point instructions.

The MC68EC040 assists the emulation process by distinguishing unimplemented floating-point instructions from other unimplemented F-line instructions. EA is calculated and is saved in the type $4 stack frame generated during exception processing for the unimplemented floating-point instruction before taking the F-line exception. This simplifies and speeds up the emulation process by eliminating the need for the emulation routine to determine the EA, and providing information required to emulate the instruction on the exception stack frame in the supervisor address space. However, the floating-point instruction could reside in user space, so the floating-point unimplemented exception handler may need to access user instruction space.

In more detail, the following processing steps occur for an unimplemented floating-point instruction:

1. When an unimplemented floating-point instruction is encountered, the instruction is partially decoded and the EA is calculated, if required.
2. The processor waits for all previous integer instructions, writebacks, and associated exception processing to complete before beginning exception processing for the unimplemented floating-point instruction. Any access error which occurs in completing the writebacks causes an access error exception, and the resulting stack frame indicates a pending unimplemented floating-point instruction exception. The writebacks are then completed in software by the access error exception handler, and exception processing for the unimplemented floating-point instruction exception begins immediately after return from the access error handler.
3. The processor begins exception processing for the unimplemented floating-point instruction by making an internal copy of the current SR. The processor then enters the supervisor mode, and clears the trace bits (T1, T0). The processor creates a type $4 stack frame, and saves the internal copy of the SR, PC, vector offset, the calculated EA, and the PC of the faulted instruction in the stack frame. The saved PC value is the address of the instruction that follows the unimplemented floating-

point instruction. The processor generates exception vector number 11, for the unimplemented F-line instruction exception vector, fetches the address of the F-line exception handler from the exception vector table, and begins execution of the handler after prefetching instructions to fill the pipeline.

Additional information on floating-point instruction emulation can be found in the MC68040DH/AD, MC68040 Designer's Handbook.

THIS PAGE
INTENTIONALY
LEFT
BLANK

SECTION 10
INSTRUCTION EXECUTION TIMING

Refer to **Section 10, Instruction Execution Timing**, of the MC68040 User's Manual. Ignore all references to the floating-point unit (FPU) in reference to the MC68EC040.

THIS PAGE
INTENTIONALY
LEFT
BLANK

SECTION 11 ELECTRICAL SPECIFICATIONS

11.1 MAXIMUM RATINGS

Signal Name	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	V
Input Voltage	V_{in}	–0.8 to +7.0	V
Maximum Operating Junction Temperature	T_J	110	°C
Storage Temperature Range	T_{stg}	–55 to 150	°C

This device contains protective circuitry 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— PGA PACKAGE

Characteristic	Symbol	Value	Rating
Thermal Resistance — Junction to Case	R_{JC}	3	°C/W

11

11.3 DC ELECTRICAL SPECIFICATIONS (V_{CC} = 5.0 Vdc ±5 %; GND = 0 Vdc)

Characteristic		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2	V_{CC}	V
Input Low Voltage		V_{IL}	GND	0.8	V
Undershoot		—	—	–0.8	V
Input Leakage Current @ 0.5/2.4 V	$\overline{AVEC}$, BCLK, $\overline{BG}$, CDIS, $\overline{IPLn}$, PCLK, $\overline{RSTI}$, SCn $\overline{TBI}$, TLNn, $\overline{TCI}$, TCK, $\overline{TEA}$	I_{in}	20	20	mA
Hi-Z (Off-State) Leakage Current @ 0.5/2.4 V	An, $\overline{BB}$, $\overline{CIOUT}$, Dn, $\overline{LOCK}$, $\overline{LOCKE}$, R/$\overline{W}$, SIZn, $\overline{TA}$, TDO $\overline{TIP}$, TMn, TLNn, $\overline{TS}$, TTn, UPAn	I_{TSI}	20	20	mA
Signal Low Input Current V_{IL} = .8 V	TMS, TDI, $\overline{TRST}$	I_{IL}	–1.1	–0.18	mA
Signal High Input Current V_{IH} = 2.0 V	TMS, TDI, $\overline{TRST}$	I_{IH}	–0.94	–0.16	mA
Output High Voltage I_{OH} = 5 mA		V_{OH}	2.4	—	V
Output Low Voltage I_{OL} = 5 mA		V_{OL}	—	0.5	V
Power Dissipation (T_J = 110°C)		P_D	TBD	TBD	W
Capacitance (see Note) V_{in} = 0 V, f = 1 MHz		C_{in}	—	20	pF

NOTE: Capacitance is periodically sampled rather than 100% tested.
TBD—To Be Determined

11.4 CLOCK AC TIMING SPECIFICATIONS (See Figure 11-1)

Num	Characteristic	25 MHz		Unit
		Min	Max	
	Frequency of Operation	16.67	25	MHz
1	PCLK Cycle Time	20	30	ns
2	PCLK Rise Time	—	1.7	ns
3	PCLK Fall Time	—	1.6	ns
4	PCLK Duty Cycle Measured at 1.5 V	47.5	52.5	%
4a[1]	PCLK Pulse Width High Measured at 1.5 V	9.5	10.5	ns
4b[1]	PCLK Pulse Width Low Measured at 1.5 V	9.5	10.5	ns
5	BCLK Cycle Time	40	60	ns
6, 7	BCLK Rise and Fall Time	—	4	ns
8	BCLK Duty Cycle Measured at 1.5 V	40	60	%
8a[1]	BCLK Pulse Width High Measured at 1.5 V	16	24	ns
8b[1]	BCLK Pulse Width Low Measured at 1.5 V	16	24	ns
9	PCLK, BCLK Frequency Stability	—	1000	ppm
10	PCLK to BCLK Skew	—	9	ns

NOTE 1: Specification value at maximum frequency of operation.

