M68040 User's Manual

Including the MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V

©MOTOROLA INC., 1990
Revised 1992, 1993

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and the Ⓜ are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

©MOTOROLA INC., 1992

PREFACE

The complete documentation package for the MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V (collectively called M68040) consists of the M68040UM/AD, *M68040 User's Manual*, and the M68000PM/AD, *M68000 Family Programmer's Reference Manual*. The *M68040 User's Manual* describes the capabilities, operation, and programming of the M68040 32-bit third-generation microprocessors. The *M68000 Family Programmer's Reference Manual* contains the complete instruction set for the M68000 family.

The introduction of this manual includes general information concerning the MC68040 and summarizes the differences between the M68040 member devices. Additionally, three appendices provide detailed information on how these M68040 dirivatives operate differently from the MC68040. For detailed information on one of these M68040 dirivatives, use the following table to determine which appendices to read in conjunction with the rest of this manual.

Device Number	Appendices
MC68040V	**Appendix A MC68LC040** and **Appendix C MC68040V and MC68EC040V**
MC68LC040	**Appendix A MC68LC040**
MC68EC040	**Appendix B MC68EC040**
MC68EC040V	**Appendix B MC68EC040** and **Appendix C MC68040V and MC68EC040V**

When reading this manual, **remember** to disregard information concerning floating-point in reference to the MC68040V and MC68LC040, and to disregard information concerning floating-point and memory management in reference to the MC68EC040 and MC68EC040V. The organization of this manual is as follows:

Section 1	Introduction
Section 2	Integer Unit
Section 3	Memory Management Unit (Except MC68EC040 and MC68EC040V)
Section 4	Instruction and Data Caches
Section 5	Signal Description
Section 6	IEEE 1149.1 Test Access Port (JTAG)
Section 7	Bus Operation
Section 8	Exception Processing
Section 9	Floating-Point Unit (MC68040)
Section 10	Instruction Timings
Section 11	MC68040 Electrical and Thermal Characteristics
Section 12	Ordering Information and Mechanical Data
Appendix A	MC68LC040
Appendix B	MC68EC040
Appendix C	MC68040V and MC68EC040V
Appendix D	M68000 Family Summary
Appendix E	Floating-Point Emulation (M68040FPSP)
Index	

TABLE OF CONTENTS

Paragraph Number	Title	Page Number
	Section 1 Introduction	
1.1	Differences	1-1
1.1.1	MC68040V and MC68LC040	1-1
1.1.2	MC68EC040 and MC68EC040V	1-2
1.2	Features	1-3
1.3	Extensions to the M68000 Family	1-3
1.4	Functional Blocks	1-3
1.5	Processing States	1-5
1.6	Programming Model	1-5
1.7	Data Format Summary	1-9
1.8	Addressing Capabilities Summary	1-9
1.9	Notational Conventions	1-11
1.10	Instruction Set Overview	1-13
	Section 2 Integer Unit	
2.1	Integer Unit Pipeline	2-1
2.2	Integer Unit Register Description	2-4
2.2.1	Integer Unit User Programming Model	2-4
2.2.1.1	Data Registers (D7–D0)	2-4
2.2.1.2	Address Registers (A6–A0)	2-4
2.2.1.3	System Stack Pointer (A7)	2-5
2.2.1.4	Program Counter	2-5
2.2.1.5	Condition Code Register	2-5
2.2.2	Integer Unit Supervisor Programming Model	2-5
2.2.2.1	Interrupt and Master Stack Pointers	2-6
2.2.2.2	Status Register	2-7
2.2.2.3	Vector Base Register	2-7
2.2.2.4	Alternate Function Code Registers	2-7
2.2.2.5	Cache Control Register	2-8

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)	
3.1	Memory Management Programming Model	3-3
3.1.1	User and Supervisor Root Pointer Registers	3-3
3.1.2	Translation Control Register	3-4
3.1.3	Transparent Translation Registers	3-5
3.1.4	MMU Status Register	3-6
3.2	Logical Address Translation	3-7
3.2.1	Translation Tables	3-7
3.2.2	Descriptors	3-12
3.2.2.1	Table Descriptors	3-12
3.2.2.2	Page Descriptors	3-13
3.2.2.3	Descriptor Field Definitions	3-13
3.2.3	Translation Table Example	3-16
3.2.4	Variations in Translation Table Structure	3-16
3.2.4.1	Indirect Action	3-16
3.2.4.2	Table Sharing Between Tasks	3-18
3.2.4.3	Table Paging	3-19
3.2.4.4	Dynamically Allocated Tables	3-21
3.2.5	Table Search Accesses	3-21
3.2.6	Address Translation Protection	3-23
3.2.6.1	Supervisor and User Translation Tables	3-23
3.2.6.2	Supervisor Only	3-23
3.2.6.3	Write Protect	3-24
3.3	Address Translation Caches	3-26
3.4	Transparent Translation	3-29
3.5	Address Translation Summary	3-30
3.6	MMU Effect on $\overline{\text{RSTI}}$ and $\overline{\text{MDIS}}$	3-31
3.6.1	Effect of $\overline{\text{RSTI}}$ on the MMUs	3-31
3.6.2	Effect of $\overline{\text{MDIS}}$ on Address Translation	3-31
3.7	MMU Instructions	3-33
3.7.1	MOVEC	3-33
3.7.2	PFLUSH	3-33
3.7.3	PTEST	3-33
3.7.4	Register Programming Considerations	3-34

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	Section 4 Instruction and Data Caches	
4.1	Cache Operation	4-2
4.2	Cache Management	4-5
4.3	Caching Modes	4-6
4.3.1	Cachable Accesses	4-6
4.3.1.1	Write-Through Mode	4-6
4.3.1.2	Copyback Mode	4-6
4.3.2	Cache-Inhibited Accesses	4-7
4.3.3	Special Accesses	4-7
4.4	Cache Protocol	4-7
4.4.1	Read Miss	4-8
4.4.2	Write Miss	4-8
4.4.3	Read Hit	4-8
4.4.4	Write Hit	4-8
4.5	Cache Coherency	4-9
4.6	Memory Accesses for Cache Maintenance	4-11
4.6.1	Cache Filling	4-11
4.6.2	Cache Pushes	4-13
4.7	Cache Operation Summary	4-13
4.7.1	Instruction Cache	4-14
4.7.2	Data Cache	4-15
	Section 5 Signal Description	
5.1	Address Bus (A31–A0)	5-4
5.2	Data Bus (D31–D0)	5-5
5.3	Transfer Attribute Signals	5-5
5.3.1	Transfer Type (TT1, TT0)	5-5
5.3.2	Transfer Modifier (TM2–TM0)	5-6
5.3.3	Transfer Line Number (TLN1, TLN0)	5-6
5.3.4	User-Programmable Attributes (UPA1, UPA0)	5-7
5.3.5	Read/Write (R/$\overline{W}$)	5-7
5.3.6	Transfer Size (SIZ1, SIZ0)	5-7
5.3.7	Lock ($\overline{LOCK}$)	5-7
5.3.8	Lock End ($\overline{LOCKE}$)	5-7
5.3.9	Cache Inhibit Out ($\overline{CIOUT}$)	5-8
5.4	Bus Transfer Control Signals	5-8
5.4.1	Transfer Start ($\overline{TS}$)	5-8

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
5.4.2	Transfer in Progress ($\overline{TIP}$)	5-8
5.4.3	Transfer Acknowledge ($\overline{TA}$)	5-8
5.4.4	Transfer Error Acknowledge ($\overline{TEA}$)	5-8
5.4.5	Transfer Cache Inhibit ($\overline{TCI}$)	5-9
5.4.6	Transfer Burst Inhibit ($\overline{TBI}$)	5-9
5.5	Snoop Control Signals	5-9
5.5.1	Snoop Control (SC1, SC0)	5-9
5.5.2	Memory Inhibit ($\overline{MI}$)	5-9
5.6	Arbitration Signals	5-10
5.6.1	Bus Request ($\overline{BR}$)	5-10
5.6.2	Bus Grant ($\overline{BG}$)	5-10
5.6.3	Bus Busy ($\overline{BB}$)	5-10
5.7	Processor Control Signals	5-10
5.7.1	Cache Disable ($\overline{CDIS}$)	5-10
5.7.2	Reset In ($\overline{RSTI}$)	5-11
5.7.3	Reset Out ($\overline{RSTO}$)	5-11
5.8	Interrupt Control Signals	5-11
5.8.1	Interrupt Priority Level ($\overline{IPL2}$–$\overline{IPL0}$)	5-11
5.8.2	Interrupt Pending Status ($\overline{IPEND}$)	5-12
5.8.3	Autovector ($\overline{AVEC}$)	5-12
5.9	Status And Clock Signals	5-12
5.9.1	Processor Status (PST3–PST0)	5-12
5.9.2	Bus Clock (BCLK)	5-14
5.9.3	Processor Clock (PCLK)—Not on MC68040V and MC68EC040V	5-14
5.10	MMU Disable ($\overline{MDIS}$)—Not on MC68EC040	5-14
5.11	Data Latch Enable (DLE)—Only on MC68040	5-14
5.12	Test Signals	5-15
5.12.1	Test Clock (TCK)	5-15
5.12.2	Test Mode Select (TMS)	5-15
5.12.3	Test Data In (TDI)	5-15
5.12.4	Test Data Out (TDO)	5-15
5.12.5	Test Reset ($\overline{TRST}$)—Not on MC68040V and MC68EC040V	5-15
5.13	Power Supply Connections	5-15
5.14	Signal Summary	5-16

Section 6
IEEE 1149.1 Test Access Port (JTAG)

6.1	Overview	6-2
6.2	Instruction Shift Register	6-3
6.2.1	EXTEST	6-3

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
6.2.2	HIGHZ	6-4
6.2.3	SAMPLE/PRELOAD	6-4
6.2.4	DRVCTL.T	6-4
6.2.5	SHUTDOWN	6-5
6.2.6	PRIVATE	6-5
6.2.7	DRVCTL.S	6-5
6.2.8	BYPASS	6-6
6.3	Boundary Scan Register	6-6
6.4	Restrictions	6-12
6.5	Disabling The IEEE Standard 1149.1A Operation	6-13
6.6	Motorola M68040 BSDL Description (Version 2.2)	6-15
6.7	MC68040, MC68LC040, MC68EC040 JTAG Electrical Characteristics	6-21

Section 7
Bus Operation

7.1	Bus Characteristics	7-1
7.2	Data Transfer Mechanism	7-3
7.3	Misaligned Operands	7-6
7.4	Processor Data Transfers	7-9
7.4.1	Byte, Word, and Long-Word Read Transfers	7-10
7.4.2	Line Read Transfer	7-12
7.4.3	Byte, Word, and Long-Word Write Transfers	7-20
7.4.4	Line Write Transfers	7-22
7.4.5	Read-Modify-Write Transfers (Locked Transfers)	7-26
7.5	Acknowledge Bus Cycles	7-29
7.5.1	Interrupt Acknowledge Bus Cycles	7-29
7.5.1.1	Interrupt Acknowledge BUS Cycle (Terminated Normally)	7-31
7.5.1.2	Autovector Interrupt Acknowledge bus Cycle	7-33
7.5.1.3	Spurious Interrupt Acknowledge Bus Cycle	7-34
7.5.2	Breakpoint Interrupt Acknowledge Bus Cycle	7-35
7.6	Bus Exception Control Cycles	7-36
7.6.1	Bus Errors	7-37
7.6.2	Retry Operation	7-41
7.6.3	Double Bus Fault	7-43
7.7	Bus Synchronization	7-43
7.8	Bus Arbitration And Examples	7-44
7.8.1	Bus Arbitration	7-45
7.8.2	Bus Arbitration Examples	7-52
7.8.2.1	Dual M68040 Fairness Arbitration	7-52
7.8.2.2	Dual M68040 Prioritized Arbitration	7-54

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
7.8.2.3	M68040 Synchronous DMA Arbitration	7-55
7.8.2.4	M68040 Asynchronous DMA Arbitration	7-57
7.9	Bus Snooping Operation	7-59
7.9.1	Snoop-Inhibited Cycle	7-60
7.9.2	Snoop-Enabled Cycle (No Intervention Required)	7-61
7.9.3	Snoop Read Cycle (Intervention Required)	7-63
7.9.4	Snoop Write Cycle (Intervention Required)	7-63
7.10	Reset Operation	7-65
7.11	Special Modes of Operation	7-68
7.11.1	Output Buffer Impedance Selection	7-68
7.11.2	Multiplexed Bus Mode	7-68
7.11.3	Data Latch Enable Mode	7-69

Section 8
Exception Processing

8.1	Exception Processing Overview	8-1
8.2	Integer Unit Exceptions	8-5
8.2.1	Access Fault Exception	8-6
8.2.2	Address Error Exception	8-8
8.2.3	Instruction Trap Exception	8-8
8.2.4	Illegal Instruction and Unimplemented Instruction Exceptions	8-9
8.2.5	Privilege Violation Exception	8-9
8.2.6	Trace Exception	8-10
8.2.7	Format Error Exception	8-11
8.2.8	Breakpoint Instruction Exception	8-12
8.2.9	Interrupt Exception	8-12
8.2.10	Reset Exception	8-17
8.3	Exception Priorities	8-19
8.4	Return From Exceptions	8-20
8.4.1	Four-Word Stack Frame (Format $0)	8-21
8.4.2	Four-Word Throwaway Stack Frame (Format $1)	8-21
8.4.3	Six-Word Stack Frame (Format $2)	8-22
8.4.4	Floating-Point Post-Instruction Stack Frame (Format $3)	8-23
8.4.5	Eight-Word Stack Frame (Format $4)	8-23
8.4.6	Access Error Stack Frame (Format $7)	8-24
8.4.6.1	Effective Address	8-24
8.4.6.2	Special Status Word (SSW)	8-24
8.4.6.3	Write-Back Status	8-26
8.4.6.4	Fault Address	8-26

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
8.4.6.5	Write-Back Address and Write-Back Data	8-26
8.4.6.6	Push Data	8-27
8.4.6.7	Access Error Stack Frame Return From Exception	8-27

Section 9
Floating-Point Unit (MC68040 Only)

9.1	Floating-Point Unit Pipeline	9-1
9.2	Floating-Point User Programming Model	9-2
9.2.1	Floating-Point Data Registers (FP7–FP0)	9-2
9.2.2	Floating-Point Control Register (FPCR)	9-3
9.2.2.1	Exception Enable Byte	9-3
9.2.2.2	Mode Control Byte	9-3
9.2.3	Floating-Point Status Register (FPSR)	9-4
9.2.3.1	Floating-Point Condition Code Byte	9-4
9.2.3.2	Quotient Byte	9-5
9.2.3.3	Exception Status Byte	9-5
9.2.3.4	Accrued Exception (AEXC) Byte.	9-5
9.2.4	Floating-Point Instruction Address Register (FPIAR)	9-6
9.3	Floating-Point Data Formats and Data Types	9-7
9.4	Computational Accuracy	9-11
9.4.1	Intermediate Result	9-12
9.4.2	Rounding the Result	9-13
9.5	Postprocessing Operation	9-15
9.5.1	Underflow, Round, Overflow	9-16
9.5.2	Conditional Testing	9-16
9.6	Floating-Point Exceptions	9-20
9.6.1	Unimplemented Floating-Point Instructions	9-20
9.6.2	Unsupported Floating-Point Data Types	9-22
9.7	Floating-Point Arithmetic Exceptions	9-24
9.7.1	Branch/Set on Unordered (BSUN)	9-25
9.7.1.1	Maskable Exception Conditions	9-26
9.7.1.2	Nonmaskable Exception Conditions	9-27
9.7.2	Signaling Not-a-Number (SNAN)	9-27
9.7.2.1	Maskable Exception Conditions	9-27
9.7.2.2	Nonmaskable Exception Conditions	9-27
9.7.3	Operand Error	9-28
9.7.3.1	Maskable Exception Conditions	9-29
9.7.3.2	Nonmaskable Exception Conditions	9-30
9.7.4	Overflow	9-31
9.7.4.1	Maskable Exception Conditions	9-31
9.7.4.2	Nonmaskable Exception Conditions	9-31

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
9.7.5	Underflow	9-33
9.7.5.1	Maskable Exception Conditions	9-34
9.7.5.2	Nonmaskable Exception Conditions	9-34
9.7.6	Divide by Zero	9-36
9.7.7	Inexact Result	9-36
9.8	Floating-Point State Frames	9-39

Section 10
Instruction Timings

10.1	Overview	10-3
10.2	Instruction Timing Examples	10-5
10.3	CINV and CPUSH Instruction Timing	10-8
10.4	MOVE Instruction Timing	10-9
10.5	Miscellaneous Integer Unit Instruction Timings	10-11
10.6	Integer Unit Instruction Timings	10-13
10.7	Floating-Point Unit Instruction Timings	10-29
10.7.1	Miscellaneous Integer Unit Support Timings	10-29
10.7.2	Integer Unit Support Timings	10-30
10.7.3	Timings in the Floating-Point Unit	10-35

Section 11
MC68040 Electrical and Thermal Characteristics

11.1	Maximum Ratings	11-1
11.2	Thermal Characteristics	11-1
11.3	DC Electrical Specifications	11-2
11.4	Power Dissipation	11-2
11.5	Clock AC Timing Specifications	11-3
11.6	Output AC Timing Specifications	11-4
11.7	Input AC Timing Specifications	11-5
11.8	MC68040 Thermal Device Characteristics	11-12
11.8.1	MC68040 Die and Package	11-12
11.8.2	MC68040 Power Considerations	11-12
11.9	MC68040 Thermal Management Techniques	11-14
11.9.1	Still Air	11-17
11.9.2	Forced Air	11-18
11.9.3	With Heat Sink	11-19
11.9.4	With Heat Sink and Forced Air	11-22

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
	Section 12 Ordering Information and Mechanical Data	
12.1	Ordering Information	12-1
12.2	Pin Assignments	12-1
12.2.1	MC68040 Pin Grid Array	12-2
12.2.2	MC68LC040 Pin Grid Array	12-3
12.2.3	MC68EC040 Pin Grid Array	12-4
12.2.4	MC68040V and MC68EC040V Pin Grid Array	12-5
12.2.5	MC68LC040 Quad Flat Pack	12-6
12.2.6	MC68EC040 Quad Flat Pack	12-6
12.2.7	MC68040V and MC68EC040V Quad Flat Pack	12-7
12.3	Mechanical Data	12-9
	Appendix A MC68LC040	
A.1	MC68LC040 Differences	A-5
A.2	Interrupt Priority Level ($\overline{IPL2}$–$\overline{IPL0}$)	A-5
A.3	JTAG Scan (JS0)	A-5
A.4	Data Latch And Multiplexed Bus Modes	A-5
A.5	Floating-Point Unit (FPU)	A-5
A.5.1	Unimplemented Floating-Point Instructions and Exceptions	A-6
A.5.2	MC68LC040 Stack Frames	A-7
A.6	MC68LC040 Electrical Characteristics	A-7
A.6.1	Maximum Ratings	A-8
A.6.2	Thermal Characteristics	A-8
A.6.3	DC Electrical Specifications	A-8
A.6.4	Power Dissipation	A-9
A.6.5	Clock AC Timing Specifications	A-9
A.6.6	Output AC Timing Specifications	A-11
A.6.7	Input AC Timing Specifications	A-12
	Appendix B MC68EC040	
B.1	MC68EC040 Differences	B-4
B.2	JTAG Scan (JS1–JS0)	B-5
B.3	Access Control Units	B-5
B.3.1	Access Control Registers	B-5
B.3.2	Address Comparison	B-7
B.3.3	Effect of $\overline{RSTI}$ on the ACU	B-8

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
B.4	Special Modes Of Operation	B-8
B.5	Exception Processing	B-10
B.5.1	Unimplemented Floating-Point Instructions and Exceptions	B-10
B.5.2	MC68EC040 Stack Frames	B-11
B.6	Software Considerations	B-12
B.7	MC68EC040 Electrical Characteristics	B-12
B.7.1	Maximum Ratings	B-12
B.7.2	Thermal Characteristics	B-12
B.7.3	DC Electrical Specifications	B-13
B.7.4	Power Dissipation	B-13
B.7.5	Clock AC Timing Specifications	B-14
B.7.6	Output AC Timing Specifications	B-15
B.7.7	Input AC Timing Specifications	B-16

Appendix C
MC68040V and MC68EC040V

C.1	Additional Signals	C-1
C.1.1	Low Frequency Operation (LFO)	C-2
C.1.2	Loss of Clock (LOC)	C-2
C.1.3	System Clock Disable (SCD)	C-2
C.2	Low-Power Stop Mode	C-3
C.2.1	Bus Arbitration and Snooping	C-5
C.2.2	Low Frequency Operation	C-5
C.2.3	Changing BCLK Frequency	C-5
C.2.4	LPSTOP Instruction Summary	C-6
C.3	Clocking During Normal Operation	C-7
C.4	Reset Operation	C-7
C.5	Power Cycling	C-9
C.6	MC68040V and MC68EC040V JTAG (Preliminary)	C-10
C.6.1	Instruction Shift Register	C-11
C.6.1.1	EXTEST	C-12
C.6.1.2	HIGHZ	C-12
C.6.1.3	SAMPLE/PRELOAD	C-12
C.6.1.4	CLAMP	C-12
C.6.1.5	BYPASS	C-13
C.6.2	Boundary Scan Register	C-13
C.6.3	Restrictions	C-16
C.6.4	Disabling The IEEE Standard 1149.1A Operation	C-16
C.6.5	MC68040V and MC68EC040V JTAG Electrical Characteristics	C-17

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
C.7	MC68040V and MC68EC040V Electrical Characteristics	C-19
C.7.1	Maximum Ratings	C-19
C.7.2	Thermal Characteristics	C-19
C.7.3	DC Electrical Specifications	C-20
C.7.4	Power Dissipation	C-20
C.7.5	Clock AC Timing Specifications	C-21
C.7.6	Output AC Timing Specifications	C-22
C.7.7	Input AC Timing Specifications	C-23

Appendix D
M68000 Family Summary

Appendix E
Floating-Point Emulation (M68040FPSP)

Index

LIST OF ILLUSTRATIONS

Figure Number	Title	Page Number
1-1	Block Diagram	1-4
1-2	Programming Model	1-7
2-1	Integer Unit Pipeline	2-2
2-2	Write-Back Cycle Block Diagram	2-3
2-3	Integer Unit User Programming Model	2-4
2-4	Integer Unit Supervisor Programming Model	2-6
2-5	Status Register	2-7
3-1	Memory Management Unit	3-2
3-2	Memory Management Programming Model	3-3
3-3	URP and SRP Register Formats	3-4
3-4	Translation Control Register Format	3-4
3-5	Transparent Translation Register Format	3-5
3-6	MMU Status Register Format	3-6
3-7	Translation Table Structure	3-8
3-8	Logical Address Format	3-9
3-9	Detailed Flowchart of Table Search Operation	3-10
3-10	Detailed Flowchart of Descriptor Fetch Operation	3-11
3-11	Table Descriptor Formats	3-13
3-12	Page Descriptor Formats	3-13
3-13	Example Translation Table	3-17
3-14	Translation Table Using Indirect Descriptors	3-18
3-15	Translation Table Using Shared Tables	3-19
3-16	Translation Table with Nonresident Tables	3-20
3-17	Translation Table Structure for Two Tasks	3-24
3-18	Logical Address Map with Shared Supervisor and User Address Spaces	3-24
3-19	Translation Table Using S-Bit and W-Bit To Set Protection	3-25
3-20	ATC Organization	3-26
3-21	ATC Entry and Tag Fields	3-27
3-22	Address Translation Flowchart	3-32
3-23	MMU Status Interpretation	3-35
4-1	Overview of Internal Caches	4-2
4-2	Cache Line Formats	4-3
4-3	Caching Operation	4-4
4-4	Cache Control Register	4-5

LIST OF ILLUSTRATIONS (Continued)

Figure Number	Title	Page Number
4-5	Instruction-Cache Line State Diagram	4-14
4-6	Data-Cache Line State Diagram	4-16
5-1	Functional Signal Groups	5-4
6-1	M68040 Test Logic Block Diagram	6-2
6-2	Bypass Register	6-6
6-3	Output Latch Cell (O.Latch)	6-7
6-4	Input Pin Cell (I.Pin)	6-7
6-5	Output Control Cells (IO.Ctl)	6-8
6-6	General Arrangement of Bidirectional Pins	6-8
6-7	Circuit Disabling IEEE Standard 1149.1A	6-14
6-8	Clock Input Timing Diagram	6-22
6-9	$\overline{\text{TRST}}$ Timing Diagram	6-22
6-10	Boundary Scan Timing Diagram	6-23
6-11	Test Access Port Timing Diagram	6-23
7-1	Signal Relationships to Clocks	7-2
7-2	Internal Operand Representation	7-3
7-3	Data Multiplexing	7-4
7-4	Byte Enable Signal Generation and PAL Equation	7-5
7-5	Example of a Misaligned Long-Word Transfer	7-7
7-6	Example of a Misaligned Word Transfer	7-7
7-7	Misaligned Long-Word Read Transfer Timing	7-8
7-8	Byte, Word, and Long-Word Read Transfer Flowchart	7-10
7-9	Byte, Word, and Long-Word Read Transfer Timing	7-11
7-10	Line Read Transfer Flowchart	7-14
7-11	Line Read Transfer Timing	7-15
7-12	Burst-Inhibited Line Read Transfer Flowchart	7-18
7-13	Burst-Inhibited Line Read Transfer Timing	7-19
7-14	Byte, Word, and Long-Word Write Transfer Flowchart	7-20
7-15	Long-Word Write Transfer Timing	7-21
7-16	Line Write Transfer Flowchart	7-23
7-17	Line Write Transfer Timing	7-24
7-18	Locked Transfer for TAS Instruction Timing	7-27
7-19	Interrupt Pending Procedure	7-30
7-20	Assertion of $\overline{\text{IPEND}}$	7-30
7-21	Interrupt Acknowledge Bus Cycle Flowchart	7-32
7-22	Interrupt Acknowledge Bus Cycle Timing	7-33
7-23	Autovector Interrupt Acknowledge Bus Cycle Timing	7-34
7-24	Breakpoint Interrupt Acknowledge Bus Cycle Flowchart	7-35
7-25	Breakpoint Interrupt Acknowledge Bus Cycle Timing	7-36

LIST OF ILLUSTRATIONS (Continued)

Figure Number	Title	Page Number
7-26	Word Write Access Terminated with $\overline{\text{TEA}}$ Timing	7-39
7-27	Line Read Access Terminated with $\overline{\text{TEA}}$ Timing	7-40
7-28	Retry Read Transfer Timing	7-41
7-29	Retry Operation on Line Write	7-42
7-30	M68040 Internal Interpretation State Diagram and External Bus Arbiter Circuit	7-47
7-31	Lock Violation Example	7-49
7-32	Processor Bus Request Timing	7-50
7-33	Arbitration During Relinquish and Retry Timing	7-51
7-34	Implicit Bus Ownership Arbitration Timing	7-52
7-35	Dual M68040 Fairness Arbitration State Diagram	7-53
7-36	Dual M68040 Prioritized Arbitration State Diagram	7-55
7-37	M68040 Synchronous DMA Arbitration	7-56
7-38	Sample Synchronizer Circuit	7-57
7-39	M68040 Asynchronous DMA Arbitration	7-58
7-40	Snoop-Inhibited Bus Cycle	7-61
7-41	Snoop Access with Memory Response	7-62
7-42	Snooped Line Read, Memory Inhibited	7-64
7-43	Snooped Long-Word Write, Memory Inhibited	7-65
7-44	Initial Power-On Reset Timing	7-66
7-45	Normal Reset Timing	7-67
7-46	Multiplexed Address and Data Bus (Line Write)	7-69
7-47	DLE Mode Block Diagram	7-70
7-48	DLE versus Normal Data Read Timing	7-71
8-1	General Exception Processing Flowchart	8-3
8-2	General Form of Exception Stack Frame	8-4
8-3	Interrupt Recognition Examples	8-14
8-4	Interrupt Exception Processing Flowchart	8-16
8-5	Reset Exception Processing Flowchart	8-18
8-6	Flowchart of RTE Instruction for Throwaway Four-Word Frame	8-22
8-7	Special Status Word Format	8-24
8-8	Write-Back Status Format	8-26
9-1	Floating-Point User Programming Model	9-2
9-2	Floating-Point Control Register	9-4
9-3	FPSR Condition Code Byte	9-4
9-4	FPSR Quotient Byte	9-5
9-5	FPSR Exception Status Byte	9-5
9-6	FPSR Accrued Exception Byte	9-6
9-7	Intermediate Result Format	9-12
9-8	Rounding Algorithm Flowchart	9-14

LIST OF ILLUSTRATIONS (Continued)

Figure Number	Title	Page Number
9-9	Format of Denormalized Operand in State Frame	9-24
9-10	MC68040 Floating-Point State Frames	9-40
9-11	Mapping of Command Bits for CMDREG3B Field	9-42
10-1	Simple Instruction Timing Example	10-5
10-2	Instruction Overlap with Multiple Clocks	10-6
10-3	Interlocked Stages	10-7
11-1	Clock Input Timing Diagram	11-3
11-2	Drive Levels and Test Points for AC Specifications	11-6
11-3	Read/Write Timing	11-7
11-4	Bus Arbitration Timing	11-8
11-5	Snoop Hit Timing	11-9
11-6	Snoop Miss Timing	11-10
11-7	Other Signal Timing	11-11
11-8	MC68040 Termination Network	11-15
11-9	Typical Configuration for RC Termination Network	11-15
11-10	Heat Sink with Adhesive	11-20
11-11	Heat Sink with Attachment	11-21
12-1	PGA Package Dimensions	12-9
12-2	QFP Package Dimensions	12-10
A-1	MC68LC040 Block Diagram	A-2
A-2	MC68LC040 Programming Model	A-3
A-3	MC68LC040 Functional Signal Groups	A-4
A-4	Clock Input Timing Diagram	A-10
A-5	Read/Write Timing	A-13
A-6	Bus Arbitration Timing	A-14
A-7	Snoop Hit Timing	A-15
A-8	Snoop Miss Timing	A-16
A-9	Other Signal Timing	A-17
B-1	MC68EC040 Block Diagram	B-2
B-2	MC68EC040 Programming Model	B-3
B-3	MC68EC040 Functional Signal Groups	B-4
B-4	MC68EC040 Access Control Register Format	B-6
B-5	MC68EC040 Initial Power-On Reset Timing	B-8
B-6	MC68EC040 Normal Reset Timing	B-9
B-7	Clock Input Timing Diagram	B-14
B-8	Read/Write Timing	B-17
B-9	Bus Arbitration Timing	B-18

LIST OF ILLUSTRATIONS (Continued)

Figure Number	Title	Page Number
B-10	Snoop Hit Timing	B-19
B-11	Snoop Miss Timing	B-20
B-12	Other Signal Timing	B-21
C-1	MC68040V and MC68EC040V Functional Signal Groups	C-3
C-2	MC68040V and MC68EC040V Initial Power-On Reset Timing	C-8
C-3	MC68040V and MC68EC040V Normal Reset Timing	C-9
C-4	MC68040V and MC68EC040V Test Logic Block Diagram	C-11
C-5	Bypass Register	C-13
C-6	Output Latch Cell (O.Latch)	C-14
C-7	Input Pin Cell (I.Pin)	C-14
C-8	Output Control Cells (IO.Ctl)	C-15
C-9	General Arrangement of Bidirectional Pins	C-15
C-10	Circuit Disabling IEEE Standard 1149.1A	C-17
C-11	Drive Levels and Test Points for AC Specifications	C-18
C-12	Clock Input Timing Diagram	C-21
C-13	Read/Write Timing	C-24
C-14	Bus Arbitration Timing	C-25
C-15	Snoop Hit Timing	C-26
C-16	Snoop Miss Timing	C-27
C-17	Other Signal Timing	C-28
C-18	Going into LPSTOP with Arbitration	C-29
C-19	LPSTOP no Arbitration, CPU is Master	C-30
C-20	Exiting LPSTOP with Interrupt	C-31
C-21	Exiting of LPSTOP with RESET	C-31

LIST OF TABLES

Table Number	Title	Page Number
1-1	M68040 Data Formats	1-9
1-2	Effective Addressing Modes	1-10
1-3	Notational Conventions	1-11
1-4	Instruction Set Summary	1-14
3-1	Updating U-Bit and M-Bit for Page Descriptors	3-22
3-2	SFC and DFC Values	3-22
4-1	Snoop Control Encoding	4-9
4-2	TLNx Encoding	4-11
4-3	Instruction-Cache Line State Transitions	4-15
4-4	Data-Cache Line State Transitions	4-17
5-1	Signal Index	5-2
5-2	Transfer-Type Encoding	5-5
5-3	Normal and MOVE16 Access Transfer Modifier Encoding	5-6
5-4	Alternate Access Transfer Modifier Encoding	5-6
5-5	Output Driver Control Groups	5-11
5-6	Processor Status Encoding	5-13
5-7	Signal Summary	5-16
6-1	IEEE Standard 1149.1A Instructions	6-3
6-2	Boundary Scan Bit Definitions	6-10
7-1	Data Bus Requirements for Read and Write Cycles	7-4
7-2	Summary of Access Types versus Bus Signal Encodings	7-6
7-3	Memory Alignment Influence on Noncachable and Write-Through Bus Cycles	7-9
7-4	Interrupt Acknowledge Termination Summary	7-31
7-5	$\overline{TA}$ and $\overline{TEA}$ Assertion Results	7-37
7-6	M68040 Bus Arbitration States	7-48
8-1	Exception Vector Assignments	8-5
8-2	Tracing Control	8-11
8-3	Interrupt Levels and Mask Values	8-12
8-4	Exception Priority Groups	8-19

LIST OF TABLES (Continued)

Table Number	Title	Page Number
8-5	Write-Back Data Alignment	8-27
8-6	Access Error Stack Frame Combinations	8-31
9-1	Floating-Point Control Register Encodings	9-3
9-2	MC68040 FPU Data Formats and Data Types	9-7
9-3	Single-Precision Real Format Summary	9-8
9-4	Double-Precision Real Format Summary	9-9
9-5	Extended-Precision Real Format Summary	9-10
9-6	Packed Decimal Real Format Summary	9-11
9-7	Floating-Point Condition Code Encodings	9-17
9-8	Floating-Point Conditional Tests	9-19
9-9	Floating-Point Exception Vectors	9-20
9-10	Unimplemented Instructions	9-21
9-11	Possible Operand Errors Exceptions	9-29
9-12	Overflow Rounding Mode Values	9-32
9-13	Underflow Rounding Mode Values	9-34
9-14	Possible Divide by Zero Exceptions	9-36
9-15	Divide by Zero Rounding Mode Values	9-37
9-16	State Frame Field Information	9-44
10-1	Instruction Timing Index	10-1
10-2	Number of Memory Accesses	10-3
10-3	CINV Timing	10-8
10-4	CPUSH Best and Worst Case Timing	10-8
11-1	Maximum Power Dissipation for Output Buffer Mode Configuration	11-13
11-2	Thermal Parameters with No Heat Sink or Airflow	11-17
11-3	Thermal Parameters with Forced Airflow and No Heat Sink for the MC68040	11-18
11-4	Thermal Parameters with Forced Airflow and No Heat Sink for the MC68LC040 and MC68EC040	11-19
11-5	Thermal Parameters with Heat Sink and No Airflow	11-21
11-6	Thermal Parameters with Heat Sink and Airflow	11-22
C-1	Additional MC68040V and MC68EC040V Signals	C-2
C-2	Bus Encodings During LPSTOP Broadcast Cycle	C-4
C-3	IEEE Standard 1149.1A Instructions	C-12
E-1	MC68040 Floating-Point Instructions	E-2
E-2	MC68040FPSP Floating-Point Instructions	E-3
E-3	Support for Data Types and Data Formats	E-4
E-4	Exception Conditions	E-4

SECTION 1
INTRODUCTION

The MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V (collectively called M68040) are Motorola's third generation of M68000-compatible, high-performance, 32-bit microprocessors. All five devices are virtual memory microprocessors employing multiple concurrent execution units and a highly integrated architecture that provides very high performance in a monolithic HCMOS device. They integrate an MC68030-compatible integer unit (IU) and two independent caches. The MC68040, MC68040V, and MC68LC040 contain dual, independent, demand-paged memory management units (MMUs) for instruction and data stream accesses and independent, 4-Kbyte instruction and data caches. The MC68040 contains an MC68881/MC68882-compatible floating-point unit (FPU). The use of multiple independent execution pipelines, multiple internal buses, and a full internal Harvard architecture, including separate physical caches for both instruction and data accesses, achieves a high degree of instruction execution parallelism on all three processors. The on-chip bus snoop logic, which directly supports cache coherency in multimaster applications, enhances cache functionality.

The M68040 family is user object-code compatible with previous M68000 family members and is specifically optimized to reduce the execution time of compiler-generated code. All five processors implement Motorola's latest HCMOS technology, providing an ideal balance between speed, power, and physical device size.

1.1 DIFFERENCES

Because the functionality of individual M68040 family members are similar, this manual is organized so that the reader will take the following differences into account while reading the rest of this manual. Unless otherwise noted, all references to M68040, with the exception of the differences outlined below, will apply to the MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V. The following paragraphs describe the differences of MC68040V, MC68LC040, MC68EC040, and the MC68EC040V from the MC68040.

1.1.1 MC68040V and MC68LC040

The MC68040V and MC68LC040 are derivatives of the MC68040. They implement the same IU and MMU as the MC68040, but have no FPU. The MC68LC040 is pin compatible with the MC68040. The MC68040V is not pin compatible with the MC68040 and contains some additional features. The following differences exist between the MC68040V, MC68LC040, and MC68040:

- The DLE pin name has been changed to JS0 on both the MC68040V and MC68LC040. In addition, the MC68040V contains three new pins, system clock disable ($\overline{SCD}$), low frequency operation ($\overline{LFO}$), and loss of clock ($\overline{LOC}$).
- The MC68040V and MC68LC040 do not implement the data latch enable (DLE), multiplexed, or output buffer impedance selection modes of operation. They implement only the small output buffer mode of operation. All timing and drive capabilities on both devices are equivalent to those of the MC68040 in small output buffer impedance mode. The MC68040V has an additional mode of operation, the low-power stop mode of operation.
- The MC68040V and MC68LC040 do not contain an FPU, causing unimplemented floating-point exceptions to occur using a new stack frame format.
- The MC68040V is a 3.3 volt static microprocessor that operates down to 0 MHz.

For specific details on the MC68LC040, refer to **Appendix A MC68LC040**. For specific details on the MC68040V, refer to both **Appendix A MC68LC040** and **Appendix C MC68040V and MC68EC040V**. **Disregard all information concerning the FPU** when reading the following subsections.

1.1.2 MC68EC040 and MC68EC040V

The MC68EC040 and MC68EC040V are derivatives of the MC68040. They implement the same IU as the MC68040, but have no FPU or MMU, which embedded control applications generally do not require. The MC68EC040 is pin compatible with the MC68040. The following differences exist between the MC68EC040, MC68EC040V, and the MC68040:

- The DLE and $\overline{MDIS}$ pin names have been changed to JS0 and JS1, respectively.
- PTEST and PFLUSH instructions cause an undetermined number of bus cycles; the user should not execute these instructions.
- The access control unit (ACU) replaces the MMU. The MC68EC040 and MC68EC040V ACU has two data and two instruction registers that are called data and instruction transparent translation registers in the MC68040.
- The MC68EC040 and MC68EC040V do not implement the DLE, multiplexed, or output buffer impedance selection modes of operation. They only implement the small output buffer mode of operation. All MC68EC040 and MC68EC040V timing and drive capabilities are equivalent to the MC68040 in small output buffer mode.
- The MC68EC040 and MC68EC040V do not contain an FPU, causing unimplemented floating-point exceptions to occur using a new stack frame format.
- The MC68040V is a 3.3 volt static microprocessor that operates down to 0 MHz.

Refer to **Appendix B MC68EC040** for specific details on the MC68EC040. Refer to **Appendix B MC68EC040** and **Appendix C MC68040V and MC68EC040V** for specific details on the MC68EC040V. **Disregard information concerning the FPU and MMU** when reading the following subsections.

1.2 FEATURES

The main features of the M68040 are as follows:

- 6-Stage Pipeline, MC68030-Compatible IU
- MC68881/MC68882-Compatible FPU
- Independent Instruction and Data MMUs
- Simultaneously Accessible, 4-Kbyte Physical Instruction Cache and 4-Kbyte Physical Data Cache
- Low-Latency Bus Accesses for Reduced Cache Miss Penalty
- Multimaster/Multiprocessor Support via Bus Snooping
- Concurrent IU, FPU, MMU, and Bus Controller Operation Maximizes Throughput
- 32-Bit, Nonmultiplexed External Address and Data Buses with Synchronous Interface
- User Object-Code Compatible with All Earlier M68000 Microprocessors
- 4-Gbyte Direct Addressing Range
- Software Support Including Optimizing C Compiler and UNIX® System V Port

The on-chip FPU and large physical instruction and data caches yield improved system performance and increased functionality. The independent instruction and data MMUs and increased internal parallelism also improve performance.

1.3 EXTENSIONS TO THE M68000 FAMILY

The M68040 is compatible with the ANSI/IEEE *Standard 754 for Binary Floating-Point Arithmetic*. The MC68040's FPU has been optimized to execute the most commonly used subset of the MC68881/MC68882 instruction sets and includes additional instruction formats for single- and double-precision rounding results. Software emulates floating-point instructions not directly supported in hardware. Refer to **Appendix E M68040 Floating-Point Emulation (MC68040FPSP)** for details on software emulation. The MOVE16 user instruction is new to the instruction set, supporting efficient 16-byte memory-to-memory data transfers.

1.4 FUNCTIONAL BLOCKS

Figure 1-1 illustrates a simplified block diagram of the MC68040. Refer to **Appendix A MC68LC040** for information on the MC68LC040's and MC68040V's functional blocks; and **Appendix B MC68EC040** for information on the MC68EC040's and MC68EC040V's functional blocks.

The M68040 IU pipeline has been expanded from the MC68030 to include effective address calculation (<ea> calculate) and operand fetch (<ea> fetch) stages with commonly used effective addressing modes. Conditional branches are optimized for the

® UNIX is a registered trademark of AT&T Bell Laboratories.

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.

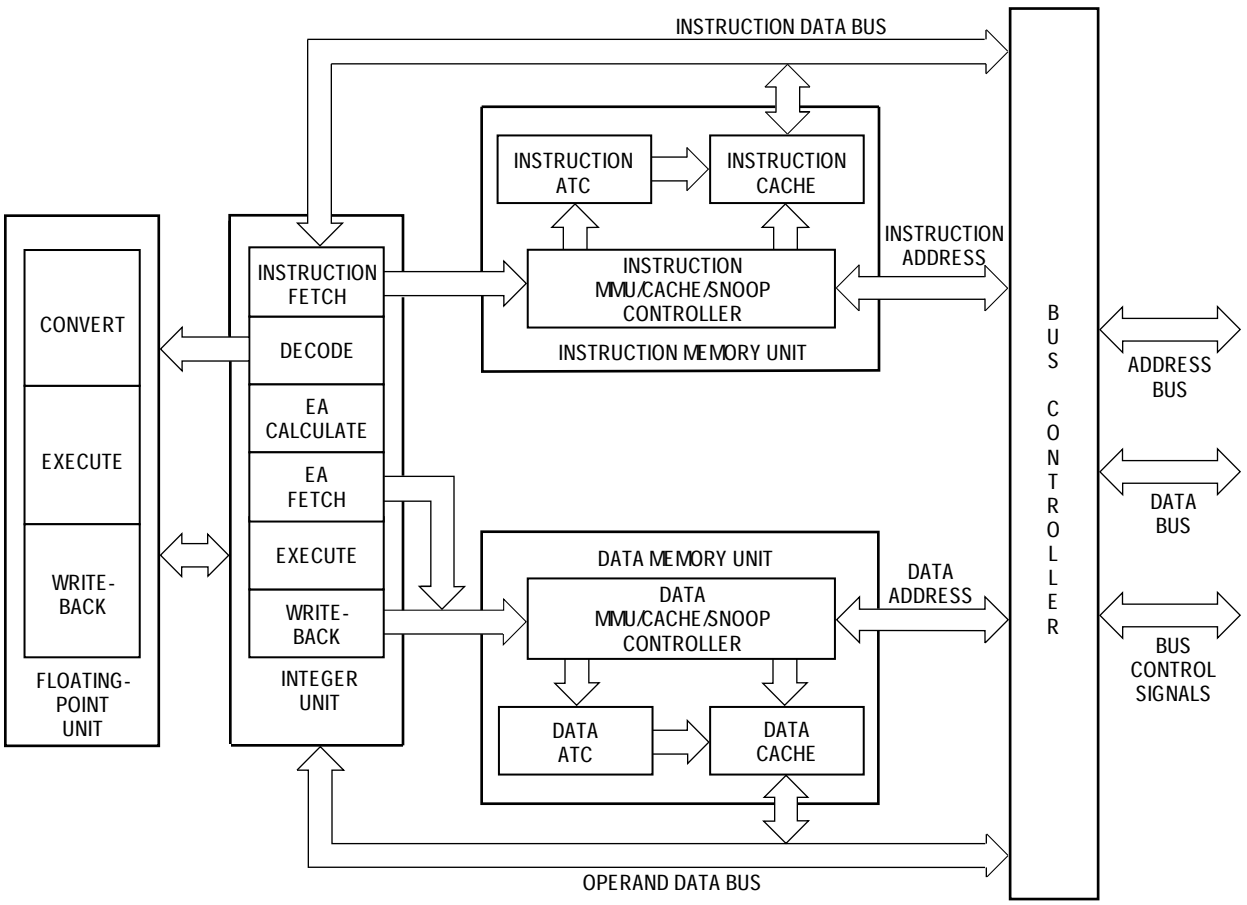


Figure 1-1. Block Diagram

To improve memory management, the M68040 includes separate, independent paged MMUs for instruction and data accesses. Each MMU stores recently used address mappings in separate 64-entry address translation caches (ATCs). Each MMU also has two transparent translation registers that define a one-to-one mapping for address space segments ranging in size from 16 Mbytes to 4 Gbytes each.