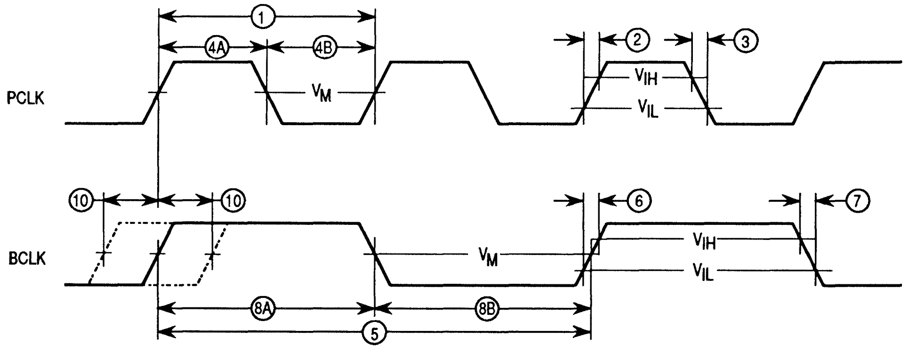


Figure 11-1. Clock Input Timing Diagram

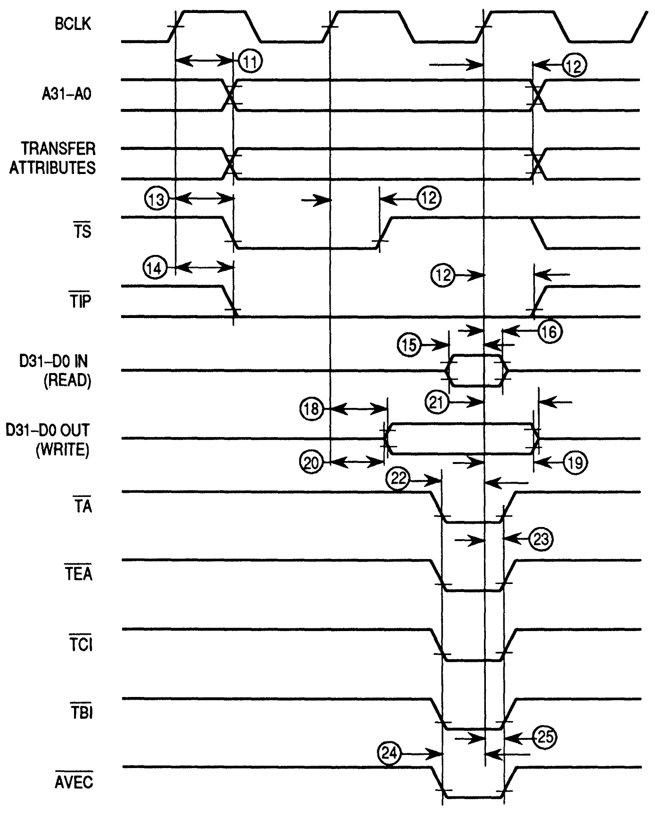
11.5 OUTPUT AC TIMING SPECIFICATIONS (See Note 1; see Figures 11-2–11-6) These output specifications are for only 25 MHz; they must be scaled for lower operating frequencies. Refer to MC68040DH/AD, *MC68040 Designer's Handbook* for further information.

Num	Characteristic	25 MHz[1]		Unit
		Min	Max	
11	BCLK to Address $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, R/$\overline{\text{W}}$, SIZn, TLNn, TMn, TTn, UPAn Valid	9	30	ns
12	BCLK to Output Invalid (Output Hold)	9	—	ns
13	TCLK to $\overline{\text{TS}}$ Valid	9	30	ns
14	BCLK to $\overline{\text{TIP}}$ Valid	9	30	ns
18	BCLK to Data-Out Valid	9	32	ns
19	BCLK to Data-Out Invalid (Output Hold)	9	—	ns
20	BCLK to Output Low Impedance	9	—	ns
21	BCLK to Data-Out High Impedance	9	20	ns
38	BCLK to Address, $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, R/$\overline{\text{W}}$, SIZn, $\overline{\text{TS}}$, TLNn, TMn, TTn, UPAn High Impedance	9	18	ns
39	BCLK to $\overline{\text{BB}}$, $\overline{\text{TA}}$, $\overline{\text{TIP}}$ High Impedance	19	28	ns
40	BCLK to $\overline{\text{BR}}$, $\overline{\text{BB}}$ Valid	9	30	ns
43	BCLK to $\overline{\text{MI}}$ Valid	9	30	ns
48	BCLK to $\overline{\text{TA}}$ Valid	9	30	ns
50	BCLK to $\overline{\text{IPEND}}$, PSTn, $\overline{\text{RSTO}}$ Valid	9	30	ns

NOTE 1: Output timing is specified for a valid signal measured at the pin. Buffer timing is specified driving an unterminated 30-ΩV transmission line with a length characterized by a 2.5-ns one-way propagation delay. Buffer output impedance is typically 30 ΩV; the buffer specifications include approximately 5 ns for the signal to propagate the length of the transmission line and back. Refer to MC68040DH/AD, *MC68040 Designer's Handbook*, for further information on transmission line environments.

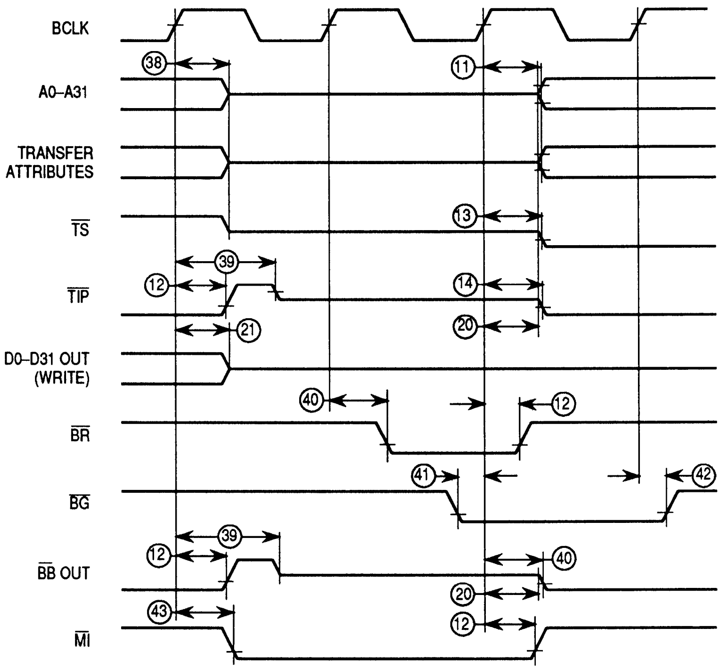
11.6 INPUT AC TIMING SPECIFICATIONS (see Figures 11-2–11-6)

Num	Characteristic	25 MHz		Unit
		Min	Max	
15	Data-In Valid to BCLK (Setup)	5	—	ns
16	BCLK to Data-In Invalid (Hold)	4	—	ns
17	BCLK to Data-In High Impedance (Read Followed by Write)	—	49	ns
22a	$\overline{\text{TA}}$ Valid to BCLK (Setup)	10	—	ns
22b	$\overline{\text{TEA}}$ Valid to BCLK (Setup)	10	—	ns
22c	$\overline{\text{TCI}}$ Valid to BCLK (Setup)	10	—	ns
22d	$\overline{\text{TBI}}$ Valid to BCLK (Setup)	11	—	ns
23	BCLK to $\overline{\text{TA}}$, $\overline{\text{TEA}}$, $\overline{\text{TCI}}$, $\overline{\text{TBI}}$ Invalid (Hold)	2	—	ns
24	$\overline{\text{AVEC}}$ Valid to BCLK (Setup)	5	—	ns
25	BCLK to $\overline{\text{AVEC}}$ Invalid (Hold)	2	—	ns
41a	$\overline{\text{BB}}$ Valid to BCLK (Setup)	7	—	ns
41b	$\overline{\text{BG}}$ Valid to BCLK (Setup)	8	—	ns
41c	$\overline{\text{CDIS}}$ Valid to BCLK (Setup)	10	—	ns
41d	IPLn Valid to BCLK (Setup)	4	—	ns
42	BCLK to $\overline{\text{BB}}$, $\overline{\text{BG}}$, $\overline{\text{CDIS}}$, $\overline{\text{IPLn}}$ Invalid (Hold)	2	—	ns
44a	Address Valid to BCLK (Setup)	8	—	ns
44b	SIZn Valid to BCLK (Setup)	12	—	ns
44c	TTn Valid to BCLK (Setup)	6	—	ns
44d	R/$\overline{\text{W}}$ Valid to BCLK (Setup)	6	—	ns
44e	SCn Valid to BCLK (Setup)	10	—	ns
45	BCLK to Address SIZn, TTn, R/$\overline{\text{W}}$, SCn Invalid (Hold)	2	—	ns
46	$\overline{\text{TS}}$ Valid to BCLK (Setup)	5	—	ns
47	BCLK to $\overline{\text{TS}}$ Invalid (Hold)	2	—	ns
49	BCLK to $\overline{\text{BB}}$ High Impedance (MC68EC040 Assumes Bus Mastership)	—	9	ns
51	$\overline{\text{RSTI}}$ Valid to BCLK	5	—	ns
52	BCLK to $\overline{\text{RSTI}}$ Invalid	2	—	ns
53	Mode Select Setup to $\overline{\text{RSTI}}$ Negated	20	—	ns
54	$\overline{\text{RSTI}}$ Negated to Mode Selects Invalid	2	—	ns



NOTE: Transfer Attribute Signals = UPAn, SIZn, TTn, TMn, TLNn, R/$\overline{\text{W}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, $\overline{\text{CIOUT}}$

Figure 11-2. Read/Write Timing



NOTE: Transfer Attribute Signals = UPAn, SIZn, TTn, TMn, TLNn, R/$\overline{W}$, $\overline{LOCK}$, $\overline{LOCKE}$, $\overline{CIOUT}$