Two memory units independently interface with the IU and FPU. Each unit consists of an MMU, an ATC, a main cache, and a snoop controller. The MMUs perform memory management on a demand-page basis. By translating logical-to-physical addresses using translation tables stored in memory, the MMUs support virtual memory systems. Each MMU stores recently used address mappings in an ATC, reducing the average translation time.

Separate on-chip instruction and data caches operate independently and are accessed in parallel with address translation. 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 alternate bus

masters in the system. Both caches are organized as four-way set associative with 64 sets of four lines. Each line contains four long words for a storage capability of 4 Kbytes for each cache (8 Kbytes total). Each cache and corresponding MMU is allocated separate internal address and data buses, allowing simultaneous access to both. The data cache provides write-through or copyback write modes that can be configured on a page-by-page basis. The caches are physically mapped, reducing software support for multitasking operating systems, and support external bus snooping to maintain cache coherency in multimaster systems.

The bus snoop logic provides cache coherency in multimaster applications. The bus controller executes bus transfers on the external bus and prioritizes external memory requests from each cache. The M68040 bus controller supports a high-speed, nonmultiplexed, synchronous, external bus interface supporting burst accesses 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.

The MC68040 contains an on-chip FPU, which is user object-code compatible with the MC68881/MC68882 floating-point coprocessors. The FPU has pipelined instruction execution. Floating-point instructions in the FPU execute concurrently with integer instructions in the IU.

1.5 PROCESSING STATES

The processor is always in one of three states: normal processing, exception processing, or halted. It is in the normal processing state when executing instructions, fetching instructions and operands, and storing instruction results.

Exception processing is the transition from program processing to system, interrupt, and exception handling. Exception processing includes fetching the exception vector, stacking operations, and refilling the instruction pipe caused after an exception. The processor enters exception processing when an exceptional internal condition arises such as tracing an instruction, an instruction results in a trap, or executing specific instructions. External conditions, such as interrupts and access errors, also cause exceptions. Exception processing ends when the first instruction of the exception handler begins to execute.

The processor halts when it receives an access error or generates an address error while in the exception processing state. For example, if during exception processing of one access error another access error occurs, the MC68040 is unable to complete the transition to normal processing and cannot save the internal state of the machine. The processor assumes that the system is not operational and halts. Only an external reset can restart a halted processor. Note that when the processor executes a STOP instruction, it is in a special type of normal processing state, one without bus cycles. The processor stops, but it does not halt.

1.6 PROGRAMMING MODEL

The MC68040 programming model is separated into two privilege modes: supervisor and user. The S-bit in the status register (SR) indicates the privilege mode that the processor

uses. The IU identifies a logical address by accessing either the supervisor or user address space, maintaining the differentiation between supervisor and user modes. The MMUs use the indicated privilege mode to control and translate memory accesses, protecting supervisor code, data, and resources from user program accesses. Refer to **Appendix B MC68EC040** for details concerning the MC68EC040 address translation.

Programs access registers based on the indicated mode. User programs can only access registers specific to the user mode; whereas, system software executing in the supervisor mode can access all registers, using the control registers to perform supervisory functions. User programs are thus restricted from accessing privileged information, and the operating system performs management and service tasks for the user programs by coordinating their activities. This difference allows the supervisor mode to protect system resources from uncontrolled accesses.

Most instructions execute in either mode, but some instructions that have important system effects are privileged and can only execute in the supervisor mode. For instance, user programs cannot execute the STOP or RESET instructions. To prevent a user program from entering the supervisor mode, except in a controlled manner, instructions that can alter the S-bit in the SR are privileged. The TRAP instructions provide controlled access to operating system services for user programs.

If the S-bit in the SR is set, the processor executes instructions in the supervisor mode. Because the processor performs all exception processing in the supervisor mode, all bus cycles generated during exception processing are supervisor references, and all stack accesses use the active supervisor stack pointer. If the S-bit of the SR is clear, the processor executes instructions in the user mode. The bus cycles for an instruction executed in the user mode are user references. The values on the transfer modifier pins indicate either supervisor or user accesses.

The processor utilizes the user mode and the user programming model when it is in normal processing. During exception processing, the processor changes from user to supervisor mode. Exception processing saves the current value of the SR on the active supervisor stack and then sets the S-bit, forcing the processor into the supervisor mode. To return to the user mode, a system routine must execute one of the following instructions: MOVE to SR, ANDI to SR, EORI to SR, ORI to SR, or RTE, which execute in the supervisor mode, modifying the S-bit of the SR. After these instructions execute, the instruction pipeline is flushed and is refilled from the appropriate address space.

The MC68040 integrates the functions of the IU, FPU, and MMU. The registers depicted in the programming model (see Figure 1-2) provide operand storage and control for these three units. The registers are partitioned into two levels of privilege modes: user and supervisor. The user programming model is the same as the user programming model of the MC68030, which consists of 16, general-purpose, 32-bit registers and two control registers. The MC68040 user programming model also incorporates the MC68881/MC68882 programming model consisting of eight, 80-bit, floating-point data registers, a floating-point control register, a floating-point status register, and a floating-point instruction address register.

Only system programmers can use the supervisor programming model to implement operating system functions, I/O control, and memory management subsystems. This supervisor/user distinction in the M68000 family architecture allows for the writing of application software that executes in the user mode and migrates to the MC68040 from any M68000 family platform without modification. The supervisor programming model contains the control features that system designers need to modify system software when porting to a new design. For example, only the supervisor software can read or write to the transparent translation registers of the MC68040. The existence of the transparent translation registers does not affect the programming resources of user application programs.

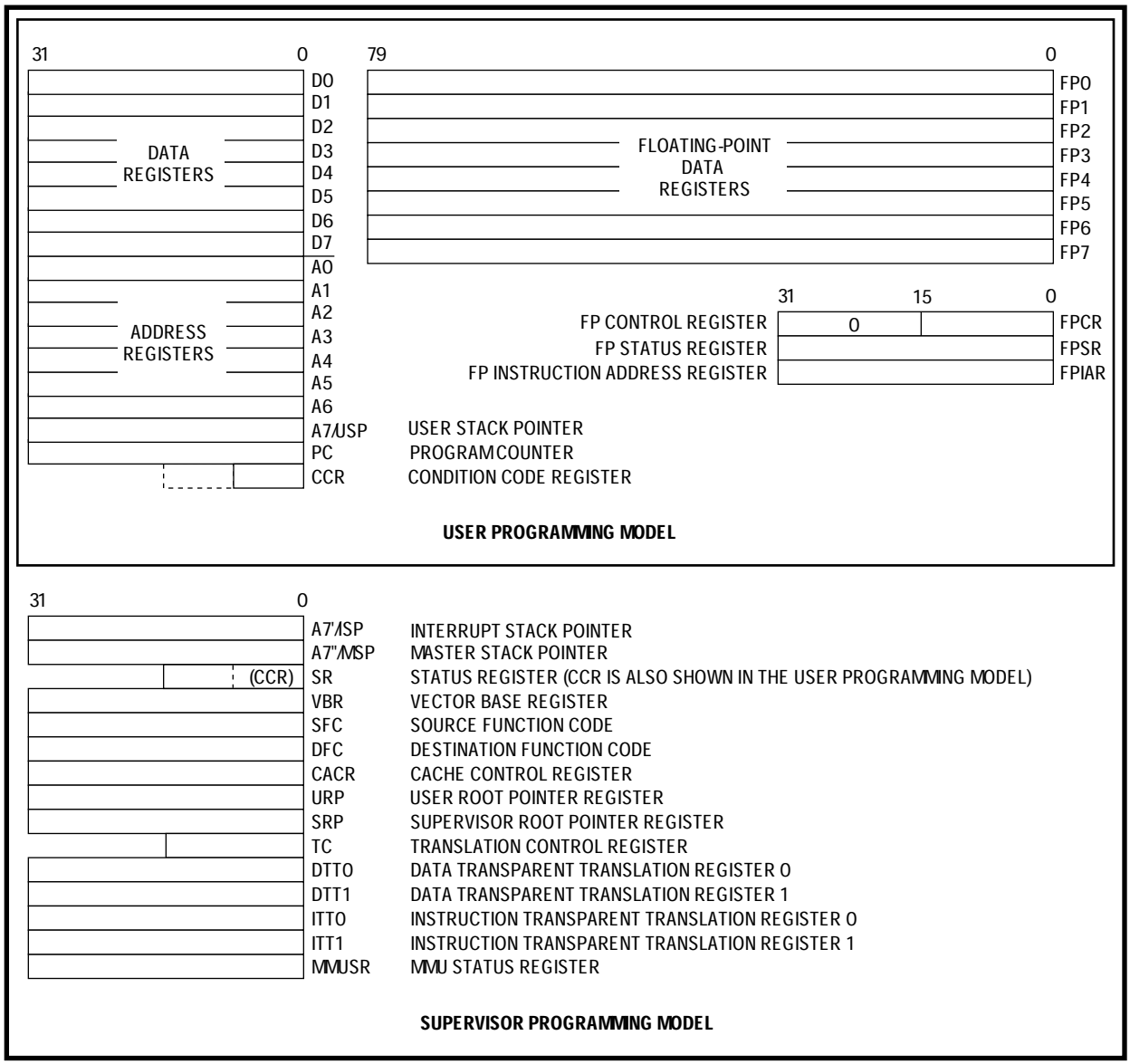


Figure 1-2. Programming Model

The user programming model includes eight data registers, seven address registers, and a stack pointer register. The address registers and stack pointer can be used as base address registers or software stack pointers, and any of the 16 registers can be used as index registers. Two control registers are available in the user mode—the program counter (PC), which usually contains the address of the instruction that the MC68040 is executing, and the lower byte of the SR, which is accessible as the condition code register (CCR). The CCR contains the condition codes that reflect the results of a previous operation and can be used for conditional instruction execution in a program.

The supervisor programming model includes the upper byte of the SR, which contains operation control information. The vector base register (VBR) contains the base address of the exception vector table, which is used in exception processing. The source function code (SFC) and destination function code (DFC) registers contain 3-bit function codes. These function codes can be considered extensions to the 32-bit logical address. The processor automatically generates function codes to select address spaces for data and program accesses in the user and supervisor modes. Some instructions use the alternate function code registers to specify the function codes for various operations.

The cache control register (CACR) controls enabling of the on-chip instruction and data caches of the MC68040. The supervisor root pointer (SRP) and user root pointer (URP) registers point to the root of the address translation table tree to be used for supervisor and user mode accesses.

The translation control register (TCR) enables logical-to-physical address translation and selects either 4- or 8-Kbyte page sizes. There are four transparent translation registers, two for instruction accesses and two for data accesses. These registers allow portions of the logical address space to be transparently mapped and accessed without the use of resident descriptors in an ATC. The MMU status register (MMUSR) contains status information derived from the execution of a PTEST instruction. The PTEST instruction searches the translation tables for the logical address, specified by this instruction's effective address field and the DFC, and returns status information corresponding to the translation.

The user programming model can also access the entire floating-point programming model. The eight 80-bit floating-point data registers are analogous to the integer data registers. A 32-bit floating-point control register (FPCR) contains an exception enable byte that enables and disables traps for each class of floating-point exceptions and a mode byte that sets the user-selectable rounding and precision modes. A floating-point status register (FPSR) contains a condition code byte, quotient byte, exception status byte, and accrued exception byte. A floating-point exception handler can use the address in the 32-bit floating-point instruction address register (FPIAR) to locate the floating-point instruction that has caused an exception. Instructions that do not modify the FPIAR can be used to read the FPIAR in the exception handler without changing the previous value.

1.7 DATA FORMAT SUMMARY

The M68040 supports the basic data formats of the M68000 family. Some data formats apply only to the IU, some only to the FPU, and some to both. In addition, the instruction set supports operations on other data formats such as memory addresses.

The operand data formats supported by the IU are the standard twos-complement data formats defined in the M68000 family architecture plus a new data format (16-byte block) for the MOVE16 instruction. Registers, memory, or instructions themselves can contain IU operands. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation.

Whenever an integer is used in a floating-point operation, the FPU automatically converts it to an extended-precision floating-point number before using the integer. The FPU implements single- and double-precision floating-point data formats as defined by the IEEE 754 standard. The FPU does not directly support packed decimal real format. However, by trapping as an unimplemented data format instead of as an illegal instruction, software emulation supports the packed decimal format. Additionally, each data format has a special encoding that represents one of five data types: normalized numbers, denormalized numbers, zeros, infinities, and not-a-numbers (NANs). Table 1-1 lists the data formats for both the IU and the FPU. Refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual,* for details on data format organization in registers and memory.

Table 1-1. M68040 Data Formats

Operand Data Format	Size	Supported In	Notes
Bit	1 Bit	IU	—
Bit Field	1–32 Bits	IU	Field of Consecutive Bits
Binary-Coded Decimal (BCD)	8 Bits	IU	Packed: 2 Digits/Byte; Unpacked: 1 Digit/Byte
Byte Integer	8 Bits	IU, FPU	—
Word Integer	16 Bits	IU, FPU	—
Long-Word Integer	32 Bits	IU, FPU	—
Quad-Word Integer	64 Bits	IU	Any Two Data Registers
16-Byte	128 Bits	IU	Memory Only, Aligned to 16-Byte Boundary
Single-Precision Real	32 Bits	FPU	1-Bit Sign, 8-Bit Exponent, 23-Bit Fraction
Double-Precision Real	64 Bits	FPU	1-Bit Sign, 11-Bit Exponent, 52-Bit Fraction
Extended-Precision Real	80 Bits	FPU	1-Bit Sign, 15-Bit Exponent, 64-Bit Mantissa

1.8 ADDRESSING CAPABILITIES SUMMARY

The M68040 supports the basic addressing modes of the M68000 family. 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. Besides these addressing modes, the M68040 provides index sizing and scaling features.

An instruction's addressing mode can specify the value of an operand, a register containing the operand, or how to derive the effective address of an operand in memory. Each addressing mode has an assembler syntax. Some instructions imply the addressing mode for an operand. These instructions include the appropriate fields for operands that use only one addressing mode. Table 1-2 lists a summary of the effective addressing modes for the M68040. Refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual,* for details on instruction format and addressing modes.

Table 1-2. Effective Addressing Modes

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

1.9 NOTATIONAL CONVENTIONS

Table 1-3 lists the notation conventions used throughout this manual unless otherwise specified.

Table 1-3. Notational Conventions

Single- And Double-Operand Operations	
+	Arithmetic addition or postincrement indicator.
–	Arithmetic subtraction or predecrement indicator.
×	Arithmetic multiplication.
÷	Arithmetic division or conjunction symbol.
~	Invert; operand is logically complemented.
Λ	Logical AND
V	Logical OR
⊕	Logical exclusive OR
⌀	Source operand is moved to destination operand.
↨ ⌀	Two operands are exchanged.
<op>	Any double-operand operation.
<operand>tested	Operand is compared to zero and the condition codes are set appropriately.
sign-extended	All bits of the upper portion are made equal to the high-order bit of the lower portion.
Other Operations	
TRAP	Equivalent to Format ÷ Offset Word ⌀ (SSP); SSP – 2 ⌀ SSP; PC ⌀ (SSP); SSP – 4 ⌀ SSP; SR ⌀ (SSP); SSP – 2 ⌀ SSP; (Vector) ⌀ PC
STOP	Enter the stopped state, waiting for interrupts.
<operand>$_{10}$	The operand is BCD; operations are performed in decimal.
If <condition> then <operations> else <operations>	Test the condition. If true, the operations after “then” are performed. If the condition is false and the optional “else” clause is present, the operations after “else” are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.
Register Specification	
An	Any Address Register n (example: A3 is address register 3)
Ax, Ay	Source and destination address registers, respectively.
BR	Base Register—An, PC, or suppressed.
Dc	Data register D7–D0, used during compare.
Dh, Dl	Data registers high- or low-order 32 bits of product.
Dn	Any Data Register n (example: D5 is data register 5)
Dr, Dq	Data register’s remainder or quotient of divide.
Du	Data register D7–D0, used during update.
Dx, Dy	Source and destination data registers, respectively.
MRn	Any Memory Register n.
Rn	Any Address or Data Register
Rx, Ry	Any source and destination registers, respectively.
Xn	Index Register—An, Dn, or suppressed.

Table 1-3. Notational Conventions (Continued)

Data Format And Type	
+ inf	Positive Infinity
<fmt>	Operand Data Format: Byte (B), Word (W), Long (L), Single (S), Double (D), Extended (X), or Packed (P).
B, W, L	Specifies a signed integer data type (twos complement) of byte, word, or long word.
D	Double-precision real data format (64 bits).
k	A twos complement signed integer (–64 to +17) specifying a number's format to be stored in the packed decimal format.
P	Packed BCD real data format (96 bits, 12 bytes).
S	Single-precision real data format (32 bits).
X	Extended-precision real data format (96 bits, 16 bits unused).
– inf	Negative Infinity
Subfields and Qualifiers	
#<xxx> or #<data>	Immediate data following the instruction word(s).
( )	Identifies an indirect address in a register.
[ ]	Identifies an indirect address in memory.
bd	Base Displacement
ccc	Index into the MC68881/MC68882 Constant ROM
d_n	Displacement Value, n Bits Wide (example: d_{16} is a 16-bit displacement).
LSB	Least Significant Bit
LSW	Least Significant Word
MSB	Most Significant Bit
MSW	Most Significant Word
od	Outer Displacement
SCALE	A scale factor (1, 2, 4, or 8, for no-word, word, long-word, or quad-word scaling, respectively).
SIZE	The index register's size (W for word, L for long word).
{offset:width}	Bit field selection.
Register Names	
CCR	Condition Code Register (lower byte of status register)
DFC	Destination Function Code Register
FPcr	Any Floating-Point System Control Register (FPCR, FPSR, or FPIAR)
FPm, FPn	Any Floating-Point Data Register specified as the source or destination, respectively.
IC, DC, IC/DC	Instruction, Data, or Both Caches
MMUSR	MMU Status Register
PC	Program Counter
Rc	Any Non Floating-Point Control Register
SFC	Source Function Code Register
SR	Status Register

Table 1-3. Notational Conventions (Concluded)

Register Codes	
*	General Case.
C	Carry Bit in CCR
cc	Condition Codes from CCR
FC	Function Code
N	Negative Bit in CCR
U	Undefined, Reserved for Motorola Use.
V	Overflow Bit in CCR
X	Extend Bit in CCR
Z	Zero Bit in CCR
—	Not Affected or Applicable.
Stack Pointers	
ISP	Supervisor/Interrupt Stack Pointer
MSP	Supervisor/Master Stack Pointer
SP	Active Stack Pointer
SSP	Supervisor (Master or Interrupt) Stack Pointer
USP	User Stack Pointer
Miscellaneous	
<ea>	Effective Address
<label>	Assemble Program Label
<list>	List of registers, for example D3–D0.
LB	Lower Bound
m	Bit m of an Operand
m–n	Bits m through n of Operand
UB	Upper Bound

1.10 INSTRUCTION SET OVERVIEW

The instruction set is tailored to support high-level languages and is optimized for those instructions most commonly executed. The floating-point instructions for the M68040 are a commonly used subset of the MC68881/MC68882 instruction set with new arithmetic instructions to explicitly select single- or double-precision rounding. The remaining unimplemented instructions are less frequently used and are efficiently emulated in the M68040FPSP, maintaining compatibility with the MC68881/MC68882 floating-point coprocessors. The M68040 instruction set includes MOVE16, a new user instruction that allows high-speed transfers of 16-byte blocks between external devices such as memory to memory or coprocessor to memory. Table 1-4 provides an alphabetized listing of the M68040 instruction set's opcode, operation, and syntax. Refer to Table 1-3 for notations used in Table 1-4. The left operand in the syntax is always the source operand, and the right operand is the destination operand. Refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual,* for details on instructions used by the M68040.

Table 1-4. Instruction Set Summary

Opcode	Operation	Syntax
ABCD	BCD Source + BCD Destination + X ⌀ Destination	ABCD Dy,Dx ABCD –(Ay),–(Ax)
ADD	Source + Destination ⌀ Destination	ADD <ea>,Dn ADD Dn,<ea>
ADDA	Source + Destination ⌀ Destination	ADDA <ea>,An
ADDI	Immediate Data + Destination ⌀ Destination	ADDI #<data>,<ea>
ADDQ	Immediate Data + Destination ⌀ Destination	ADDQ #<data>,<ea>
ADDX	Source + Destination + X ⌀ Destination	ADDX Dy,Dx ADDX –(Ay),–(Ax)
AND	Source Λ Destination ⌀ Destination	AND <ea>,Dn AND Dn,<ea>
ANDI	Immediate Data Λ Destination ⌀ Destination	ANDI #<data>,<ea>
ANDI to CCR	Source Λ CCR ⌀ CCR	ANDI #<data>,CCR
ANDI to SR	If supervisor state then Source Λ SR ⌀ SR else TRAP	ANDI #<data>,SR
ASL, ASR	Destination Shifted by count ⌀ Destination	ASd Dx,Dy[1] ASd #<data>,Dy[1] ASd <ea>[1]
Bcc	If condition true then PC + d_n ⌀ PC	Bcc <label>
BCHG	~(bit number of Destination) ⌀ Z; ~(bit number of Destination) ⌀ (bit number) of Destination	BCHG Dn,<ea> BCHG #<data>,<ea>
BCLR	~(bit number of Destination) ⌀ Z; 0 ⌀ bit number of Destination	BCLR Dn,<ea> BCLR #<data>,<ea>
BFCHG	~(bit field of Destination) ⌀ bit field of Destination	BFCHG <ea>{offset:width}
BFCLR	0 ⌀ bit field of Destination	BFCLR <ea>{offset:width}
BFEXTS	bit field of Source ⌀ Dn	BFEXTS <ea>{offset:width},Dn
BFEXTU	bit offset of Source ⌀ Dn	BFEXTU <ea>{offset:width},Dn
BFFFO	bit offset of Source Bit Scan ⌀ Dn	BFFFO <ea>{offset:width},Dn
BFINS	Dn ⌀ bit field of Destination	BFINS Dn,<ea>{offset:width}
BFSET	1s ⌀ bit field of Destination	BFSET <ea>{offset:width}
BFTST	bit field of Destination	BFTST <ea>{offset:width}
BKPT	Run breakpoint acknowledge cycle; TRAP as illegal instruction	BKPT #<data>
BRA	PC + d_n ⌀ PC	BRA <label>
BSET	~(bit number of Destination) ⌀ Z; 1 ⌀ bit number of Destination	BSET Dn,<ea> BSET #<data>,<ea>
BSR	SP – 4 ⌀ SP; PC ⌀ (SP); PC + d_n ⌀ PC	BSR <label>

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
BTST	–(bit number of Destination) ⌀ Z;	BTST Dn,<ea> BTST #<data>,<ea>
CAS	CAS Destination – Compare Operand ⌀ cc; if Z, Update Operand ⌀ Destination else Destination ⌀ Compare Operand	CAS Dc,Du,<ea>
CAS2	CAS2 Destination 1 – Compare 1 ⌀ cc; if Z, Destination 2 – Compare ⌀ cc; if Z, Update 1 ⌀ Destination 1; Update 2 ⌀ Destination 2 else Destination 1 ⌀ Compare 1; Destination 2 ⌀ Compare 2	CAS2 Dc1–Dc2,Du1–Du2,(Rn1)–(Rn2)
CHK	If Dn < 0 or Dn > Source then TRAP	CHK <ea>,Dn
CHK2	If Rn < LB or If Rn > UB then TRAP	CHK2 <ea>,Rn
CINV	If supervisor state then invalidate selected cache lines else TRAP	CINVL <caches>, (An) CINVP <caches>, (An) CINVA <caches>
CLR	0 ⌀ Destination	CLR <ea>
CMP	Destination – Source ⌀ cc	CMP <ea>,Dn
CMPA	Destination – Source	CMPA <ea>,An
CMPI	Destination – Immediate Data	CMPI #<data>,<ea>
CMPM	Destination – Source ⌀ cc	CMPM (Ay)+,(Ax)+
CMP2	Compare Rn < LB or Rn > UB and Set Condition Codes	CMP2 <ea>,Rn
CPUSH	If supervisor state then if data cache push selected dirty data cache lines; invalidate selected cache lines else TRAP	CPUSHL <caches>, (An) CPUSHP <caches>, (An) CPUSHA <caches>
DBcc	If condition false then (Dn–1 ⌀ Dn; If Dn ≠ –1 then PC + d_n ⌀ PC)	DBcc Dn,<label>
DIVS, DIVSL	Destination ÷ Source ⌀ Destination	DIVS.W <ea>,Dn 32 ÷ 16 ⌀ 16r:16q DIVS.L <ea>,Dq 32 ÷ 32 ⌀ 32q DIVS.L <ea>,Dr:Dq 64 ÷ 32 ⌀ 32r:32q DIVSL.L <ea>,Dr:Dq 32 ÷ 32 ⌀ 32r:32q
DIVU, DIVUL	Destination ÷ Source ⌀ Destination	DIVU.W <ea>,Dn 32 ÷ 16 ⌀ 16r:16q DIVU.L <ea>,Dq 32 ÷ 32 ⌀ 32q DIVU.L <ea>,Dr:Dq 64 ÷ 32 ⌀ 32r:32q DIVUL.L <ea>,Dr:Dq 32 ÷ 32 ⌀ 32r:32q
EOR	Source ⊕ Destination ⌀ Destination	EOR Dn,<ea>
EORI	Immediate Data ⊕ Destination ⌀ Destination	EORI #<data>,<ea>
EORI to CCR	Source ⊕ CCR ⌀ CCR	EORI #<data>,CCR
EORI to SR	If supervisor state then Source ⊕ SR ⌀ SR else TRAP	EORI #<data>,SR

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
EXG	Rx ↨ ⌀ Ry	EXG Dx,Dy EXG Ax,Ay EXG Dx,Ay EXG Ay,Dx
EXT EXTB	Destination Sign – Extended ⌀ Destination	EXT.W Dn extend byte to word EXT.L L Dn extend word to long word EXTB.L Dn extend byte to long word
FABS[2]	Absolute Value of Source ⌀ FPn	FABS.<fmt> <ea>,FPn FABS.X FPm,FPn FABS.X FPn FrABS.<fmt> <ea>,FPn[3] FrABS.X FPm,FPn[3] FrABS.X FPn[3]
FADD[2]	Source + FPn ⌀ FPn	FADD.<fmt> <ea>,FPn FADD.X FPm,FPn FrADD.<fmt> <ea>,FPn[3] FrADD.X FPm,FPn[3]
FBcc[2]	If condition true then PC + d_n ⌀ PC	FBcc.SIZE <label>
FCMP[2]	FPn – Source	FCMP.<fmt> <ea>,FPn FCMP.X FPm,FPn
FDBcc[2]	If condition true then no operation else Dn – 1 ⌀ Dn if Dn ≠ –1 then PC + d_n ⌀ PC else execute next instruction	FDBcc Dn,<label>
FDIV[2]	FPn ÷ Source ⌀ FPn	FDIV.<fmt> <ea>,FPn FDIV.X FPm,FPn FrDIV.<fmt> <ea>,FPn[3] FrDIV.X FPm,FPn[3]
FMOVE[2]	Source ⌀ Destination	FMOVE.<fmt> <ea>,FPn FMOVE.<fmt> FPM,<ea> FMOVE.P FPm,<ea>{Dn} FMOVE.P FPm,<ea>{#k} FrMOVE.<fmt> <ea>,FPn[3]
FMOVE[2]	Source ⌀ Destination	FMOVE.L <ea>,FPcr FMOVE.L FPcr,<ea>
FMOVEM[2]	Register List ⌀ Destination Source ⌀ Register List	FMOVEM.X <list>,<ea>[4] FMOVEM.X Dn,<ea> FMOVEM.X <ea>,<list>[4] FMOVEM.X <ea>,Dn
FMOVEM[2]	Register List ⌀ Destination Source ⌀ Register List	FMOVEM.L <list>,<ea>[5] FMOVEM.L <ea>,<list>[5]
FMUL[2]	Source × FPn ⌀ FPn	FMUL.<fmt> <ea>,FPn FMUL.X FPm,FPn FrMUL<fmt> <ea>,FPn[3] FrMUL.X FPm,FPn[3]

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
FNEG[2]	–(Source) ⌀ FPn	FNEG.<fmt> <ea>,FPn FNEG.X FPm,FPn FNEG.X FPn FrNEG.<fmt> <ea>,FPn[3] FrNEG.X FPm,FPn[3] FrNEG.X FPn[3]
FNOP[2]	None	FNOP
FRESTORE[2]	If in supervisor state then FPU State Frame ⌀ Internal State else TRAP	FRESTORE <ea>
FSAVE[2]	If in supervisor state then FPU Internal State ⌀ State Frame else TRAP	FSAVE <ea>
FScc[2]	If condition true then 1s ⌀ Destination else 0s ⌀ Destination	FScc.SIZE <ea>
FSGLDIV	FPn ÷ Source ⌀ FPn	FSGLDIV.<fmt> <ea>,FPn FSGLDIV.X FPm,FPn
FSGLMUL	Source × FPn ⌀ FPn	FSGMUL.<fmt> <ea>,FPn FSGLMUL.X FPm, FPn
FSQRT[2]	Square Root of Source ⌀ FPn	FSQRT.<fmt> <ea>,FPn FSQRT.X FPm,FPn FSQRT.X FPn FrSQRT.<fmt> <ea>,FPn[3] FrSQRT FPm,FPn[3] FrSQRT FPn[3]
FSUB[2]	FPn – Source ⌀ FPn	FSUB.<fmt> <ea>,FPn FSUB.X FPm,FPn FrSUB.<fmt> <ea>,FPn[3] FrSUB.X FPm,FPn[3]
FTRAPcc[2]	If condition true then TRAP	FTRAPcc FTRAPcc.W #<data> FTRAPcc.L #<data>
FTST[2]	Condition Codes for Operand ⌀ FPCC	FTST.<fmt> <ea> FTST.X FPm
ILLEGAL	SSP – 2 ⌀ SSP; Vector Offset ⌀ (SSP); SSP – 4 ⌀ SSP; PC ⌀ (SSP); SSp – 2 ⌀ SSP; SR ⌀ (SSP); Illegal Instruction Vector Address ⌀ PC	ILLEGAL
JMP	Destination Address ⌀ PC	JMP <ea>
JSR	SP – 4 ⌀ SP; PC ⌀ (SP) Destination Address ⌀ PC	JSR <ea>
LEA	<ea> ⌀ An	LEA <ea>,An
LINK	SP – 4 ⌀ SP; An ⌀(SP) SP ⌀ An, SP+d ⌀SP	LINK An,d_n

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
LPSTOP[6]	If supervisor state immediate data ⌀ SR SR ⌀ broadcast cycle STOP else TRAP	LPSTOP #<data>
LSL, LSR	Destination Shifted by count ⌀ Destination	LSd Dx,Dy[1] LSd #<data>,Dy[1] LSd <ea>[1]
MOVE	Source ⌀ Destination	MOVE <ea>,<ea>
MOVEA	Source ⌀ Destination	MOVEA <ea>,An
MOVE from CCR	CCR ⌀ Destination	MOVE CCR,<ea>
MOVE to CCR	Source ⌀ CCR	MOVE <ea>,CCR
MOVE from SR	If supervisor state then SR ⌀ Destination else TRAP	MOVE SR,<ea>
MOVE to SR	If supervisor state then Source ⌀ SR else TRAP	MOVE <ea>,SR
MOVE USP	If supervisor state then USP ⌀ An or An ⌀ USP else TRAP	MOVE USP,An MOVE An,USP
MOVE16	Source block ⌀ Destination block	MOVE16 (Ax)+, (Ay)+[7] MOVE16 (xxx).L, (An) MOVE16 (An), (xxx).L MOVE16 (An)+, (xxx).L
MOVEC	If supervisor state then Rc ⌀ Rn or Rn ⌀Rc else TRAP	MOVEC Rc,Rn MOVEC Rn,Rc
MOVEM	Registers ⌀ Destination Source ⌀ Registers	MOVEM <list>,<ea>[4] MOVEM <ea>,<list>[4]
MOVEP	Source ⌀ Destination	MOVEP Dx,(d_n,Ay) MOVEP (d_n,Ay),Dx
MOVEQ	Immediate Data ⌀ Destination	MOVEQ #<data>,Dn
MOVES	If supervisor state then Rn ⌀ Destination [DFC] or Source [SFC] ⌀ Rn else TRAP	MOVES Rn,<ea> MOVES <ea>,Rn
MULS	Source × Destination ⌀ Destination	MULS.W <ea>,Dn 16 × 16 ⌀ 32 MULS.L <ea>,Dl 32 × 32 ⌀ 32 MULS.L <ea>,Dh–Dl 32 × 32 ⌀ 64
MULU	Source × Destination ⌀ Destination	MULU.W <ea>,Dn 16 × 16 ⌀ 32 MULU.L <ea>,Dl 32 × 32 ⌀ 32 MULU.L <ea>,Dh–Dl 32 × 32 ⌀ 64
NBCD	0 – (Destination$_{10}$) – X ⌀ Destination	NBCD <ea>
NEG	0 – (Destination) ⌀ Destination	NEG <ea>
NEGX	0 – (Destination) – X ⌀ Destination	NEGX <ea>

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
NOP	None	NOP
NOT	~ Destination ⌀ Destination	NOT <ea>
OR	Source V Destination ⌀ Destination	OR <ea>,Dn OR Dn,<ea>
ORI	Immediate Data V Destination ⌀ Destination	ORI #<data>,<ea>
ORI to CCR	Source V CCR ⌀ CCR	ORI #<data>,CCR
ORI to SR	If supervisor state then Source V SR ⌀ SR else TRAP	ORI #<data>,SR
PACK	Source (Unpacked BCD) + adjustment ⌀ Destination (Packed BCD)	PACK –(Ax),–(Ay),#(adjustment) PACK Dx,Dy,#(adjustment)
PEA	SP – 4 ⌀ SP; <ea> ⌀ (SP)	PEA <ea>
PFLUSH[8]	If supervisor state then invalidate instruction and data ATC entries for destination address else TRAP	PFLUSH (An) PFLUSHN (An) PFLUSHA PFLUSHAN
PTEST[8]	If supervisor state then logical address status ⌀ MMUSR; entry ⌀ ATC else TRAP	PTESTR (An) PTESTW (An)
RESET	If supervisor state then Assert RSTO Line else TRAP	RESET
ROL, ROR	Destination Rotated by count ⌀ Destination	ROd Rx,Dy[1] ROd #<data>,Dy[1]
ROXL, ROXR	Destination Rotated with X by count ⌀ Destination	ROXd Dx,Dy[1] ROXd #<data>,Dy[1] ROXd <ea>[1]
RTD	(SP) ⌀ PC; SP + 4 + d_n ⌀ SP	RTD #(d_n)
RTE	If supervisor state then (SP) ⌀ SR; SP + 2 ⌀ SP; (SP) ⌀PC; SP + 4 ⌀ SP; restore state and deallocate stack according to (SP) else TRAP	RTE
RTR	(SP) ⌀ CCR; SP + 2 ⌀ SP; (SP) ⌀ PC; SP + 4 ⌀ SP	RTR
RTS	(SP) ⌀ PC; SP + 4 ⌀ SP	RTS
SBCD	Destination$_{10}$ – Source$_{10}$ – X ⌀ Destination	SBCD Dx,Dy SBCD –(Ax),–(Ay)
Scc	If condition true then 1s ⌀ Destination else 0s ⌀ Destination	Scc <ea>
STOP	If supervisor state then Immediate Data ⌀ SR; STOP else TRAP	STOP #<data>
SUB	Destination – Source ⌀ Destination	SUB <ea>,Dn SUB Dn,<ea>

Table 1-4. Instruction Set Summary (Concluded)

Opcode	Operation	Syntax
SUBA	Destination – Source ⌀ Destination	SUBA <ea>,An
SUBI	Destination – Immediate Data ⌀ Destination	SUBI #<data>,<ea>
SUBQ	Destination – Immediate Data ⌀ Destination	SUBQ #<data>,<ea>
SUBX	Destination – Source – X ⌀ Destination	SUBX Dx,Dy SUBX –(Ax),–(Ay)
SWAP	Register 31–16 ¯ ⌀ Register 15–0	SWAP Dn
TAS	Destination Tested ⌀ Condition Codes; 1 ⌀ bit 7 of Destination	TAS <ea>
TRAP	SSP – 2 ⌀ SSP; Format ÷ Offset ⌀ (SSP); SSP – 4 ⌀ SSP; PC ⌀ (SSP); SSP – 2 ⌀SSP; SR ⌀ (SSP); Vector Address ⌀ PC	TRAP #<vector>
TRAPcc	If cc then TRAP	TRAPcc TRAPcc.W #<data> TRAPcc.L #<data>
TRAPV	If V then TRAP	TRAPV
TST	Destination Tested ⌀ Condition Codes	TST <ea>
UNLK	An ⌀ SP; (SP) ⌀An; SP + 4 ⌀ SP	UNLK An
UNPK	Source (Packed BCD) + adjustment ⌀ Destination (Unpacked BCD)	UNPACK –(Ax),–(Ay),#(adjustment) UNPACK Dx,Dy,#(adjustment)

NOTES:
1. Where d is direction, left or right.
2. Available only on the MC68040.
3. Where r is rounding precision, single or double precision.
4. List refers to register.
5. List refers to control registers only.
6. Available only on the MC68040V and MC68EC040V.
7. MOVE16 (ax)+,(ay)+ is functionally the same as MOVE16 (ax),(ay)+ when ax = ay. The address register is only incremented once, and the line is copied over itself rather than to the next line.
8. Not available for the MC68EC040 or MC68EC040V.

SECTION 2
INTEGER UNIT

This section describes the organization of the M68040 integer unit (IU) and presents a brief description of the associated registers. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for details concerning the memory management unit (MMU) programming model, and to **Section 9 Floating-Point Unit (MC68040 Only)** for details concerning the floating-point unit (FPU) programming model.

2.1 INTEGER UNIT PIPELINE

The IU carries out logical and arithmetic operations using six separate subunits. Each unit is dedicated to a different stage of the IU pipeline, handling a total of six separate instructions simultaneously. Pipelining is a technique that overlaps the processing of different parts of several instructions. Pipelining simulates an assembly line with the IU containing a number of instructions in different phases of processing. The IU pipeline consists of six stages:

1. Instruction Fetch—Fetching an instruction from memory.
2. Decode—Converting an instruction into micro-instructions.
3. <ea> Calculate—If the instruction calls for data from memory, the location of the data, its memory address is calculated.
4. <ea> Fetch—Data is fetched from memory.
5. Execute—The data is manipulated during execution.
6. Write-Back—The result of the computation is written back to on-chip caches or external memory.

The pipeline contains special shadow registers that can begin processing future instructions for conditional branches while the main pipeline is processing current instructions. The <ea> calculate stage eliminates pipeline blockage for instructions with postincrement, postdecrement, or immediate add and load to address register for updates that occur in the <ea> calculate stage. The write-back stage can write data over the system bus to store a result in external memory or directly to on-chip caches. These write-backs to memory can be deferred until the most opportune moment because of the M68040 bus interface. Figure 2-1 illustrates the IU pipeline.

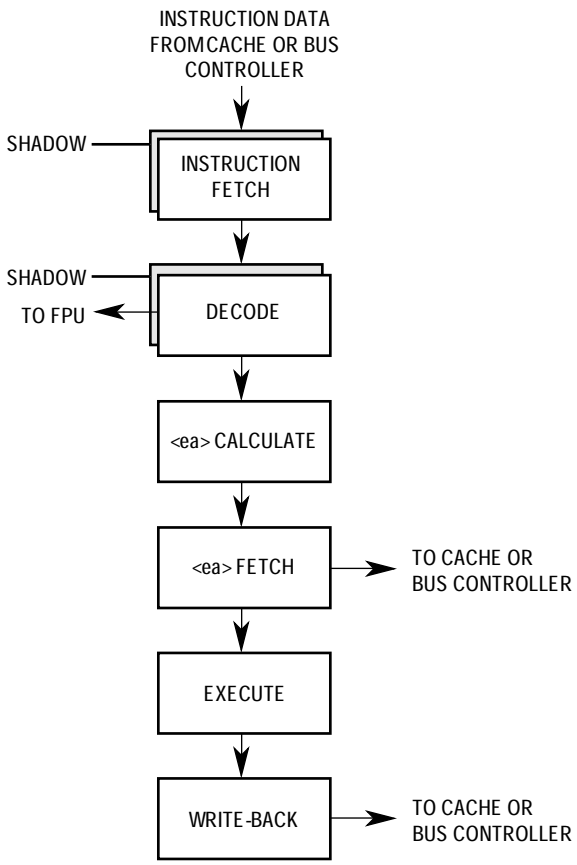


Figure 2-1. Integer Unit Pipeline

An instruction stream is fetched from the instruction memory unit and decoded on an instruction-by-instruction basis in the decode stage. Multiple instructions are fetched to keep the pipeline stages full so that the pipeline will not stall.

The decoded instruction is then passed to the <ea> calculate stage to calculate the effective addresses that the instruction requires. The <ea> calculate stage initiates additional fetches from the instruction stream to obtain the effective address extension words and performs the effective address calculation. The initial execution of the instruction in the execute stage handles any data registers required for the calculation, which passes the register back to the <ea> calculate stage.

The resulting effective address is passed to the <ea> fetch stage, which initiates an operand fetch from the data memory controller if the effective address is for a source operand. The fetched operand is returned to the execute stage, which completes execution of the instruction and writes any result to either a data register, memory, or back to the <ea> calculate stage for storage in an address register. For a memory destination, the <ea> fetch stage passes the address to the execution stage.

The previously described sequence of effective address calculation and fetch can occur multiple times for an instruction, depending on the source and/or destination addressing modes. For memory indirect addressing modes, the <ea> calculate stage initiates an operand fetch from the intermediate indirect memory address, then calculates the final

effective address. Also, some instructions access multiple memory operands and initiate fetches for each operand.

The instruction finishes execution in the execute stage. Instructions with write-back operands to memory generate pending write accesses that are passed to the write-back stage. The write occurs to the data memory unit if it is not busy. If the following instruction, which is in the <ea> fetch stage, requires an operand fetch, the write-back stalls in the write-back stage since it is at a lower priority. The write-back can stall indefinitely until either the data memory unit is free or another write is pending from the execution stage.