Figure 11-3. Bus Arbitration Timing

11

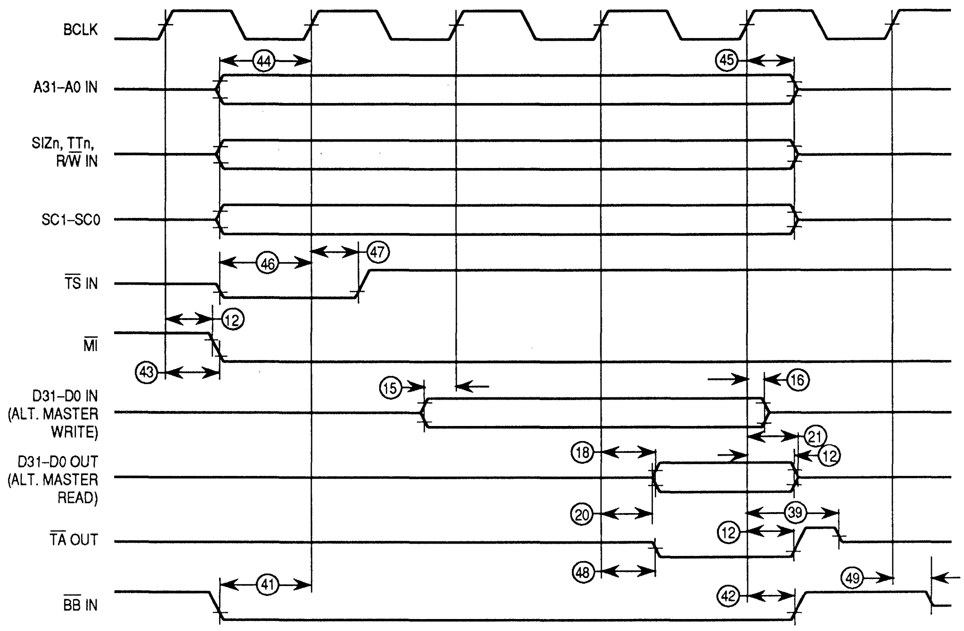


Figure 11-4. Snoop Hit Timing

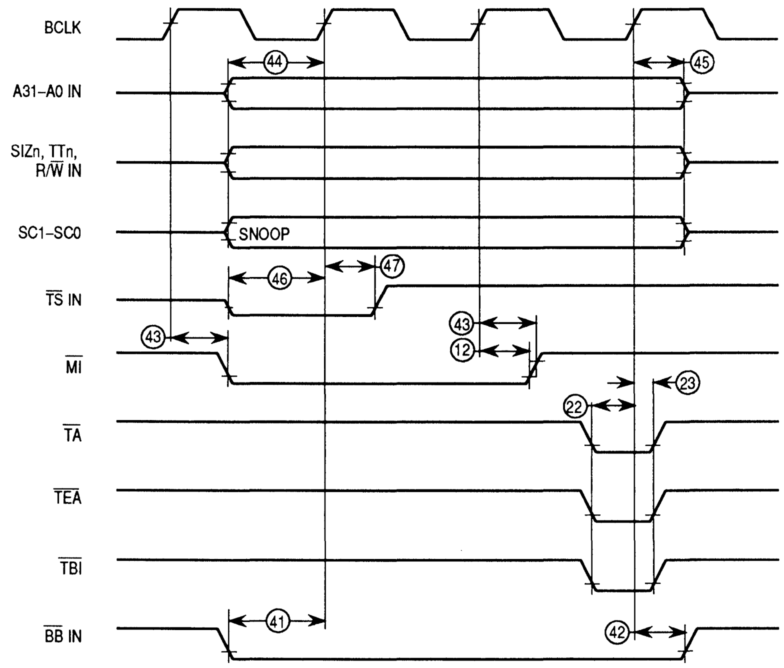


Figure 11-5. Snoop Miss Timing

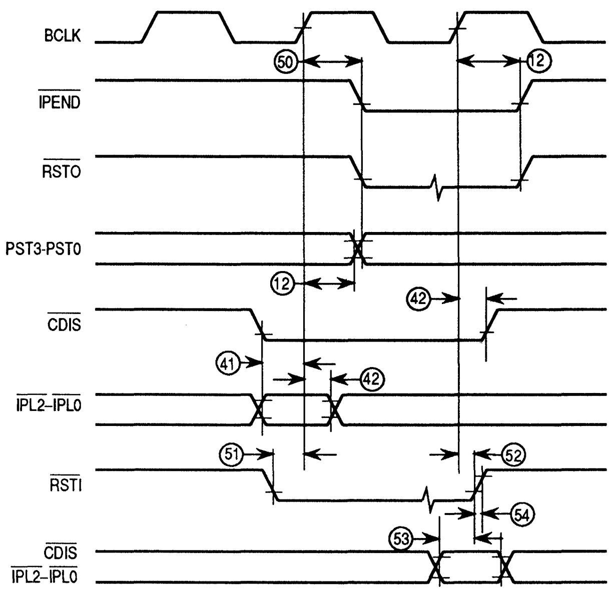


Figure 11-6. Other Signal Timing

SECTION 12 ORDERING INFORMATION AND MECHANICAL DATA

This section contains the pin assignments and package dimensions of the MC68EC040. In addition, detailed information is provided to be used as a guide when ordering.