Figure 2-2 illustrates a write cycle, which begins in the IU pipeline. The IU stores the logical address and data for a write operation in a temporary holding register (WB3). Write operation control passes from the IU to the data memory unit once the data memory unit is idle. When the data memory unit receives the logical address and data from the IU, it stores the logical address and data to a second temporary holding register (WB2). The data memory unit then translates the logical address into a physical address. If the address translation is successful, the data memory unit either stores an address translation in the data cache (write hit) or passes it to the bus controller (write-through with write miss). Once the bus controller is ready to execute the external write operation, it multiplexes the data to the correct data byte lanes and stores the multiplexed data and physical address into a third holding register (WB1). WB1 is used in the actual write operation seen on the address and data buses. **Appendix B MC68EC040** contains details on address translation in the MC68EC040.

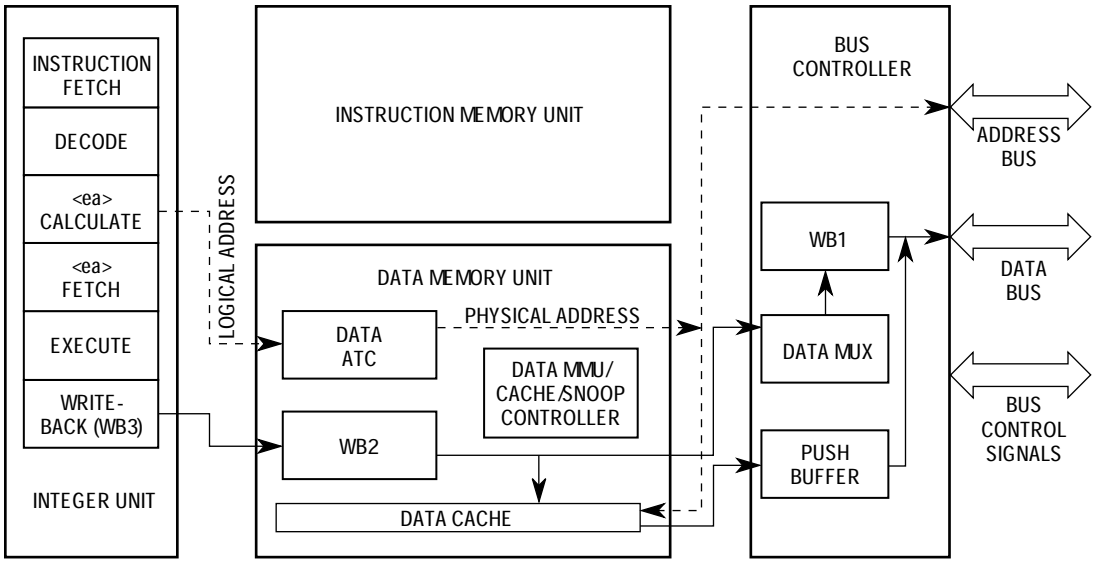


Figure 2-2. Write-Back Cycle Block Diagram

2.2 INTEGER UNIT REGISTER DESCRIPTION

The following paragraphs describe the IU registers in the user and supervisor programming models. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for details on the MMU programming model and **Section 9 Floating-Point Unit (MC68040 Only)** for details on the FPU programming model.

2.2.1 Integer Unit User Programming Model

Figure 2-3 illustrates the IU portion of the user programming model. The model is the same as for previous M68000 family microprocessors, consisting of the following registers:

- 16 General-Purpose 32-Bit Registers (D7–D0, A7–A0)
- 32-Bit Program Counter (PC)
- 8-Bit Condition Code Register (CCR)

2.2.1.1 DATA REGISTERS (D7–D0). These registers are used as data registers for bit and bit field (1 to 32 bits), byte (8 bit), word (16 bit), long-word (32 bit), and quad-word (64 bit) operations. These registers may also be used as index registers.

2.2.1.2 ADDRESS REGISTERS (A6–A0). These registers can be used as software stack pointers, index registers, or base address registers. The address registers may be used for word and long-word operations.

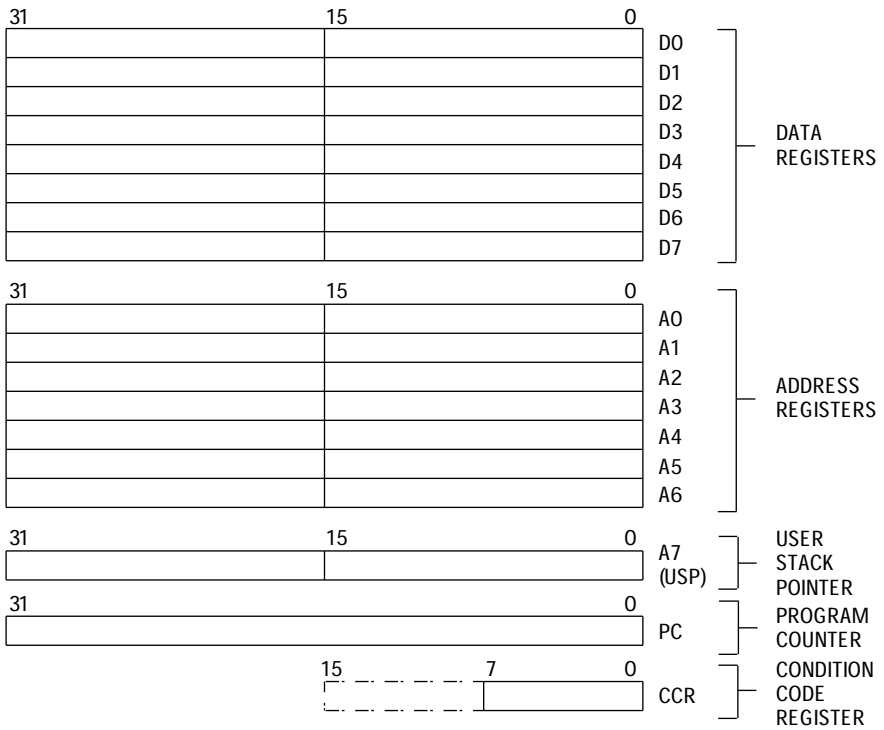


Figure 2-3. Integer Unit User Programming Model

2.2.1.3 SYSTEM STACK POINTER (A7). A7 is used as a hardware stack pointer during stacking for subroutine calls and exception handling. The register designation A7 refers to three different uses of the register: the user stack pointer (USP) (A7) in the user programming model and either the interrupt stack pointer (ISP) or master stack pointer (MSP) (A7' or A7", respectively) in the supervisor programming model. When the S-bit in the status register (SR) is clear, the USP is the active stack pointer. Explicit references to the system stack pointer (SSP) refer to the USP while the processor is operating in the user mode.

A subroutine call saves the program counter (PC) on the active system stack, and the return restores it from the active system stack. Both the PC and the SR are saved on the supervisor stack (either ISP or MSP) during the processing of exceptions and interrupts. Thus, the execution of supervisor level code is independent of user code and condition of the user stack. Conversely, user programs use the USP independently of supervisor stack requirements.

2.2.1.4 PROGRAM COUNTER. The PC contains the address of the currently executing instruction. 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. For some addressing modes, the PC can be used as a pointer for PC-relative addressing.

2.2.1.5 CONDITION CODE REGISTER. The CCR consists of five bits of the SR least significant byte. The first four bits represent a condition of the result generated by a processor operation. The fifth bit, the extend bit (X-bit), is an operand for multiprecision computations. The carry bit (C-bit) and the X-bit are separate in the M68000 family to simplify programming techniques that use them.

2.2.2 Integer Unit Supervisor Programming Model

Only system programmers use the supervisor programming model (see Figure 2-4) to implement sensitive operating system functions, I/O control, and MMU subsystems. All accesses that affect the control features of the M68040 are in the supervisor programming model. Thus, all application software is written to run in the user mode and migrates to the M68040 from any M68000 platform without modification.

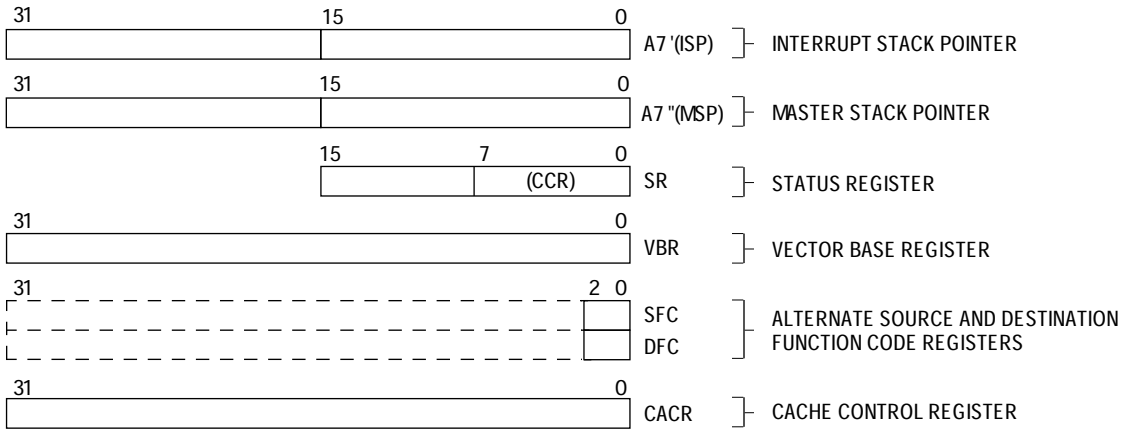


Figure 2-4. Integer Unit Supervisor Programming Model

The supervisor programming model consists of the registers available to the user as well as the following control registers:

- Two 32-Bit Supervisor Stack Pointers (ISP, MSP)
- 16-Bit Status Register (SR)
- 32-Bit Vector Base Register (VBR)
- Two 32-Bit Alternate Function Code Registers: Source Function Code (SFC) and Destination Function Code (DFC)
- 32-Bit Cache Control Register (CACR)

The following paragraphs describe the supervisor programming model registers. Additional information on the ISP, MSP, SR, and VBR registers can be found in **Section 8 Exception Processing.**

2.2.2.1 INTERRUPT AND MASTER STACK POINTERS. In a multitasking operating system, it is more efficient to have a supervisor stack pointer associated with each user task and a separate stack pointer for interrupt-associated tasks. The M68040 provides two supervisor stack pointers, master and interrupt. Explicit references to the SSP refer to either the MSP or ISP while the processor is operating in the supervisor mode. All instructions that use the SSP implicitly reference the active stack pointer. The ISP and MSP are general-purpose registers and can be used as software stack pointers, index registers, or base address registers. The ISP and MSP can be used for word and long-word operations.

The M-bit of the SR selects whether the ISP or MSP is active. SSP references access the ISP when the M-bit is clear, putting the processor into the interrupt mode. If an exception being processed is an interrupt and the M-bit is set, the M-bit is cleared, putting the processor into the interrupt mode. The interrupt mode is the default condition after reset, and all SSP references access the ISP. The ISP can be used for interrupt control information and for workspace area as interrupt exception handling requires.

SSP references access the MSP when the M-bit is set. The operating system uses the MSP for each task pointing to a task-related area of supervisor data space. This

procedure separates task-related supervisor activity from asynchronous, I/O-related supervisor tasks that can only be coincidental to the currently executing task. The MSP can separately maintain task control information for each currently executing user task, and the software updates the MSP when a task switch is performed, providing an efficient means for transferring task-related stack items. The value of the M-bit does not affect execution of privileged instructions. Instructions that affect the M-bit are MOVE to SR, ANDI to SR, EORI to SR, ORI to SR, and RTE. The processor automatically saves the M-bit value and clears it in the SR as part of the exception processing for interrupts.

2.2.2.2 STATUS REGISTER. The SR (see Figure 2-5) stores the processor status. In the supervisor mode, software can access the full SR, including the CCR available in user mode (see **2.2.1.5 Condition Code Register**) and the interrupt priority mask and additional control bits available only in the supervisor mode. These bits indicate the following states for the processor: one of two trace modes (T1, T0), supervisor or user mode (S), and master or interrupt mode (M).

The term SSP refers to the ISP and MSP. The M and S bits of the SR decide which SSP to use. When the S-bit is one and the M-bit is zero, the ISP is the active stack pointer; when the S-bit is one and the M-bit is one, the MSP is the active stack pointer. The ISP is the default mode after reset and corresponds to the MC68000, MC68008, MC68010, and CPU32 supervisor mode.

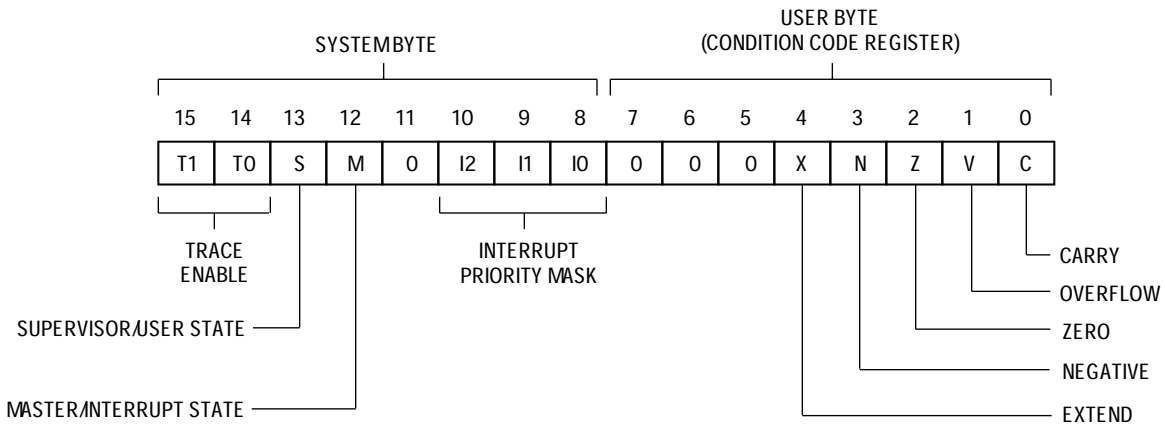


Figure 2-5. Status Register

2.2.2.3 VECTOR BASE REGISTER. The VBR contains the base address of the exception vector table in memory. The displacement of an exception vector is added to the value in this register to access the vector table. Refer to **Section 8 Exception Processing** for information on exception vectors.

2.2.2.4 ALTERNATE FUNCTION CODE REGISTERS. The alternate function code registers contain 3-bit function codes. Function codes can be considered extensions of the 32-bit logical address that optionally provides as many as eight 4-Gbyte address spaces. The processor automatically generates function codes to select address spaces for data and programs at the user and supervisor modes. Certain instructions use the SFC and DFC registers to specify the function codes for operations.

2.2.2.5 CACHE CONTROL REGISTER. The CACR contains two enable bits that allow the instruction and data caches to be independently enabled or disabled. Setting an enable bit enables the associated cache without affecting the state of any lines within the cache. A hardware reset clears the CACR, disabling both caches.

SECTION 3
MEMORY MANAGEMENT UNIT
(EXCEPT MC68EC040 AND MC68EC040V)

NOTE

This section does not apply to the MC68EC040 and MC68EC040V. Refer to **Appendix B MC68EC040** for details. All references to M68040 in this section only, refer to the MC68040, MC68040V, and MC68LC040.

The M68040 supports a demand-paged virtual memory environment. Demand means that programs request memory accesses through logical addresses, and paged means that memory is divided into blocks of equal size, called page frames. Each page frame is divided into pages of the same size. The operating system assigns pages to page frames as they are required to meet the needs of the program.

The M68040 memory management includes the following features:

- Independent Instruction and Data Memory Management Units (MMUs)
- 32-Bit Logical Address Translation to 32-Bit Physical Address
- User-Defined 2-Bit Physical Address Extension
- Addresses Translated in Parallel with Indexing into Data or Instruction Cache
- 64-Entry Four-Way Set-Associative Address Translation Cache (ATC) for Each MMU (128 Total Entries)
- Global Bit Allowing Flushes of All Nonglobal Entries from ATCs
- Selectable 4K or 8K Page Size
- Separate Supervisor and User Translation tables
- Two Independent Blocks for Each MMU Can Be Defined as Transparent (Untranslated)
- Three-Level Translation Tables with Optional Indirection
- Supervisor and Write Protections
- History Bits Automatically Maintained in Descriptors
- External Translation Disable Input Signal (`MDIS`) for Emulator Support
- Caching Mode Selected on Page Basis

The MMUs completely overlap address translation time with other processing activities when the translation is resident in one of the ATCs. ATC accesses operate in parallel with

indexing into the on-chip instruction and data caches. The MMU MDIS signal dynamically disables address translation for emulation and diagnostic support.

Figure 3-1 illustrates the MMUs contained in the two memory units, one for instructions (supporting instruction prefetches) and one for data (supporting all other accesses). Each unit contains an MMU, main cache, and snoop controller. The corresponding MMUs contain two transparent translation registers, which identify blocks of memory that can be accessed without translation. The MMUs also contain control logic and corresponding address translation caches (ATCs) in which recently used logical-to-physical address translations are stored. The data memory unit contains a data write and data read buffer, and the instruction memory unit contains an instruction line read buffer. These buffers temporarily hold data until an opportune moment arises to write the data to external memory or read the operand/instruction into the integer unit.

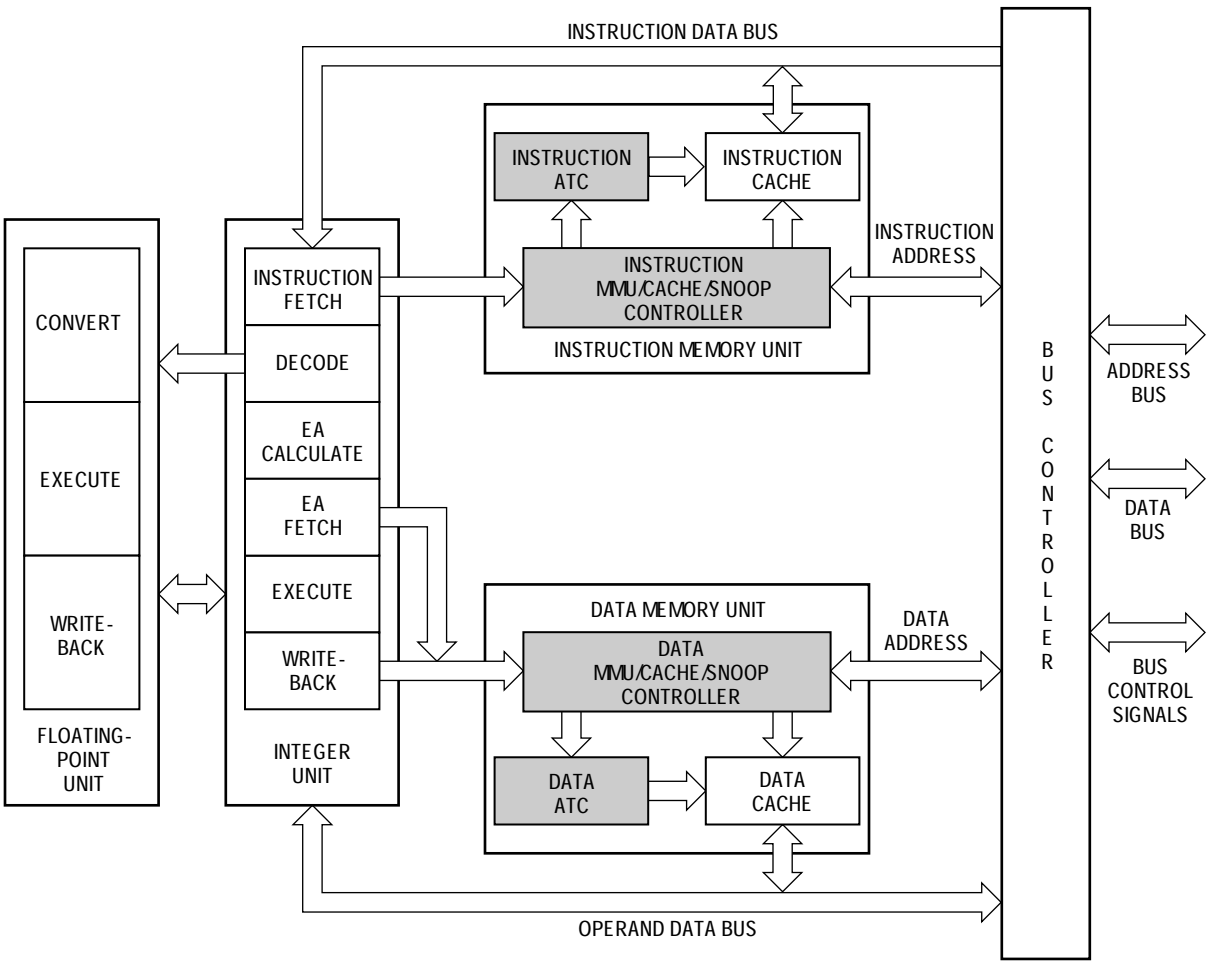


Figure 3-1. Memory Management Unit

The principal MMU function is to translate logical addresses to physical addresses using translation tables stored in memory. As the MMU receives a logical address from the integer unit, it searches its ATC for the corresponding physical address using the upper

logical address bits. If the translation is resident, the MMU provides the physical address to the cache controller, which determines if the instruction or data being accessed is cached. The cache controller uses the lower address bits to index into memory. An external bus cycle is performed only when explicitly requested by the cache controller. When the translation is not in the ATC, the MMU searches the translation tables in memory for the translation information. Microcode and dedicated logic perform the address calculations and bus cycles required for this search.

3.1 MEMORY MANAGEMENT PROGRAMMING MODEL

The memory management programming model is part of the supervisor programming model for the M68040. The eight registers that control and provide status information for address translation in the M68040 are: the user root pointer register (URP), the supervisor root pointer register (SRP), the translation control register (TCR), four independent transparent translation registers (ITT0, ITT1, DTT0, and DTT1), and the MMU status register (MMUSR). Only programs that execute in the supervisor mode can directly access these registers. Figure 3-2 illustrates the memory management programming model.

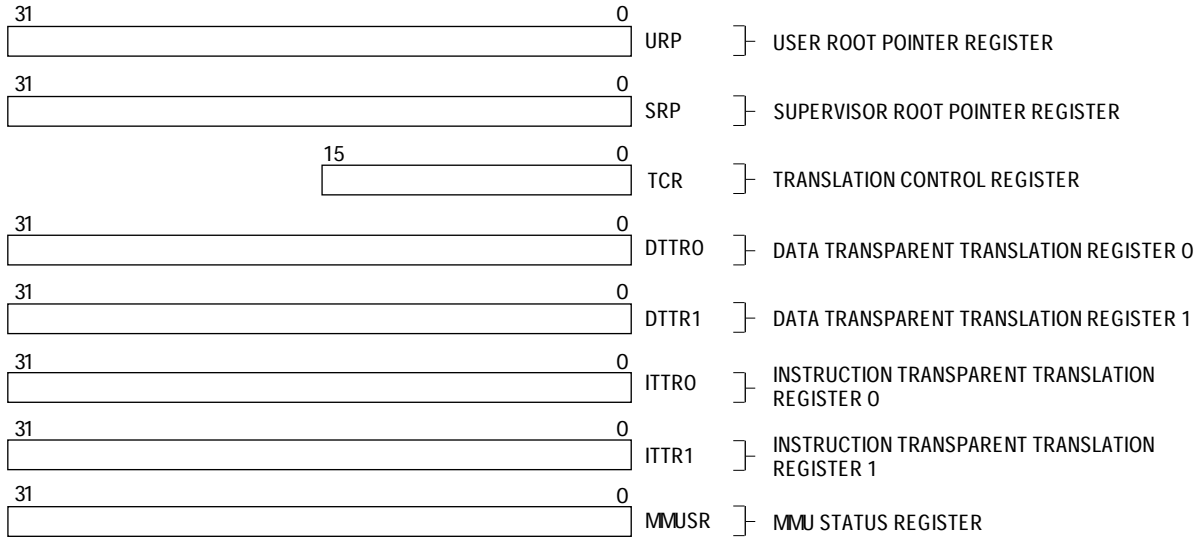


Figure 3-2. Memory Management Programming Model

3.1.1 User and Supervisor Root Pointer Registers

The SRP and URP registers each contain the physical address of the translation table's root, which the MMU uses for supervisor and user accesses, respectively. The URP points to the translation table for the current user task. When a new task begins execution, the operating system typically writes a new root pointer to the URP. A new translation table address implies that the contents of the ATCs may no longer be valid. A PFLUSH instruction should be executed to flush the ATCs before loading a new root pointer value, if necessary. Figure 3-3 illustrates the format of the 32-bit URP and SRP registers. Bits 8–

0 of an address loaded into the URP or the SRP must be zero. Transfers of data to and from these 32-bit registers are long-word transfers.

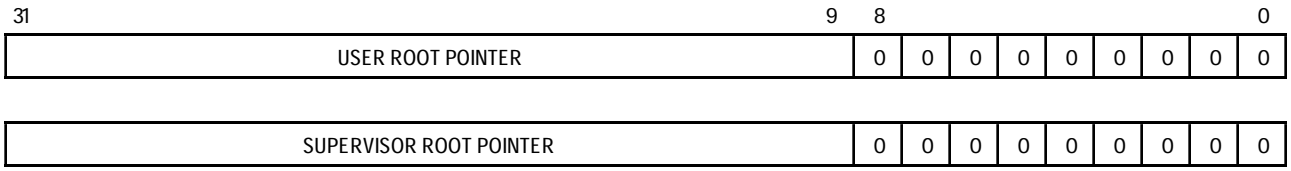
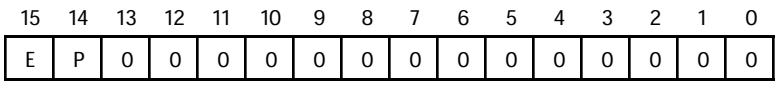


Figure 3-3. URP and SRP Register Formats

3.1.2 Translation Control Register

The 16-bit TCR contains two control bits to enable paged address translation and to select page size. The operating system must flush the ATCs before enabling address translation since the TCR accesses and reset do not flush the ATCs. All unimplemented bits of this register are read as zeros and must always be written as zeros. The M68040 always uses word transfers to access this 16-bit register. The fields of the TCRs are defined following Figure 3-4, which illustrates the TCR.



NOTE: Bits 13–0 are undefined (reserved).

Figure 3-4. Translation Control Register Format

E—Enable

This bit enables and disables paged address translation.

0 = Disable
1 = Enable

A reset operation clears this bit. When translation is disabled, logical addresses are used as physical addresses. The MMU instruction, PFLUSH, can be executed successfully despite the state of the E-bit. PTEST results are undefined if the MMU is disabled and no table search occurs. If translation is disabled and an access does not match a transparent translation register (TTR), the access has the following default attributes on the TTR: the caching mode is cachable/write-through, write protection is disabled, and the user attribute signals (UPA1 and UPA0) are zero.

P—Page Size

This bit selects the memory page size.

0 = 4 Kbytes
1 = 8 Kbytes

A reset operation does not affect this bit. The bit must be initialized after a reset.

3.1.3 Transparent Translation Registers

The data transparent translation registers (DTTR0 and DTTR1) and instruction transparent translation registers (ITTR0 and ITTR1) are 32-bit registers that define blocks of logical address space. The TTRs operate independently of the E-bit in the TCR and the state of the MDIS signal. Data transfers to and from these registers are long-word transfers. The TTR fields are defined following Figure 3-5, which illustrates TTR format. Bits 12–10, 7, 4, 3, 1, and 0 always read as zero.

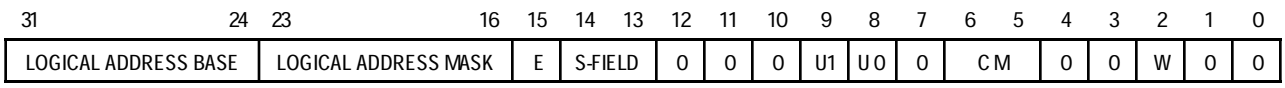


Figure 3-5. Transparent Translation Register Format

Logical Address Base

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

Logical Address Mask

Since this 8-bit field contains a mask for the Logical Address Mask field, setting a bit in this field causes the corresponding bit in the Logical Address Base field to be ignored. Blocks of memory larger than 16 Mbytes can be transparently translated by setting some of the logical address mask bits to ones. The low-order bits of this field can be set to define contiguous blocks larger than 16 Mbytes.

E—Enable

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

0 = Transparent translation disabled
1 = Transparent translation enabled

S—Supervisor Mode

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

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

U0, U1—User Page Attributes

The user defines these bits, and the M68040 does not interpret them. U0 and U1 are echoed to the UPA0 and UPA1 signals, respectively, if an external bus transfer results from an access. These bits can be programmed by the user to support external addressing, bus snooping, or other applications.

CM—Cache Mode

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

00 = Cachable, Write-through
01 = Cachable, Copyback
10 = Noncachable, Serialized
11 = Noncachable

Section 4 Instruction and Data Caches provides detailed information on caching modes, and **Section 7 Bus Operation** provides information on serialization.

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 bit in a table descriptor were clear.

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

3.1.4 MMU Status Register

The MMUSR is a 32-bit register that contains the status information returned by execution of the PTEST instruction. The PTEST instruction searches the translation tables to determine status information about the translation of a specified logical address. Transfers to and from the MMUSR are long-word transfers. The fields of the MMUSR are defined following Figure 3-6, which illustrates the MMUSR.

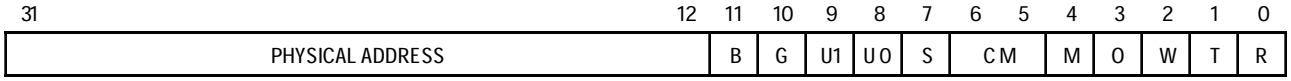


Figure 3-6. MMU Status Register Format

Physical Address

This 20-bit field contains the upper bits of the translated physical address. Merging these bits with the lower bits of the logical address forms the actual physical address. Bit 12 is undefined if a PTEST is executed with 8-Kbyte pages selected.

B—Bus Error

The B-bit is set if a transfer error is encountered during the table search for the PTEST instruction. If the B-bit is set, all other bits are zero.

G—Global

This bit is set if the G-bit is set in the page descriptor.

U1, U0—User Page Attributes

These bits are set if corresponding bits in the page descriptor are set.

S—Supervisor Protection

This bit is set if the S-bit in the page descriptor is set. Setting this bit does not indicate that a violation has occurred.

CM—Cache Mode

This 2-bit field is copied from the CM bits in the page descriptor.

M—Modified

This bit is set if the M-bit is set in the page descriptor associated with the address.

W—Write Protect

This bit is set if the W-bit is set in any of the descriptors encountered during the table search. Setting this bit does not indicate that a violation has occurred.

T—Transparent Translation Register Hit

If the T-bit is set, then the PTEST address matches an instruction or data TTR, the R-bit is set, and all other bits are zero.

R—Resident

The R-bit is set if the PTEST address matches an instruction or data TTR or if the table search completes by obtaining a valid page descriptor.

3.2 LOGICAL ADDRESS TRANSLATION

The function of the MMUs is to translate logical addresses to physical addresses. The MMUs perform translations according to control information in translation tables. The operating system creates these translation tables and stores them in memory. The processor then fetches a translation table as needed and stores it in an ATC.

3.2.1 Translation Tables

The M68040 uses the ATCs in the instruction and data memory units with translation tables stored in memory to perform the translations from logical to physical addresses. The operating system loads the translation tables for a program into memory. No distinction is made in the translation of instruction accesses versus data accesses because the instruction and data MMUs access the same translation table for a specific privilege mode, either user or supervisor. This lack of distinction results in a merged instruction and data address space.

Figure 3-7 illustrates the three-level tree structure of a general translation table supported by the M68040. The root- and pointer-level tables contain the base addresses of the tables at the next level. The page-level tables contain either the physical address for the translation or a pointer to the memory location containing the physical address. Only a portion of the translation table for the entire logical address space is required to be resident in memory at any time—specifically, only the portion of the table that translates

the logical addresses of the currently executing process. Portions of translation tables can be dynamically allocated as the process requires additional memory.

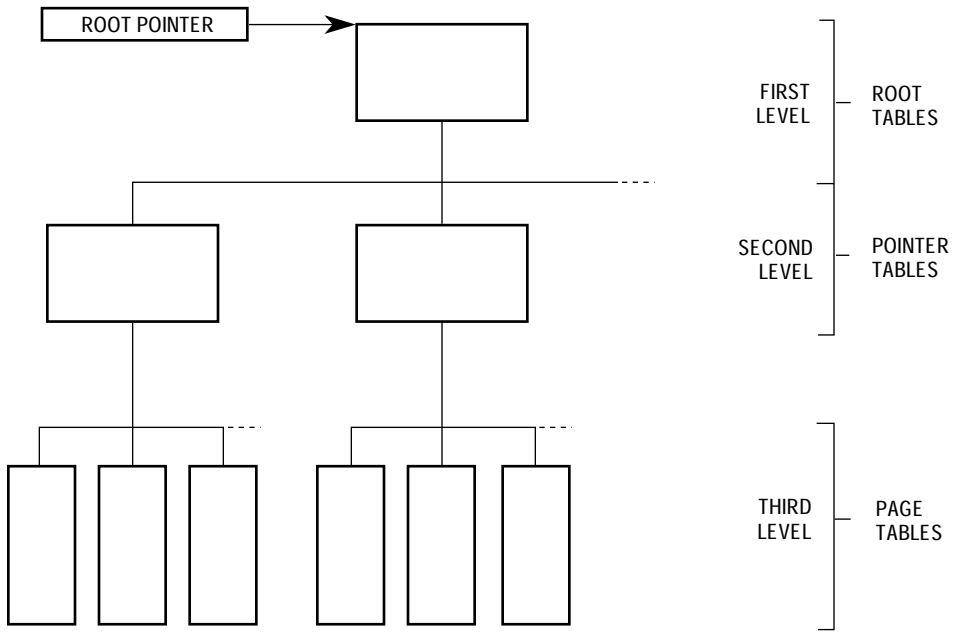


Figure 3-7. Translation Table Structure

The current privilege mode determines the use of the URP or SRP for translation of the access. The root pointer contains the base address of the translation table's root-level table. The translation table consists of tables of descriptors. The table descriptors of the root- and pointer-levels can be either resident or invalid. The page descriptors of the page-level table can be resident, indirect, or invalid. A page descriptor defines the physical address of a page frame in memory that corresponds to the logical address of a page. An indirect descriptor, which contains a pointer to the actual page descriptor, can be used when two or more logical addresses access a single page descriptor.

The table search uses logical addresses to access the translation tables. Figure 3-8 illustrates a logical address format, which is segmented into four fields: root index (RI), pointer index (PI), page index (PGI), and page offset. The first three fields extracted from the logical address index the base address for each table level. The seven bits of the logical address RI field are multiplied by 4 or shifted to the left by two bits. This sum is concatenated with the upper 23 bits of the appropriate root pointer (URP or SRP) to yield the physical address of a root-level table descriptor. Each of the 128 root-level table descriptors corresponds to a 32-Mbyte block of memory and points to the base of a pointer-level table.

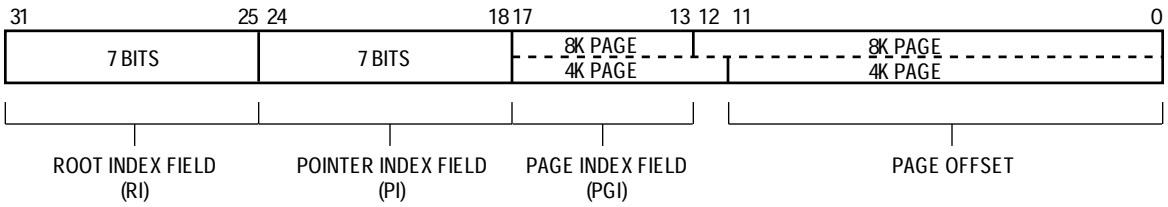


Figure 3-8. Logical Address Format

The seven bits of a logical address PI field are multiplied by 4 (shifted to the left by two bits) and concatenated with the fetched root-level descriptor's upper 23 bits to produce the physical address of the pointer-level table descriptor. Each of the 128 pointer-level table descriptors corresponds to a 256-Kbyte block of memory.

For 8-Kbyte pages, the five bits of the PGI field are multiplied by 4 (shifted to the left by two bits) and concatenated with the fetched pointer-level descriptor's upper 25 bits to produce the physical address of the 8-Kbyte page descriptor. The upper 19 bits of the page descriptor are the page frame's physical address. There are 32 8-Kbyte page descriptors in a page-level table.

Similarly, for 4-Kbyte pages, the six bits of the PGI field are multiplied by 4 (shifted to the left by two bits) and concatenated with the fetched pointer-level descriptor's upper 24 bits to produce the physical address of the 4-Kbyte page descriptor. The upper 20 bits of the page descriptor are the page frame's physical address. There are 64 4-Kbyte page descriptors in a page-level table.

Write-protect status is accumulated from each level's descriptor and combined with the status from the page descriptor to form the ATC entry status. The M68040 creates the ATC entry from the page frame address and the associated status bits and retries the original bus access. Refer to **3.3 Address Translation Caches** for details on ATC entries.

If the descriptor from a page table is an indirect descriptor, the page descriptor pointed to by this descriptor is fetched. Invalid descriptors can be used at any level of the tree except the root. When a table search for a normal translation encounters an invalid descriptor, the processor takes an access fault exception. The invalid descriptor can be used to identify either a page or branch of the tree that has been stored on an external device and is not resident in memory or a portion of the translation table that has not yet been defined. In these two cases, the exception routine can either restore the page from disk or add to the translation table. Figures 3-9 and 3-10 illustrate detailed flowcharts of table search and descriptor fetch operations.

A table search terminates successfully when a page descriptor is encountered. The occurrence of an invalid descriptor or a transfer error acknowledge also terminates a table search, and the M68040 takes an exception on the retry of the cycle because of these conditions. The exception handler should distinguish between anticipated conditions and true error conditions. The exception handler can correct an invalid descriptor that indicates a nonresident page or one that identifies a portion of the translation table yet to be allocated. An access error due to a system malfunction can require the exception handler to write an error message and terminate the task.

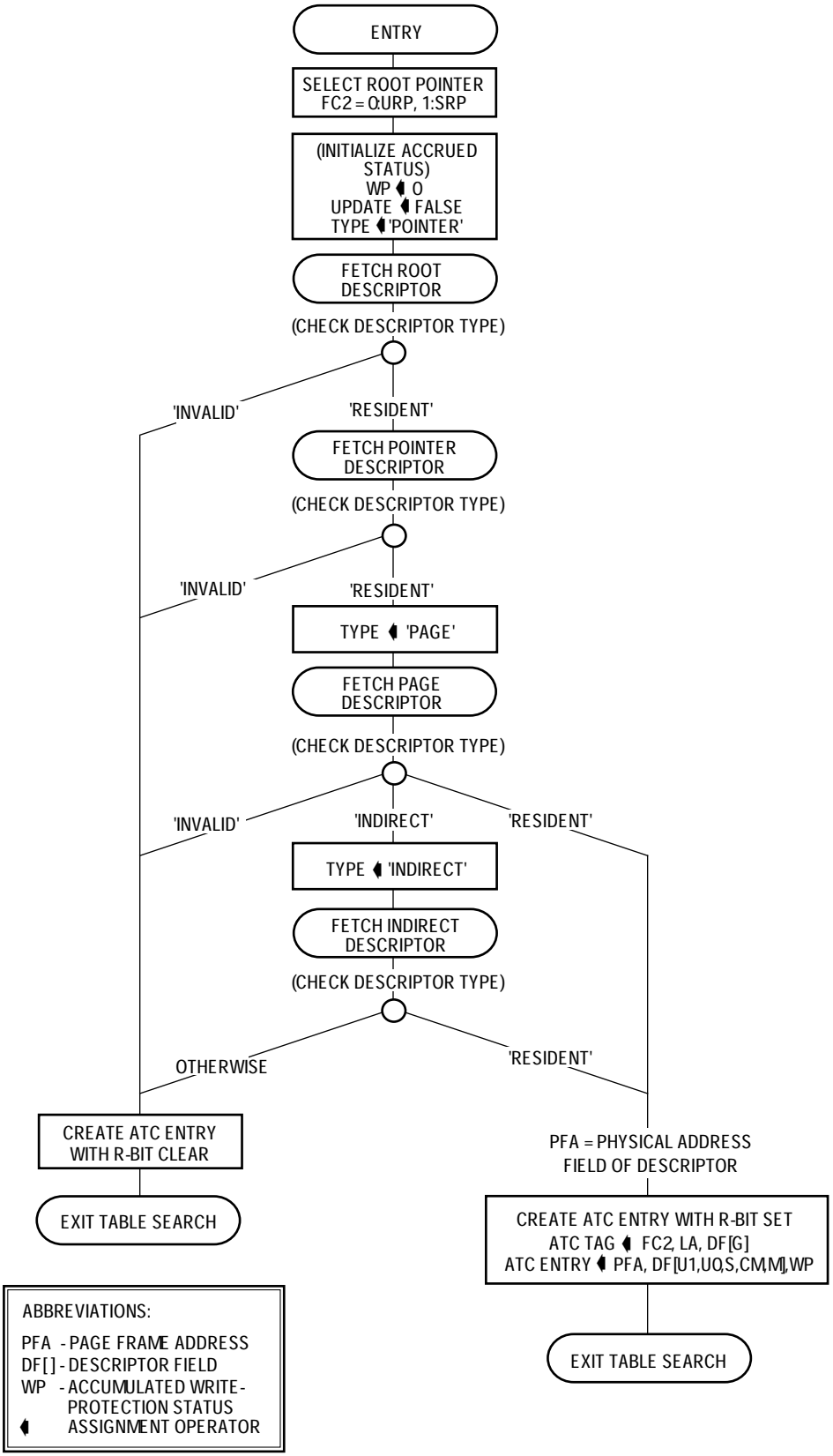


Figure 3-9. Detailed Flowchart of Table Search Operation

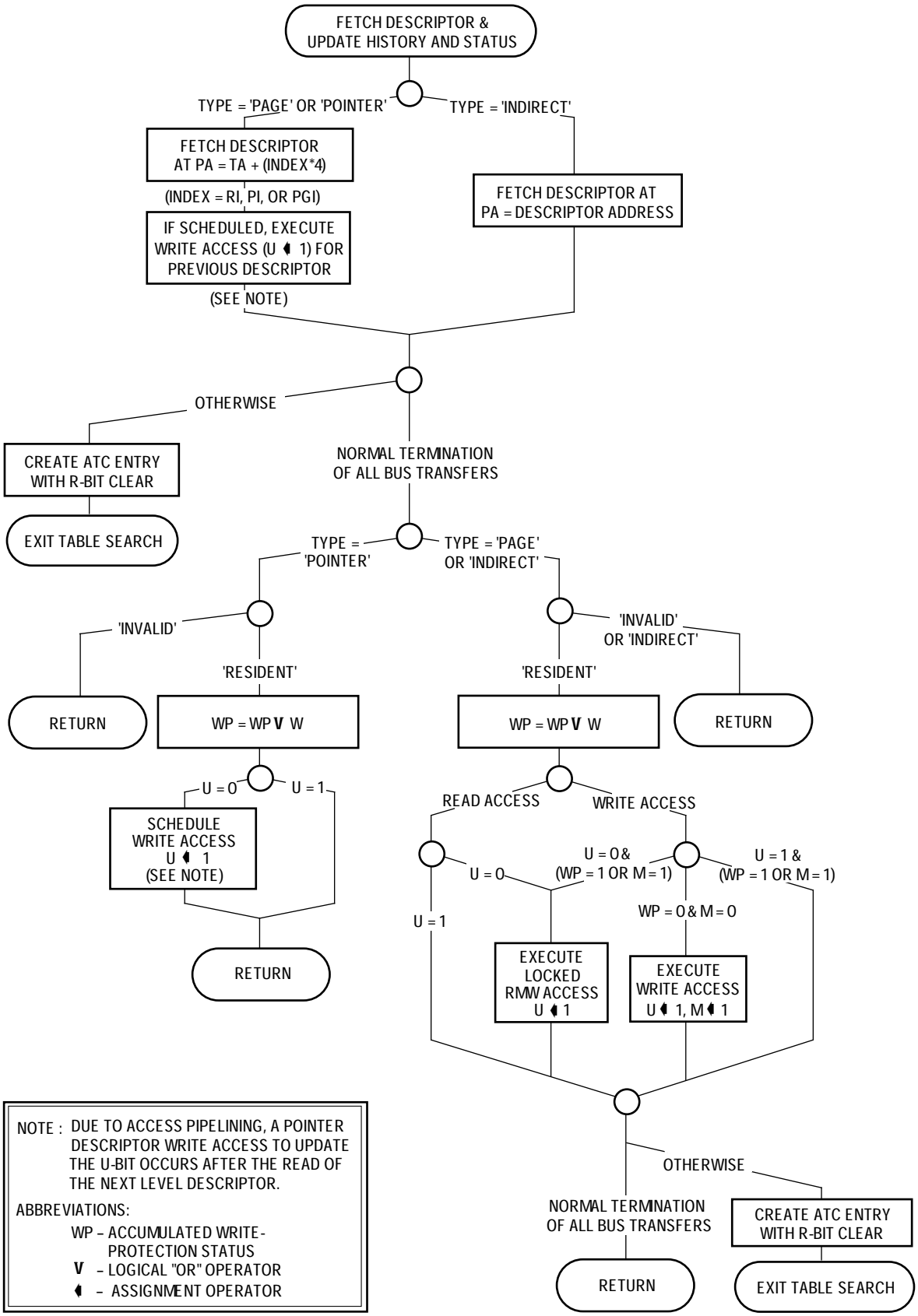


Figure 3-10. Detailed Flowchart of Descriptor Fetch Operation

Motorola highly recommends that the translation tables be placed in cache-inhibited memory space. Motorola also highly recommends table descriptors must not be left in states that are incoherent to the processor. Future processors may treat these recommendations as mandatory. The following paragraphs apply only to M68040 systems that cannot meet these recommendations.

The processor never allocates table descriptors in the data cache when the processor performs a table search. Only normal accesses to the translation tables cause descriptors to be allocated in the data cache. If table descriptors are allocated in the data cache and the cache is disabled, the processor locks up trying to access a cached descriptor during a table search. Ensuring that the data cache is invalidated before enabling the MMU or disabling the data cache and ensuring that the pages containing table descriptors are pushed and invalidated prevents lockup during table searches.

Table and page descriptors must not be left in a state that is incoherent to the processor. Violation of this restriction can result in an undefined operation. Page descriptors must not have an encoding of U-bit = 0, M-bit = 1 and PDT field = 01 or 11. This encoding indicates that the page descriptor is resident, not used, and modified. The processor's table search algorithm never leaves a descriptor in this state. This state is possible through direct manipulation by the operating system for this specific instance. A table search for a MOVE16 write can corrupt the cache line being written if the table descriptors are marked copyback.

3.2.2 Descriptors

There are two types of descriptors used in the translation tables, table and page. Table- and page-level descriptors can be further divided into types of descriptors. Root table descriptors are used in root-level tables and pointer table descriptors are used in pointer-level tables. Descriptors in the page-level tables contain either a page descriptor for the translation or an indirect descriptor that points to a memory location containing the page descriptor. The P-bit in the TCR selects the page size as either 4 or 8 Kbytes.

3.2.2.1 TABLE DESCRIPTORS. Figure 3-11 illustrates the formats of the root and pointer table descriptors. Two descriptor formats are possible at the pointer-level tables to support 4-Kbyte and 8-Kbyte page sizes.

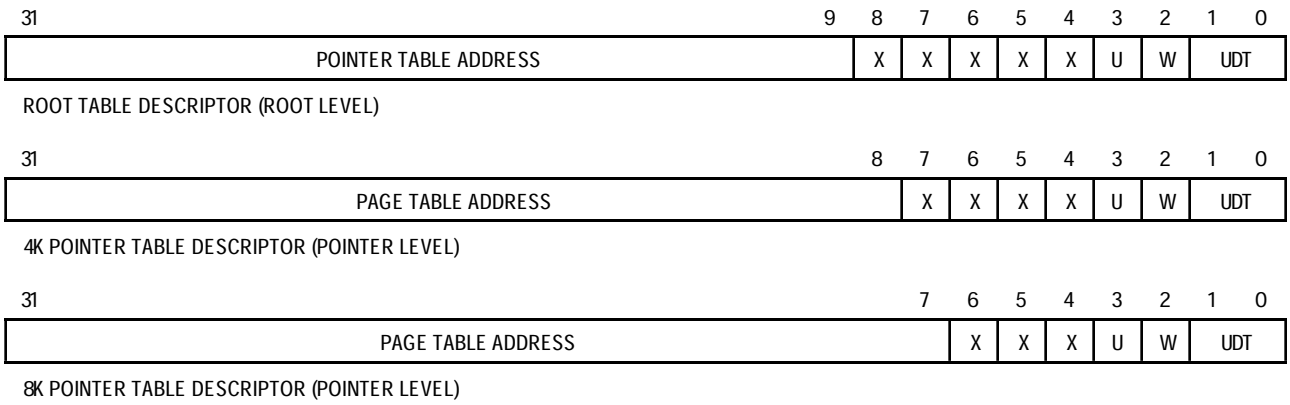


Figure 3-11. Table Descriptor Formats

3.2.2.2 PAGE DESCRIPTORS. Figure 3-12 illustrates the page descriptors for both 4-Kbyte and 8-Kbyte page sizes. Refer to **Section 4 Instruction and Data Caches** for details concerning caching page descriptors.

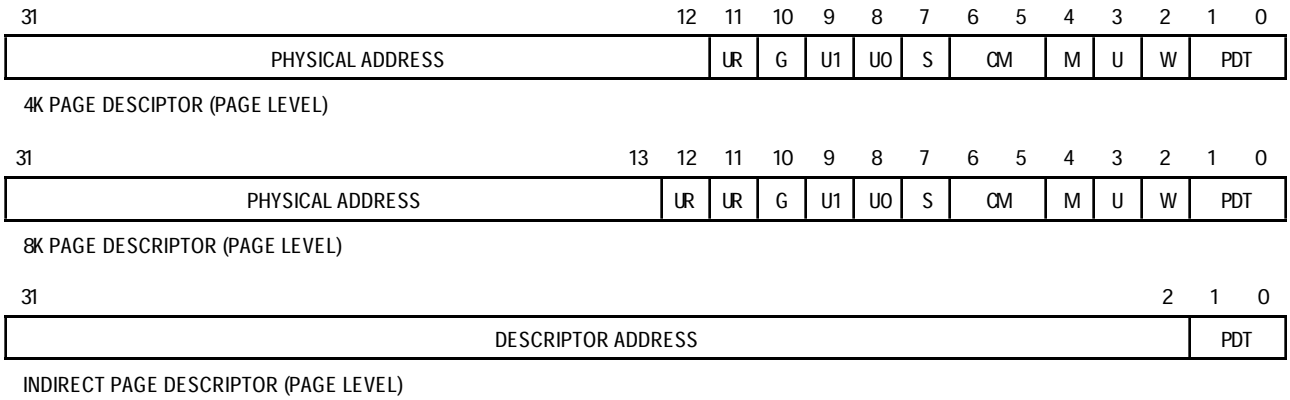


Figure 3-12. Page Descriptor Formats

3.2.2.3 DESCRIPTOR FIELD DEFINITIONS. The field definitions for the table- and page-level descriptors are listed in alphabetical order:

CM—Cache Mode

This field selects the cache mode and accesses serialization as follows:

00 = Cachable, Write-through
01 = Cachable, Copyback
10 = Noncachable, Serialized
11 = Noncachable

Section 4 Instruction and Data Caches provides detailed information on caching modes, and **Section 7 Bus Operation** provides information on serialization.

Descriptor Address

This 30-bit field, which contains the physical address of a page descriptor, is only used in indirect descriptors.

G—Global

When this bit is set, it indicates the entry is global. PFLUSH instruction variants that specify nonglobal entries do not invalidate global entries, even when all other selection criteria are satisfied. If these PFLUSH variants are not used, then system software can use this bit.

M—Modified

This bit identifies a modified page. The M68040 sets the M-bit in the corresponding page descriptor before a write operation to a page for which the M-bit is clear, except for write-protect or supervisor violations. The read portion of a read-modify-write access is considered a write for updating purposes. The M68040 never clears this bit.

PDT—Page Descriptor Type

This field identifies the descriptor as an invalid descriptor, a page descriptor for a resident page, or an indirect pointer to another page descriptor.

00 = Invalid
This code indicates that the descriptor is invalid. An invalid descriptor can represent a nonresident page or a logical address range that is out of bounds. All other bits in the descriptor are ignored. When an invalid descriptor is encountered, an ATC entry is created for the logical address with the resident bit in the MMUSR clear.

01 or 11 = Resident
These codes indicate that the page is resident.

10 = Indirect
This code indicates that the descriptor is an indirect descriptor. Bits 31–2 contain the physical address of the page descriptor. This encoding is invalid for a page descriptor pointed to by an indirect descriptor.

Physical Address

This 20-bit field contains the physical base address of a page in memory. The logical address supplies the low-order bits of the address required to index into the page. When the page size is 8-Kbyte, the least significant bit of this field is not used.

S—Supervisor Protected

This bit identifies a page as supervisor only. Only programs operating in the supervisor mode are allowed to access the portion of the logical address space mapped by this descriptor when the S-bit is set. If the bit is clear, both supervisor and user accesses are allowed.

Page Table Address

This field contains the physical base address of a table of page descriptors. The low-order bits of the address required to index into the page table are supplied by the logical address.

U—Used

The processor automatically sets this bit when a descriptor is accessed in which the U-bit is clear. In a page descriptor table, this bit is set to indicate that the page corresponding to the descriptor has been accessed. In a pointer table, this bit is set to indicate that the pointer has been accessed by the M68040 as part of a table search. The U-bit is updated before the M68040 allows a page to be accessed. The processor never clears this bit.

U0, U1—User Page Attributes

These bits are user defined and the processor does not interpret them. U0 and U1 are echoed to the UPA0 and UPA1 signals, respectively, if an external bus transfer results from the access. Applications for these bits include extended addressing and snoop protocol selection.

UDT—Upper Level Descriptor Type

These bits indicate whether the next level table descriptor is resident.

00 or 01 = Invalid
These codes indicate that the table at the next level is not resident or that the logical address is out of bounds. All other bits in the descriptor are ignored. When an invalid descriptor is encountered, an ATC entry is created for the logical address with the resident bit in the MMUSR clear.

10 or 11 = Resident
These codes indicate that the page is resident.

UR—User Reserved

These single bit fields are reserved for use by the user.

W—Write Protected

Setting the W-bit in a table descriptor write protects all pages accessed with that descriptor. When the W-bit is set, a write access or a read-modify-write access to the logical address corresponding to this entry causes an access error exception to be taken.

X—Motorola Reserved

These bit fields are reserved for future use by Motorola.

3.2.3 Translation Table Example

Figure 3-13 illustrates an access example to the logical address $76543210 while in the supervisor mode with an 8-Kbyte memory page size. The RI field of the logical address, $3B, is mapped into bits 8–2 of the SRP value to select a 32-bit root table descriptor at a root-level table. The selected root table descriptor points to the base of a pointer-level table, and the PI field of the logical address, $15, is mapped into bits 8–2 of this base address to select a pointer descriptor within the table. This pointer table descriptor points to the base of a page-level table, and the PGI field of the logical address, $1, is mapped into bits 6–2 of this base address to select a page descriptor within the table.

3.2.4 Variations in Translation Table Structure

Several aspects of the MMU translation table structure are software configurable, allowing the system designer flexibility to optimize the performance of the MMUs for a particular system. The following paragraphs discuss the variations of the translation table structure.

3.2.4.1 INDIRECT ACTION. The M68040 provides the ability to replace an entry in a page table with a pointer to an alternate entry. The indirection capability allows multiple tasks to share a physical page while maintaining only a single set of history information for the page (i.e., the modified indication is maintained only in the single descriptor). The indirection capability also allows the page frame to appear at arbitrarily different addresses in the logical address spaces of each task.

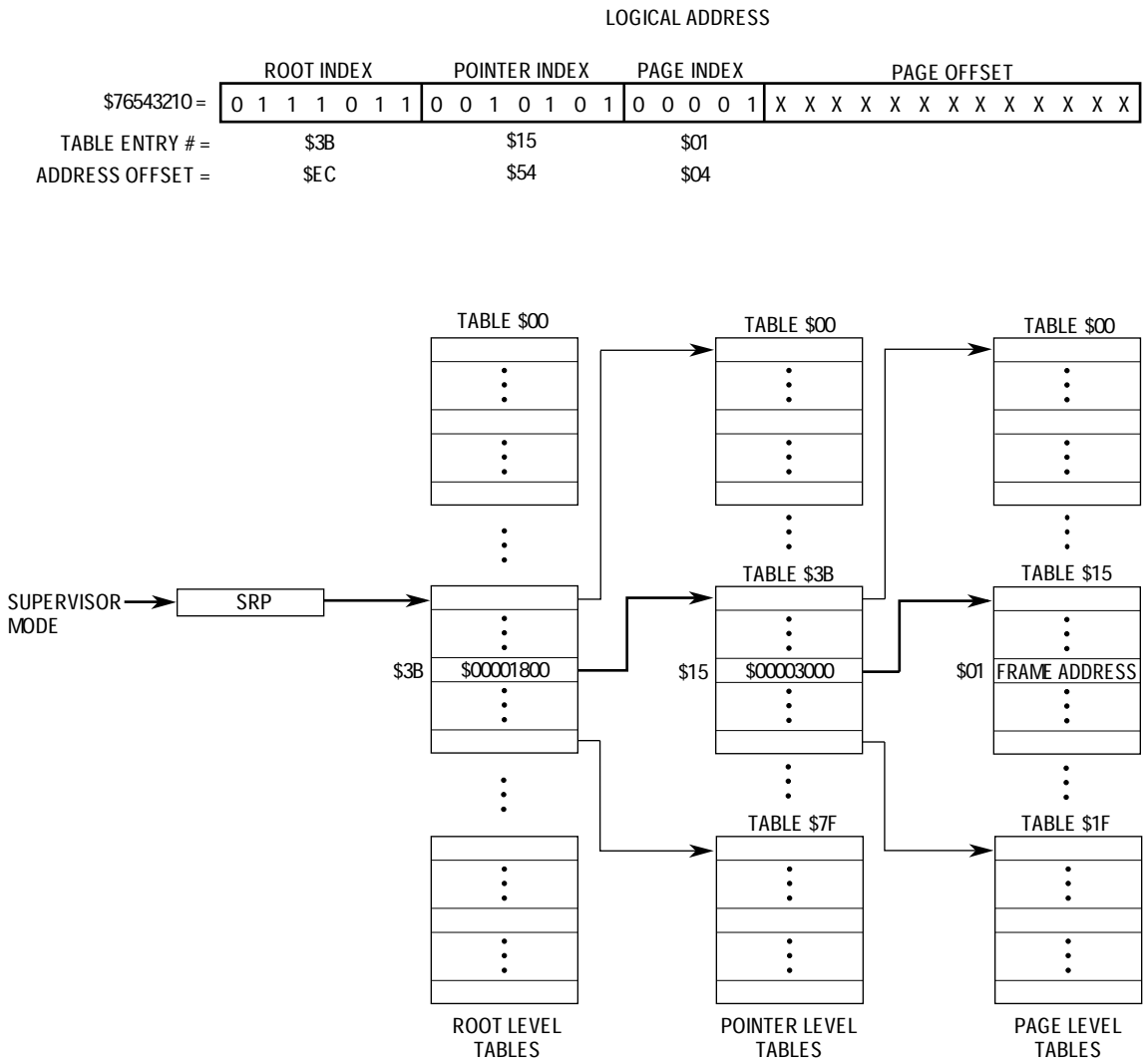


Figure 3-13. Example Translation Table

Using the indirection capability, single entries or entire tables can be shared between multiple tasks. Figure 3-14 illustrates two tasks sharing a page using indirect descriptors.

When the M68040 has completed a normal table search, it examines the PDT field of the last entry fetched from the page tables. If the PDT field contains an indirect ($2) encoding, it indicates that the address contained in the highest order 30 bits of the descriptor is a pointer to the page descriptor that is to be used to map the logical address. The processor then fetches the page descriptor from this address and uses the physical address field of the page descriptor as the physical mapping for the logical address.

The page descriptor located at the address given by the indirect descriptor must not have a PDT field with an indirect encoding (it must be either a resident descriptor or invalid). Otherwise, the descriptor is treated as invalid, and the M68040 creates an ATC entry with a signaled error condition (R-bit in MMUSR is clear).

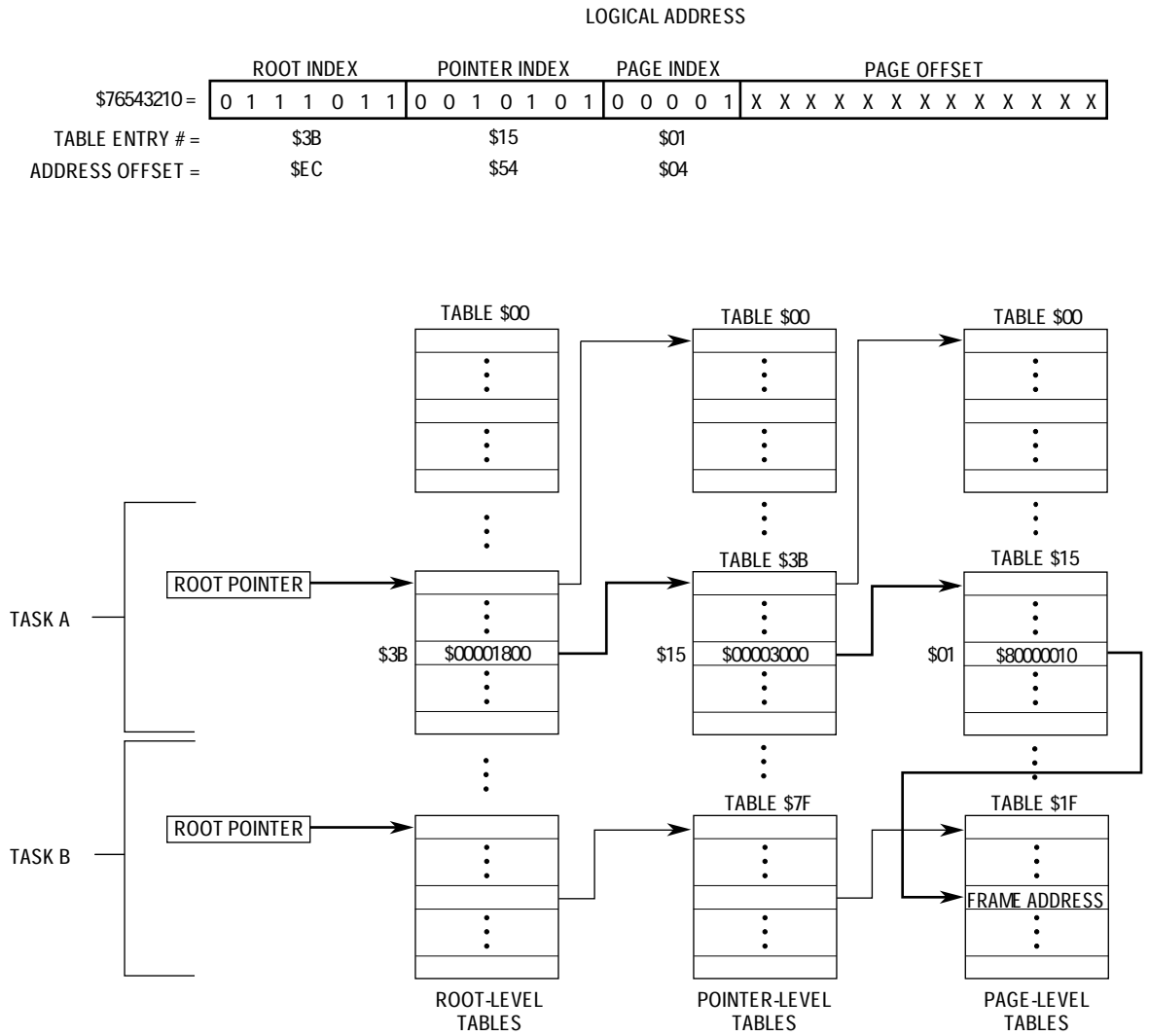
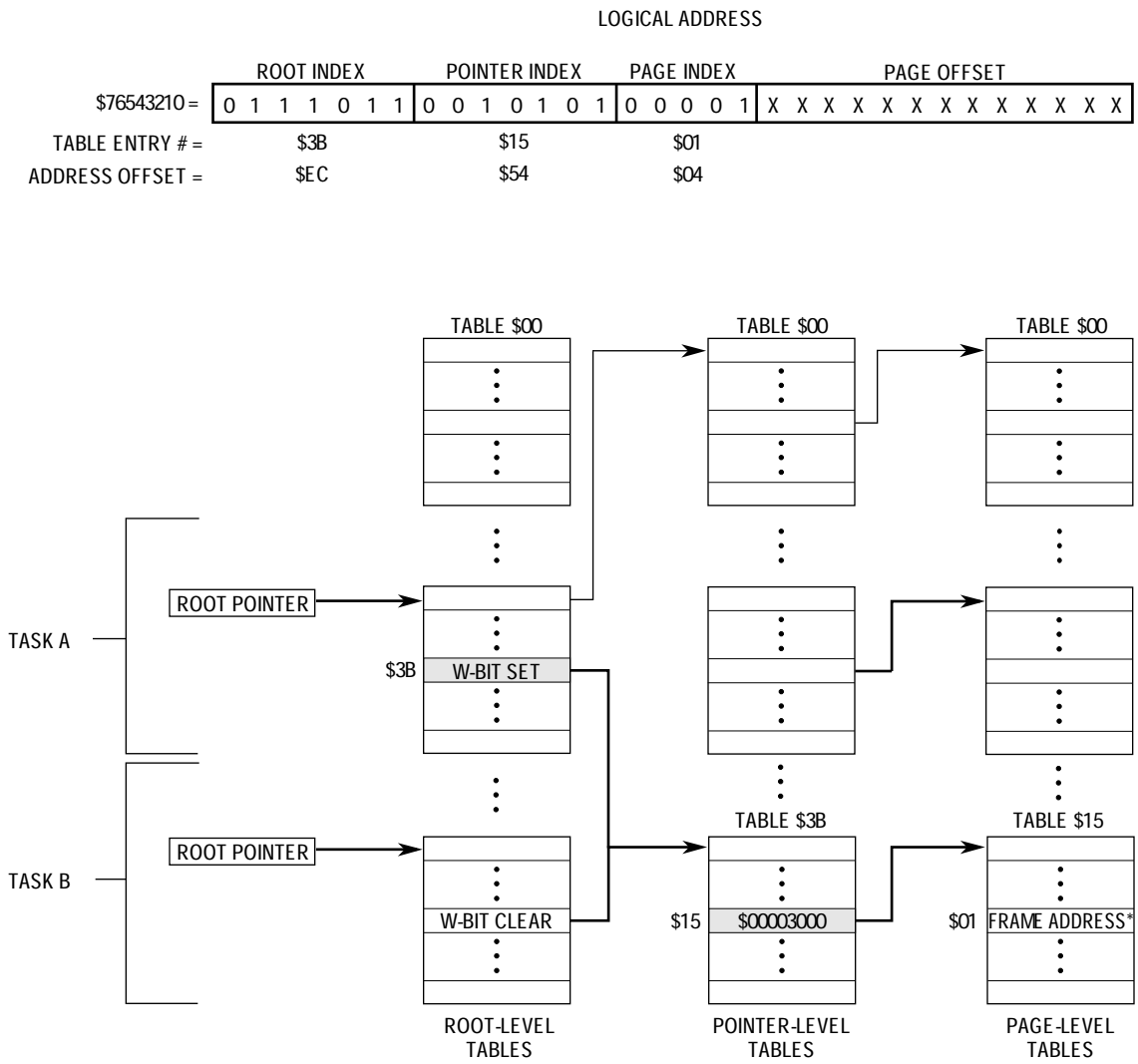


Figure 3-14. Translation Table Using Indirect Descriptors

3.2.4.2 TABLE SHARING BETWEEN TASKS. More than one task can share a pointer- or page-level table by placing a pointer to a shared table in the address translation tables. The upper (nonshared) tables can contain different write-protected settings, allowing different tasks to use the memory areas with different write permissions. In Figure 3-15, two tasks share the memory translated by the table at the pointer table level. Task A cannot write to the shared area; task B, however, has the W-bit clear in its pointer to the shared table so that it can read and write the shared area. Also, the shared area appears at different logical addresses for each task. Figure 3-15 illustrates shared tables in a translation table structure.



* Page frame address shared by task A and B; write protected from task A.

Figure 3-15. Translation Table Using Shared Tables

3.2.4.3 TABLE PAGING. The entire translation table for an active task need not be resident in main memory. In the same way that only the working set of pages must be allocated in main memory, only the tables that describe the resident set of pages need be available. Placing the invalid code ($0 or $1) in the UDT field of the table descriptor that points to the absent table(s) implements this paging of tables. When a task attempts to use an address that an absent table would translate, the M68040 is unable to locate a translation and takes access error exception when the execution unit retries the bus access that caused the table search to be initiated.

The operating system determines that the invalid code in the descriptor corresponds to nonresident tables. This determination can be facilitated by using he unused bits in the descriptor to store status information concerning the invalid encoding. The M68040 does not interpret or modify an invalid descriptor's fields except for the UDT field. This

interpretation allows the operating system to store system-defined information in the remaining bits. Information typically stored includes the reason for the invalid encoding (tables paged out, region unallocated, etc.) and possibly the disk address for nonresident tables. Figure 3-16 illustrates an address translation table in which only a single page table (table $15) is resident; all other page tables are not resident.

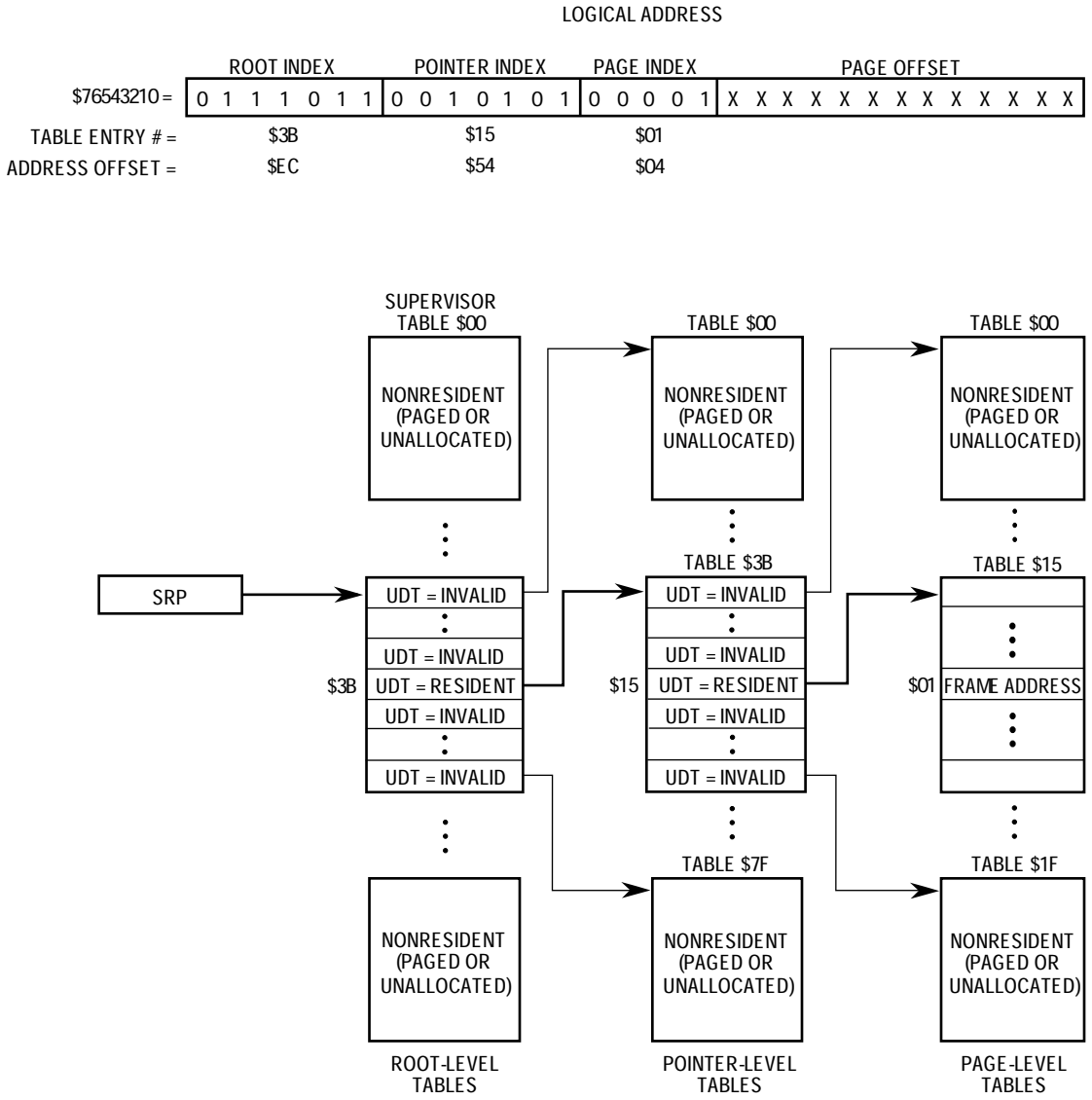


Figure 3-16. Translation Table with Nonresident Tables

3.2.4.4 DYNAMICALLY ALLOCATED TABLES. Similar to paged tables, a complete translation table need not exist for an active task. The operating system can dynamically allocate the translation table based on requests for access to particular areas.

As in demand paging, it is difficult, if not impossible, to predict the areas of memory that a task uses over any extended period. Instead of attempting to predict the requirements of the task, the operating system performs no action for a task until a demand is made requesting access to a previously unused area or an area that is no longer resident in memory. This technique can be used to efficiently create a translation table for a task.

For example, consider an operating system that is preparing the system to execute a previously unexecuted task that has no translation table. Rather than guessing what the memory-usage requirements of the task are, the operating system creates a translation table for the task that maps one page corresponding to the initial value of the program counter (PC) for that task and one page corresponding to the initial stack pointer of the task. All other branches of the translation table for this task remain unallocated until the task requests access to the areas mapped by these branches. This technique allows the operating system to construct a minimal translation table for each task, conserving physical memory utilization and minimizing operating system overhead.

3.2.5 Table Search Accesses

The cache treats table search accesses that are not read-modify-write accesses as cachable/write-through but do not allocate in the cache for misses. Read-modify-write table search accesses (required to update some descriptor U-bit and M-bit combinations) are treated as noncachable and force a matching cache line to be pushed and invalidated. Table search bus accesses are locked only for the specific portions of the table search that requires a read-modify-write access.

During a table search, the U-bit in each encountered descriptor is checked and set if not already set. Similarly, when the table search is for a write access and the M-bit of the page descriptor is clear, the processor sets the bit if the table search does not encounter a set W-bit or a supervisor violation. Repeating the descriptor access as part of a read-modify-write access updates specific combinations of the U and M bits, allowing the external arbiter to prevent the update operation from being interrupted.

The M68040 asserts the LOCK signal during certain portions of the table search to ensure proper maintenance of the U-bit and M-bit. The U-bit and M-bit are updated before the M68040 allows a page to be accessed or written. As descriptors are fetched, the U-bit and M-bit are monitored. Write cycles modify these bits when required. For a table descriptor, a write cycle that sets the U-bit occurs only if the U-bit was clear. Table 3-1 lists the page descriptor update operations for each combination of U-bit, M-bit, write-protected, and read or write access type.

Table 3-1. Updating U-Bit and M-Bit for Page Descriptors

Previous Status		WP Bit	Access Type	Page Descriptor	New Status	
U-Bit	M-Bit			Update Operation	U-Bit	M-Bit
0	0	X	Read	Locked RMW Access to Set U	1	0
0	1			Locked RMW Access to Set U	1	1
1	0			None	1	0
1	1			None	1	1
0	0	0	Write	Write to Set U and M	1	1
0	1			Locked RMW Access to Set U	1	1
1	0			Write to Set M	1	1
1	1			None	1	1
0	0	1		Locked RMW Access to Set U	1	0
0	1			Locked RMW Access to Set U	1	1
1	0			None	1	0
1	1			None	1	1

NOTE: WP indicates the accumulated write-protect status.

An alternate address space access is a special case that is immediately used as a physical address without translation. Because the M68040 implements a merged instruction and data space, the integer unit translates MOVES accesses to instruction address spaces (SFC/DFC = $6 or $2) into data references (SFC/DFC = $5 or $1). The data memory unit handles these translated accesses as normal data accesses. If the access fails due to an ATC fault or a physical bus error, the resulting access error stack frame contains the converted function code in the TM field for the faulted access. Invalidation of the instruction cache line containing the referenced location to maintain cache coherency must precede MOVES accesses that write the instruction address space. The SFC and DFC values and results are listed in Table 3-2.

Table 3-2. SFC and DFC Values

SFC/DFC Value	Results	
	TT	TM
000	10	000
001	00	001
010	00	001
011	10	011
100	10	100
101	00	101
110	00	101
111	10	111

3.2.6 Address Translation Protection

The M68040 MMUs provide separate translation tables for supervisor and user address spaces. The translation tables contain both mapping and protection information. Each table and page descriptor includes a write-protect (W) bit that can be set to provide write protection at any level. Page descriptors also contain a supervisor-only (S) bit that can limit access to programs operating at the supervisor privilege level.

The protection mechanisms can be used individually or in any combination to protect:

- Supervisor address space from accesses by user programs.
- User address space from accesses by other user programs.
- Supervisor and user program spaces from write accesses (implicitly supported by designating all memory pages used for program storage as write protected).
- One or more pages of memory from write accesses.

3.2.6.1 SUPERVISOR AND USER TRANSLATION TABLES. One way of protecting supervisor and user address spaces from unauthorized accesses is to use separate supervisor and user translation tables. Separate trees protect supervisor programs and data from accesses by user programs and user programs and data from access by supervisor programs. Access is granted to the supervisor programs that can accesses any area of memory with MOVES. The translation table pointed to by the SRP is selected for all other supervisor mode accesses. This translation table can be common to all tasks. Figure 3-17 illustrates separate translation tables for supervisor accesses and for two user tasks that share the common supervisor space. Each user task has an translation table with unique mappings for the logical addresses in its user address space.

3.2.6.2 SUPERVISOR ONLY. A second mechanism protects supervisor programs and data without requiring segmenting of the logical address space into supervisor and user address spaces. Page descriptors contain S-bits to protect areas of memory from access by user programs. When a table search for a user access encounters an S-bit set in a page descriptor, the table search ends, and an ATC descriptor corresponding to the logical address is created with the S-bit set. A subsequent retry of the user access results in an access error exception being taken. The S-bit can be used to protect one or more pages from user program access. Supervisor and user mode accesses can share descriptors by using indirect descriptors or by sharing tables. The entire user and supervisor address spaces can be mapped together by loading the same root pointer address into both the SRP and URP registers.

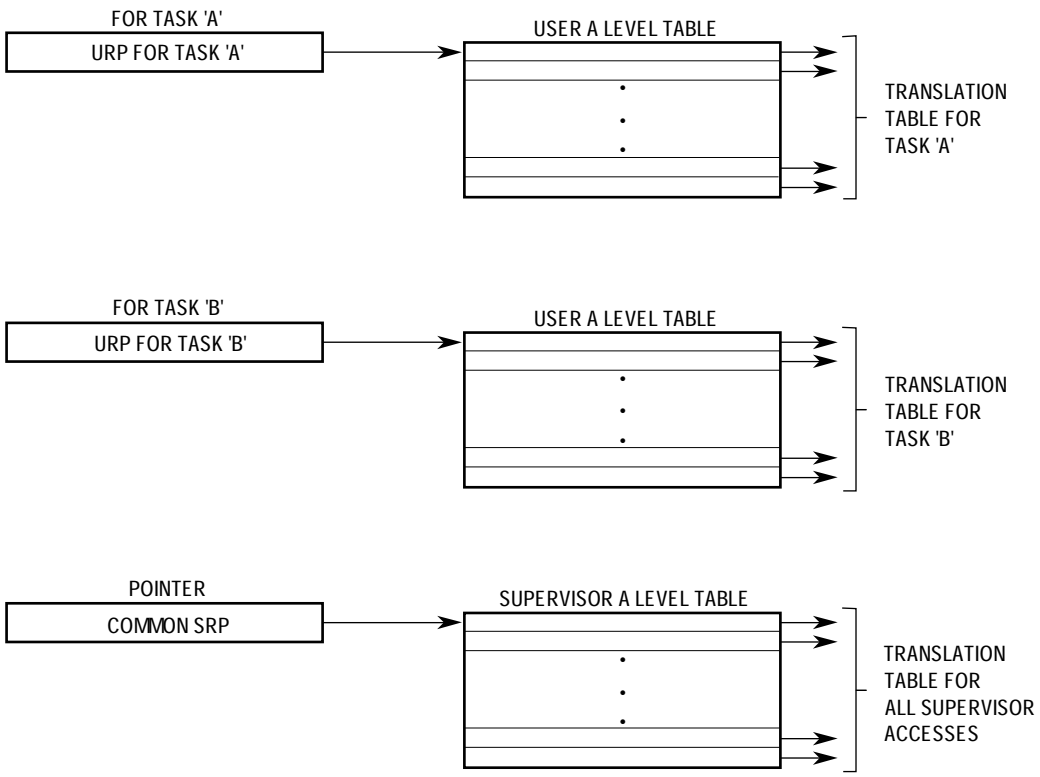


Figure 3-17. Translation Table Structure for Two Tasks

3.2.6.3 WRITE PROTECT. The M68040 provides write protection independent of other protection mechanisms. All table and page descriptors contain W-bits to protect areas of memory from write accesses of any kind, including supervisor writes. An ATC descriptor corresponding to the logical address is created with the W-bit set after the table search is completed when a table search encounters a W-bit set in any table or page descriptor. The subsequent retry of the write access results in an access error exception being taken. The W-bit can be used to protect the entire area of memory defined by a branch of the translation table or protect only one or more pages from write accesses. Figure 3-18 illustrates a memory map of the logical address space organized to use supervisor-only and write-protect bits for protection. Figure 3-19 illustrates an example translation table for this technique.

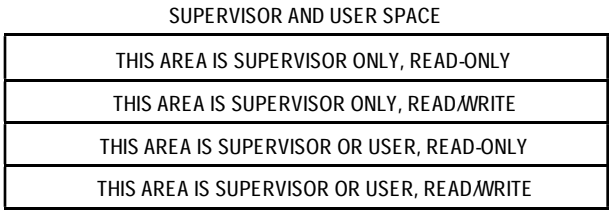
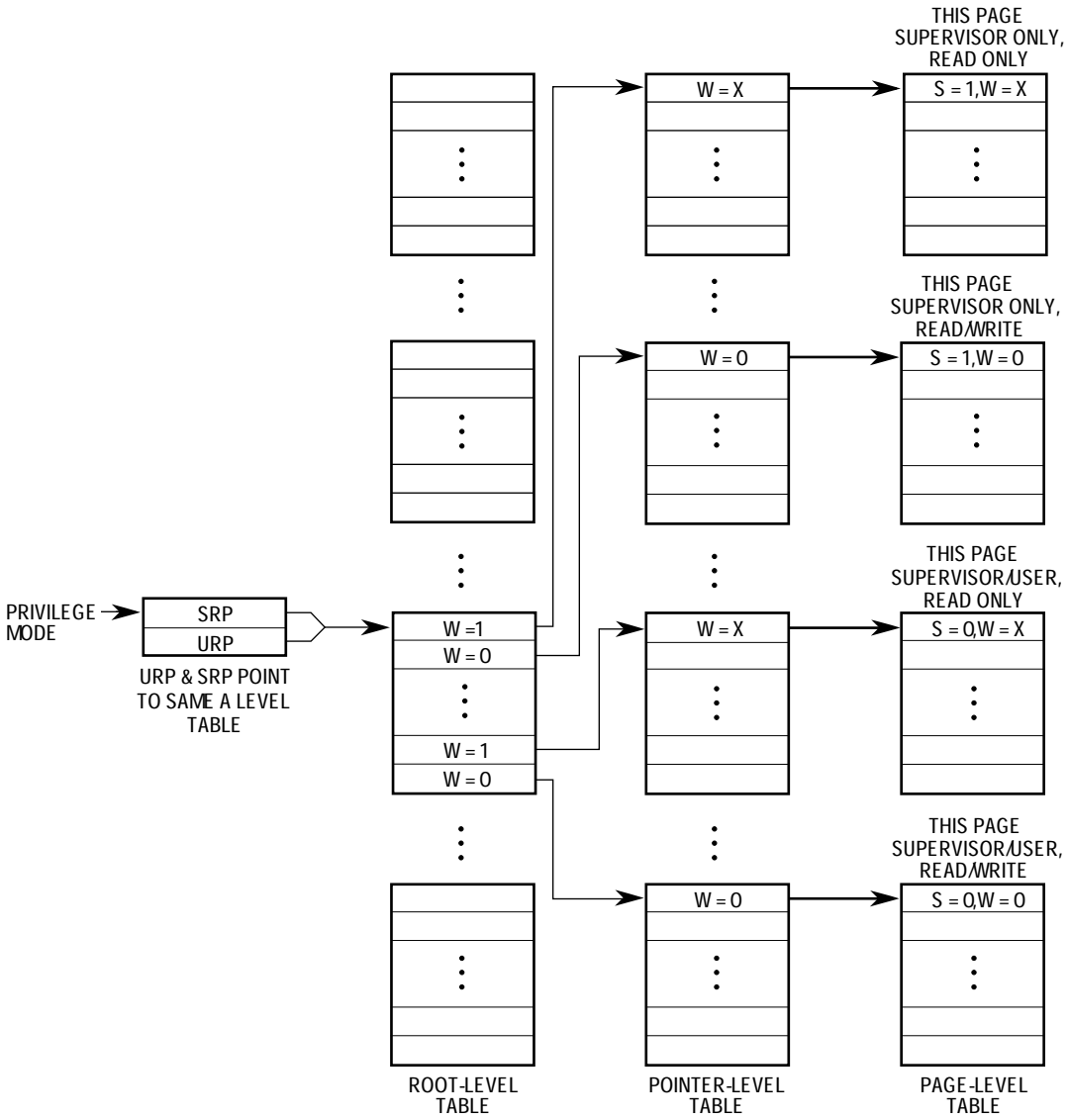


Figure 3-18. Logical Address Map with Shared Supervisor and User Address Spaces



NOTE: X = Don't care.

Figure 3-19. Translation Table Using S-Bit and W-Bit To Set Protection

3.3 ADDRESS TRANSLATION CACHES

The ATCs in the MMUs are four-way set-associative caches that each store 64 logical-to-physical address translations and associated page information similar in form to the corresponding page descriptors in memory. The purpose of the ATC is to provide a fast mechanism for address translation by avoiding the overhead associated with a table search of the logical-to-physical mapping of recently used logical addresses. Figure 3-20 illustrates the organization of the ATC.