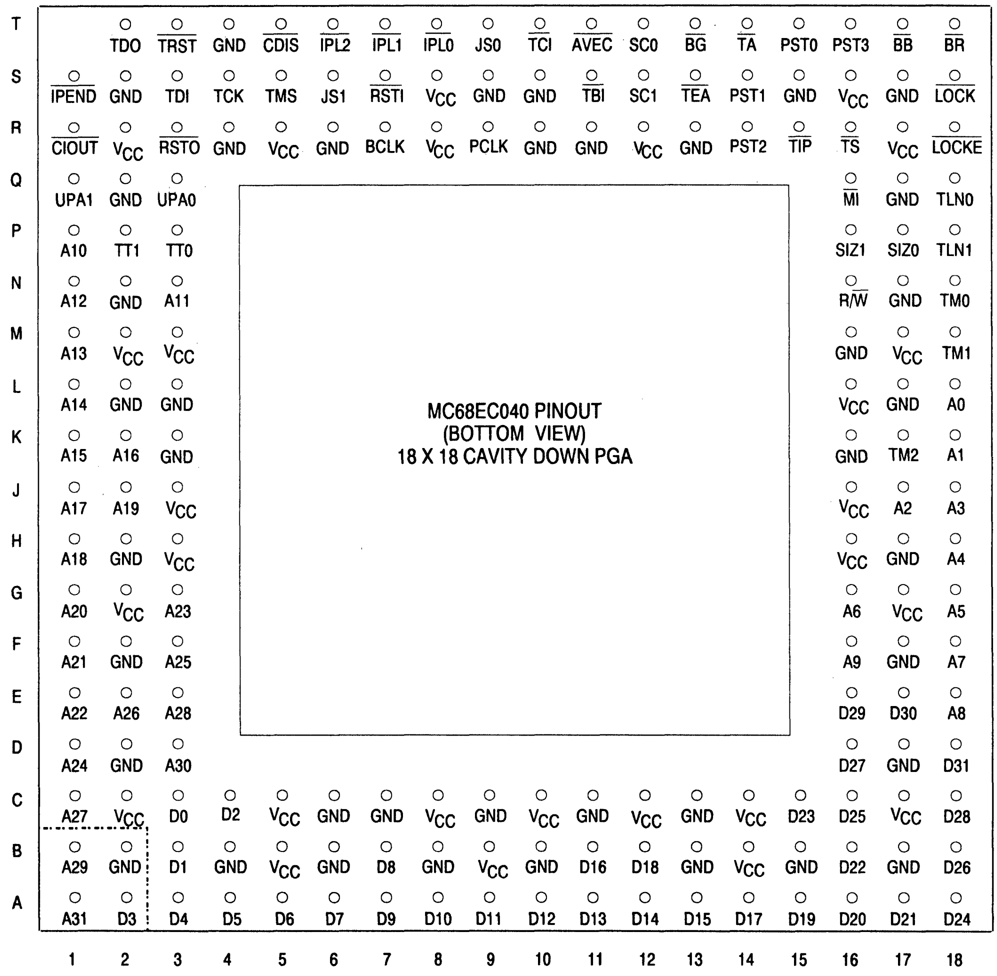
12.1 ORDERING INFORMATION

The following table provides ordering information pertaining to the package type frequency, temperature and Motorola order number for the MC68EC040.

Package Type	Frequency (MHz)	Temperature	Order Number
Pin Grid Array R Suffix	20.0	TBD	MC68EC040R20
Pin Grid Array R Suffix	25.0	TBD	MC68EC040R25
Pin Grid Array R Suffix	33.33	TBD	MC68EC040R33

12.2 PIN ASSIGNMENTS

The MC68EC040 is available in an 179-pin package. The following figure shows the pin assignment of the MC68EC040.

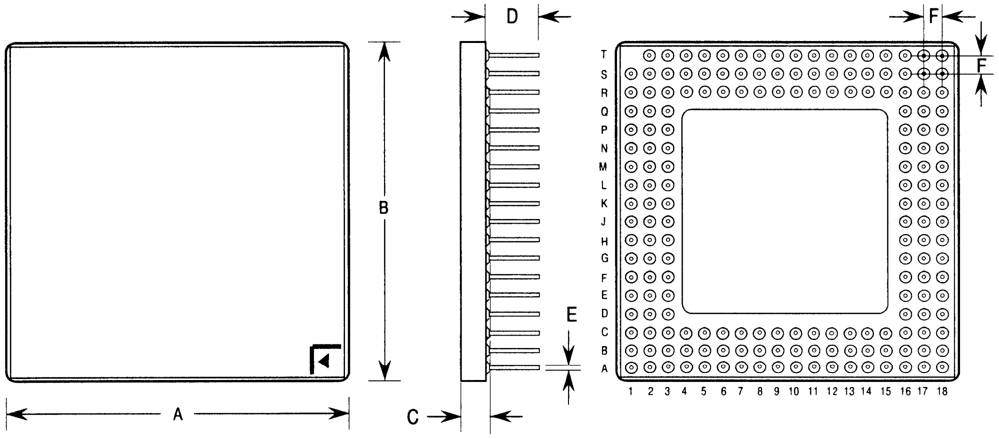

T TDO TRST GND CDIS IPL2 IPL1 IPL0 JS0 TCI AVEC SC0 BG TA PST0 PST3 BB BR
S IPEND GND TDI TCK TMS JS1 RSTI VCC GND GND TBI SC1 TEA PST1 GND VCC GND LOCK
R CIOUT VCC RSTO GND VCC GND BCLK VCC PCLK GND GND VCC GND PST2 TIP TS VCC LOCKE
Q UPA1 GND UPA0 MI GND TLN0
P A10 TT1 TT0 SIZ1 SIZ0 TLN1
N A12 GND A11 R/W GND TM0
M A13 VCC VCC GND VCC TM1
L A14 GND GND VCC GND A0
K A15 A16 GND GND TM2 A1
J A17 A19 VCC VCC A2 A3
H A18 GND VCC VCC GND A4
G A20 VCC A23 A6 VCC A5
F A21 GND A25 A9 GND A7
E A22 A26 A28 D29 D30 A8
D A24 GND A30 D27 GND D31
C A27 VCC D0 D2 VCC GND GND VCC GND VCC GND VCC GND VCC D23 D25 VCC D28
B A29 GND D1 GND VCC GND D8 GND VCC GND D16 D18 GND VCC GND D22 GND D26
A A31 D3 D4 D5 D6 D7 D9 D10 D11 D12 D13 D14 D15 D17 D19 D20 D21 D24
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
MC68EC040 PINOUT
(BOTTOM VIEW)
18 X 18 CAVITY DOWN PGA

	GND	VCC
PLL	S9, R6, R10	R8, S8
Internal Logic	C6, C7, C9, C11, C13,K3, K16, L3, M16, R4,R11, R13, S10, S10,T4	C5, C8, C10, C12, C14,H3, H16, J3, J16, L16,M3, R5, R12
Output Drivers	B2, B4, B6, B8, B10,, B13, B15, B17, D2, D17,F2, F17, H2, H17, L2,L17, N2, N17, Q2, Q17,S2, S15, S17	B5, B9, B14, C2, C17, G2, G17, M2, M17, R2, R17, S16

12.3 MECHANICAL DATA

The following figure provides the package dimensions for the MC68EC040.

Case To Be Determined



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	46.863	47.625	1.845	1.875
B	46.863	47.625	1.845	1.875
C	2.3876	2.9464	.094	.116
D	4.318	4.826	.170	.190
E	0.44	0.55	0.017	0.022
F	2.54 BSC		0.100 BSC	

THIS PAGE
INTENTIONALY
LEFT
BLANK

APPENDIX A
M68000 FAMILY SUMMARY

This Appendix summarizes the characteristics of the microprocessors in the M68000 Family. The M68000PM/AD, *M68000 Programmer's Reference Manual* includes more detailed information on the M68000 Family differences.

Attribute	68000	68008	68010	68020	68030	68040	EC000	EC020	EC030	EC040
Data Bus Size (Bits)	16	8	16	8, 16, 32	8, 16, 32	32	16	8, 16, 32	8, 16, 32	32
Address Bus Size (Bits)	24	20	24	32	32	32	24	24	32	32
Instruction Cache (In Bytes)	—	—	3[1] (Words)	256	256	4096	—	256	256	4096
Data Cache (In Bytes)	—	—	—	—	256	4096	—	—	256	4096

NOTE 1: The MC68010 supports a 3-word cache for the loop mode.

Virtual Interfaces

MC68010, MC68020, MC68030, MC68EC020	Virtual Memory/Machine
M68040, MC68LC040	Virtual Memory
MC68010, MC68020, MC68030, MC68040,MC68LC040	Provide Bus Error Detection, Fault Recovery
MC68030, MC68040, MC68LC040	On-Chip MMU

Coprocessor Interface

MC68000, MC68008, MC68010, MC68EC000, MC68EC040, MC68LC040	Emulated in Software
MC68020, MC68030, MC68EC020, MC68EC030	In Microcode
MC68040	Emulated in Software (On-Chip Floating-Point Unit)

Word/Long Word Data Alignment

MC68000, MC68008, MC68010, MC68EC000	Word/Long Data, Instructions, and Stack Must be Word Aligned
MC68020, MC68030, MC68040, MC68EC020, MC68EC030, MC68EC040, MC68LC040	Only Instructions Must be Word Aligned (Data Alignment Improves Performance)

Control Registers

MC68000, MC68008, MC68EC000	None
MC68010	SFC, DFC, VBR
MC68020, MC68EC020	SFC, DFC, VBR, CACR, CAAR
MC68030	SFC, DFC, VBR, CACR, CAAR, CRP, SRP, TC, TT0, TT1, MMUSR
MC68EC030	SFC, DFC, VBR, CACR, CAAR, ACR0, ACR1, ACUSR
MC68040, MC68LC040	SFC, DFC, VBR, CACR, URP, SRP, TC, DTT0, DTT1, ITT0, ITT1, MMUSR
MC68EC040	SFC, DFC, VBR, CACR, DACR0, DACR1, IACR0, IACR1

Stack Pointer

MC68000, MC68008, MC68010, MC68EC000	USP, SSP
MC68020, MC68030, MC68040, MC68EC040, MC68LC040, MC68EC020, MC68EC030	USP, SSP (MSP, ISP)

Status Register Bits

MC68000, MC68008, MC68010, MC68EC000	T, S, I0/I1/I2, X/N/Z/V/C
MC68020, MC68030, MC68040, MC68EC020, MC68EC030, MC68EC040, MC68LC040	T0, T1, S, M, I0/I1/I2, X/N/Z/V/C

Function Code/Address Space

MC68000, MC68008, MC68EC000	FC2-FC0=7 is Interrupt Acknowledge Only
MC68010, MC68020, MC68030, MC68040, MC68EC000, MC68EC020, MC68EC030, MC68EC040, MC68LC040	FC2-FC0=7 is CPU Space
MC68040, MC68EC040, MC68LC040	User, Supervisor, and Acknowledge

Indivisible Bus Cycles

MC68000, MC68008, MC68010, MC68EC000	Use AS Signal
MC68020, MC68030, MC68EC020, MC68EC030	Use RMC Signal
MC68040, MC68EC040, MC68LC040	Use LOCK and LOCKE Signal

Stack Frames

MC68000, MC68008, MC68EC000	Supports Original set
MC68010	Supports Formats $0, $8
MC68020/MC68030, MC68EC020, MC68EC030	Supports Formats $0, $1, $2, $9, $A, $B
MC68040	Supports Formats $0, $1, $2, $3, $7
MC68EC040, MC68LC040	Supports Formats $0, $1, $2, $3, $4, $7

Addressing Modes

MC68020, MC68030, and MC68040, MC68EC020, MC68EC030, MC68EC040, MC68LC040 Extensions	Memory indirect addressing modes, scaled index, and larger displacements. Refer to specific data sheets for details.

MC68020, MC68030, MC68040, MC68EC040, MC68LC040 Instruction Set Extensions		Applies To				
Instruction	**Notes**	**68020**	**68030**	**68040**	**EC040**	**LC040**
Bcc	Supports 32-Bit Displacements	√	√	√	√	√
BFxxxx	Bit Field Instructions (BCHG, BFCLR, BFEXTS, BFEXTU, BFFFO, BFINS, BFSET, BFTST)	√	√	√	√	√
BKPT	New Instruction Functionally	√	√			
BRA	Supports 32-Bit Displacements	√	√	√	√	√
BSR	Supports 32-Bit Displacement	√	√	√	√	√
CALLM	New Instruction	√				
CAS, CAS2	New Instructions	√	√	√	√	√
CHK	Supports 32-Bit Operands	√	√	√	√	√
CHK2	New Instruction	√	√	√	√	√
CINV	Cache Maintenance Instruction			√	√	√
CMPI	Supports Program Counter Relative Addressing Modes	√	√	√	√	√
CMP2	New Instruction	√	√	√	√	√
CPUSH	Cache Maintenance Instruction			√	√	√
cp	Coprocessor Instructions	√	√			
DIVS/DIVU	Supports 32-Bit and 64-Bit Operands	√	√	√	√	√
EXTB	Supports 8-Bit Extend to 32-Bits	√	√	√	√	√
FABS	New Instruction			√		
FADD	New Instruction					
FBcc	New Instruction			√		
FCMP	New Instruction			√		
FDBcc	New Instruction			√		
FDIV	New Instruction			√		
FMOVE	New Instruction			√		
FMOVEM	New Instruction			√		
FMUL	New Instruction			√		
FNEG	New Instruction			√		
FRESTORE	New Instruction			√		
FSAVE	New Instruction			√		
FScc	New Instruction			√		
FSQRT	New Instruction			√		
FSUB	New Instruction			√		
FTRAPcc	New Instruction			√		