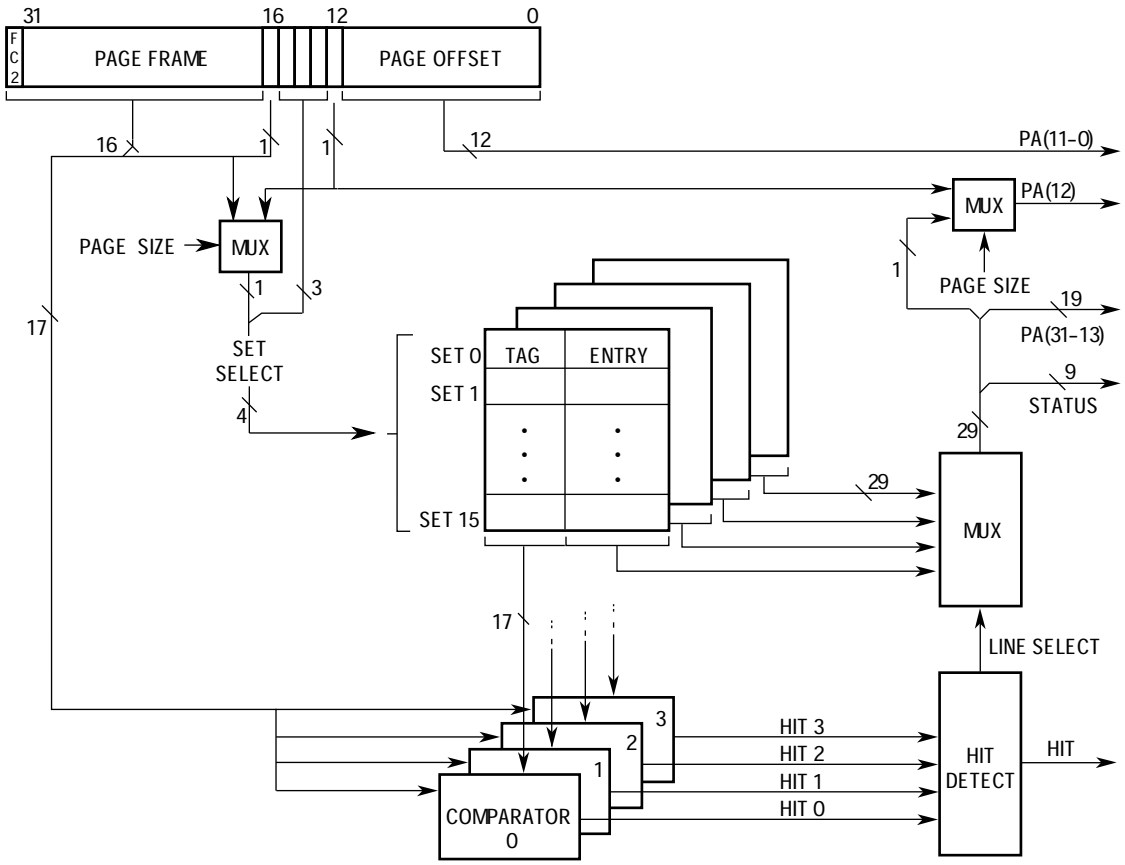


Figure 3-20. ATC Organization

Each ATC entry consists of a physical address, attribute information from a corresponding page descriptor, and a tag that contains a logical address and status information. Figure 3-21, which illustrates the entry and tag fields, is followed by field definitions listed in alphabetical order.

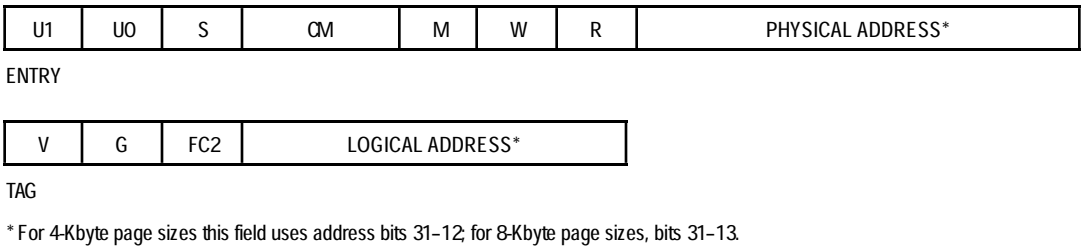


Figure 3-21. ATC Entry and Tag Fields

CM—Cache Mode

This field selects the cache mode and accesses serialization as follows:

00 = Cachable, Write-through
01 = Cachable, Copyback
10 = Noncachable, Serialized
11 = Noncachable

Section 4 Instruction and Data Caches provides detailed information on caching modes, and **Section 7 Bus Operation** provides information on serialization.

FC2—Function Code Bit 2 (Supervisor/User)

This bit contains the function code corresponding to the logical address in this entry. FC2 is set for supervisor mode accesses and cleared for user mode accesses.

G—Global

When set, this bit indicates the entry is global. Global entries are not invalidated by the PFLUSH instruction variants that specify nonglobal entries, even when all other selection criteria are satisfied.

Logical Address

This 13-bit field contains the most significant logical address bits for this entry. All 16 bits of this field are used in the comparison of this entry to an incoming logical address when the page size is 4 Kbytes. For 8-Kbytes pages, the least significant bit of this field is ignored.

M—Modified

The modified bit is set when a valid write access to the logical address corresponding to the entry occurs. If the M-bit is clear and a write access to this logical address is attempted, the M68040 suspends the access, initiates a table search to set the M-bit in the page descriptor, and writes over the old ATC entry with the current page descriptor information. The MMU then allows the original write access to be performed. This

procedure ensures that the first write operation to a page sets the M-bit in both the ATC and the page descriptor in the translation tables, even when a previous read operation to the page had created an entry for that page in the ATC with the M-bit clear.

Physical Address

The upper bits of the translated physical address are contained in this field.

R—Resident

This bit is set if the table search successfully completes without encountering either a nonresident page or a transfer error acknowledge during the search.

S—Supervisor Protected

This bit identifies a pointer table or a page as a supervisor-only table or page. Only programs operating in the supervisor privilege mode are allowed to access the portion of the logical address space mapped by this descriptor when the S-bit is set. If the bit is clear, both supervisor and user accesses are allowed.

U0, U1—User Page Attributes

These user-defined bits are not interpreted by the M68040. U0 and U1 are echoed to the UPA0 and UPA1 signals, respectively, if an external bus transfer results from the access.

V—Valid

When set, this bit indicates the validity of the entry. This bit is set when the M68040 loads an entry. A flush operation by a PFLUSH or PFLUSHA instruction that selects this entry clears the bit.

W—Write Protected

This write-protect bit is set when a W-bit is set in any of the descriptors encountered during the table search for this entry. Setting a W-bit in a table descriptor write protects all pages accessed with that descriptor. When the W-bit is set, a write access or a read-modify-write access to the logical address corresponding to this entry causes an access error exception to be taken immediately.

For each access to a memory unit, the MMU uses the four bits of the logical address located just above the page offset (LA16–LA13 for 8K pages, LA15–LA12 for 4K pages) to index into the ATC. The tags are compared with the remaining upper bits of the logical address and FC2. If one of the tags matches and is valid, then the multiplexer choses the corresponding entry to produce the physical address and status information. The ATC outputs the corresponding physical address to the cache controller, which accesses the data within the cache and/or requests an external bus cycle. Each ATC entry contains a logical address, a physical address, and status bits.

When the ATC does not contain the translation for a logical address, a miss occurs. The MMU aborts the current access and searches the translation tables in memory for the correct translation. If the table search completes without any errors, the MMU stores the

translation in the ATC and provides the physical address for the access, allowing the memory unit to retry the original access.

There are some variations in the logical-to-physical mapping because of the two page sizes. If the page size is 4 Kbytes, then logical address bit 12 is used to access the ATC's memory, the tag comparators use bit 16, and physical address bit 12 is an ATC output. If the page size is 8 Kbytes, then logical address bit 16 is used to access the ATC's memory, and physical address bit 12 is driven by logical address bit 12. It is advisable that a translation always be disabled before changing size and that the ATCs are flushed before enabling translation again.

The M68040 is organized such that other operations always completely overlap the translation time of the ATCs; thus, no performance penalty is associated with ATC searches. The address translation occurs in parallel with indexing into the on-chip instruction and data caches.

The MMU replaces an invalid entry when the ATC stores a new address translation. When all entries in an ATC set are valid, the ATC selects a valid entry to be replaced, using a pseudo-random replacement algorithm. A 2-bit counter, which is incremented for each ATC access, points to the entry to replace when an access misses in the ATC. ATC hit rates are application and page-size dependent, but hit rates ranging from 98% to greater than 99% can be expected. These high rates are achieved because the ATCs are relatively large (64 entries) and utilization efficiency is high with 8-Kbyte and 4-Kbyte page sizes.

3.4 TRANSPARENT TRANSLATION

Four independent TTRs (DTT0 and DTT1 in the data MMU, ITT0 and ITT1 in the instruction MMU) define four blocks of logical address space to be translated to physical address space. These logical address spaces must be at least 16 Mbytes and can overlap or be separate. Each TTR can be disabled and completely ignored. The following description assumes that the TTRs are enabled.

When an MMU receives an address to be translated, the privilege mode and the eight high-order bits of the address are compared to the logical address spaces defined by the two TTRs for the corresponding MMU. The logical address space for each TTR is defined by an S-field, logical base address field, and logical address mask field. The S-field allows matching either user or supervisor accesses or both accesses. When a bit in the logical address mask field is set, the corresponding bit of the logical 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 physical address space.

The address for the current bus cycle and a TTR address match when the privilege mode and logical base address bits are equal. Each TTR 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.

By appropriately configuring a TTR, flexible transparent mappings can be specified (refer to **3.1.3 Transparent Translation Registers** for field identification). For instance, to transparently translate the user address space, the S-field is set to $0, and the logical address mask is set to $FF in both an instruction and data TTR. To transparently translate supervisor accesses of addresses $00000000–$0FFFFFFF with write protection, the logical base address field is set to $0x, the logical 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 TTRs in both the instruction and data MMUs provides an exception to the merged instruction and data address space, allowing different translations for instruction and operand accesses. Also, since the instruction memory unit is only used for instruction prefetches, different instruction and data TTRs can cause PC relative operand fetches to be translated differently from instruction prefetches.

If either of the TTRs matched during an access to a memory unit (either instruction or data), the access is transparently translated. If both registers match, the TT0 status bits are used for the access. Transparent translation can also be implemented by the translation tables of the translation tables if the physical addresses of pages are set equal to their logical addresses.

3.5 ADDRESS TRANSLATION SUMMARY

The instruction and data MMUs process translations by first comparing the logical address and privilege mode with the parameters of the TTRs. If there is a match, the MMU uses the logical address as a physical address for the access. If there is no match, the MMU compares the logical address and privilege mode with the tag portions of the entries in the ATC and uses the corresponding physical address for the access when a match occurs. When neither a TTR nor a valid ATC entry matches, the MMU initiates a table search operation to obtain the corresponding physical address from the translation table. When a table search is required, the processor suspends instruction execution activity and, at the end of a successful table search, stores the address mapping in the appropriate ATC and retries the access. The MMU creates a valid ATC entry for the logical address, and the access is retried. If an access hits in the ATC but an access error or invalid page descriptor was detected during the table search that created the ATC entry, the access is aborted, and a bus error exception is taken.

If a write or read-modify-write access results in an ATC hit but the page is write protected, the access is aborted, and an access error exception is taken. If the page is not write protected and the modified bit of the ATC entry is clear, a table search proceeds to set the modified bit in both the page descriptor in memory and in the ATC; the access is retried. The ATC provides the address translation for the access if the modified bit of the ATC entry is set for a write or read-modify-write access to an unprotected page, if the resident bit is set (indicating the table search for the entry completed successfully), and if none of the TTRs (instruction or data, as appropriate) match.

An ATC access error is not reported immediately, if the last 16 bits of a page is either an A-line, illegal, CHK, or unimplemented instruction and the next page is non-resident. Instead, the M68040 attempts to prefetch the next instruction on the missing page, then the ATC access error exception is reported. The stacked PC points to the exceptional

instruction, and the stacked FA points to the first longword in the missing page. When an ATC access error occurs while prefetching the next instruction on the non-existant page after a change of flow instruction, the exception should be cleared by execution of the new instruction flow. Either avoid this scenario, or have a dummy resident page following the exceptional instruction.

Figure 3-22 illustrates a general flowchart for address translation. The top branch of the flowchart applies to transparent translation. The bottom three branches apply to ATC translation.

3.6 MMU EFFECT ON RSTI AND MDIS

The following paragraphs describe MMU effects on the RSTI and MDIS pins.

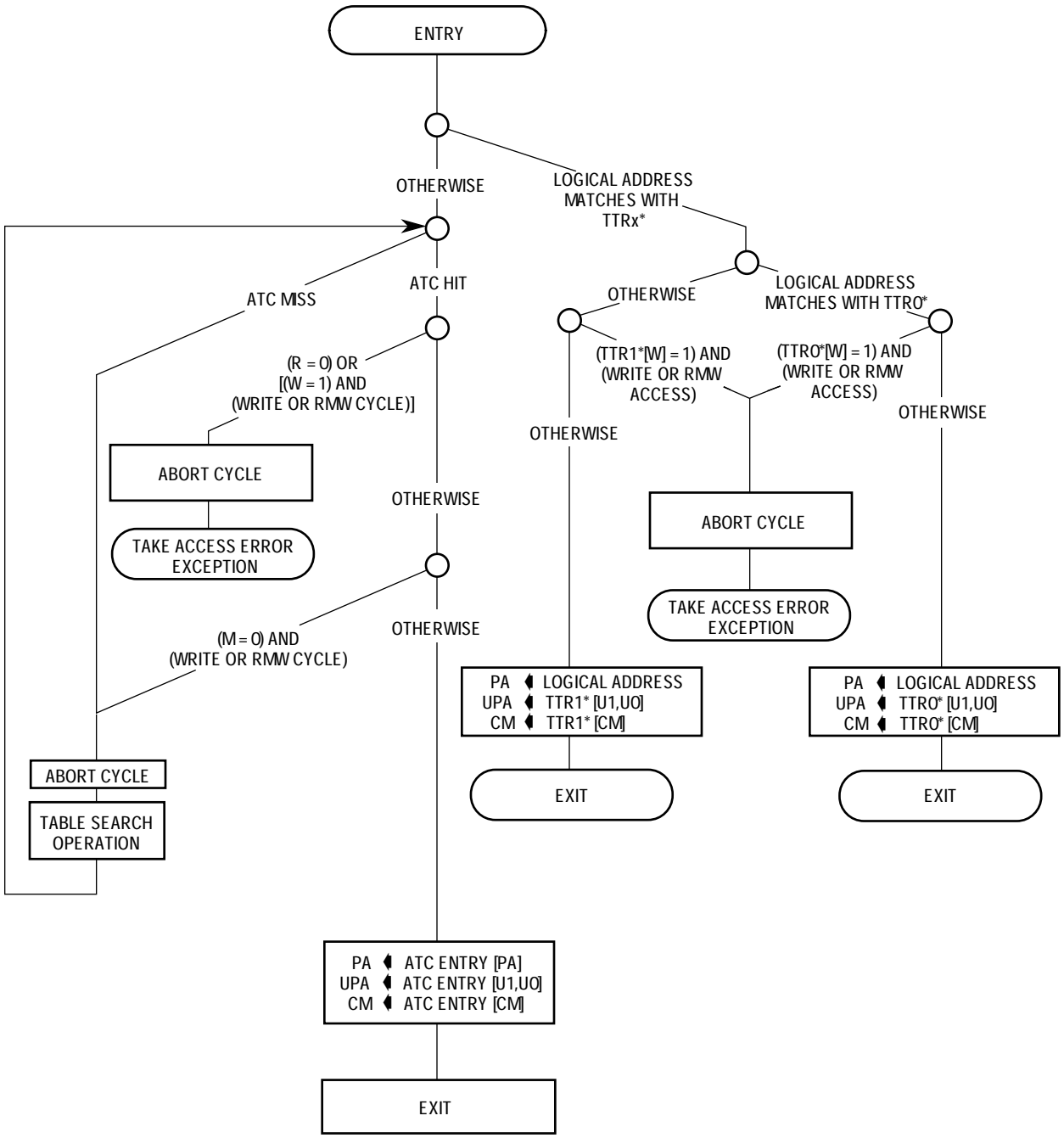
3.6.1 Effect of RSTI on the MMUs

When the M68040 is reset by the assertion of the reset input signal, the E-bits of the TCR and TTRs are cleared, disabling address translation. This reset causes logical addresses to be passed through as physical addresses, allowing an operating system to set up the translation tables and MMU registers as required. After the translation tables and registers are initialized, the E-bit of the TCR can be set, enabling paged address translation. While address translation is disabled, the attribute bits for an access that an ATC entry or a TTR normally supplies are zero, selecting write-through cachable mode, no write protection, and user page attribute bits cleared. RSTI does not affect the P-bit of the TCR.

A reset of the processor does not invalidate any entries in the ATCs or alter the page size. A PFLUSH instruction must be executed to flush all existing valid entries from the ATCs after a reset operation and before translation is enabled. PFLUSH can be executed even if the E-bit is cleared.

3.6.2 Effect of MDIS on Address Translation

The assertion of MDIS prevents the MMUs from performing ATC searches and the execution unit from performing table searches. With address translation disabled, logical addresses are used as physical addresses. MDIS disables the MMUs on the next internal access boundary when asserted and enables the MMUs on the next boundary after the signal is negated. The assertion of this signal does not affect the operation of the transparent translation registers or execution of the PFLUSH or PTEST instructions.



* Refers to either instruction or data transparent translation register.

Figure 3-22. Address Translation Flowchart

3.7 MMU INSTRUCTIONS

The M68040 instruction set includes three privileged instructions that perform MMU operations. The following paragraphs briefly describe each of these instructions. For detailed descriptions of these instructions, refer to M68000PR/AD, *M68000 Family Programmer's Reference Manual*.

3.7.1 MOVEC

The MOVEC instruction transfers data between an integer data register, or memory location, and any of the M68040 control and status registers. The operating system uses the MOVEC instruction to control and monitor MMU operation by manipulating and reading the eight MMU registers.

3.7.2 PFLUSH

The PFLUSH instruction flushes or invalidates address translation descriptors in the ATCs. PFLUSHA, a version of the PFLUSH instruction, flushes all entries. The PFLUSH instruction flushes a user or supervisor entry with a specified logical address. The PFLUSHAN and PFLUSHN instruction variants qualify entry selection further by flushing only entries that are nonglobal, indicated by a cleared G-bit in the entry.

3.7.3 PTEST

The PTEST instruction performs a table search operation for a specified function code and logical address and sets the appropriate bit fields in the MMUSR to indicate conditions encountered during the search. PTEST automatically flushes the corresponding entry from the cache before searching the tables and loads the latest information from the translation tables into the ATC. The exception routines of the operating system can use this instruction to identify MMU faults.

PTEST is primarily used in access error exception handlers. For example, if a bus error has occurred, the handler can execute an instruction sequence such as the following sequence:

```
MOVE.B (A7,offset1),D0      Copy transfer modifier field from stack frame
MOVEC D0,DFC                into DFC register
MOVEA.L (A7,offset2),A0     Copy fault address from stack frame into address register
PTESTW (A0)                 Test address in A0 with function code in DFC registers
```

The transfer modifier field copied into the destination function code (DFC) register indicates whether the faulted access was a supervisor or user mode access and whether it was an instruction prefetch or data access. The PTEST instruction uses the DFC value to determine which translation table (supervisor or user) to search and which ATC (data or instruction) to create the entry in. After executing this code sequence, the handler can examine the MMUSR for the source of the fault.

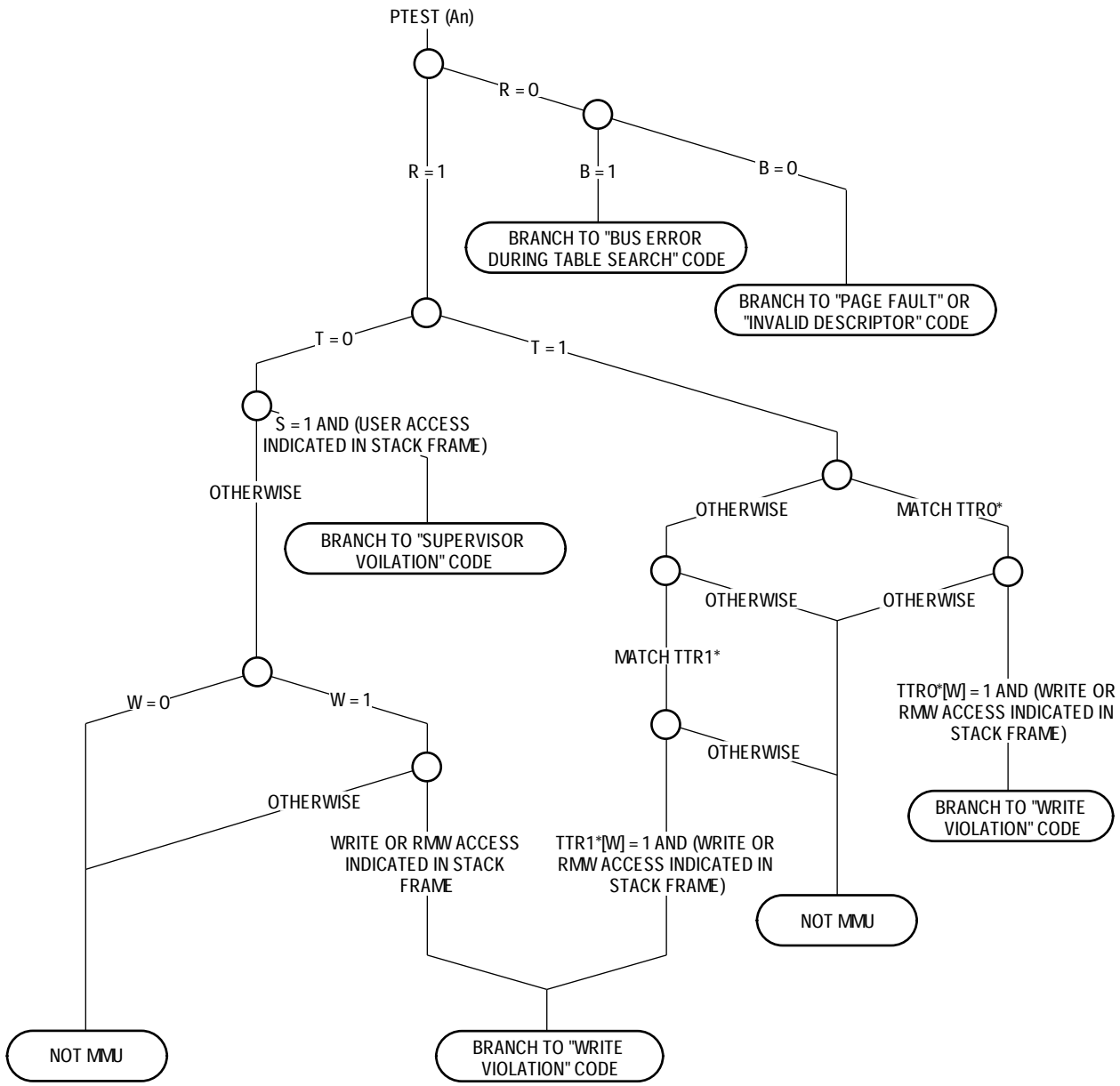
The M68040 MMU instructions use opcodes that are different from those for the corresponding instructions in the MC68030 and MC68851. All MMU opcodes for the

MC68030 and MC68851 cause F-line unimplemented instruction exceptions if executed in either supervisor or user mode by the M68040.

3.7.4 Register Programming Considerations

If the entries in the ATCs are no longer valid when a reset operation occurs (as is normally expected), an explicit flush operation must be specified by the system software. The assertion of `RSTI` disables translation by clearing the E-bits of the TCR, DTTRx, and ITTRx, but it does not flush the ATCs. Reading or writing any of the MMU registers (URP, SRP, TCR, MMUSR, DTTR0, DTTR1, ITTR0, ITTR1) does not flush the ATCs. Since a write to these registers can cause some or all the address translations to change, the write should be followed by a PFLUSH operation to flush the ATCs if necessary.

The status bits in the MMUSR indicate conditions to which the operating system should respond. In a typical access error exception handler, the flowchart illustrated in Figure 3-23 can be used to determine the cause of an MMU fault. The PTEST instruction sets the bits in the MMUSR appropriately, and the program can branch to the appropriate code segment for the condition.



* Refers to either instruction or data transparent translation register.

Figure 3-23. MMU Status Interpretation

SECTION 4
INSTRUCTION AND DATA CACHES

NOTE

Ignore all references to the memory management unit (MMU) when reading for the MC68EC040 and MC68EC040V. The functionality of the MC68040 transparent translation registers has been changed in the MC68EC040 and MC68EC040V to the access control registers. Refer to **Appendix B MC68EC040** for details.

The M68040 contains two independent, 4-Kbyte, on-chip caches located in the physical address space. Accessing instruction words and data simultaneously through separate caches increases instruction throughput. The M68040 caches improve system performance by providing cached data to the on-chip execution unit with very low latency. Systems with an alternate bus master receive increased bus availability.

Figure 4-1 illustrates the instruction and data caches contained in the instruction and data memory units. The appropriate memory unit independently services instruction prefetch and data requests from the integer unit (IU). The memory units translate the logical address in parallel with indexing into the cache. If the translated address matches one of the cache entries, the access hits in the cache. For a read operation, the memory unit supplies the data to the IU, and for a write operation, the memory unit updates the cache. If the access does not match one of the cache entries (misses in the cache) or a write access must be written through to memory, the memory unit sends an external bus request to the bus controller. The bus controller then reads or writes the required data.

Cache coherency in the M68040 is optimized for multimaster applications in which the M68040 is the caching master sharing memory with one or more noncaching masters (such as DMA controllers). The M68040 implements a bus snooper that maintains cache coherency by monitoring an alternate bus master's access and performing cache maintenance operations as requested by the alternate bus master. Matching cache entries can be invalidated during the alternate bus master's access to memory, or memory can be inhibited to allow the M68040 to respond to the access as a slave. For an external write operation, the processor can intervene in the access and update its internal caches (sink data). For an external read operation, the processor supplies cached data to the alternate bus muster (source data). This prevents the M68040 caches from accumulating old or invalid copies of data (stale data). Alternate bus masters are allowed access to locally modified data within the caches that is no longer consistent with external memory (dirty data). Allowing memory pages to be specified as write-through instead of copyback also supports cache coherency. When a processor writes to write-through pages, external

memory is always updated through an external bus access after updating the cache, keeping memory and cached data consistent.

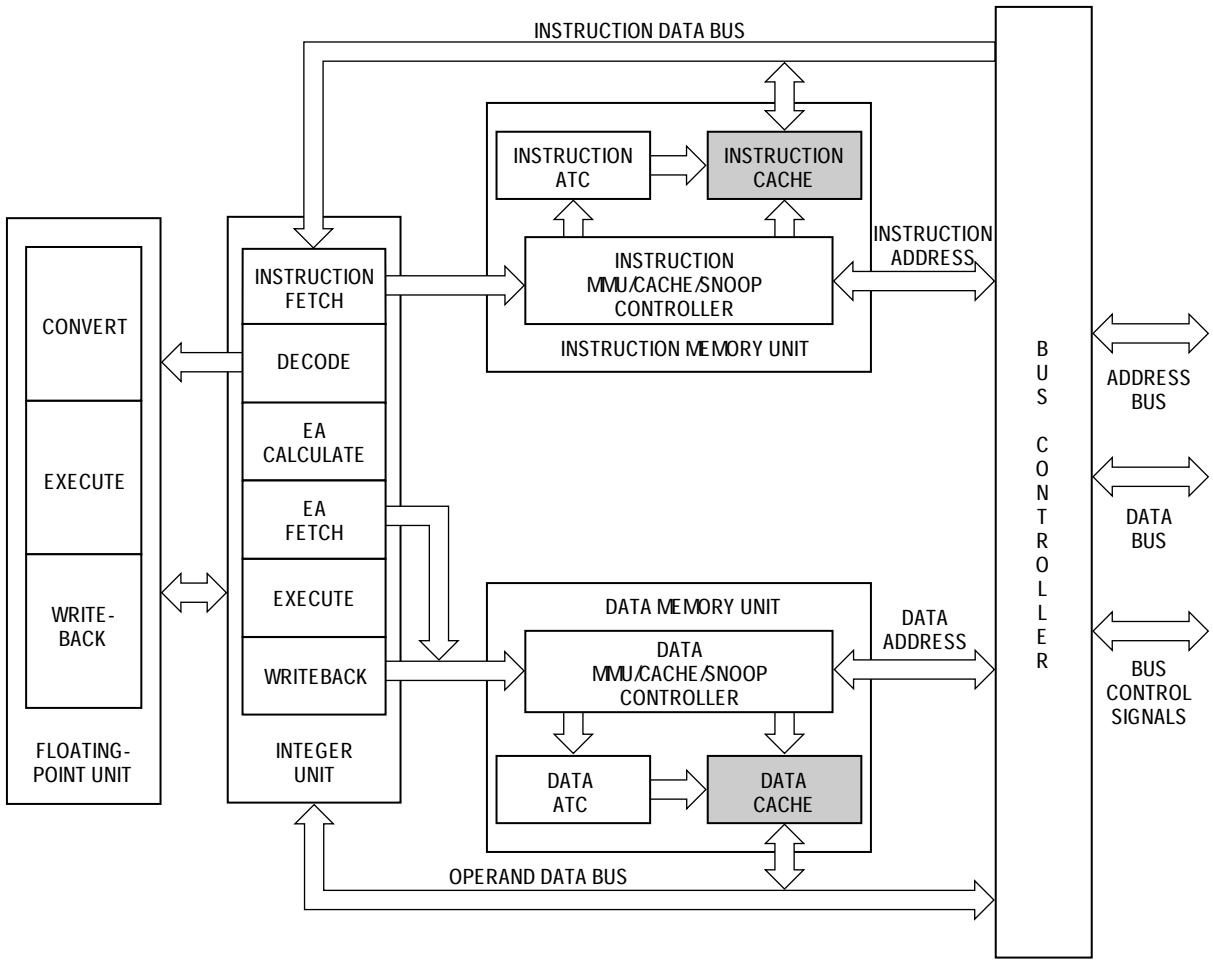


Figure 4-1. Overview of Internal Caches

4.1 CACHE OPERATION

Both four-way set-associative caches have 64 sets of four 16-byte lines. There are two formats that define each cache line, an instruction cache line format and a data cache line format. Each format contains an address tag consisting of the upper 22 bits of the physical address, status information, and four long words (128 bits) of data. The status information for the instruction cache line address tag consists of a single valid bit for the entire line. The status information for the data cache line address tag contains a valid bit and four additional bits to indicate dirty status for each long word in the line. Note that only the data cache supports dirty cache lines. Figure 4-2 illustrates the instruction cache line format (a) and the data cache line format (b).

TAG	V	LW3	LW2	LW1	LW0

(a) Instruction Cache Line

TAG	V	LW3	D3	LW2	D2	LW1	D1	LW0	D0

TAG — 22-Bit Physical Address Tag
V — Line VALID Bit
LW — Long Word n (32-Bit) Data Entry
Dn — DIRTY Bit for Long Word n

(b) Data Cache Line

Figure 4-2. Cache Line Formats

The cache stores an entire line, providing validity on a line-by-line basis. Only burst mode accesses that successfully read four long words can be cached. Memory devices unable to support bursting can respond to a cache line read or write access by asserting the transfer burst inhibit (TBI) signal, forcing the processor to complete the access as a sequence of three long-word accesses. The cache recognizes burst accesses as if the access were never inhibited, detecting no difference.

A cache line is always in one of three states: invalid, valid, or dirty. For invalid lines, the V-bit is clear, causing the cache line to be ignored during lookups. Valid lines have their V-bit set and D-bits cleared, indicating all four long words in the line contain valid data consistent with memory. Dirty cache lines have the V-bit and one or more D-bits set, indicating that the line has valid long-word entries that have not been written to memory (long words whose D-bit is set). A cache line changes from valid to invalid if the execution of the CINV or CPUSH instruction explicitly invalidates the cache line; if a snooped write access hits the cache line and the line is not dirty; or if the SCx signals for a snooped read access invalidates the line. Both caches should be explicitly cleared after a hardware reset of the processor since reset does not invalidate the cache lines.

Figure 4-3 illustrates the general flow of a caching operation. The corresponding memory unit translates the logical address of each access to a physical address allowing the IU to access the data in the cache. To minimize latency of the requested data, the lower untranslated bits of the logical address map directly to the physical address bits and are used to access a set of cache lines in parallel with the translation. Physical address bits 9–4 are used to index into the cache and select one of the 64 sets of four cache lines. The four tags from the selected cache set are compared with the translated physical address bits 31–12 and bits 11 and 10 of the untranslated page offset. If any one of the four tags matches and the tag status is either valid or dirty, then the cache has a hit. During read accesses, a half-line (two long words) is accessed at a time, requiring two cache accesses for reads that are greater than a half-line or two long words. Write accesses within a cache line require a single cache access. If a misaligned access crosses two pages, then the partial access to the first page always happens twice, even if the pages are serialized. Consequently, if the accesses span page boundaries, misaligned accesses to peripherals are not possible unless the peripheral can tolerate double reads or writes.

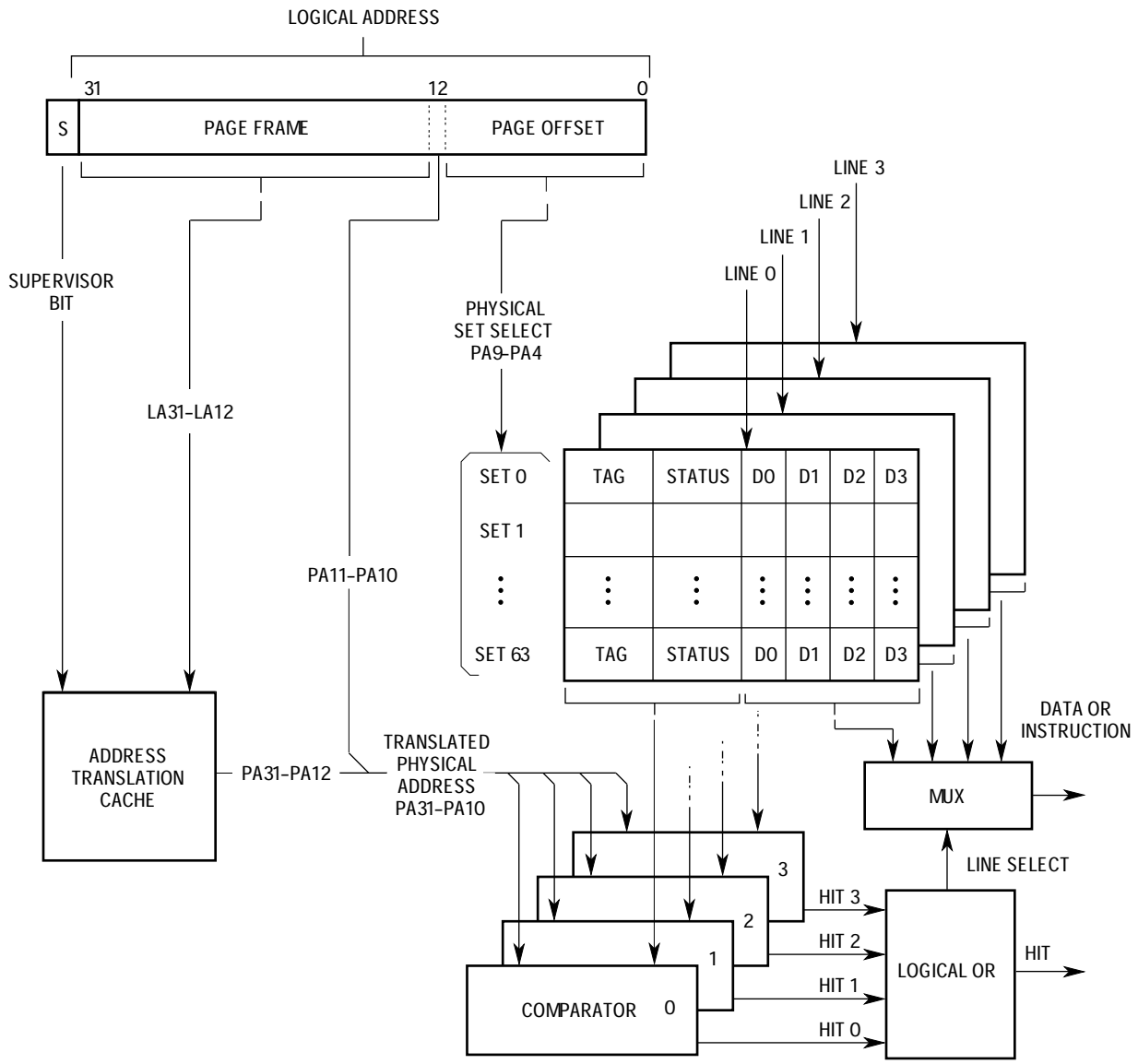
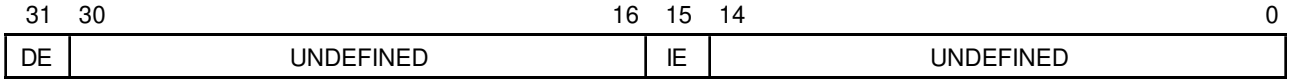


Figure 4-3. Caching Operation

Both caches contain circuitry to automatically determine which cache line in a set to use for a new line. The cache controller locates the first invalid line and uses it; if no invalid lines exist, then a pseudo-random replacement algorithm is used to select a valid line, replacing it with the new line. Each cache contains a 2-bit counter, which is incremented for each access to the cache. The instruction cache counter is incremented for each half-line accessed in the instruction cache. The data cache counter is incremented for each half-line accessed during reads, for each full line accessed during writes in copyback mode, and for each bus transfer resulting from a write in write-through mode. When a miss occurs and all four lines in the set are valid, the line pointed to by the current counter value is replaced, after which the counter is incremented.

4.2 CACHE MANAGEMENT

Using the MOVEC instruction, the caches are individually enabled to access the 32-bit cache control register (CACR) illustrated in Figure 4-4. The CACR contains two enable bits that allow the instruction and data caches to be independently enabled or disabled. Setting one of these bits enables the associated cache without affecting the state of any lines within the cache. A hardware reset clears the CACR, disabling both caches; however, reset does not affect the tags, state information, and data within the caches. The CINV instruction must clear the caches before enabling them. It is not recommended that page descriptors be cached. Specifically, the M68040 does not support the caching of page descriptors in copyback mode with the bit pattern U = 0, M = 1, and R = 1 in a page descriptor. The M68040 table search algorithm will never leave this bit pattern for a page descriptor.



DE = Enable Data Cache
IE = Enable Instruction Cache

Figure 4-4. Cache Control Register

System hardware can assert the cache disable (CDIS) signal to dynamically disable both caches, regardless of the state of the enable bits in the CACR. The caches are disabled immediately after the current access completes. If CDIS is asserted during the access for the first half of a misaligned operand spanning two cache lines, the data cache is disabled for the second half of the operand. Accesses by the execution units bypass the caches while they are disabled and do not affect their contents (with the exception of CINV and CPUSH instructions). Disabling the caches with CDIS does not affect snoop operations. CDIS is intended primarily for use by in-circuit emulators to allow swapping between the tags and emulator memories.

Even if the instruction cache is disabled, the M68040 can cache instructions because of an internal cache line register. This happens for instruction loops that are completely resident within the first six bytes of a half-line. Thus, the cache line holding register can operate as a small cache. If a loop fits anywhere within the first three words of a half-line, then it becomes cached.

The CINV and CPUSH instructions support cache management in the supervisor mode. CINV allows selective invalidation of cache entries. CPUSH performs two operations: 1) any selected data cache lines containing dirty data are pushed to memory; 2) all selected cache lines are invalidated. This operation can be used to update a page in memory before swapping it out with snooping disabled or to push dirty data when changing a page caching mode to write-through. Because of the size of the caches, pushing pages or an entire cache incurs a significant time penalty. However, these instructions are interruptable to avoid large interrupt latencies. The state of the CDIS signal or the cache enable bits in the CACR does not affect the operation of CINV and CPUSH. Both instructions allow operation on a single cache line, all cache lines in a specific page, or an

entire cache, and can select one or both caches for the operation. For line and page operations, a physical address in an address register specifies the memory address.

4.3 CACHING MODES

Every IU access to the cache has an associated caching mode that determines how the cache handles the access. An access can be cachable in either the write-through or copyback modes, or it can be cache inhibited in nonserialized or serialized modes. The CM field corresponding to the logical address of the access normally specifies, on a page-by-page basis, one of these caching modes. The default memory access caching mode is nonserialized. When the cache is enabled and memory management is disabled, the default caching mode is write-through. The transparent translation registers and MMUs allow the defaults to be overridden. In addition, some instructions and IU operations perform data accesses that have an implicit caching mode associated with them. The following paragraphs discuss the different caching accesses and their related cache modes.

4.3.1 Cachable Accesses

If a page descriptor's CM field indicates write-through or copyback, then the access is cachable. A read access to a write-through or copyback page is read from the cache if matching data is found. Otherwise, the data is read from memory and used to update the cache. Since instruction cache accesses are always reads, the selection of write-through or copyback modes do not affected them. The following paragraphs describe the write-through and copyback modes in detail.

4.3.1.1 WRITE-THROUGH MODE. Accesses to pages specified as write-through are always written to the external address, although the cycle can be buffered, keeping memory and cache data consistent. Writes in write-through mode are handled with a no-write-allocate policy—i.e., writes that miss in a data cache are written to memory but do not cause the corresponding line in memory to be loaded into the cache. Write accesses always write through to memory and update matching cache lines. Specifying write-through mode for the shared pages maintains cache coherency for shared memory areas in a multiprocessing environment. The cache supplies data to instruction or data read accesses that hit in the appropriate cache; misses cause a new cache line to be loaded into the cache, replacing a valid cache line if there are no invalid lines.

4.3.1.2 COPYBACK MODE. Copyback pages are typically used for local data structures or stacks to minimize external bus usage and reduce write access latency. Write accesses to pages specified as copyback that hit in the data cache update the cache line and set the corresponding D-bits without an external bus access. The dirty cached data is only written to memory if 1) the line is replaced due to a miss, 2) a cache inhibited access matches the line, or 3) the CPUSH instruction explicitly pushes the line. If a write access misses in the cache, the memory unit reads the needed cache line from memory and updates the cache. When a miss causes a dirty cache line to be selected for replacement, the memory unit places the line in an internal copyback buffer. The replacement line is read into the cache, and writing the dirty cache line back to memory updates memory.

4.3.2 Cache-Inhibited Accesses

Address space regions containing targets such as I/O devices and shared data structures in multiprocessing systems can be designated cache inhibited. If a page descriptor's CM field indicates nonserialized or serialized, then the access is cache inhibited. The caching operation is identical for both cache-inhibited modes. If the CM field of a matching address indicates either nonserialized or serialized modes, the cache controller bypasses the cache and performs an external bus transfer. The data associated with the access is not cached internally, and the cache inhibited out (CIOUT) signal is asserted during the bus transfer to indicate to external memory that the access should not be cached. If the data cache line is already resident in an internal cache, then the data cache line is pushed from the cache if it is dirty or the data cache line is invalidated if it is valid.

If the CM field indicates serialized, then the sequence of read and write accesses to the page is guaranteed to match the sequence of the instruction order. Without serialization, the IU pipeline allows read accesses to occur before completion of a write-back for a previous instruction. Serialization forces operand read accesses for an instruction to occur only once by preventing the instruction from being interrupted after the operand fetch stage. Otherwise, the instruction is aborted, and the operand is accessed when the instruction is restarted. These guarantees apply only when the CM field indicates the serialized mode and the accesses are aligned. Regardless of the selected cache mode, locked accesses are implicitly serialized. The TAS, CAS, and CAS2 instructions use locked accesses for operands in memory and for updating translation table entries during table search operations.

4.3.3 Special Accesses

Several other processor operations result in accesses that have special caching characteristics besides those with an implied cache-inhibited access in the serialized mode. Exception stack accesses, exception vector fetches, and table searches that miss in the cache do not allocate cache lines in the data cache, preventing replacement of a cache line. Cache hits by these accesses are handled in the normal manner according to the caching mode specified for the accessed address.

Accesses by the MOVE16 instruction also do not allocate cache lines in the data cache for either read or write misses. Read hits on either valid or dirty cache lines are read from the cache. Write hits invalidate a matching line and perform an external access. Interacting with the cache in this manner prevents a large block move or block initialization implemented with a MOVE16 from being cached, since the data may not be needed immediately.

If the data cache is re-enabled after a locked access has hit and the data cache was disabled, the next non-locked access that results in a data cache miss will not be cached.

4.4 CACHE PROTOCOL

The cache protocol for processor and snooped accesses is described in the following paragraphs. In all cases, an external bus transfer will cause a cache line state to change

only if the bus transfer is marked as snoopable on the bus. The protocols described in the following paragraphs assume that the data is cachable (i.e., write-through and copyback).

4.4.1 Read Miss

A processor read that misses in the cache causes the cache controller to request a bus transaction that reads the needed line from memory and supplies the required data to the IU. The line is placed in the cache in the valid state. Snooped external reads that miss in the cache have no affect on the cache.

4.4.2 Write Miss

The cache controller handles processor writes that miss in the cache differently for write-through and copyback pages. Write misses to copyback pages cause the processor to perform a bus transaction that writes the needed cache line into its cache from memory in the same manner as for a read miss. The new cache line is then updated with the write data, and the D-bits are set for each long word that has been modified, leaving the cache line in the dirty state. Write misses to write-through pages write directly to memory without loading the corresponding cache line in the cache. Snooped external writes that miss in the cache have no affect on the cache.

4.4.3 Read Hit

The cache controller handles processor reads that hit in the cache differently for write-through and copyback pages. No bus transaction is performed, and the state of the cache line does not change. Physical address bit 3 selects either the upper or lower half-line containing the required operand. This half-line is driven onto the internal bus. If the required data is allocated entirely within the half-line, only one access into the cache is required. Because the organization of the cache does not allow selection of more than one half-line at a time, misalignment across a half-line boundary requires two accesses into the cache.

A snooped external read that hits in the cache is ignored if the cache line is valid. If the snooped access hits a dirty line, memory is inhibited from responding, and the data is sourced from the cache directly to the alternate bus master. A snooped read hit does not change the state of the cache line unless the snooped access also indicates mark invalid, which causes the line to be invalidated after the access, even if it is dirty. Alternate bus masters should indicate mark invalid only for line reads to ensure the entire line is transferred before invalidating.

4.4.4 Write Hit

The cache controller handles processor writes that hit in the cache differently for write-through and copyback pages. For write-through accesses, a processor write hit causes the cache controller to update the affected long-word entries in the cache line and to request an external memory write transfer to update memory. The cache line state does not change. A write-through access to a line containing dirty data constitutes a system programming error even if the D-bits for the line are unchanged. This situation can be

avoided by pushing cache lines when a page descriptor is changed and ensuring that alternate bus masters indicate the appropriate snoop operation for writes to corresponding pages (i.e., mark invalid for write-through pages and sink data for copyback pages). If the access is copyback, the cache controller updates the cache line and sets the D-bit for of the appropriate long words in the cache line. An external write is not performed, and the cache line state changes to, or remains in, the dirty state.

An alternate bus master can drive the SCx signals for a write access with an encoding that indicates to the M68040 that it should sink the data, inhibit memory, and respond as a slave if the access hits in the cache. The cache operation depends on the access size and current line state. A snooped line write that hits a valid line always causes the corresponding cache line to be invalidated. For snooped writes of byte, word, or long-word size that hit a dirty line, the processor inhibits memory and responds to the alternate bus master as a slave, sinking the data. Data received from the alternate bus master is written to the appropriate long word in the cache line, and the D-bit is set for that entry. The cache controller invalidates a cache line if the snoop control pins have indicated that a matching cache line is marked invalid for a snoop write.

4.5 CACHE COHERENCY

The M68040 provides several different mechanisms to assist in maintaining cache coherency in multimaster systems. Both write-through and copyback memory update techniques are supported to maintain coherency between the data cache and memory.

Alternate bus master accesses can reference data that the M68040 caches, causing coherency problems if the accesses are not handled properly. The M68040 snoops the bus during alternate bus master transfers. If a write access hits in the cache, the M68040 can update its internal caches, or if a read access hits, it can intervene in the access to supply dirty data. Caches can be snooped even if they are disabled. The alternate bus master controls snooping through the snoop control signals, indicating which access can be snooped and the required operation for snoop hits. Table 4-1 lists the requested snoop operation for each encoding of the snoop control signals. Since the processor and the bus snooper must both access the caches, the snoop controller has priority over the processor for snoopable accesses to maintain cache coherency.

Table 4-1. Snoop Control Encoding

		Requested Snoop Operation	
SC1	**SC0**	**Alternate Bus Master Read Access**	**Alternate Bus Master Write Access**
0	0	Inhibit Snooping	Inhibit Snooping
0	1	Supply Dirty Data and Leave Dirty Data	Sink Byte/Word/Long/Long Word
1	0	Supply Dirty Data and Mark Line Invalid	Invalidate Line
1	1	Reserved (Snoop Inhibited)	Reserved (Snoop Inhibited)

The snooping protocol and caching mechanism supported by the M68040 are optimized to support multimaster systems with the M68040 as the single caching master. In systems

implementing multiple MC68040s as bus masters, shared data should be stored in write-through pages. This procedure allows each processor to cache shared data for read access while forcing a processor write to shared data to appear as an external write to memory, which the other processors can snoop.

If shared data is stored in copyback pages, only one processor at a time can cache the data since writes to copyback pages do not access the external bus. If a processor accesses shared data cached by another processor, the slave can source the data to the master without invalidating its own copy only if the transfer to the master is cache inhibited. For the master processor to cache the data, it must force invalidation of the slave processor's copy of the data (by specifying mark invalid for the snoop operation), and the memory controller must monitor the data transfer between the processors and update memory with the transferred data. The memory update is required since the master processor is unaware of the sourced data (valid data from memory or dirty data from a snooping processor) and initially creates a valid cache line, losing dirty status if a snooping processor supplies the data.

Coherency between the instruction cache and the data cache must be maintained in software since the instruction cache does not monitor data accesses. Processor writes that modify code segments (i.e., resulting from self-modifying code or from code executed to load a new page from disk) access memory through the data memory unit. Because the instruction cache does not monitor these data accesses, stale data occurs in the instruction cache if the corresponding data in memory is modified. Invalidating instruction cache lines before writing to the corresponding memory lines can prevent this coherency problem, but only if the data cache line is in write-through mode and the page is marked serialized. A cache coherency problem could arise if the data cache line is configured as copyback and no serialization is done.

To fully support self-modifying code in any situation, it is imperative that a CPUSHA instruction be executed before the execution of the first self-modified instruction. The CPUSHA instruction has the effect of ensuring that there is no stale data in memory, the pipeline is flushed, and instruction prefetches are repeated and taken from external memory.

Another potential coherency problem exists due to the relationship between the cache state information and the translation table descriptors. Because each cache line reflects page state information, a page should be flushed from the cache before any of the page attributes are changed. The presence of a valid or dirty cache line implicitly indicates that accesses to the page containing the line are cachable. The presence of a dirty cache line implies that the page is not write protected and that writes to the page are in copyback mode. A system programming error occurs when page attributes are changed without flushing the corresponding page from the cache, resulting in cache line states inconsistent with their page definitions. Even with these inconsistencies, the cache is defined and predictable.

4.6 MEMORY ACCESSES FOR CACHE MAINTENANCE

The cache controller in each memory unit performs all maintenance activities that supply data from the cache to the execution units. The activities include requesting accesses to the bus interface unit for reading new cache lines and writing dirty cache lines to memory. The following paragraphs describe the memory accesses resulting from cache fill operations (by both caches) and push operations (by the data cache). Refer to **Section 7 Bus Operation** for detailed information about the bus cycles required.

4.6.1 Cache Filling

When a new cache line is required, the cache controller requests a line read from the bus controller. The bus controller requests a burst read transfer by indicating a line access with the size signals (SIZ1, SIZ0) and indicates which line in the set is being loaded with the transfer line number signals (TLN1, TLN0). TLN1 and TLN0 are undefined for the instruction cache. These pins indicate the appropriate line numbers for data cache transfers only. Table 4-2 lists the definition of the TLNx encoding.

Table 4-2. TLNx Encoding

TLN1	TLN0	Line
0	0	Zero
0	1	One
1	0	Two
1	1	Three

The responding device sequentially supplies four long words of data and can assert the transfer cache inhibit signal (TCI) if the line is not cachable. If the responding device does not support the burst mode, it should assert the TBI signal for the first long word of the line access. The bus controller responds by terminating the line access and completes the remainder of the line read as three, sequential, long-word reads.

Bus controller line accesses implicitly request burst mode operations from external memory. To operate in the burst mode, the device or external hardware must be able to increment the low-order address bits as described in **Section 7 Bus Operation**. The device indicates its ability to support the burst access by acknowledging the initial long-word transfer with transfer acknowledge (TA) asserted and TBI negated. This procedure causes the processor to continue to drive the address and bus control signals and to latch a new data value for the cache line at the completion of each subsequent cycle (as defined by TA) for a total of four cycles. The bursting mechanism requires addresses to wrap around so that the entire four long words in the cache line are filled in a single operation.

When a cache line read is initiated, the first cycle attempts to load the line entry corresponding to the instruction half-line or data item requested by the IU. Subsequent transfers are for the remaining entries in the cache line. In the case of a misaligned

access in which the operand spans two line entries, the first cycle corresponds to the line entry containing the portion of the operand at the lower address.

The cache controller temporarily stores the data from each cycle in a line read buffer, where it is immediately available to the IU. If a misaligned access spans two entries in the line, the second portion of the operand is available to the IU as soon as the second memory cycle completes. A new IU access that hits the cache line being filled is also supplied data as soon as the required long word has been received from the bus controller. During the period required to fill the buffer, other IU accesses that hit in the cache are supplied data. This is vertical for a short cache-inhibited code loop that is less than eight bytes in length. Subsequent interactions of the loop hit in the buffer, but appear to hit in the cache since there is no external bus activity associated with the reads.

The assertion of TCI during the first cycle of a burst read operation inhibits loading of the buffered line into the cache, but it does not cause the burst transfer (or pseudo-burst transfer if TBI is asserted with TCI) to be terminated early. The data placed in the buffer is accessible by the IU until the last long word of the burst is transferred from the bus controller, after which the contents of the buffer are invalidated without being copied into the cache. The assertion of TCI is ignored during the second, third, or fourth cycle of a burst operation and is ignored for write operations.

A bus error occurring during a burst operation causes the burst operation to abort. If the bus error occurs during the first cycle of a burst, the data from the bus is ignored. If the access is a data cycle, exception processing proceeds immediately. If the cycle is for an instruction prefetch, a bus error exception is pending. The bus error is processed only if the IU attempts to use either instruction word. Refer to **Section 7 Bus Operation** for more information about pipeline operation.

For either cache, when a bus error occurs on the second cycle or later, the burst operation is aborted and the line buffer is invalidated. The processor may or may not take an exception, depending on the status of the pending data request. If the bus error cycle contains a portion of a data operand that the processor is specifically waiting for (e.g., the second half of a misaligned operand), the processor immediately takes an exception. Otherwise, no exception occurs, and the cache line fill is repeated the next time data within the line is required. In the case of an instruction cache line fill, the data from the aborted cycle is completely ignored.

On the initial access of a line read, a retry (indicated by the assertion of TA and TEA) causes the bus controller to retry the bus cycle. However, a retry signaled during the remaining cycles of the line access (either burst or pseudo-burst) is recognized as a bus error, and the processor handles it as described in the previous paragraphs.

A cache inhibit or bus error on a line read can change the state of the line being replaced, even though the new line is not copied into the cache. Before loading a new line, the cache line being replaced is copied to the push buffer; if it is dirty, the cache line is invalidated. If a cache inhibit or bus error occurs on a replacement line read, a dirty line is restored to the cache from the push buffer. However, the line being replaced is not restored in the cache if it was originally valid and the cache line remains invalid. If the line

read resulting from a write miss in copyback mode is cache inhibited, the write access misses in the cache and writes through to memory.

4.6.2 Cache Pushes

When the cache controller selects a dirty data cache line for replacement, memory must be updated with the dirty data before the line is replaced. This occurs when a CPUSH instruction execution explicitly selects the cache and when a cache inhibit access hits in the cache. To reduce the requested data's latency in the new line, the dirty line being replaced is temporarily placed in a push buffer while the new line is fetched from memory. When a line is allocated to the push buffer, an alternate bus master can snoop it, but the execution units cannot access it. After the bus transfer for the new line successfully completes, the dirty cache line is copied back to memory, and the push buffer is invalidated. If the operation to access the replacement line is abnormally terminated or signaled as cache inhibited, the line in the push buffer is copied back into its original position in the cache, and the processor continues operation as described in the previous paragraphs.

The number of dirty long words in the line to be pushed determines the size of the push transfer on the bus, minimizing bus bandwidth required for the push. A single long word is written to memory using a long-word push transfer if it is dirty. A push transfer is distinguished from a normal write transfer by an encoding of 000 on the transfer modifier signals (TM2–TM0) for the push. Asserting TA and TEA retries the transfer; a bus-error-asserted TEA terminates it. If a bus error terminates a push transfer, the processor immediately takes an exception.

A line containing two or more dirty long words is copied back to memory, using a line push transfer. For a line push, the bus controller requests a burst write transfer by indicating a line access with SIZ1 and SIZ0. The responding device sequentially accepts four long words of data. If the responding device does not support the burst mode, it should assert TBI for the first long word of the line access. The bus controller responds by terminating the line access and completes the remainder of the line push as three, sequential, long-word writes. The first cycle of the burst can be retried, but the bus controller interprets a retry for any of the three remaining cycles as a bus error. If a bus error occurs in any cycle in the line push transfer, the processor immediately takes an exception.

A dirty cache line hit by a cache-inhibited access is pushed before the external bus access occurs. If the access is part of a locked transfer sequence for TAS, CAS, or CAS2 operand accesses or translation table updates, the LOCK signal is also asserted for the push access.

4.7 CACHE OPERATION SUMMARY

The instruction and data caches function independently when servicing access requests from the IU. The following paragraphs discuss the operational details for the caches and present state diagrams depicting the cache line state transitions.

4.7.1 Instruction Cache

The IU uses the instruction cache to store instruction prefetches as it requests them. Instruction prefetches are normally requested from sequential memory locations except when a change of program flow occurs (e.g., a branch taken) or when an instruction that can modify the status register (SR) is executed, in which case the instruction pipe is automatically flushed and refilled. The instruction cache supports a line-based protocol that allows individual cache lines to be in either the invalid or valid states.

For instruction prefetch requests that hit in the cache, the half-line selected by physical address bit 3 is multiplexed onto the internal instruction data bus. When an access misses in the cache, the cache controller requests the line containing the required data from memory and places it in the cache. If available, an invalid line is selected and updated with the tag and data from memory. The line state then changes from invalid to valid by setting the V-bit. If all lines in the set are already valid, a pseudo-random replacement algorithm is used to select one of the four cache lines replacing the tag and data contents of the line with the new line information. Figure 4-5 illustrates the instruction-cache line state transitions resulting from processor and snoop controller accesses. Transitions are labeled with a capital letter, indicating the previous state, followed by a number indicating the specific case listed in Table 4-3.

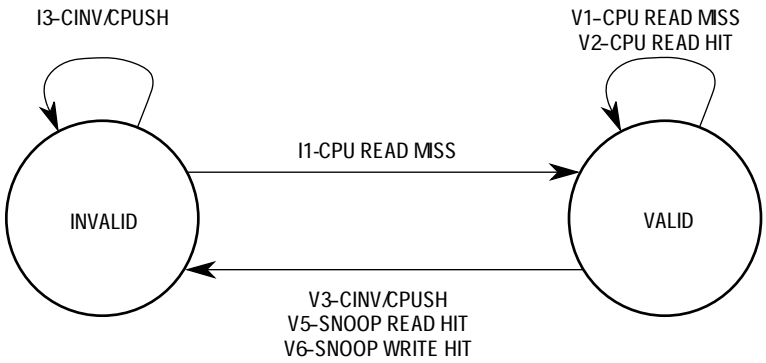


Figure 4-5. Instruction-Cache Line State Diagram

Table 4-3. Instruction-Cache Line State Transitions

Cache Operation	Current State			
	Invalid Cases		Valid Cases	
CPU Read Miss	I1	Read line from memory; supply data to CPU and update cache; go to valid state.	V1	Read line from memory; supply data to CPU and update cache (replacing old line); remain in current state.
CPU Read Hit	I2	Not Possible	V2	Supply data to CPU; remain in current state.
Cache Invalidate or Push (CINV or CPUSH)	I3	No action; remain in current state.	V3	No action; go to invalid state.
Alternate Master Read Hit (Snoop Control = 01 — Leave Dirty)	I4	Not possible; not snooped.	V4	Not possible; not snooped.
Alternate Master Read Hit (Snoop Control = 10 — Invalidate)	I5	Not Possible	V5	No action; go to invalid state.
Alternate Master Write Hit (Snoop Control = 01 — Leave Dirty or Snoop Control = 10 — Invalidate)	I6	Not Possible	V6	No action; go to invalid state.

4.7.2 Data Cache

The IU uses the data cache to store operand data as it generates the data. The data cache supports a line-based protocol allowing individual cache lines to be in one of three states: invalid, valid, or dirty. To maintain coherency with memory, the data cache supports both write-through and copyback modes, specified by the CM field for the page.

Read misses and write misses to copyback pages cause the cache controller to read a new cache line from memory into the cache. If available, an invalid line in the selected set is updated with the tag and data from memory. The line state then changes from invalid to valid by setting the V-bit for the line. If all lines in the set are already valid or dirty, the pseudo-random replacement algorithm is used to select one of the four lines and replace the tag and data contents of the line with the new line information. Before replacement, dirty lines are temporarily buffered and later copied back to memory after the new line has been read from memory. If a snoop access occurs before the buffered line is written to memory, the snoop controller snoops the buffer and the caches. Figure 4-6 illustrates the three possible states for a data cache line, with the possible transitions caused by either the processor or snooped accesses. Transitions are labeled with a capital letter, indicating the previous state, followed by a number indicating the specific case listed in Table 4-4.

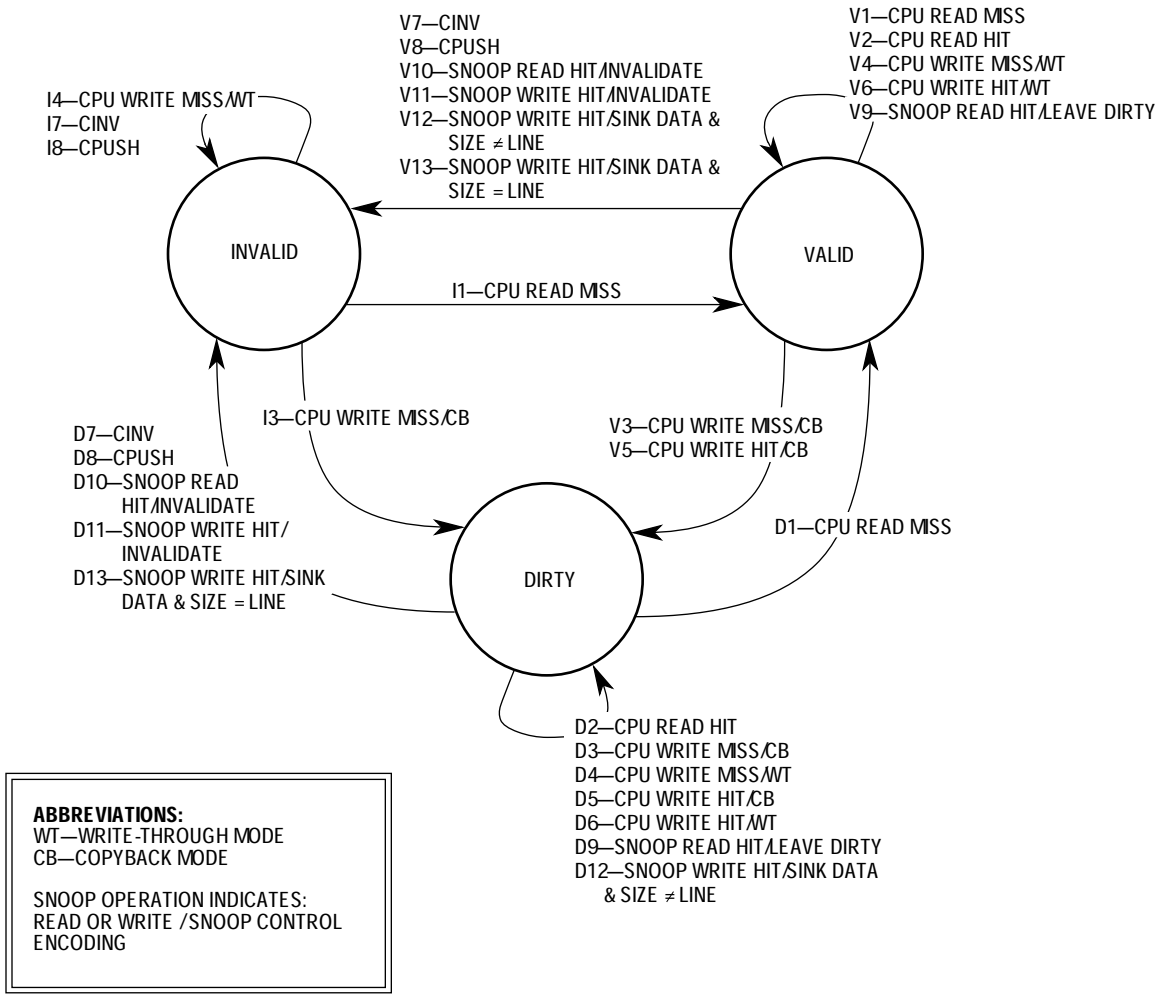


Figure 4-6. Data-Cache Line State Diagram

Table 4-4. Data-Cache Line State Transitions

Cache Operation	Current State					
	Invalid Cases		Valid Cases		Dirty Cases	
CPU Read Miss	I1	Read line from memory; supply data to CPU and update cache; go to valid state.	V1	Read line from memory; supply data to CPU and update cache (replacing old line); remain in current state.	D1	Buffer dirty cache line; read new line from memory; supply data to CPU and update cache; write buffered dirty data to memory; go to valid state.
CPU Read Hit	I2	Not Possible	V2	Supply data to CPU; remain in current state.	D2	Supply data to CPU; remain in current state.
CPU Write Miss (Copyback)	I3	Read line from memory into cache; write data to cache; set Dn bits of modified long words; go to dirty state.	V3	Read line from memory into cache (replacing old line); write data to cache and set Dn bits; go to dirty state.	D3	Buffer dirty cache line; read new line from memory; write data to cache and set Dn bits; write buffered dirty data to memory; remain in current state.
CPU Write Miss (Write-through)	I4	Write data to memory; remain in current state.	V4	Write data to memory; remain in current state.	D4	Write data to memory; remain in current state (see note).
CPU Write Hit (Copyback)	I5	Not Possible	V5	Write data into cache; set Dn bits of modified long words; go to dirty state.	D5	Write data in cache; set Dn bits of modified long words; remain in current state.
CPU Write Hit (Write-through)	I6	Not Possible	V6	Write data to cache; write data to memory; remain in current state.	D6	Write data into cache (no change to Dn bits); write data to memory; remain in current state (see note).
Cache Invalidate (CINV)	I7	No action; remain in current state.	V7	No action; go to invalid state.	D7	No action (dirty data lost); go to invalid state.
Cache Push (CPUSH)	I8	No action; remain in current state.	V8	No action; go to invalid state.	D8	Write dirty data to memory; go to invalid state.
Alternate Master Read Hit (Snoop Control = 01 — Leave Dirty)	I9	Not Possible	V9	No action; remain in current state.	D9	Inhibit memory and source data; remain in current state.

NOTE: Dirty state transitions D4 and D6 are the result of a system programming error and should be avoided even though they are technically valid.

Table 4-4. Data-Cache Line State Transitions (Continued)

Cache Operation	Current State					
	Invalid Cases		Valid Cases		Dirty Cases	
Alternate Master Read Hit (Snoop Control = 10 — Invalidate)	I10	Not Possible	V10	No action; go to invalid state.	D10	Inhibit memory and source data; go to invalid state
Alternate Master Write Hit (Snoop Control = 10 —Invalidate)	I11	Not Possible	V11	No action; go to invalid state.	D11	No action; go to invalid state.
Alternate Master Write Hit (Snoop Control = 01 — Sink Data and Size ≠ Line)	I12	Not Possible	V12	No action; go to invalid state.	D12	Inhibit memory and sink data; set Dn bits of modified long words; remain in current state.
Alternate Master Write Hit (Snoop Control = 01 — Sink Data and Size = Line)	I13	Not Possible	V13	No action; go to invalid state.	D13	No action; go to invalid state.

SECTION 5
SIGNAL DESCRIPTION

This section contains brief descriptions of the input and output signals in their functional groups (see Figure 5-1). Each signal's function is briefly explained, referencing other sections that contain detailed information about the signal and related operations. Table 5-1 lists the signal names, mnemonics, and functional descriptions of the input and output signals for the M68040. Timing specifications for these signals can be found in **Section 11 MC68040 Electrical and Thermal Characteristics**.

NOTES

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 independent of the voltage level (high or low) that they represent.

For the MC68040V, MC68LC040, MC68EC040, and MC68EC040V ignore all references to the floating-point unit (FPU). For the MC68EC040 and MC68EC040V only, ignore all references to the memory management unit (MMU). Some pin names are different on these parts; please refer to the appropriate appendix in the back of this book for more information.

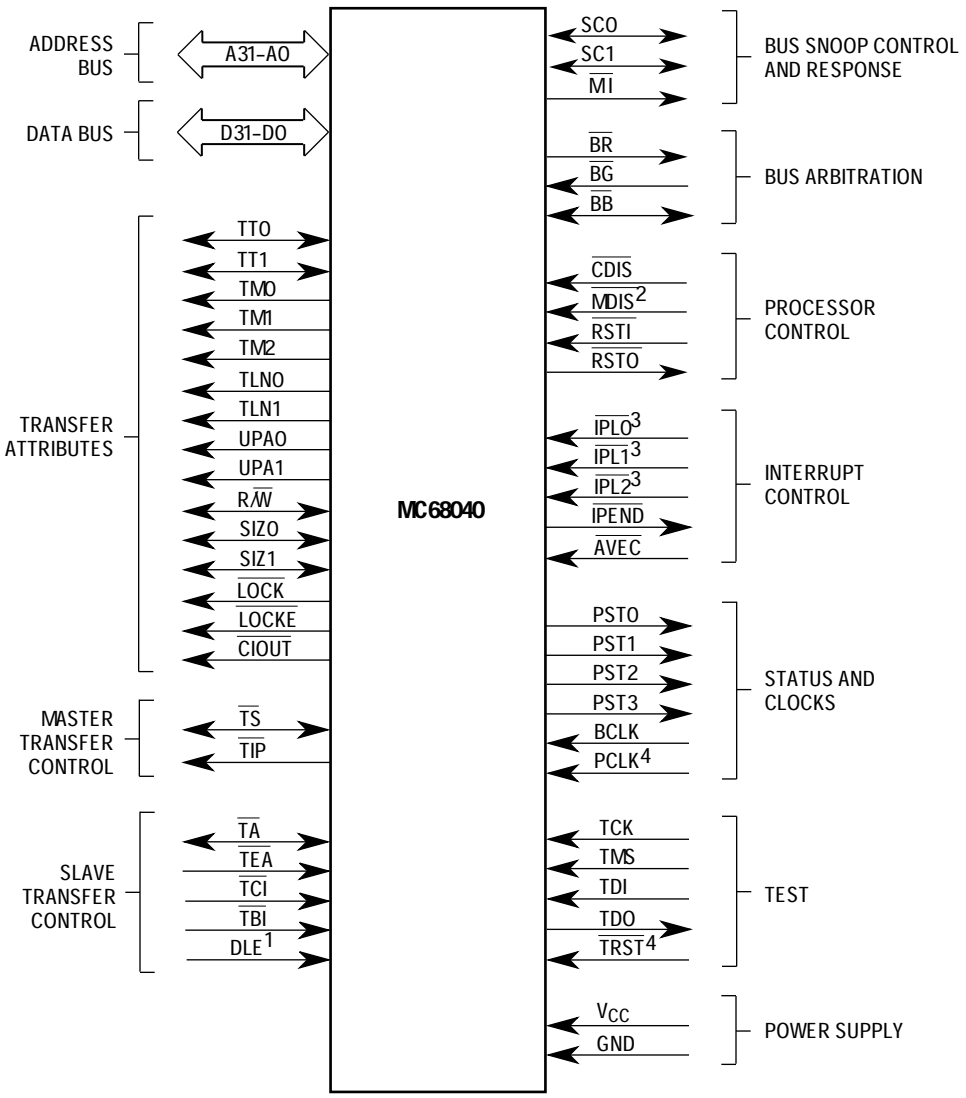
Table 5-1. Signal Index

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 address translation entry.
Read/Write	R/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	LOCK	Indicates a bus transfer is part of a read-modify-write operation, and the sequence of transfers should not be interrupted.
Bus Lock End	LOCKE	Indicates the current transfer is the last in a locked sequence of transfers.
Cache Inhibit Out	CIOUT	Indicates the processor will not cache the current bus transfer.
Transfer Start	TS	Indicates the beginning of a bus transfer.
Transfer in Progress	TIP	Asserted for the duration of a bus transfer.
Transfer Acknowledge	TA	Asserted to acknowledge a bus transfer.
Transfer Error Acknowledge	TEA	Indicates an error condition exists for a bus transfer.
Transfer Cache Inhibit	TCI	Indicates the current bus transfer should not be cached.
Transfer Burst Inhibit	TBI	Indicates the slave cannot handle a line burst access.
Data Latch Enable[1]	DLE	Alternate clock input used to latch input data when the processor is operating in DLE mode.
Snoop Control	SC1,SC0	Indicates the snooping operation required during an alternate master access.
Memory Inhibit	MI	Inhibits memory devices from responding to an alternate master access during snooping operations.
Bus Request	BR	Asserted by the processor to request bus mastership.
Bus Grant	BG	Asserted by an arbiter to grant bus mastership to the processor.
Bus Busy	BB	Asserted by the current bus master to indicate it has assumed ownership of the bus.
Cache Disable	CDIS	Dynamically disables the internal caches to assist emulator support.
MMU Disable[2]	MDIS	Disables the translation mechanism of the MMUs.
Reset In	RSTI	Processor reset.
Reset Out	RSTO	Asserted during execution of a RESET instruction to reset external devices.
Interrupt Priority Level[3]	IPL2–IPL0	Provides an encoded interrupt level to the processor.
Interrupt Pending	IPEND	Indicates an interrupt is pending.
Autovector	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.

Table 5-1. Signal Index (Continued)

Signal Name	Mnemonic	Function
Processor Clock	PCLK[4]	Clock input used for internal logic timing. The PCLK frequency is exactly 2 × 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	TRST[4]	Provides an asynchronous reset of the TAP controller.
Power Supply	V_{CC}	Power supply.
Ground	GND	Ground connection.

NOTES:
1. This signal is only available on the MC68040.
2. This signal is not available on the MC68EC040 and the MC68EC040V.
3. These signals are different on power-up for the MC68LC040 and MC68EC040.
4. These signals are not available on the MC68040V and MC68EC040V.



NOTES:
1. This signal is only available on the MC68040.
2. This signal is not available on the MC68EC040 and MC68EC040V.
3. These signals are different on power-up for the MC68LC040 and MC68EC040.
4. These signals are not available on the MC68040V and MC68EC040V.

Figure 5-1. Functional Signal Groups

5.1 ADDRESS BUS (A31–A0)

These three-state bidirectional signals provide the address of the first item of a bus transfer (except for acknowledge transfers) when the M68040 is the bus master. When an alternate bus master is controlling the bus, the processor examines (snoops) these signals to determine whether the processor should intervene in the access to maintain cache coherency.

The level on CDIS can select a multiplexed bus mode during processor reset, which allows the address bus and data bus to be physically tied together for multiplexed bus

applications. Refer to **Section 7 Bus Operation** for detailed information about the relationship of the address bus to bus operation and the multiplexed bus mode. Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for details concerning the CDIS level and multiplexed bus mode.

5.2 DATA BUS (D31–D0)

These three-state bidirectional signals provide the general-purpose data path between the M68040 and all other devices. The data bus can transfer 8, 16, or 32 bits of data per bus transfer. During a burst transfer, the data lines are time-multiplexed to carry all 128 bits of the burst request using four 32-bit transfers.

The level on CDIS can select a multiplexed bus mode during processor reset, which allows the data bus and address bus to be physically tied together for multiplexed bus applications. The level on MDIS can select a data latch mode during processor reset, which allows the memory interface to specify when the processor should latch input data through the DLE signal. **Section 7 Bus Operation** provides detailed information about the relationship of the data bus to bus operation, the multiplexed bus mode, and the data latch mode. Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for details concerning the CDIS level and multiplexed bus mode.

5.3 TRANSFER ATTRIBUTE SIGNALS

The following paragraphs describe the transfer attribute signals, which provide additional information about the bus transfer. Refer to **Section 7 Bus Operation** for detailed information about the relationship of the transfer attribute signals to bus operation.

5.3.1 Transfer Type (TT1, TT0)

The processor drives these three-state bidirectional signals to indicate the type of access for the current bus transfer. During bus transfers by an alternate bus master, the processor samples these signals to determine if it should snoop the transfer; only normal and MOVE16 accesses can be snooped. Table 5-2 lists the definition of the transfer-type encoding. The acknowledge access (TT1 = 1 and TT0 = 1) is used for both interrupt and breakpoint acknowledge transfers, and for LPSTOP broadcast cycles on the MC68040V and MC68EC040V.

Table 5-2. Transfer-Type Encoding

TT1	TT0	Transfer Type
0	0	Normal Access
0	1	MOVE16 Access
1	0	Alternate Logical Function Code Access
1	1	Acknowledge Access

5.3.2 Transfer Modifier (TM2–TM0)

These three-state outputs provide supplemental information for each transfer type. Table 5-3 lists the encoding for normal and MOVE16 transfers, and Table 5-4 lists the encoding for alternate access transfers. For interrupt acknowledge transfers, the TMx signals carry the interrupt level being acknowledged; for breakpoint acknowledge transfers and LPSTOP broadcast cycles on the MC68040V and MC68EC040V, the TMx signals are low. When the M68040 is not the bus master, the TMx signals are set to a high-impedance state.

Table 5-3. Normal and MOVE16 Access Transfer Modifier Encoding

TM2	TM1	TM0	Transfer Modifier
0	0	0	Data Cache Push Access
0	0	1	User Data Access*
0	1	0	User Code Access
0	1	1	MMU Table Search Data Access
1	0	0	MMU Table Search Code Access
1	0	1	Supervisor Data Access*
1	1	0	Supervisor Code Access
1	1	1	Reserved

* MOVE16 accesses use only these encodings.

Table 5-4. Alternate Access Transfer Modifier Encoding

TM2	TM1	TM0	Transfer Modifier
0	0	0	Logical Function Code 0
0	0	1	Reserved
0	1	0	Reserved
0	1	1	Logical Function Code 3
1	0	0	Logical Function Code 4
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Logical Function Code 7

5.3.3 Transfer Line Number (TLN1, TLN0)

These three-state outputs indicate which line in the set of four data cache lines is being accessed for normal push and line data read accesses. TLNx signals are undefined for all other accesses to instruction space and are placed in a high-impedance state when the processor relinquishes the bus.

The TLNx signals can be used in high-performance systems to build an external snoop filter with a duplicate set of cache tags. The TLNx signals and address bus provide a direct indication of the state of the data caches and can be used to help maintain the duplicate tag store. The TLNx pins do not indicate the correct TLN number when an instruction cache burst fill occurs.

5.3.4 User-Programmable Attributes (UPA1, UPA0)

The UPAx signals are three-state outputs. If they match the logical address, the user-programmable attribute bits in the address translation entry or the transparent translation register determine the UPAx signal level. These signals are only for normal code, data, and MOVE16 accesses. For all other accesses, including table search and cache line push accesses, which may result from a normal access, the UPAx signals are zero. If the transparent translation register and the memory management unit are disabled, the UPAx signals are also zero. When the M68040 is not the bus master, these signals are set to a high-impedance state.

5.3.5 Read/Write (R/W)

This bidirectional three-state signal defines the data transfer direction for the current bus cycle. A high level indicates a read cycle, and a low level indicates a write cycle. The bus snoop controller examines this signal when the processor is not the bus master.

5.3.6 Transfer Size (SIZ1, SIZ0)

These bidirectional three-state signals indicate the data size for the bus transfer. The bus snoop controller examines this signal when the processor is not the bus master. Refer to **Section 7 Bus Operation** for more information on the encoding of these signals.

5.3.7 Lock (LOCK)

This three-state output indicates that the current transfer is part of a sequence of locked transfers for a read-modify-write operation. The external arbiter can use LOCK to prevent an alternate bus master from gaining control of the bus and accessing the same operand between processor accesses for the locked sequence of transfers. Although LOCK indicates that the processor requests the bus be locked, the processor will give up the bus if the external arbiter negates the BG signal. When the M68040 is not the bus master, the LOCK signal is set to a high-impedance state. LOCK drives high before three-stating. Refer to **Section 7 Bus Operation** for information on locked transfers.

5.3.8 Lock End (LOCKE)

This three-state output indicates that the current transfer is the last in a sequence of locked transfers for a read-modify-write operation. The external arbiter can use LOCKE to support arbitration between unrelated locked transfer sequences while still maintaining the indivisible nature of each read-modify-write operation. When the M68040 is not the bus master, the LOCKE signal is set to a high-impedance state. LOCKE drives high before

three-stating. Do not use LOCKE if it is possible to retry the last write of a read-write-modify operation.

5.3.9 Cache Inhibit Out (C)

This three-state output reflects the state of the cache mode field in one of the address translation caches and is asserted for accesses to noncachable pages to indicate that an external cache should ignore the bus transfer. When the referenced logical address is within an area specified for transparent translation, the cache mode field of the appropriate transparent translation register controls the state of CIOUT. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for more information about the address translation caches and transparent translation. When the M68040 is not the bus master, the CIOUT signal is set to a high-impedance state.

5.4 BUS TRANSFER CONTROL SIGNALS

The following signals provide control functions for bus transfers. Refer to **Section 7 Bus Operation** for detailed information about the relationship of the bus transfer control signals to bus operation.

5.4.1 Transfer Start ( )

The processor asserts this three-state bidirectional signal for one clock period to indicate the start of each transfer. During alternate bus master accesses, the processor monitors this signal to detect the start of each transfer to be snooped.

5.4.2 Transfer in Progress ( )

This three-state output is asserted to indicate that a bus transfer is in progress and is negated during idle bus cycles if the bus is still granted to the processor. When the processor loses the bus, TIP negates after completion of the current transfer and goes to a high-impedance state. Note that TIP is kept asserted on back-to-back bus cycles.

5.4.3 Transfer Acknowledge ( )

This three-state bidirectional signal indicates the completion of a requested data transfer operation. During transfers by the M68040, TA is an input signal from the referenced slave device indicating completion of the transfer. During alternate bus master accesses, TA is normally three-stated to allow the referenced slave device to respond, and the M68040 samples it to detect the completion of each bus transfer. The M68040 can inhibit memory and intervene in the access to source or sink data in its internal caches by asserting TA to acknowledge the data transfer. This capability applies to alternate bus master accesses that reference modified (dirty) data in the M68040 caches.

5.4.4 Transfer Error Acknowledge ( )

The current slave asserts this input signal to indicate an error condition for the bus transaction. When asserted with TA, this signal indicates that the processor should retry

the access. During alternate bus master accesses, the M68040 samples TEA to detect completion of each bus transfer.

5.4.5 Transfer Cache Inhibit ( )

This input signal inhibits read data from being loaded into the M68040 instruction or data caches. TCI is ignored during all writes and after the first data transfer for both burst line reads and burst-inhibited line reads. TCI is also ignored during all alternate bus master transfers.

5.4.6 Transfer Burst Inhibit ( )

This input signal indicates to the processor that the accessed device cannot support burst mode accesses and that the requested line transfer should be divided into individual long-word transfers. Asserting TBI with TA terminates the first data transfer of a line access, which causes the processor to terminate the burst and access the remaining data for the line as three successive long-word transfers. During alternate bus master accesses, the M68040 samples the TBI to detect completion of each bus transfer.

5.5 SNOOP CONTROL SIGNALS

The following signals control the operation of the M68040 on-chip snoop logic. **Section 4 Instruction and Data Caches** provides information about the relationship of the snoop control signals to the caches, and **Section 7 Bus Operation** discusses the relationship of these signals to bus operation.

5.5.1 Snoop Control (SC1, SC0)

These input signals specify the snoop operation to be performed by the M68040 for an alternate bus master transfer. If the M68040 is allowed to snoop an alternate bus master read transfer, it can intervene in the access to supply data from its data cache when the memory copy is stale, ensuring that the alternate bus master receives valid data. Writes by an alternate bus master can also be snooped to either update the M68040 internal data cache with the new data or invalidate the matching cache lines, ensuring that subsequent M68040 reads access valid data. These signals are ignored when the processor is the bus master.

5.5.2 Memory Inhibit ( )

This output signal prevents an alternate bus master from accessing possibly stale data in memory while the M68040 is unable to respond. MI is asserted during reset preventing external memory from responding. When the SCx signals indicate an access should be snooped, the M68040 keeps MI asserted until it determines if intervention in the access is required. If no intervention is required, MI is negated and memory is allowed to respond to complete the access. Otherwise, MI remains asserted and the M68040 completes the transfer as a slave. It updates its caches on a write or supplies data to the alternate bus master on a read. MI is negated when the M68040 is the bus master. During a snoop

cycle, the M68040 ignores all TA and TEA assertions while MI is asserted; when RSTI is asserted, MI is asserted.

5.6 ARBITRATION SIGNALS

The following control signals support requests to an external arbiter to become the bus master. Refer to **Section 7 Bus Operation** for detailed information about the relationship of the arbitration signals to bus operation.

5.6.1 Bus Request ( )

This output signal indicates to the external arbiter that the processor needs to become bus master for one or more bus transfers. BR is negated when the M68040 begins an access to the external bus with no other accesses pending, and BR remains negated until another access is required. There are some situations in which the M68040 asserts BR and then negates it without having run bus transfers; this is a disregard request condition. Refer to **Section 7 Bus Operation** for details about this state.

5.6.2 Bus Grant ( )

This input signal from an external arbiter indicates that the bus is available to the M68040 as soon as the current bus access completes. BG must be asserted and BB must be negated (indicating the bus is free) before the M68040 assumes ownership of the bus.

5.6.3 Bus Busy ( )

This three-state bidirectional signal indicates that the bus is currently owned. BB is monitored as a processor input to determine when a alternate bus master has released control of the bus. BG must be asserted and BB must be negated (indicating the bus is free) before the M68040 asserts BB as an output to assume ownership of the bus. The processor keeps BB asserted until the external arbiter negates BG and the processor completes the bus transfer in progress. When releasing the bus, the processor negates BB, then sets it to a high-impedance state for use again as an input.

5.7 PROCESSOR CONTROL SIGNALS

The following signals control disabling caches and memory management units (MMUs) and support processor and external device initialization.

5.7.1 Cache Disable (C)

CDIS dynamically disables the on-chip caches on the next internal cache access boundary. CDIS does not flush the data and instruction caches; entries remain unaltered and become available after CDIS is negated. The assertion of CDIS does not affect snooping. During a processor reset, the level on CDIS is latched and used to select the normal bus mode (CDIS high) or multiplexed bus mode (CDIS low). Refer to **Section 4 Instruction and Data Caches** for information about the caches and to **Section 7 Bus Operation** for information about the multiplexed bus mode. Refer to **Appendix E**

MC68040 Floating-Point Emulation (MC68040FPSP) for descriptions of emulator use of this signal.

5.7.2 Reset In ( )

This input signal causes the M68040 to enter reset exception processing. The 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 outputs, except MI, are negated when RSTI is recognized. The assertion of RSTI does not affect the test pins. Refer to **Section 7 Bus Operation** for a description of reset operation and to **Section 8 Exception Processing** for information about the reset exception.

5.7.3 Reset Out ( )

The M68040 asserts this output during execution of the RESET instruction to initialize external devices. Refer to **Section 7 Bus Operation** for a description of reset out bus operation.

5.8 INTERRUPT CONTROL SIGNALS

The following signals control the interrupt functions.

5.8.1 Interrupt Priority Level (–)

These input signals provide an indication of an interrupt condition and the encoding of the interrupt level from a peripheral or external prioritizing circuitry. IPL2is the most significant bit of the level number. For example, since the IPL¯signals are active low, IPL2–IPL0 = $5 corresponds to an interrupt request at interrupt priority level 2.

During a processor reset, the levels on the IPL¯lines are latched and used to select the output driver characteristics for three signal groups listed in Table 5-5. Refer to **Section 8 Exception Processing** for information on interrupts and to **Section 11 MC68040 Electrical and Thermal Characteristics** for information on driver characteristics. Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for how these signals are different on power-up.

Table 5-5. Output Driver Control Groups

Signal	Output Buffers Controlled
IPL2	Data-Bus: D31–D0
IPL1	Address Bus and Transfer Attributes: A31–A0, CIOUT, LOCK, LOCKE , R/W, SIZ1–SIZ0, TLN1–TLN0, TM2–TM0, TT1–TT0, UPA1–UPA0
IPL0	Miscellaneous Control Signals: BB, BR , IPEND, MI, PST3–PST0, RSTO , TA, TDO, TIP, TS

NOTE: High input level = small buffers enabled; low input level = large buffers enabled.