MC68020, MC68030, MC68040, MC68EC040, MC68LC040 Instruction Set Extensions		**Applies To**				
Instruction	**Notes**	**68020**	**68030**	**68040**	**EC040**	**LC040**
FTST	New Instruction			√		
LINK	Supports 32-Bit Displacement	√	√	√	√	√
MOVE16	New Instruction			√	√	√
MOVEC	Supports New Control Registers	√	√	√	√	√
MULS/MULU	Supports 32-Bit Operands	√	√	√	√	√
PACK	New Instruction	√	√	√	√	√
PFLUSH	MMU Instruction		√	√		√
PLOAD	MMU Instruction		√			
PMOVE	MMU Instruction		√			
PTEST	MMU Instruction		√	√		√
RTM	New Instruction	√				
TST	Supports Program Counter Relative Addressing Modes	√	√	√	√	√
TRAPcc	New Instruction	√	√	√	√	√
UNPK	New Instruction	√	√	√	√	√

THIS PAGE
INTENTIONALY
LEFT
BLANK

APPENDIX B SOFTWARE CONSIDERATIONS

The MC68EC040 has six instruction or instruction opcodes which are different from the MC68040 and require consideration from a programming perspective. The instructions are: PTEST, PFLUSH, CPUSH, CINV, MOVEC, and all floating point instructions.

The PTEST and PFLUSH instructions should not be executed. Execution of the PTEST instruction could cause random bus cycles to occur. Execution of the PFLUSH instruction will have indeterminate results. Both the PTEST and the PFLUSH instructions will not cause the MC68EC040 to generate an exception.

The CPUSH and CINV instruction require special consideration. A page is defined as a 4 Kbyte block of external memory. The CPUSH page and CINV page instruction opcodes can be used to push or invalidate 4 Kbyte blocks of memory.

The MOVEC instruction has two opcodes that are not valid and two that provide limited functionality. The MOVEC to URP and SRP registers are not valid and will produce indeterminate results. The access control unit (ACU) has a status register and translation control register (TCR) which replaces the MMUSR and TCR. The ACU status register can be modified by using the MMUSR opcode of the MOVEC instruction; however, the ACU status register does not provide additional functionality to the ACU and is only provided for compatibility with ACUSR register in the MC68EC030. The ACU status register may not be implemented in future M68EC0x0 products.

To initialize the access control registers (ACR) the MOVEC instruction should be used. The instruction and data ACRs can be modified using the MOVEC instruction for the instruction and data transparent translation register (TTR) opcodes of the MC68040, respectively.

The floating point instructions are not valid and will result in an unimplemented floating point instruction exception and a new unimplemented floating point instruction stack frame. More detailed information is available in the **SECTION 9 EXCEPTION PROCESSING** of this manual.

The MC68EC040 will not generate stack frames $3 which the MC68040 can generate. The MC68040 will not generate the unimplemented floating point instruction exception stack frame $4 which the MC68EC040 and MC68LC040 will generate when a floating point instruction execution is attempted. The MC68EC040 will correctly accept or read stack frame type $3 which can be generated by MC68040. The MC68040 will not accept the unimplemented floating point instruction stack frame $4.

THIS PAGE
INTENTIONALY
LEFT
BLANK

— A —

A-line 9-11
Access Control Register Format 6-3
Access Control Registers 6-1, 6-2
Access Control Unit 1-3, 2-1, 6-1
Access Error Exception 9-9
Access Error Stack Frame 9-23, 9-24
Access Fault Recovery 9-23
Address Error Exception 9-10
Address Errors 9-27
Address Registers 1-6
Addressing Modes 1-7, 1-8
  Register Indirect 1-7
Autovector 9-5
Autovector Signal 5-5, 9-18

— B —

Breakpoint Instruction Exception 9-19
Breakpoint Operation 9-19
Bus Accesses 8-1
Bus Error 9-9
Bus Operation 8-1
Bus Snooping 1-4, 1-10
Bus Transfers 8-1

— C —

Cache Control Register 1-6
Cache Disable Signal 6-4
Cache Inhibit Out Signal 6-2
Cache Inhibited, Not Serialized 6-4
Cache Inhibited, Serialized 6-4
Cache Mode 6-4
Caches
  Bus snooping 1-10
  Data 1-10
  Instructions 1-10
CACR 1-6
Copyback 1-3, 1-4, 6-2

— D —

Data Cache
  Access 1-10
  Copyback 1-3
  Writethrough 1-3
Data Latch Enable Signal 5-4
Data Registers 1-6
Data Types 1- 6, 1-7, 6-1
Destination Function Code (DFC) 1-6
Double Bus Fault 9-9

— E —

Exception Priorities 9-19
Exception Processing 9-1—9-25
Exception Processing
  Access Error 9-9
  Address Error 9-10
  BCLK 9-14
  Breakpoints 9-19
  Exception Vectors 9-2
  Format Error 9-14
  Illegal Instruction 9-11
  Integer Unit 9-7
  Interrupts 9-14
  Priorities 9-19
  Privileged Instructions 9-12
  Reset 9-7
  Returns 9-21
  Stack Frames 9-4
  Trace Exception 9-12
  Trap Exception 9-10
  Unimplemented Floating Point 9-11
  Unimplemented Instructions 9-1
Exception Processing Sequence 9-1
Exception Stack Frames 9-5
Exception Vector Assignments 9-3