5.8.2 Interrupt Pending Status ( )

This output signal indicates that an interrupt request has been recognized internally and exceeds the current interrupt priority mask in the status register (SR). External devices (other bus masters) can use IPEND to predict processor operation on the next instruction boundaries. IPEND is not intended for use as an interrupt acknowledge to external peripheral devices. Refer to **Section 7 Bus Operation** for bus information related to interrupts and to **Section 8 Exception Processing** for interrupt information.

5.8.3 Autovector ( )

This input signal is asserted with TA during an interrupt acknowledge transfer to request internal generation of the vector number. Refer to **Section 7 Bus Operation** for more information about automatic vectors.

5.9 STATUS AND CLOCK SIGNALS

The following paragraphs explain the signals that provide timing, test control, and the internal processor status.

5.9.1 Processor Status (PST3–PST0)

These outputs indicate the internal execution unit's status. The timing is synchronous with BCLK, and the status may have nothing to do with the current bus transfer. The PSTx signal is updated depending on the type of PSTx encoding. There are two classes of PSTx encodings. The first class is associated with instruction boundaries, and the second class indicates the processor's present status. Table 5-6 lists the definition of the encodings.

The encodings 0, 8, 4, 5, C, D, E, and F indicate the present status and do not reflect a specific stage of the pipe. These encodings persist as long as the processor stays in the indicated state. The default encoding 0 (user) or 8 (supervisor) is indicated if none of the above conditions apply. The encodings 1, 2, 3, 9, A, and B belong to the first class of PSTx encoding. This class indicates that the instruction is in its last instruction execution stage. These encodings exist for only one BCLK period per instruction and are mutually exclusive.

Table 5-6. Processor Status Encoding

Hex	PST3	PST2	PST1	PST0	Internal Status
0	0	0	0	0	User, Start/Continue Current Instruction
1	0	0	0	1	User, End Current Instruction
2	0	0	1	0	User, Branch Not Taken/End Current Instruction
3	0	0	1	1	User, Branch Taken/End Current Instruction
4	0	1	0	0	User, Table Search
5	0	1	0	1	Halted State (Double Bus Fault)
6	0	1	1	0	Low-Power Stop Mode (Supervisor Instruction)*
7	0	1	1	1	Reserved
8	1	0	0	0	Supervisor, Start/Continue Current Instruction
9	1	0	0	1	Supervisor, End Current Instruction
A	1	0	1	0	Supervisor, Branch Not Taken/End Current Instruction
B	1	0	1	1	Supervisor, Branch Taken/End Current Instruction
C	1	1	0	0	Supervisor, Table Search
D	1	1	0	1	Stopped State (Supervisor Instruction)
E	1	1	1	0	RTE Executing
F	1	1	1	1	Exception Stacking

NOTE: *MC68040V and MC68EC040V only.

When a ‘branch taken/end current instruction’ is indicated, it means that a change of instruction flow is pending. Along with the following instructions, an exception stacking (encoding F) sequence is ended with the ‘supervisor, branch taken/end current instruction’ encoding as though it were a virtual JMP instruction. This includes all the possible exceptions listed in the processor’s vector table. Instructions that cause a ‘branch taken/end current instruction’ encoding when they are executed are as follows:

ANDI to SR	DBcc (Taken)	MOVE to SR	RTD
Bcc (Taken)	FBcc (Taken)	MOVE USP	RTE
BRA	FDBcc (Always)	MOVEC	RTR
BSR	FMOVEM Rc,MRn	MOVES	RTS
CAS	FMOVEM FPm,MRn	NOP	STOP
CAS2	FSAVE	ORI to SR	TAS
CINV	JMP	PFLUSH	
CPUSH	JSR	PTEST	

The Bcc (not taken) and DBcc (not taken) are the only instructions that cause a ‘branch not taken/end current instruction’ encoding. Note that the FBcc (not taken) is not included in this category. The FBcc (not taken) instruction ends with an ‘end current instruction’ encoding. All other instructions and conditions end with the ‘end current instruction’ encoding. For instance, if the processor is running back-to-back single clock instructions, the encoding ‘end current instruction’ remains asserted for as many clock cycles as instructions.

The following examples are for PSTx encodings:

1. An access error terminates an instruction such that the instruction execution stage is not reached. In this case, an ‘end current instruction’ is not indicated. Exception processing starts, the exception stacking status is indicated, and then the virtual JMP causes the ‘supervisor, branch taken/end current instruction’ encoding.
2. An FTRAPcc that does not take an exception ending with the ‘end current instruction’ encoding. The exception stacking status is indicated and then reaches the ‘supervisor, branch taken/end current instruction’ encoding if the FTRAPcc ends in an exception.
3. Two simultaneous interrupt exception processing sequences follow an ADD instruction. The ADD instruction ends with ‘end current instruction’, followed by exception stacking, followed by ‘branch taken/end current instruction’, followed by exception stacking, followed by ‘branch taken/end current instruction’.
4. An RTE instruction follows an ADD instruction. The ‘end current instruction’ is followed by RTE executing followed by a branch taken/end current instruction.

5.9.2 Bus Clock (BCLK)

This input signal is used as a reference for all bus timing. It is a TTL-compatible signal and cannot be gated off. Refer to **Section 11 MC68040 Electrical and Thermal Characteristics** for electrical specifications.

5.9.3 Processor Clock (PCLK)—Not on MC68040V and MC68EC040V

PCLK is used to derive all internal timing. This clock is also TTL compatible and cannot be gated off. Refer to **Section 11 MC68040 Electrical and Thermal Characteristics** for electrical specifications.

5.10 MMU DISABLE ( )—NOT ON MC68EC040

The MMU disable signal dynamically disables the translation of addresses by the MMUs. The assertion of MDIS does not flush the address translation caches (ATCs); ATC entries become available again when MDIS is negated. During a processor reset, the level on MDIS is latched and used to select the normal data latch mode (MDIS high) or DLE mode (MDIS low). Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for a description of address translation and to **Section 7 Bus Operation** for information about DLE mode.

5.11 DATA LATCH ENABLE (DLE)—ONLY ON MC68040

This input signal is used in DLE mode to latch the input data bus on read transfers. DLE mode can be used to support asynchronous memory interfaces by allowing the interface to specify when data should be latched instead of requiring data to be valid on the rising edge of BCLK.

5.12 TEST SIGNALS

The M68040 includes dedicated user-accessible test logic that is fully compatible with the IEEE 1149.1 *Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to the development of this standard under the IEEE Test Technology Committee and Joint Test Action Group (JTAG) sponsorship. The M68040 implementation supports circuit board test strategies based on this standard. However, the JTAG interface is not intended to provide an in-circuit test to verify M68040 operations; therefore, it is impossible to test M68040 operations using this interface. **Section 6 IEEE 1149.1 Test Access Port (JTAG)** describes the M68040 implementation of the IEEE 1149.1 and is intended to be used with the supporting IEEE document.

5.12.1 Test Clock (TCK)

This input signal is used as a dedicated clock for the test logic. Since clocking of the test logic is independent of the normal operation of the MC68040, several other components on a board can share a common test clock with the processor even though each component may operate from a different system clock. The design of the test logic allows the test clock to run at low frequencies, or to be gated off entirely as required for test purposes.

5.12.2 Test Mode Select (TMS)

This input signal is decoded by the TAP controller and distinguishes the principle operationas of the test support circuitry.

5.12.3 Test Data In (TDI)

This input signal provides a serial data input to the TAP.

5.12.4 Test Data Out (TDO)

This three-state output signal provides a serial data output from the TAP. The TDO output can be placed in a high-impedance mode to allow parallel connection of board-level test data paths.

5.12.5 Test Reset ( )—Not on MC68040V and MC68EC040V

This input signal provides an asynchronous reset of the TAP controller.

5.13 POWER SUPPLY CONNECTIONS

The M68040 requires connection to a V_{CC} power supply, positive with respect to ground. The V_{CC} and ground connections are grouped to supply adequate current to the various sections of the processor. **Section 12 Ordering Information and Mechanical Data** describes the groupings of V_{CC} and ground connections.

5.14 SIGNAL SUMMARY

Table 5-7 provides a summary of the electrical characteristics of the signals discussed in this section.

Table 5-7. Signal Summary

Signal Name	Mnemonic	Type	Active	Three-State
Address Bus	A31–A0	Input/Output	High	Yes
Autovector	AVEC	Input	Low	—
Bus Busy	BB	Input/Output	Low	Yes
Bus Clock	BCLK	Input	—	—
Bus Grant	BG	Input	Low	—
Bus Request	BR	Output	Low	No
Cache Disable	CDIS	Input	Low	—
Cache Inhibit Out	CIOUT	Output	Low	Yes
Data Bus	D31–D0	Input/Output	High	Yes
Data Latch Enable[1]	DLE	Input	High	—
Ground	GND	Ground	—	—
Interrupt Pending	IPEND	Output	Low	No
Interrupt Priority Level[2]	IPL2–IPL0	Input	Low	—
Bus Lock	LOCK	Output	Low	Yes
Bus Lock End	LOCKE	Output	Low	Yes
Memory Inhibit	MI	Output	Low	No
MMU Disable[3]	MDIS	Input	Low	—
Processor Clock	PCLK	Input	—	—
Processor Status	PST3–PST0	Output	High	No
Read/Write	R/W	Input/Output	High/Low	Yes
Reset In	RSTI	Input	Low	—
Reset Out	RSTO	Output	Low	No
Snoop Control	SC1, SC0	Input	High	—
Transfer Acknowledge	TA	Input/Output	Low	Yes
Transfer Burst Inhibit	TBI	Input	Low	—
Transfer Cache Inhibit	TCI	Input	Low	—
Transfer Error Acknowledge	TEA	Input	Low	—
Transfer in Progress	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

Table 5-7 Signal Summary (Continued)

Signal Name	Mnemonic	Type	Active	Three-State
Transfer Start	TS	Input/Output	Low	Yes
Transfer Type	TT1, TT0	Input/Output	High	Yes
Test Clock	TCK	Input	—	—
Test Data Input	TDI	Input	High	—
Test Data Output	TDO	Output	High	Yes
Test Mode Select	TMS	Input	High	—
Test Reset	TRST	Input	Low	—
User-Programmable Attributes	UPA1, UPA0	Output	High	Yes
Power Supply	V_{CC}	Power	—	—

NOTES:
1. This signal is not available on the MC68LC040 and MC68EC040.
2. These signals are different on power-up for the MC68LC040 and MC68EC040.
3. This signal is not available on the MC68EC040.

SECTION 6
IEEE 1149.1A TEST ACCESS PORT (JTAG)

NOTE

This section does not apply to the MC68040V and MC68EC040V. Refer to **Appendix C MC68040V and MC68EC040** for details. All references to M68040 in this section only, refer to the MC68040, MC68LC040, and MC68EC040.

The M68040 includes dedicated user-accessible test logic that is fully compatible with the IEEE standard 1149.1A *Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to the standard's development under the sponsorship of the IEEE Test Technology Committee and the Joint Test Action Group (JTAG).

This section is to be used in conjunction with the supporting IEEE document and includes those chip-specific items that the IEEE standard requires to be defined and additional information specific to the M68040 implementation. For example, the IEEE standard 1149.1A test access port (TAP) controller states are referenced in this section but are not described. For these details and application information regarding the standard, refer to the IEEE standard 1149.1A document.

The M68040 implementation supports circuit board test strategies based on the standard. The test logic utilizes static logic design and is system logic independent of the device. The M68040 implementation provides capabilities to:

a. Perform boundary scan operations to test circuit board electrical continuity,
b. Bypass the M68040 by reducing the shift register path to a single cell,
c. Sample the M68040 system pins during operation and transparently shift out the result,
d. Disable the output drive to output-only pins during circuit board testing, and
e. Select one of two output drivers on a pin-by-pin basis.

NOTE

The IEEE standard 1149.1A test logic cannot be considered completely benign to those planning not to use this capability. Certain precautions must be observed to ensure that this logic does not interfere with system operation. Refer to **6.5 Disabling the IEEE Standard 1149.1A Operation**.

6.1 OVERVIEW

Figure 6-1 illustrates a block diagram of the M68040 implementation of IEEE standard 1149.1A. The test logic includes a 16-state dedicated TAP controller. These 16 controller states are defined in detail in the IEEE standard 1149.1A, but only 8 are included in this section.

Test-Logic-Reset	Run-Test/Idle
Capture-IR	Capture-DR
Update-IR	Update-DR
Shift-IR	Shift-DR

The TAP controller provides access to five dedicated signal pins:

TCK—A test clock input that synchronizes the test logic.

TMS—A test mode select input with an internal pullup resistor sampled on the rising edge of TCK to sequence the TAP controller.

TDI—A test data input with an internal pullup resistor sampled on the rising edge of TCK.

TDO—A three-state test data output actively driven only in the shift-IR and shift-DR controller states that changes on the falling edge of TCK.

$\overline{\text{TRST}}$—An active-low asynchronous reset with an internal pullup resistor that forces the TAP controller into the test-logic-reset state.

The test logic also includes an instruction shift register and two test data registers, a boundary scan register and a bypass register. The boundary scan register links all device signal pins into the instruction shift register.

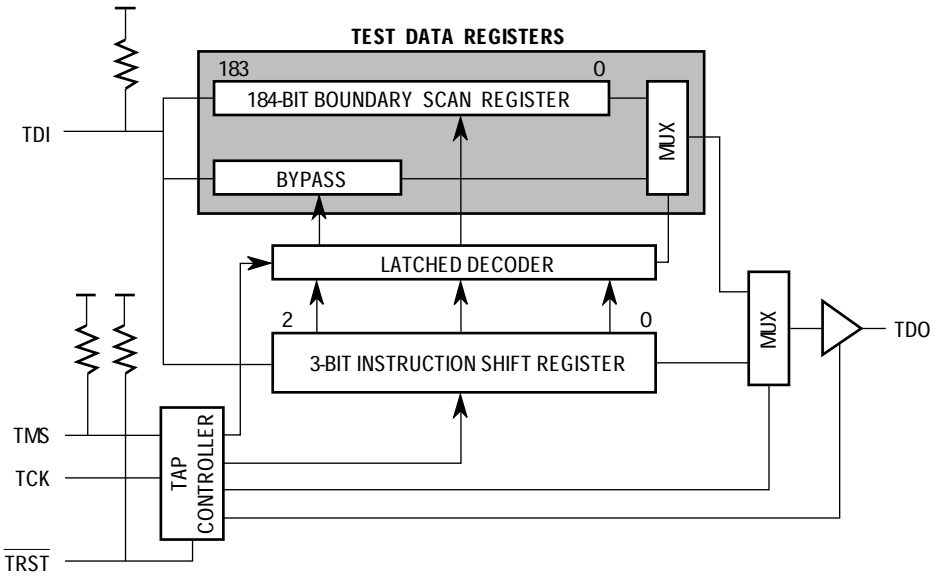


Figure 6-1. M68040 Test Logic Block Diagram

6.2 INSTRUCTION SHIFT REGISTER

The M68040 IEEE standard 1149.1A implementation includes a 3-bit instruction shift register without parity. The register shifts one of eight instructions, which can either select the test to be performed or access a test data register, or both. Data is transferred from the instruction shift register to latched decoded outputs during the update-IR state. The instruction shift register is reset to all ones in the TAP controller test-logic-reset state, which is equivalent to selecting the BYPASS instruction. During the capture-IR state, the binary value 001 is loaded into the parallel inputs of the instruction shift register.

The M68040 IEEE standard 1149.1A implementation includes three mandatory public instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) and four manufacturer's public instructions. The four manufacturer's public instructions provide the capability to disable all device output drivers, operate the device in a BYPASS configuration without a system clocking requirement, and select one of two output drive capabilities on a pin-by-pin basis. The M68040 implementation does not support the optional standard public instructions. Table 6-1 lists the three bits used in the instruction shift register to decode the instructions and their related encodings. Note that the least significant bit of the instruction (bit 0) is the first bit to be shifted into the instruction shift register.

Table 6-1. IEEE Standard 1149.1A Instructions

Bit 2	Bit 1	Bit 0	Instruction Selected	Test Data Register Accessed
0	0	0	EXTEST	Boundary Scan
0	0	1	HIGHZ	Bypass
0	1	0	SAMPLE/PRELOAD	Boundary Scan
0	1	1	DRVCTL.T	Boundary Scan
1	0	0	SHUTDOWN	Bypass
1	0	1	PRIVATE	Bypass
1	1	0	DRVCTL.S	Boundary Scan
1	1	1	BYPASS	Bypass

EXTEST, HIGHZ, DRVCTL.T, SHUTDOWN, and PRIVATE have a PCLK and BCLK restriction. Failure to comply with this restriction results in potential internal damage to the device (see **6.4 Restrictions**). Once the restriction is complied with, SHUTDOWN, EXTEST, HIGHZ, and DRVCTL.T can be entered regardless of order. The system clocks (PCLK and BCLK) must be kept running while in the SAMPLE/PRELOAD, DRVCLT.S, and BYPASS instructions. Failure to do so could result in potential internal damage to the device.

6.2.1 EXTEST

The external test instruction (EXTEST) selects the 184-bit boundary scan register. This instruction also activates two internal functions that are intended to protect the device from potential damage while performing boundary scan operations.

EXTEST asserts internal reset for the M68040 system logic to force a predictable benign internal state and activates an internal keep-alive clock to protect the device from potential internal damage. This internal clock eliminates the requirement to keep the system clocks (PCLK and BCLK) running during EXTEST operations and allows these two system clock pins to be included in boundary scan testing.

6.2.2 HIGHZ

The HIGHZ instruction is an optional instruction provided as a Motorola public instruction to anticipate the need to backdrive output pins during circuit board testing. The HIGHZ instruction activates an internal keep-alive clock, asserts internal system reset, selects the bypass register, and forces all output and bidirectional pins to the high-impedance state.

Asserting TRST or holding TMS high and clocking TCK for at least five rising edges causes the TAP controller to enter the test-logic-reset state. Using only the TMS and TCK pins and the capture-IR and update-IR states invokes the HIGHZ instruction. This scheme works because the value captured by the instruction shift register during the capture-IR state is identical to the HIGHZ opcode.

6.2.3 SAMPLE/PRELOAD

The SAMPLE/PRELOAD instruction provides two separate functions. First, it provides a means to obtain a sample system data and control signal. Sampling occurs on the rising edge of TCK in the capture-DR state. The user can observe the data by shifting it through the boundary scan register to output TDO using the shift-DR state. Both the data capture and the shift operations are transparent to system operation. The user must provide some form of external synchronization to achieve meaningful results since there is no internal synchronization between TCK and BCLK.

The second function of the SAMPLE/PRELOAD instruction is to initialize the boundary scan register output cells before selecting EXTEST, which is accomplished by ignoring data being shifted out of TDO while shifting in initialization data. The update-DR state can then be used to initialize the boundary scan register and ensure that known data and output state will occur on the outputs after entering the EXTEST instruction.

6.2.4 DRVCTL.T

The DRVCTL.T instruction is a Motorola public instruction that provides the ability to select one of two output drivers on a pin-by-pin basis. It is intended for use with EXTEST or SHUTDOWN to provide an IEEE-compatible environment to select the output drivers for board-level test environments. This instruction allows data in the boundary scan register to select the output driver. A logic zero in the appropriate boundary scan output cell (see Table 6-1) selects the large buffer, and a logic one selects the small buffer (see **Section 7 Bus Operation**). Data captured in the capture-DR state for this instruction is identical to that captured during EXTEST: output data cells for outputs and pin state for inputs. Note that no data relevant to the drive control function is captured during the capture-DR state.

The DRVCTL.T instruction is intended to be used in test applications in conjunction with the EXTEST and SHUTDOWN instructions and not for system applications. It therefore differs from DRVCTL.S in that this instruction invokes the keep-alive clock, asserts the internal reset, and the test logic, not the system logic, has control of the I/O pins.

When the system logic has control of signal pin I/O directions and levels, the drive control latch is loaded from the IPL2–IPL0 pins during the negation of RSTI. DRVCTL.T overwrites this value with boundary scan data in the update-DR state. The selected output driver state remains unchanged if only the DRVCTL.T, EXTEST, or SHUTDOWN instructions are invoked. If an instruction other than one of these three is executed, the system logic protocol regains control of the output driver state and overwrites the value that the DRVCTL.T instruction previously defined.

Note that the output drive control state does not change while the 1149.1A instruction is one of the three instructions DRVCTL.T, EXTEST, or SHUTDOWN. If DRVCTL.T changes the output driver state and then the test-logic-reset state is entered, the instruction shift register is reset to BYPASS, and the system logic can change the output driver state.

6.2.5 SHUTDOWN

This instruction provides an opcode for automatic test pattern generation (ATPG) programs to cope with the clocking protocol required to stop the system clocks. This instruction asserts internal system reset, activates an internal keep alive clock, and selects the bypass register. Internal decoding of the instruction selects the bypass register, and the test logic, not the system logic, has control of the I/O ports. Note that initializing the boundary scan data register and then selecting the SHUTDOWN instruction provides a clamping function. The test logic controls the I/O state, and the bypass register is selected.

6.2.6 PRIVATE

Motorola reserves this instruction for manufacturing use. The instruction does not change pin I/O as defined for system operation.

6.2.7 DRVCTL.S

The DRVCTL.S instruction controls the output driver selection on a pin-by-pin basis. This instruction allows data in the boundary scan register to select the output driver during the update-DR state when the system logic has control of the signal I/O directions and levels. A logic zero selects the large buffer or driver; a logic one selects the small buffer or driver (see Table 6-1).

The DRVCTL.S instruction is intended to be used in system applications and not in test applications. In system applications, the system logic has control of the signal pin I/O directions and levels; whereas, in test applications, the 1149.1A test logic has control of it. It therefore differs from DRVCTL.T in that this instruction does not invoke the internal keep alive clock, it does not assert the internal reset, and the system logic, not the test logic,

has control of the I/O pins. The 1149.1A interface is transparent to system operation except for drive control selection during execution of this instruction.

When the system logic has control of the signal I/O directions and levels, the drive control latches are loaded from the IPL2–IPL0 pins at the negation of the RSTI signal. After RSTI has been negated, and the 128-clock internal reset cycle has expired (see **Section 7 Bus Operation**), the DRVCTL.S instruction is executed. Each drive control latch is modified during the update-DR state. Any subsequent RSTI signal negation while in a system configuration (i.e., system logic has control of the signal I/O directions and levels) can cause the drive control latches to be overwritten with new IPL- signal values. The system bus can be suspended in a wait state while this function is being performed.

6.2.8 BYPASS

The BYPASS instruction selects the single-bit bypass register, creating a single-bit shift-register path from TDI to the bypass register to TDO. The instruction enhances test efficiency when a component other than the M68040 becomes the device under test. When the bypass register is initially selected, the instruction shift register stage is set to a logic zero on the rising edge of TCK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the bypass register is always a logic zero. Figure 6-2 illustrates the bypass register.

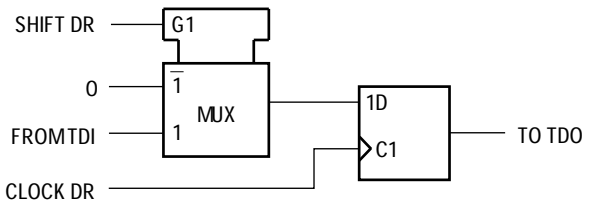


Figure 6-2. Bypass Register

6.3 BOUNDARY SCAN REGISTER

The 184-bit boundary scan register uses the TAP controller to scan user-defined values into the output buffers, capture values presented to input pins, and control the direction of bidirectional pins. The instruction shift register cell nearest TDO (i.e., first to be shifted out) is defined as bit zero. The last bit to be shifted out is bit 183. This register includes cells for all device signal pins and clock pins along with associated control signals.

The M68040 boundary scan register consists of three cell structure types, O.Latch, I.Pin, and IO.Ctl, that are associated with a boundary scan register bit. All boundary scan output cells capture the logic level of the device output latch during the capture-DR state. Figures 6-3 through 6-5 illustrate these three cell types. Figure 6-6 illustrates the general arrangement of these cells.

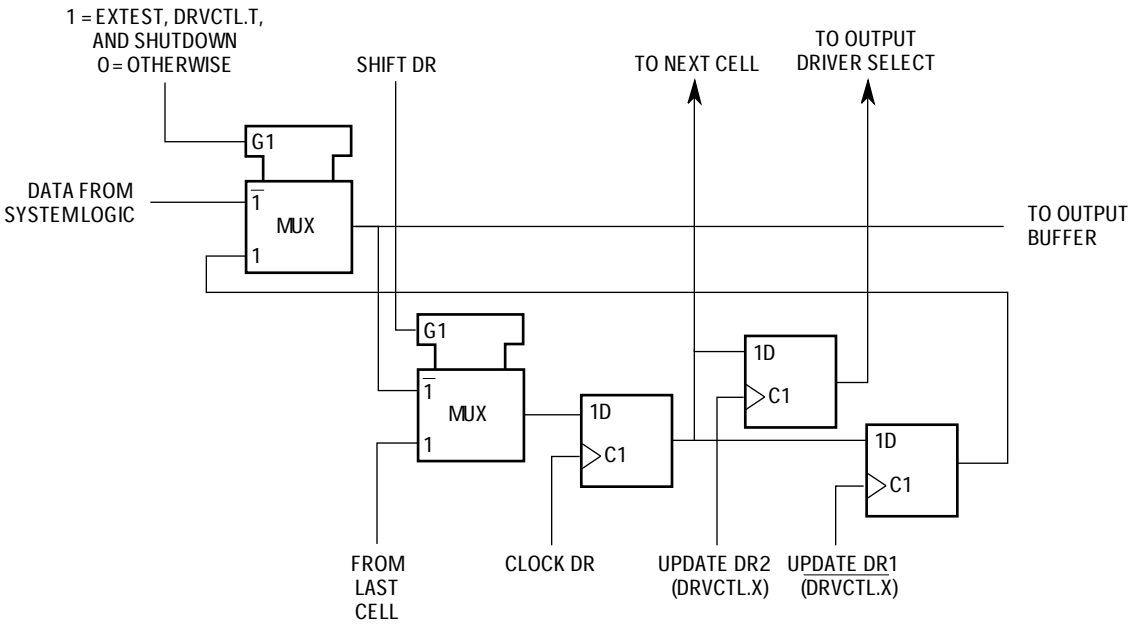


Figure 6-3. Output Latch Cell (O.Latch)

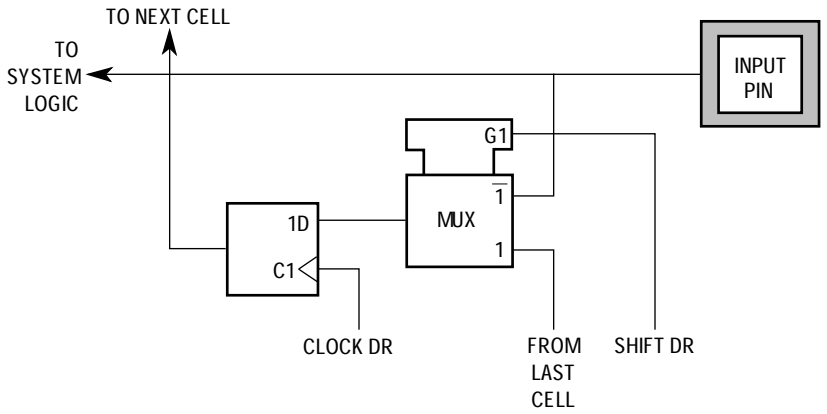


Figure 6-4. Input Pin Cell (I.Pin)

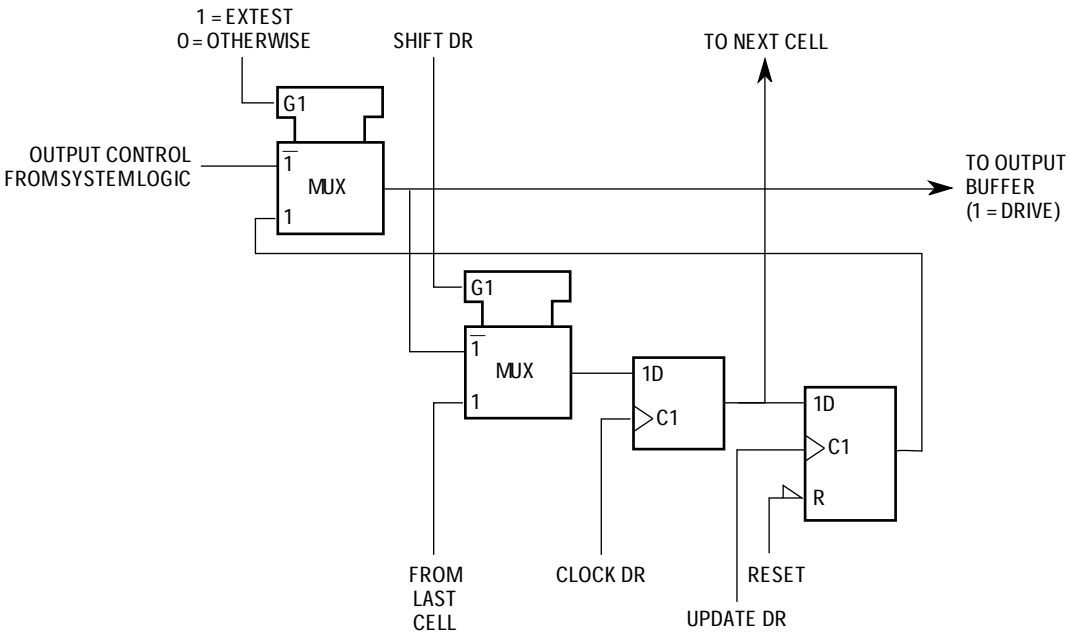


Figure 6-5. Output Control Cells (IO.Ctl)

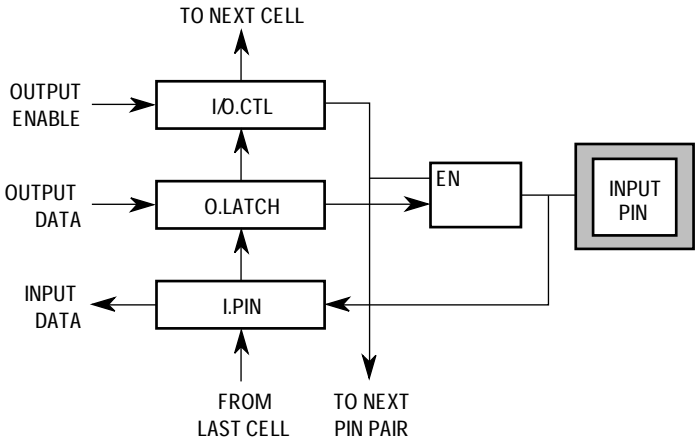


Figure 6-6. General Arrangement of Bidirectional Pins

All M68040 bidirectional pins include two boundary scan data cells, an input, and an output. One of five associated boundary scan control cells controls each bidirectional pin. If these cells contain a logic one, the associated bidirectional or three-state pin will be configured as an output and enabled. The cell captures the current value during the capture-DR state. All five control cells are reset (i.e., logic zero) in the test-logic-reset state. The five bidirectional/three-state control cells and their boundary scan register bit positions are as follows:

Cell Name	Bit
io.ab	150
io.db	151
io.2	154
io.1	155
io.0	156

Table 6-2 lists the 184 boundary scan bit definitions. The first column in the table defines the bit position in the boundary scan register. The second column references one of the three cell types. The third column lists the pin name for all pin-related cells. The fourth column lists the system pin type for convenience where TS-Output indicates a three-state output pin and I/O indicates a bidirectional pin. The last column lists the name of the associated control bit of the boundary scan register for three-state output and bidirectional pins. The boundary scan description language (BSDL) type for each cell can be found in note 1.

Table 6-2. Boundary Scan Bit Definitions[1]

Bit	Cell Type	Pin/Cell Name	Pin Type	Output Ctrl Cell
0	O.Latch	RSTO	Output[2]	(Note 3)
1	O.Latch	IPEND	Output[2]	(Note 3)
2	O.Latch	CIOUT	TS-Output[2]	io.0
3	O.Latch	UPA0	TS-Output[2]	io.0
4	O.Latch	UPA1	TS-Output[2]	io.0
5	O.Latch	TT0	I/O[2]	io.0
6	I.Pin	TT0	I/O	io.0
7	O.Latch	TT1	I/O[2]	io.0
8	I.Pin	TT1	I/O	io.0
9	O.Latch	A10	I/O[2]	io.ab
10	I.Pin	A10	I/O	io.ab
11	O.Latch	A11	I/O[2]	io.ab
12	I.Pin	A11	I/O	io.ab
13	O.Latch	A12	I/O[2]	io.ab
14	I.Pin	A12	I/O	io.ab
15	O.Latch	A13	I/O[2]	io.ab
16	I.Pin	A13	I/O	io.ab
17	O.Latch	A14	I/O[2]	io.ab
18	I.Pin	A14	I/O	io.ab
19	O.Latch	A15	I/O[2]	io.ab
20	I.Pin	A15	I/O	io.ab
21	O.Latch	A16	I/O[2]	io.ab
22	I.Pin	A16	I/O	io.ab
23	O.Latch	A17	I/O[2]	io.ab
24	I.Pin	A17	I/O	io.ab
25	O.Latch	A18	I/O[2]	io.ab
26	I.Pin	A18	I/O	io.ab
27	O.Latch	A19	I/O[2]	io.ab
28	I.Pin	A19	I/O	io.ab
29	O.Latch	A20	I/O[2]	io.ab
30	I.Pin	A20	I/O	io.ab
31	O.Latch	A21	I/O[2]	io.ab
32	I.Pin	A21	I/O	io.ab
33	O.Latch	A22	I/O[2]	io.ab
34	I.Pin	A22	I/O	io.ab
35	O.Latch	A23	I/O[2]	io.ab
36	I.Pin	A23	I/O	io.ab
37	O.Latch	A24	I/O[2]	io.ab
38	I.Pin	A24	I/O	io.ab
39	O.Latch	A25	I/O[2]	io.ab
40	I.Pin	A25	I/O	io.ab
41	O.Latch	A26	I/O[2]	io.ab
42	I.Pin	A26	I/O	io.ab
43	O.Latch	A27	I/O[2]	io.ab
44	I.Pin	A27	I/O	io.ab
45	O.Latch	A28	I/O[2]	io.ab
46	I.Pin	A28	I/O	io.ab
47	O.Latch	A29	I/O[2]	io.ab
48	I.Pin	A29	I/O	io.ab
49	O.Latch	A30	I/O[2]	io.ab
50	I.Pin	A30	I/O	io.ab
51	O.Latch	A31	I/O[2]	io.ab
52	I.Pin	A31	I/O	io.ab
53	O.Latch	D0	I/O[2]	io.db
54	O.Latch	D1	I/O[2]	io.db
55	O.Latch	D2	I/O[2]	io.db
56	O.Latch	D3	I/O[2]	io.db
57	O.Latch	D4	I/O[2]	io.db
58	O.Latch	D5	I/O[2]	io.db
59	O.Latch	D6	I/O[2]	io.db
60	O.Latch	D7	I/O[2]	io.db
61	O.Latch	D8	I/O[2]	io.db
62	O.Latch	D9	I/O[2]	io.db
63	O.Latch	D10	I/O[2]	io.db
64	O.Latch	D11	I/O[2]	io.db
65	O.Latch	D12	I/O[2]	io.db
66	O.Latch	D13	I/O[2]	io.db
67	O.Latch	D14	I/O[2]	io.db
68	O.Latch	D15	I/O[2]	io.db
69	O.Latch	D16	I/O[2]	io.db
70	O.Latch	D17	I/O[2]	io.db
71	O.Latch	D18	I/O[2]	io.db
72	O.Latch	D19	I/O[2]	io.db
73	O.Latch	D20	I/O[2]	io.db

Table 6-2. Boundary Scan Bit Definitions (Continued)

Bit	Cell Type	Pin/Cell Name	Pin Type	Output Ctrl Cell
74	O.Latch	D21	I/O[2]	io.db
75	O.Latch	D22	I/O[2]	io.db
76	O.Latch	D23	I/O[2]	io.db
77	O.Latch	D24	I/O[2]	io.db
78	O.Latch	D25	I/O[2]	io.db
79	O.Latch	D26	I/O[2]	io.db
80	O.Latch	D27	I/O[2]	io.db
81	O.Latch	D28	I/O[2]	io.db
82	O.Latch	D29	I/O[2]	io.db
83	O.Latch	D30	I/O[2]	io.db
84	O.Latch	D31	I/O[2]	io.db
85	I.Pin	D0	I/O	io.db
86	I.Pin	D1	I/O	io.db
87	I.Pin	D2	I/O	io.db
88	I.Pin	D3	I/O	io.db
89	I.Pin	D4	I/O	io.db
90	I.Pin	D5	I/O	io.db
91	I.Pin	D6	I/O	io.db
92	I.Pin	D7	I/O	io.db
93	I.Pin	D8	I/O	io.db
94	I.Pin	D9	I/O	io.db
95	I.Pin	D10	I/O	io.db
96	I.Pin	D11	I/O	io.db
97	I.Pin	D12	I/O	io.db
98	I.Pin	D13	I/O	io.db
99	I.Pin	D14	I/O	io.db
100	I.Pin	D15	I/O	io.db
101	I.Pin	D16	I/O	io.db
102	I.Pin	D17	I/O	io.db
103	I.Pin	D18	I/O	io.db
104	I.Pin	D19	I/O	io.db
105	I.Pin	D20	I/O	io.db
106	I.Pin	D21	I/O	io.db
107	I.Pin	D22	I/O	io.db
108	I.Pin	D23	I/O	io.db
109	I.Pin	D24	I/O	io.db
110	I.Pin	D25	I/O	io.db
111	I.Pin	D26	I/O	io.db
112	I.Pin	D27	I/O	io.db
113	I.Pin	D28	I/O	io.db
114	I.Pin	D29	I/O	io.db
115	I.Pin	D30	I/O	io.db
116	I.Pin	D31	I/O	io.db
117	O.Latch	A9	I/O[2]	io.ab
118	I.Pin	A9	I/O	io.ab
119	O.Latch	A8	I/O[2]	io.ab
120	I.Pin	A8	I/O	io.ab
121	O.Latch	A7	I/O[2]	io.ab
122	I.Pin	A7	I/O	io.ab
123	O.Latch	A6	I/O[2]	io.ab
124	I.Pin	A6	I/O	io.ab
125	O.Latch	A5	I/O[2]	io.ab
126	I.Pin	A5	I/O	io.ab
127	O.Latch	A4	I/O[2]	io.ab
128	I.Pin	A4	I/O	io.ab
129	O.Latch	A3	I/O[2]	io.ab
130	I.Pin	A3	I/O	io.ab
131	O.Latch	A2	I/O[2]	io.ab
132	I.Pin	A2	I/O	io.ab
133	O.Latch	A1	I/O[2]	io.ab
134	I.Pin	A1	I/O	io.ab
135	O.Latch	A0	I/O[2]	io.ab
136	I.Pin	A0	I/O	io.ab
137	O.Latch	TM2	TS-Output[2]	io.0
138	O.Latch	TM1	TS-Output[2]	io.0
139	O.Latch	TM0	TS-Output[2]	io.0
140	O.Latch	TLN1	TS-Output[2]	io.0
141	O.Latch	TLN0	TS-Output[2]	io.0
142	O.Latch	SIZ0	I/O[2]	io.0
143	I.Pin	SIZ0	I/O	io.0
144	O.Latch	R/W	I/O[2]	io.0
145	I.Pin	R/W	I/O	io.0
146	O.Latch	LOCKE	TS-Output[2]	io.1
147	O.Latch	SIZ1	I/O[2]	io.0

Table 6-2. Boundary Scan Bit Definitions (Concluded)

Bit	Cell Type	Pin/Cell Name	Pin Type	Output Ctrl Cell
148	I.Pin	SIZ1	I/O	io.0
149	O.Latch	LOCK	TS-Output[2]	io.1
150	IO.Ctl	io.ab	—	(Note 4)
151	IO.Ctl	io.db	—	(Note 4)
152	O.Latch	MI	Output[2]	(Note 3)
153	O.Latch	BR	Output[2]	(Note 3)
154	IO.Ctl	io.2	—	(Note 4)
155	IO.Ctl	io.1	—	(Note 4)
156	IO.Ctl	io.0	—	(Note 4)
157	O.Latch	TS	I/O[2]	io.0
158	I.Pin	TS	I/O	io.0
159	O.Latch	BB	I/O[2]	io.1
160	I.Pin	BB	I/O	io.1
161	O.Latch	TIP	TS-Output[2]	io.1
162	O.Latch	PST3	Output[2]	(Note 3)
163	O.Latch	PST2	Output[2]	(Note 3)
164	O.Latch	PST1	Output[2]	(Note 3)
165	O.Latch	PST0	Output[2]	(Note 3)
166	O.Latch	TA	I/O[2]	io.2
167	I.Pin	TA	I/O	io.2
168	I.Pin	TEA	Input	—
169	I.Pin	BG	Input	—
170	I.Pin	SC1	Input	—
171	I.Pin	SC0	Input	—
172	I.Pin	TBI	Input	—
173	I.Pin	AVEC	Input	—
174	I.Pin	TCI	Input	—
175	I.Pin	DLE[5]	Input	—
176	I.Pin	PCLK	Input	—
177	I.Pin	BCLK	Input	—
178	I.Pin	IPL0	Input	—
179	I.Pin	IPL1	Input	—
180	I.Pin	IPL2	Input	—
181	I.Pin	RSTI	Input	—
182	I.Pin	CDIS	Input	—
183	I.Pin	MDIS[6]	Input	—

NOTES:
1. I.Pin, IO.Ctl, and O.Latch are equivalent to the BSDL descriptions: BC_4, BC_2, and BC_2, respectively.
2. Boundary scan register bit positions that are used during the drive control (DRVCTL.X) instructions.
3. These output-only cells can be turned off (high impedance) by using the HIGHZ instruction.
4. All of the control signals (IO.Ctl) are cleared in the test-logic-reset state.
5. Renamed JS0 on the MC68LC040 and MC68EC040.
6. Renamed JS1 on the MC68EC040.

6.4 RESTRICTIONS

The test logic is implemented using static logic design, and TCK can be stopped in either a high or low state without loss of data. The system logic, however, includes considerable dynamic logic. For this reason, the system clocks (PCLK and BCLK) cannot be stopped or allowed to run slower than the specified frequency except when the EXTEST, HIGHZ, DRVCTL.T, or SHUTDOWN instructions have been properly invoked.

PCLK and BCLK must be kept running for two additional BCLK periods upon initial entry into any of the four instructions, EXTEST, HIGHZ, DRVCTL.T, or SHUTDOWN. This restriction is necessary to allow time for an internal reset to propagate through an internal synchronizer. After this period, the user has complete time-domain freedom with the two system clock pins. After any of the four instructions has been properly entered, these instructions can be executed in any order without a time-domain clocking restriction. Entering any instruction other than one of these four requires that the system clocks be

restarted, and a proper reentry into any of the four instructions is again required before the system clocks can be stopped.

Control over the output enable signals using the boundary scan register and the EXTEST and HIGHZ instructions requires a compatible circuit-board test environment to avoid destructive configurations. The user is responsible for avoiding situations in which the M68040 output drivers are enabled into actively driven networks.

The TRST signal provides the ability for an asynchronous reset of the test logic and requires no internal clocking to force the TAP controller into the test-logic-reset state. This signal should be asserted during system power-up to initialize the 1149.1A test interface and avoid the potential for board-level bus conflicts. Essentially the TRST signal provides the ability to prevent possible board-level bus contention during power-up due to the test logic having control of the pins. The device has no internal power-up reset circuit. The TRST signal should be treated similar to the RSTI signal for board design considerations concerning power-up conditions.

Negation of the TRST signal requires certain precautions to achieve a predictable TAP controller state. The TMS signal is sampled on the rising edge of TCK and sequences the TAP controller. If TMS is low and TRST is negated simultaneously with the rising edge of TCK, the resultant TAP controller state is unpredictable but will be either test-logic-reset or run-test/idle. To avoid this uncertainty, either 1) the negation of TRST can be synchronized with the falling edge of TCK or 2) TMS can remain high until after TRST negation. Alternatively, holding TMS low for two or more TCK periods following TRST negation ensures that the TAP controller is in the run-test/idle state.