— F —

F-line 9-11, 9-28
Faults
  Double Bus 9-9
  Access 9-23
Floating Point Exceptions 9-28
Floating-Point Post-Instruction Stack Frame 9-22
Format Error Exception 9-14

— I —

Illegal Instruction 9-11
Instruction Set 1-8
Instruction Trap Exception 9-10
Interrupt Exceptions 9-14
Interrupt Levels and Mask Values 9-15
Interrupt Pending 9-15
Interrupt Recognition 9-16, 9-17
Interrupts
  Exceptions 9-1, 9-14
  Interrupt Stack Pointer 9-2
  $\overline{\text{IPEND}}$ 5-5
  $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ 5-5

Integer Unit Exception Processing 9-7

— J —

JTAG Scan Signal 5-4

— M —

$\overline{\text{MDIS}}$ 5-4
Memory Management Unit Disable Signal 5-4
MOVEM Instruction Execution Pending 9-25
MOVE16 Instruction 1-9, 9-26

— N —

Non-maskable Interrupt 9-16
Normal Six Word Stack Frame 9-22

— P —

Privilege Violation Exception 9-12
Privileged Instructions 9-12
Program Counter (PC) 1-6, 9-2, 9-9–9-14, 9-18, 9-21, 9-23, 9-27–9-29
Programming Model 1-4

— R —

Registers
  Access Control Registers 1-6, 6-3
  Access Control Unit 1-1
  Address Registers 1-6
  Cache Control Register 1-6
  Data Registers 1-6, 6-2
    DACR0 6-1
    DACR1 6-1
  Exception Processing
    Status Register 9-1
  Instruction Registers 9-2
    IACR0 6-1
    IACR1 6-1
  Program Counter 1-6, 9-2, 9-9–9-14, 9-18, 9-21, 9-23, 9-27–9-29
  Special Status Word 9-24
  Status Register 1-6, 5-5, 9-1, 9-2, 9-7, 9-9, 9-10–9-14, 9-16, 9-18, 9-21, 9-23, 9-28, 9-29
  Vector Base Register 1-6, 9-2
RESET instruction 8-4, 9-4
Reset Signal 5-4 5-5, 6-1, 8-2, 8-3, 8-4, 9-7
Reset Exception 9-7
Return From Exceptions 9-21
Returning From Access Errors 9-27
RTE Instruction 9-3, 9-9, 9-14, 9-16, 9-21, 9-23

— S —

SFC 1-6
Signal Index 5-3
Signal Summary 5-6
Signals
  $\overline{\text{AVEC}}$ 5-5, 9-18
  $\overline{\text{BB}}$ 8-3
  BCLK 8-2
  $\overline{\text{CDIS}}$ 5-4, 8-3
  $\overline{\text{CIOUT}}$ 6-2, 6-4
  DLE 5-4
  $\overline{\text{IPEND}}$ 5-5, 9-17
  $\overline{\text{IPL2}}$-$\overline{\text{IPL0}}$ 5-5, 8-3, 9-14
  JS1–JS0 5-4, 5-5, 8-3
  $\overline{\text{MDIS}}$ 5-3
  PCLK 8-2
  $\overline{\text{RSTI}}$ 5-4, 6-1, 8-2, 8-3, 9-7
  $\overline{\text{RSTO}}$ 5-5, 8-4
  $\overline{\text{TA}}$ 8-3, 9-11, 9-19
  $\overline{\text{TIP}}$ 8-3, 9-11, 9-19
  UPA0–UPA1 6-2, 6-4
Source Function Code 1-6
Special Modes of Operation 8-2
Stack Frames 9-3, 9-4
STOP Instruction 9-13
Status Register (SR) 1-6, 5-5, 9-1, 9-2, 9-7, 9-9, 9-10–9-14, 9-16, 9-18, 9-21, 9-23, 9-28, 9-29
Supervisor Programming Model 1-5
Supervisor/User Mode 1-4, 6-3

— T —

Trace Exception 9-12, 9-13
Trace Exception Pending 9-25
Tracing Control 9-13

— U —

Unimplemented Floating-Point Instruction Exception 9-11, B-1
Unimplemented Floating-Point Instructions 9-28
Unimplemented Floating-Point Instruction Stack Frame 9-22
Unimplemented Floating-Point-Instruction Exception Pending 9-25
Unimplemented Instruction Exceptions 9-11
User Page Attribute 6-2

User Programming Model 1-5

— V —

Vector Base Register (VBR) 1-6, 9-2

— W —

Writeback 9-9
Writethrough 1-3, 1-4, 6-2

NOTES

Introduction 1

Programming Model 2

Data Organization and Addressing Capabilities 3

Instruction Set Summary 4

Signal Description 5

Access Control Unit 6

Instruction and Data Caches 7

Bus Operation 8

Exception Processing 9

Instruction Execution Timing 10

Electrical Characteristics 11

Ordering Information and Mechanical Data 12

Appendix A A

Appendix B B

Index I

1 Introduction

2 Programming Model

3 Data Organization and Addressing Capabilities

4 Instruction Set Summary

5 Signal Description

6 Access Control Unit

7 Instruction and Data Caches

8 Bus Operation

9 Exception Processing

10 Instruction Execution Timing

11 Electrical Characteristics

12 Ordering Information and Mechanical Data

A Appendix A

B Appendix B

I Index

A30319-0 Printed in USA EVANS PRESS EMTR 3020 10,000 MPU YGABAA