6.5 DISABLING THE IEEE STANDARD 1149.1A OPERATION

There are two considerations for non-IEEE standard 1149.1A operation. First, TCK does not include an internal pullup resistor and should not be left unconnected to preclude mid-level inputs. The second consideration is to ensure that the IEEE standard 1149.1A test logic remains transparent to the system logic by providing the ability to force the test-logic-reset state.

Figure 6-7 illustrates disabling the IEEE standard 1149.1A operation through connecting TRST directly or through a resistor to ground or a suitable logic network. Connecting TRST to RSTI while TCK is held either high or low meets the two considerations. If a pulse asserts TRST, the TAP controller is forced into the test-logic-reset state and can remain in this state as long as a rising edge on the TCK signal does not occur when TMS is low.

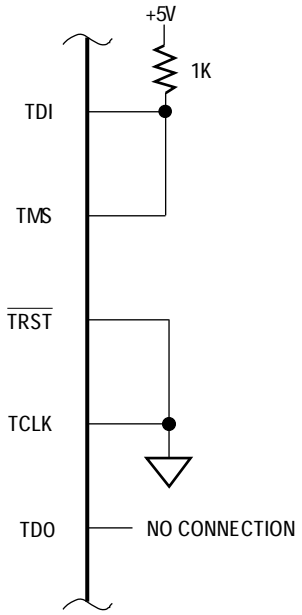


Figure 6-7. Circuit Disabling IEEE Standard 1149.1A

6.6 MOTOROLA M68040 BSDL DESCRIPTION (VERSION 2.2)

Revision List:

1. LOCK and LOCKE controlled by io.1 vice io.0 (4D98D).
3. No other changes to Version 2.1 BSDL.
2. Instruction opcodes changed for SAMPLE, SHUTDOWN, and BYPASS.
3. New instructions DRVCTL.T, DRVCTL.S and PRIVATE added.
4. New instructions DRVCTL.T and DRVCTL.S renamed to DRVCTL_T and DRVCTL_S for syntax compatibility.
5. Register access specified for DRVCTL_T, DRVCTL_S, and PRIVATE instructions.
6. No other changes to Version 1.0 BSDL.

Package Type: 18 x 18 PGA

This BSDL is for the newer MC68040 mask sets of E26A and after (roughly after the second half of 1992). It does not include the 0.8-µm mask sets D43B, D50D, and D98D. For MC68LC040 and MC68EC040, two pin names have changed. To make the necessary modifications, change all occurrences of DLE to JS0 and `MDIS` to JS1.

```
entity MC68040 is

generic(PHYSICAL_PIN_MAP:string := "PGA_18x18");

port  (TDI:     in          bit;
       TDO:     out         bit;
       TMS:     in          bit;
       TCK:     in          bit;
       TRST:    in          bit;
       RSTO:    buffer      bit;
       IPEND:   buffer      bit;
       CIOUT:   out         bit;
       UPA:     out         bit_vector(0 to 1);
       TT:      inout       bit_vector(0 to 1);
       A:       inout       bit_vector(0 to 31);
       D:       inout       bit_vector(0 to 31);
       LOCKE:   out         bit;
       LOCK:    out         bit;
       R_W:     inout       bit;
       TLN:     out         bit_vector(0 to 1);
       TM:      out         bit_vector(0 to 2);
       SIZ:     inout       bit_vector(0 to 1);
       MI:      buffer      bit;
       BR:      buffer      bit;
       TS:      inout       bit;
       BB:      inout       bit;
       TIP:     out         bit;
       PST:     buffer      bit_vector(0 to 3);
       TA:      inout       bit;
       TEA:     in          bit;
       BG:      in          bit;
       SC:      in          bit_vector(0 to 1);
       TBI:     in          bit;
       AVEC:    in          bit;
       TCI:     in          bit;
```

```
        DLE:    in          bit;
        PCLK:   in          bit;
        BCLK:   in          bit;
        IPL:    in          bit_vector(0 to 2);
        RSTI:   in          bit;
        CDIS:   in          bit;
        MDIS:   in          bit;
        EGND:   linkage bit_vector(1 to 23);
        EVDD:   linkage   bit_vector(1 to 12);
        IGND:   linkage   bit_vector(1 to 12);
        IVDD:   linkage   bit_vector(1 to 7);
        CGND:   linkage   bit_vector(1 to 2);
        CVDD:   linkage   bit_vector(1 to 6);
        PGND:   linkage   bit_vector(1 to 3);
        PVDD:   linkage   bit_vector(1 to 2)
        );


    use STD_1149_1_1990.all;
    attribute PIN_MAP of MC68040 : entity is PHYSICAL_PIN_MAP;


—18x18 PGA Pin Map

constant PGA_18x18 : PIN_MAP_STRING :=
        "TDI:       S3,                                                         "  &
        "TDO:       T2,                                                         "  &
        "TMS:       S5,                                                         "  &
        "TCK:       S4,                                                         "  &
        "TRST:      T3,                                                         "  &
        "RSTO:      R3,                                                         "  &
        "IPEND:     S1,                                                         "  &
        "CIOUT:     R1,                                                         "  &
        "UPA:       (Q3, Q1),                                                   "  &
        "TT:        (P3, P2),                                                   "  &
        "A:         (L18, K18, J17, J18, H18, G18, G16, F18, E18, F16, P1,  N3,  "  &
        "            N1,  M1,  L1,  K1,  K2,  J1,  H1,  J2,  G1,  F1,  E1,  G3,  "  &
        "            D1,  F3,  E2,  C1,  E3,  B1,  D3,  A1),                    "  &
        "D:         (C3,  B3,  C4,  A2,  A3,  A4,  A5,  A6,  B7,  A7,  A8,  A9,  "  &
        "            A10, A11, A12, A13, B11, A14, B12, A15, A16, A17, B16, C15,  "  &
        "            A18, C16, B18, D16, C18, E16, E17, D18),                    "  &
        "LOCKE:     R18,                                                        "  &
        "LOCK:      S18,                                                        "  &
        "R_W:       N16,                                                        "  &
        "TLN:       (Q18, P18),                                                 "  &
        "TM:        (N18, M18, K17),                                            "  &
        "SIZ:       (P17, P16),                                                 "  &
        "MI:        Q16,                                                        "  &
        "BR:        T18,                                                        "  &
        "TS:        R16,                                                        "  &
        "BB:        T17,                                                        "  &
        "TIP:       R15,                                                        "  &
        "PST:       (T15, S14, R14, T16),                                       "  &
        "TA:        T14,                                                        "  &
        "TEA:       S13,                                                        "  &
        "BG:        T13,                                                        "  &
        "SC:        (T12, S12),                                                 "  &
        "TBI:       S11,                                                        "  &
        "AVEC:      T11,                                                        "  &
```

```
        "TCI:     T10,                                                  "  &
        "DLE:     T9,                                                   "  &
        "PCLK:    R9,                                                   "  &
        "BCLK:    R7,                                                   "  &
        "IPL:     (T8, T7, T6),                                         "  &
        "RSTI:    S7,                                                   "  &
        "CDIS:    T5,                                                   "  &
        "MDIS:    S6,                                                   "  &
        "EGND:    (S2,  Q2,  N2,  L2,  H2,  F2,  D2,  B2,  B4,  B6,  B8,  B10,  "  &
        "          B13, B15, B17, D17, F17, H17, L17, N17, Q17, S17, S15),  "  &
        "EVDD:    (R2,  M2,  G2,  C2,  B5,  B9,  B14, C17, G17, M17, R17, S16),  "  &
        "IGND:    (T4,  R4,  L3,  K3,  C7,  C9,  C11, K16, M16, R13, R11, S10),  "  &
        "IVDD:    (R5,  M3,  C8,  C10, C12, L16, R12),                  "  &
        "CGND:    (C6,  C13),                                           "  &
        "CVDD:    (J3,  H3,  C5,  C14, H16, J16),                       "  &
        "PGND:    (S9,  R10, R6),                                       "  &
        "PVDD:    (S8,  R8)                                             "  ;
```

—Other Pin Maps here when documented

```
    attribute TAP_SCAN_IN of TDI:signal is true;
    attribute TAP_SCAN_OUT of TDO:signal is true;
    attribute TAP_SCAN_MODE of TMS:signal is true;
    attribute TAP_SCAN_CLOCK of TCK:signal is (10.0e6, BOTH);
    attribute TAP_SCAN_RESET of TRST:signal is true;
    attribute INSTRUCTION_LENGTH of MC68040:entity is 3;
    attribute INSTRUCTION_OPCODE of MC68040:entity is

        "EXTEST      (000),                                             "  &
        "HI_Z        (001),                                             "  &
        "SAMPLE      (010),                                             "  &
        "DRVCTL.T    (011),                                             "  &
        "SHUTDOWN    (100),                                             "  &
        "PRIVATE     (101),                                             "  &
        "DRVCTL.S    (110),                                             "  &
        "BYPASS      (111)                                              "  ;

    attribute INSTRUCTION_CAPTURE of MC68040:entity is "001";
    attribute INSTRUCTION_DISABLE of MC68040:entity is "HI_Z";
    attribute REGISTER_ACCESS of MC68040:entity is

        "BYPASS (SHUTDOWN, HI_Z, PRIVATE),                              "  &
        "BOUNDARY (DRVCTL_T, DRVCTL_S)                                  "  ;

    attribute BOUNDARY_CELLS of MC68040:entity is

        "BC_2, BC_4                                                     "  ;

    attribute BOUNDARY_LENGTH of MC68040:entity is 184;
    attribute BOUNDARY_REGISTER of MC68040:entity is
```

```
num    cell    port    function  safe  ccell  dsval  rslt

"0     (BC_2,  RSTO,   output2,  X),                          "  &
"1     (BC_2,  IPEND,  output2,  X),                          "  &
"2     (BC_2,  CIOUT,  output3,  X,    156,   0,     Z),      "  &    —156 = io.0
"3     (BC_2,  UPA(0),  output3,  X,    156,   0,     Z),      "  &
"4     (BC_2,  UPA(1),  output3,  X,    156,   0,     Z),      "  &
"5     (BC_2,  TT(0),  output3,  X,    156,   0,     Z),      "  &
"6     (BC_4,  TT(0),  input,    X),                          "  &
"7     (BC_2,  TT(1),  output3,  X,    156,   0,     Z),      "  &
"8     (BC_4,  TT(1),  input,    X),                          "  &
"9     (BC_2,  A(10),  output3,  X,    150,   0,     Z),      "  &    —150 = io.ab
"10    (BC_4,  A(10),  input,    X),                          "  &
"11    (BC_2,  A(11),  output3,  X,    150,   0,     Z),      "  &
"12    (BC_4,  A(11),  input,    X),                          "  &
"13    (BC_2,  A(12),  output3,  X,    150,   0,     Z),      "  &
"14    (BC_4,  A(12),  input,    X),                          "  &
"15    (BC_2,  A(13),  output3,  X,    150,   0,     Z),      "  &
"16    (BC_4,  A(13),  input,    X),                          "  &
"17    (BC_2,  A(14),  output3,  X,    150,   0,     Z),      "  &
"18    (BC_4,  A(14),  input,    X),                          "  &
"19    (BC_2,  A(15),  output3,  X,    150,   0,     Z),      "  &
"20    (BC_4,  A(15),  input,    X),                          "  &
"21    (BC_2,  A(16),  output3,  X,    150,   0,     Z),      "  &
"22    (BC_4,  A(16),  input,    X),                          "  &
"23    (BC_2,  A(17),  output3,  X,    150,   0,     Z),      "  &
"24    (BC_4,  A(17),  input,    X),                          "  &
"25    (BC_2,  A(18),  output3,  X,    150,   0,     Z),      "  &
"26    (BC_4,  A(18),  input,    X),                          "  &
"27    (BC_2,  A(19),  output3,  X,    150,   0,     Z),      "  &
"28    (BC_4,  A(19),  input,    X),                          "  &
"29    (BC_2,  A(20),  output3,  X,    150,   0,     Z),      "  &
"30    (BC_4,  A(20),  input,    X),                          "  &
"31    (BC_2,  A(21),  output3,  X,    150,   0,     Z),      "  &
"32    (BC_4,  A(21),  input,    X),                          "  &
"33    (BC_2,  A(22),  output3,  X,    150,   0,     Z),      "  &
"34    (BC_4,  A(22),  input,    X),                          "  &
"35    (BC_2,  A(23),  output3,  X,    150,   0,     Z),      "  &
"36    (BC_4,  A(23),  input,    X),                          "  &
"37    (BC_2,  A(24),  output3,  X,    150,   0,     Z),      "  &
"38    (BC_4,  A(24),  input,    X),                          "  &
"39    (BC_2,  A(25),  output3,  X,    150,   0,     Z),      "  &
"40    (BC_4,  A(25),  input,    X),                          "  &
"41    (BC_2,  A(26),  output3,  X,    150,   0,     Z),      "  &
"42    (BC_4,  A(26),  input,    X),                          "  &
"43    (BC_2,  A(27),  output3,  X,    150,   0,     Z),      "  &
"44    (BC_4,  A(27),  input,    X),                          "  &
"45    (BC_2,  A(28),  output3,  X,    150,   0,     Z),      "  &
"46    (BC_4,  A(28),  input,    X),                          "  &
"47    (BC_2,  A(29),  output3,  X,    150,   0,     Z),      "  &
"48    (BC_4,  A(29),  input,    X),                          "  &
"49    (BC_2,  A(30),  output3,  X,    150,   0,     Z),      "  &
"50    (BC_4,  A(30),  input,    X),                          "  &
"51    (BC_2,  A(31),  output3,  X,    150,   0,     Z),      "  &
"52    (BC_4,  A(31),  input,    X),                          "  &
"53    (BC_2,  D(0),   output3,  X,    151,   0,     Z),      "  &    — 151 = io.db
"54    (BC_2,  D(1),   output3,  X,    151,   0,     Z),      "  &
"55    (BC_2,  D(2),   output3,  X,    151,   0,     Z),      "  &
"56    (BC_2,  D(3),   output3,  X,    151,   0,     Z),      "  &
```

```
num    cell    port    function    safe    ccell    dsval    rslt

"57    (BC_2,  D(4),   output3,    X,      151,     0,       Z),          "  &
"58    (BC_2,  D(5),   output3,    X,      151,     0,       Z),          "  &
"59    (BC_2,  D(6),   output3,    X,      151,     0,       Z),          "  &
"60    (BC_2,  D(7),   output3,    X,      151,     0,       Z),          "  &
"61    (BC_2,  D(8),   output3,    X,      151,     0,       Z),          "  &
"62    (BC_2,  D(9),   output3,    X,      151,     0,       Z),          "  &
"63    (BC_2,  D(10),  output3,    X,      151,     0,       Z),          "  &
"64    (BC_2,  D(11),  output3,    X,      151,     0,       Z),          "  &
"65    (BC_2,  D(12),  output3,    X,      151,     0,       Z),          "  &
"66    (BC_2,  D(13),  output3,    X,      151,     0,       Z),          "  &
"67    (BC_2,  D(14),  output3,    X,      151,     0,       Z),          "  &
"68    (BC_2,  D(15),  output3,    X,      151,     0,       Z),          "  &
"69    (BC_2,  D(16),  output3,    X,      151,     0,       Z),          "  &
"70    (BC_2,  D(17),  output3,    X,      151,     0,       Z),          "  &
"71    (BC_2,  D(18),  output3,    X,      151,     0,       Z),          "  &
"72    (BC_2,  D(19),  output3,    X,      151,     0,       Z),          "  &
"73    (BC_2,  D(20),  output3,    X,      151,     0,       Z),          "  &
"74    (BC_2,  D(21),  output3,    X,      151,     0,       Z),          "  &
"75    (BC_2,  D(22),  output3,    X,      151,     0,       Z),          "  &
"76    (BC_2,  D(23),  output3,    X,      151,     0,       Z),          "  &
"77    (BC_2,  D(24),  output3,    X,      151,     0,       Z),          "  &
"78    (BC_2,  D(25),  output3,    X,      151,     0,       Z),          "  &
"79    (BC_2,  D(26),  output3,    X,      151,     0,       Z),          "  &
"80    (BC_2,  D(27),  output3,    X,      151,     0,       Z),          "  &
"81    (BC_2,  D(28),  output3,    X,      151,     0,       Z),          "  &
"82    (BC_2,  D(29),  output3,    X,      151,     0,       Z),          "  &
"83    (BC_2,  D(30),  output3,    X,      151,     0,       Z),          "  &
"84    (BC_2,  D(31),  output3,    X,      151,     0,       Z),          "  &
"85    (BC_4,  D(0),   input,      X),                                   "  &
"86    (BC_4,  D(1),   input,      X),                                   "  &
"87    (BC_4,  D(2),   input,      X),                                   "  &
"88    (BC_4,  D(3),   input,      X),                                   "  &
"89    (BC_4,  D(4),   input,      X),                                   "  &
"90    (BC_4,  D(5),   input,      X),                                   "  &
"91    (BC_4,  D(6),   input,      X),                                   "  &
"92    (BC_4,  D(7),   input,      X),                                   "  &
"93    (BC_4,  D(8),   input,      X),                                   "  &
"94    (BC_4,  D(9),   input,      X),                                   "  &
"95    (BC_4,  D(10),  input,      X),                                   "  &
"96    (BC_4,  D(11),  input,      X),                                   "  &
"97    (BC_4,  D(12),  input,      X),                                   "  &
"98    (BC_4,  D(13),  input,      X),                                   "  &
"99    (BC_4,  D(14),  input,      X),                                   "  &
"100   (BC_4,  D(15),  input,      X),                                   "  &
"101   (BC_4,  D(16),  input,      X),                                   "  &
"102   (BC_4,  D(17),  input,      X),                                   "  &
"103   (BC_4,  D(18),  input,      X),                                   "  &
"104   (BC_4,  D(19),  input,      X),                                   "  &
"105   (BC_4,  D(20),  input,      X),                                   "  &
"106   (BC_4,  D(21),  input,      X),                                   "  &
"107   (BC_4,  D(22),  input,      X),                                   "  &
"108   (BC_4,  D(23),  input,      X),                                   "  &
"109   (BC_4,  D(24),  input,      X),                                   "  &
"110   (BC_4,  D(25),  input,      X),                                   "  &
"111   (BC_4,  D(26),  input,      X),                                   "  &
"112   (BC_4,  D(27),  input,      X),                                   "  &
"113   (BC_4,  D(28),  input,      X),                                   "  &
```

```
num    cell    port    function  safe  ccell  dsval  rslt

"114   (BC_4,  D(29),  input,    X),                          "  &
"115   (BC_4,  D(30),  input,    X),                          "  &
"116   (BC_4,  D(31),  input,    X),                          "  &
"117   (BC_2,  A(9),   output3,  X,    150,   0,     Z),      "  &     —150 = io.ab
"118   (BC_4,  A(9),   input,    X),                          "  &
"119   (BC_2,  A(8),   output3,  X,    150,   0,     Z),      "  &
"120   (BC_4,  A(8),   input,    X),                          "  &
"121   (BC_2,  A(7),   output3,  X,    150,   0,     Z),      "  &
"122   (BC_4,  A(7),   input,    X),                          "  &
"123   (BC_2,  A(6),   output3,  X,    150,   0,     Z),      "  &
"124   (BC_4,  A(6),   input,    X),                          "  &
"125   (BC_2,  A(5),   output3,  X,    150,   0,     Z),      "  &
"126   (BC_4,  A(5),   input,    X),                          "  &
"127   (BC_2,  A(4),   output3,  X,    150,   0,     Z),      "  &
"128   (BC_4,  A(4),   input,    X),                          "  &
"129   (BC_2,  A(3),   output3,  X,    150,   0,     Z),      "  &
"130   (BC_4,  A(3),   input,    X),                          "  &
"131   (BC_2,  A(2),   output3,  X,    150,   0,     Z),      "  &
"132   (BC_4,  A(2),   input,    X),                          "  &
"133   (BC_2,  A(1),   output3,  X,    150,   0,     Z),      "  &
"134   (BC_4,  A(1),   input,    X),                          "  &
"135   (BC_2,  A(0),   output3,  X,    150,   0,     Z),      "  &
"136   (BC_4,  A(0),   input,    X),                          "  &
"137   (BC_2,  TM(2),  output3,  X,    156,   0,     Z),      "  &     —156 = io.0
"138   (BC_2,  TM(1),  output3,  X,    156,   0,     Z),      "  &
"139   (BC_2,  TM(0),  output3,  X,    156,   0,     Z),      "  &
"140   (BC_2,  TLN(1),  output3,  X,    156,   0,     Z),      "  &
"141   (BC_2,  TLN(0),  output3,  X,    156,   0,     Z),      "  &
"142   (BC_2,  SIZ(0),  output3,  X,    156,   0,     Z),      "  &
"143   (BC_4,  SIZ(0),  input,    X),                          "  &
"144   (BC_2,  R_W,    output3,  X,    156,   0,     Z),      "  &
"145   (BC_4,  R_W,    input,    X),                          "  &
"146   (BC_2,  LOCKE,  output3,  X,    156,   0,     Z),      "  &
"147   (BC_2,  SIZ(1),  output3,  X,    156,   0,     Z),      "  &
"148   (BC_4,  SIZ(1),  input,    X),                          "  &
"149   (BC_2,  LOCK,   output3,  X,    156,   0,     Z),      "  &
"150   (BC_2,  *,      controlr,  0),                          "  &     — io.ab
"151   (BC_2,  *,      controlr,  0),                          "  &     — io.db
"152   (BC_2,  MI,     output2,  X),                          "  &
"153   (BC_2,  BR,     output2,  X),                          "  &
"154   (BC_2,  *,      controlr,  0),                          "  &     — io.2
"155   (BC_2,  *,      controlr,  0),                          "  &     — io.1
"156   (BC_2,  *,      controlr,  0),                          "  &     — io.0
"157   (BC_2,  TS,     output3,  X,    156,   0,     Z),      "  &     — 156 = io.0
"158   (BC_4,  TS,     input,    X),                          "  &
"159   (BC_2,  BB,     output3,  X,    155,   0,     Z),      "  &     — 155 = io.1
"160   (BC_4,  BB,     input,    X),                          "  &
"161   (BC_2,  TIP,    output3,  X,    155,   0,     Z),      "  &     — 155 = io.1
"162   (BC_2,  PST(3),  output2,  X),                          "  &
"163   (BC_2,  PST(2),  output2,  X),                          "  &
"164   (BC_2,  PST(1),  output2,  X),                          "  &
"165   (BC_2,  PST(0),  output2,  X),                          "  &
"166   (BC_2,  TA,     output3,  X,    154,   0,     Z),      "  &     — 154 = io.2
"167   (BC_4,  TA,     input,    X),                          "  &
"168   (BC_4,  TEA,    input,    X),                          "  &
"169   (BC_4,  BG,     input,    X),                          "  &
"170   (BC_4,  SC(1),  input,    X),                          "  &
```

```
num     cell       port       function     safe     ccell     dsval     rslt

"171    (BC_4,     SC(0),     input,       X),                                  "  &
"172    (BC_4,     TBI,       input,       X),                                  "  &
"173    (BC_4,     AVEC,      input,       X),                                  "  &
"174    (BC_4,     TCI,       input,       X),                                  "  &
"175    (BC_4,     DLE,       input,       X),                                  "  &
"176    (BC_4,     PCLK,      input,       X),                                  "  &
"177    (BC_4,     BCLK,      input,       X),                                  "  &
"178    (BC_4,     IPL(0),    input,       X),                                  "  &
"179    (BC_4,     IPL(1),    input,       X),                                  "  &
"180    (BC_4,     IPL(2),    input,       X),                                  "  &
"181    (BC_4,     RSTI,      input,       X),                                  "  &
"182    (BC_4,     CDIS,      input,       X),                                  "  &
"183    (BC_4,     MDIS,      input,       X)                                   "  ;

attribute DESIGN_WARNING of MC68040: entity is
        "A non-standard clocking protocol on BCLK and PCLK must be             "  &
        "observed when entering Boundary Scan Test Mode.                       "  ;

end MC68040                                                                       ;
```

6.7 MC68040, MC68LC040, MC68EC040 JTAG ELECTRICAL CHARACTERISTICS

The following paragraphs provide information on JTAG electrical and timing specifications. This section is subject to change. For the most recent specifications, contact a Motorola sales office or complete the registration card at the beginning of this manual.

JTAG DC Electrical Specifications

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
TCK Input Leakage Current @ 0.5–2.4 V	I_{in}	20	20	μA
TDO Hi-Z (Off-State) Leakage Current @ 0.5–2.4 V	I_{TST}	20	20	μA
Signal Low Input Current, V_{IL} = 0.8 V TMS, TDI, TRST	I_L	–1.1	–0.18	mA
Signal High Input Current, V_{IH} = 2.0 V TMS, TDI, TRST	I_H	–0.94	–0.16	mA
TDO Output High Voltage	V_{OH}	2.4	—	V
TDO Output Low Voltage	V_{OL}	—	0.5	V
Capacitance*, V_{in} = 0 V, f = 1 MHz	C_{in}	—	25	pF

*Capacitance is periodically sampled rather than 100% tested.

JTAG Timing Specifications (All Operating Frequencies)

Num	Characteristic	Min	Max	Unit
	TCK Frequency of Operation	0	10	MHz
1	TCK Cycle Time	100	—	ns
2	TCK Clock Pulse Width Measured at 1.5 V	40	—	ns
3	TCK Rise and Fall Times	0	10	ns
4	TRST Setup Time to TCK Falling Edge	40	—	ns
5	TRST Assert Time	100	—	ns
6	Boundary Scan Input Data Setup Time	50	—	ns
7	Boundary Scan Input Data Hold Time	50	—	ns
8	TCK to Output Data Valid	0	50	ns
9	TCK to Output High Impedance	0	50	ns
10	TMS, TDI Data Setup Time	20	—	ns
11	TMS, TDI Data Hold Time	5	—	ns
12	TCK to TDO Data Valid	0	20	ns
13	TCK to TDO High Impedance	0	20	ns

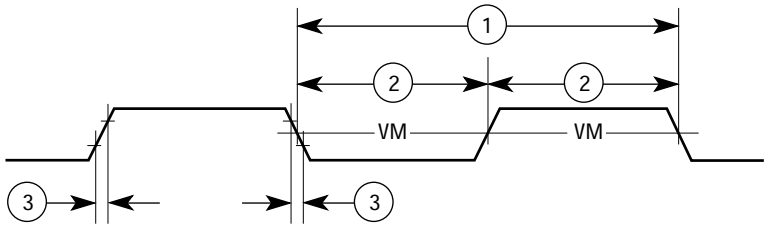


Figure 6-8. Clock Input Timing Diagram

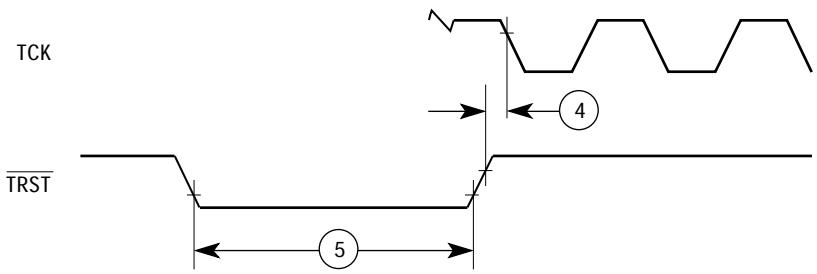


Figure 6-9. TRST Timing Diagram

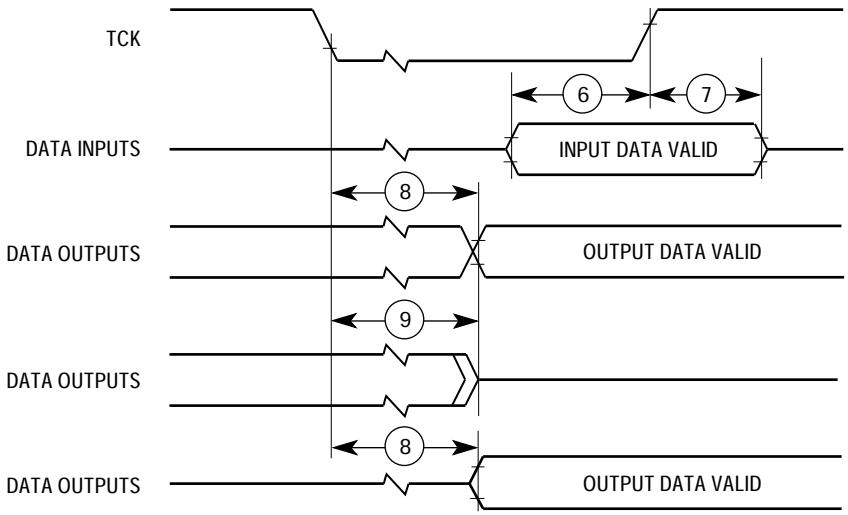


Figure 6-10. Boundary Scan Timing Diagram

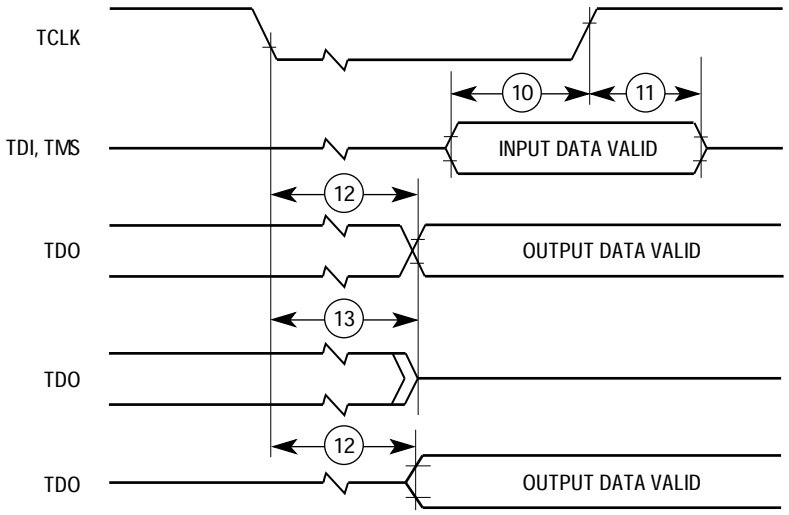


Figure 6-11. Test Access Port Timing Diagram

SECTION 7
BUS OPERATION

The M68040 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 bus master, which the processor snoops as a slave device. Descriptions of the error and halt conditions, bus arbitration, and the reset operation are also included. For timing specifications, refer to **Section 11 MC68040 Electrical and Thermal Characteristics.**

NOTE

For the MC68040V, MC68LC040, and MC68EC040 ignore all references to floating-point. For the MC68EC040 and MC68EC040V ignore all references to the memory management unit (MMU). Special modes of operation do not apply to these devices. Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for details.

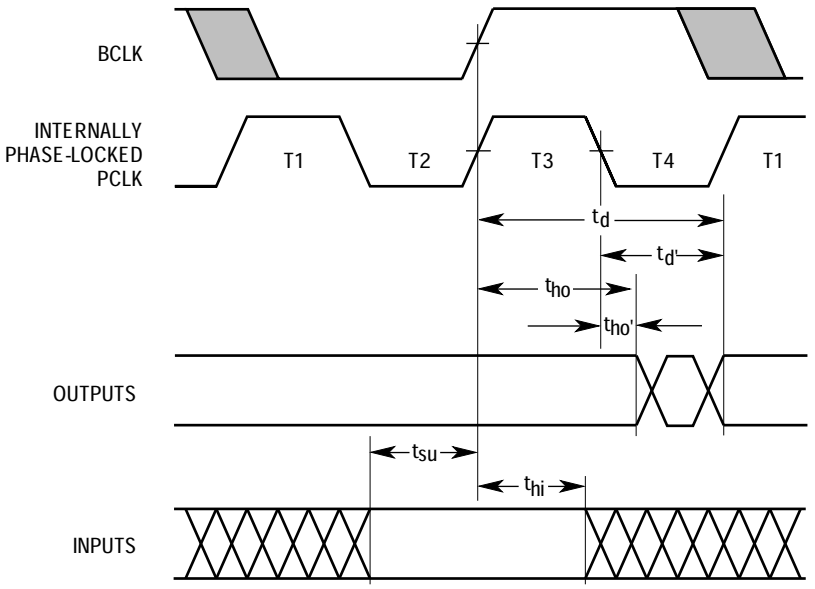
7.1 BUS CHARACTERISTICS

The M68040 uses the address bus (A31–A0) to specify the address for a data transfer and the data bus (D31–D0) to transfer the data. Control signals indicate the beginning and type of a bus cycle as well as the address space and size of the transfer. The selected device then controls the length of the cycle by terminating it using the control signals.

The M68040 uses two clocks to generate timing: a processor clock (PCLK) and a bus clock (BCLK). The PCLK signal is twice the frequency of the BCLK signal and is internally phase-locked to BCLK. PCLK is also distributed throughout the device to generate additional timing for additional edges for internal logic blocks and has no bearing on bus timing. The use of dual clock inputs allows the bus interface to operate at half the speed of the internal logic of the processor, requiring less stringent memory interface requirements. Since the rising edge of BCLK is used as the reference point for the phase-locked loop (PLL), all timing specifications are referenced to this edge.

Figure 7-1 illustrates the general relationship between the two clock signals and most input and output signals. The rising edge of the internally phase-locked PCLK is aligned with the rising edge of BCLK, and the two PCLK cycles corresponding to each BCLK cycle are divided into four states, T1–T4. Most outputs change during state T4, whether transitioning between a driven and high-impedance state or switching between assert and

negate logic levels. The exceptions to this rule are the $\overline{TIP}$, $\overline{TA}$, and $\overline{BB}$ signals that transition between logic levels during T4 but transition from a driven state to a high-impedance state during T1. The input setup time (t_{su}), input hold time (t_{hi}), output hold time (t_{ho}), and delay time (t_d) illustrated in Figure 7-1 are described in the AC electrical timing specifications in **Section 11 MC68040 Electrical and Thermal Characteristics**.



NOTES:
1. t_d = Propagation delay of signal relative to BLK rising edge.
2. $t_{d'}$ = Propagation delay of signal relative to PCLK falling edge; $t_{d'} = t_d - 1/2$ PCLK except for $\overline{TIP}$, $\overline{TA}$, $\overline{BB}$ when used as outputs.
3. t_{ho} = Output hold time relative to BCLK rising edge.
4. $t_{ho'}$ = Output hold time relative to BCLK rising edge; $t_{ho'} = t_h - 1/2$ PCLK.
5. t_{su} = Required input setup time relative to BCLK rising edge.
6. t_{hi} = Required input hold time relative to BCLK rising edge.

Figure 7-1. Signal Relationships to Clocks

Inputs to the M68040 (other than the $\overline{IPL2}$–$\overline{IPL0}$ and $\overline{RSTI}$ signals) are synchronously sampled and must be stable during the sample window defined by t_{su}, t_{hi}, and t_{ho} (see Figure 7-1) to guarantee proper operation. The asynchronous $\overline{IPLx}$ and $\overline{RSTI}$ signals are also sampled on the rising edge of BCLK, but are internally synchronized to resolve the input to a valid level before using it. Since the timing specifications for the M68040 are referenced to the rising edge of BCLK, they are valid only for the specified operating frequency and must be scaled for lower operating frequencies.

7.2 DATA TRANSFER MECHANISM

Figure 7-2 illustrates how the bus designates operands for transfers on a byte boundary system. The integer unit handles floating-point operands as a sequence of related long-word operands. These designations are used in the figures and descriptions that follow.

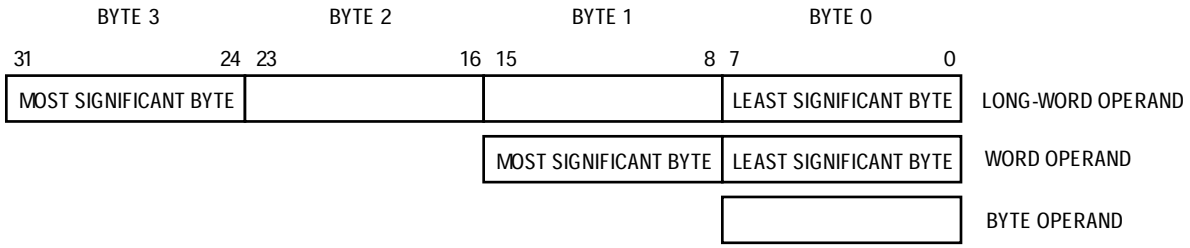


Figure 7-2. Internal Operand Representation

Figure 7-3 illustrates general multiplexing between an internal register and the external bus. The internal register connects to the external data bus through the internal data bus and multiplexer. The data multiplexer establishes the necessary connections for different combinations of address and data sizes.

Unlike the MC68020 and MC68030 processors, the M68040 does not support dynamic bus sizing and expects the referenced device to accept the requested access width. The MC68150 dynamic bus sizer is designed to allow the 32-bit M68040, MC68EC040, MC68LC040 bus to communicate bidirectionally with 32-, 16-, or 8-bit peripherals and memories. It dynamically recognizes the size of the selected peripheral or memory device and then reads or writes the appropriate data from that location. Refer to MC68150/D, *MC68150 Dynamic Bus Sizer*, for information on this device.

Blocks of memory that must be contiguous, such as for code storage or program stacks, must be 32 bits wide. Byte- and word-sized I/O ports that return an interrupt vector during interrupt acknowledge cycles must be mapped into the low-order 8 or 16 bits, respectively, of the data bus.

The multiplexer takes the four bytes of the 32-bit bus transfer and routes them to their required positions. For example, byte 0 would normally be routed to D31–D24, but it can also be routed to any other byte position supporting a misaligned data transfer. The same is true for any of the other operand bytes. The transfer size (SIZ0 and SIZ1) and byte offset (A1 and A0) signals determine the positioning of the bytes (see Table 7-1). The size indicated on the SIZx signals corresponds to the size of the operand transfer for the entire bus cycle. During an operand transfer, A31–A2 indicate the long-word base address for the first byte of the operand to be accessed; A1 and A0 indicate the byte offset from the base. For a burst-inhibited line transfer, A1 and A0 for each of the four accesses (the burst-inhibited line transfer and three long-word transfers) are copied from the lowest two bits of the access address used to initiate the line transfer.

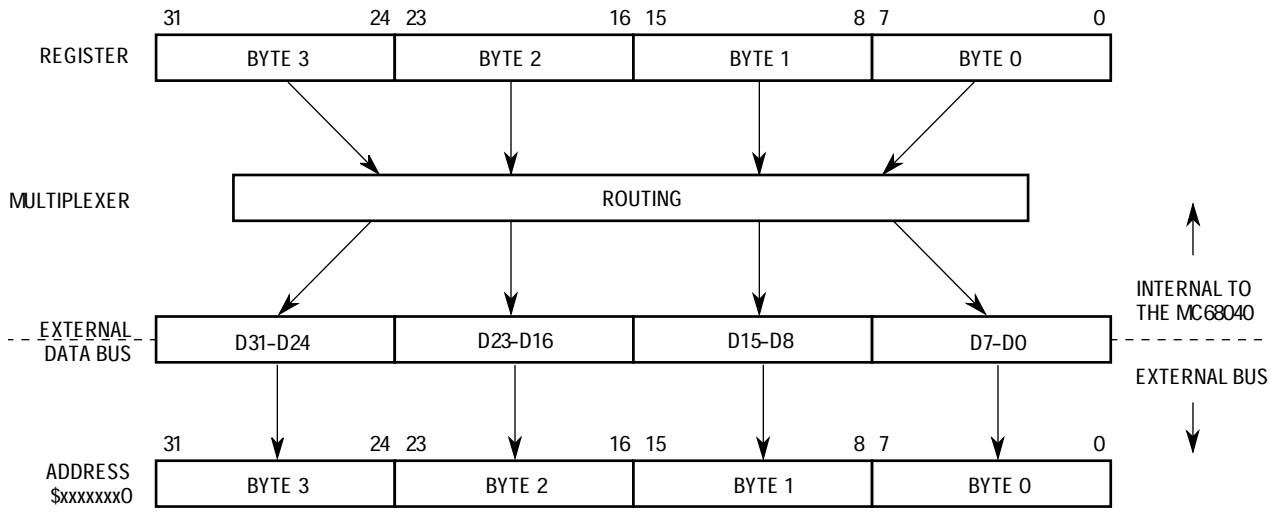
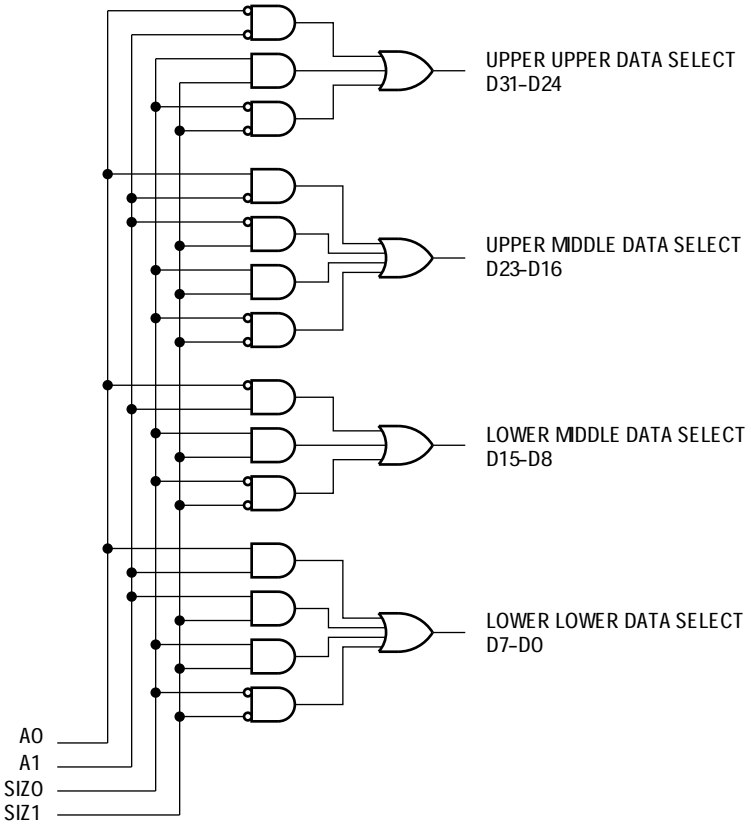


Figure 7-3. Data Multiplexing

Table 7-1 lists the combinations of the SIZx, A1, and A0 signals, collectively called byte enable signals, that are used for each of the four sections of the data bus. In the table, BYTEn indicates the data bus section that is active, the portion of the requested operand that is read or written during that bus transfer. For line transfers, all bytes are valid as listed and can correspond to portions of the requested operand or to data required to fill the remainder of the cache line. The bytes labeled with a dash are not required; they are ignored on read transfers and driven with undefined data on write transfers. Not selecting these bytes prevents incorrect accesses in sensitive areas such as I/O devices. Figure 7-4 illustrates a logic diagram for one method for generating byte enable signals from the SIZx, A1, and A0 and the associated PAL equation. These byte enable signals can be combined with the address decode logic.

Table 7-1. Data Bus Requirements for Read and Write Cycles

Transfer Size	Signal Encodings				Active Data Bus Sections			
	SIZ1	SIZ0	A1	A0	D31–D24	D23–D16	D15–D8	D7–D0
Byte	0	1	0	0	BYTEn	—	—	—
	0	1	0	1	—	BYTEn	—	—
	0	1	1	0	—	—	BYTEn	—
	0	1	1	1	—	—	—	BYTEn
Word	1	0	0	0	BYTEn	BYTEn	—	—
	1	0	1	0	—	—	BYTEn	BYTEn
Long Word	0	0	X	X	BYTEn	BYTE n	BYTEn	BYTEn
Line	1	1	X	X	BYTEn	BYTEn	BYTEn	BYTEn




```
PAL16L8
U1
MC68040 Byte Data Select Generation.
Motorola Worldwide Marketing Training Organization
A0 A1 SIZ0 SIZ1 NC NC NC NC NC GND NC UUD UMD LMD LLD
NC NC NC NC VCC

/UUD =   /A0 * /A1              ; directly addressed, any size
       + /SIZ1 * /SIZ0          ; enable every byte for long word size
       + SIZ1 * SIZ0            ; enable every byte for line size
/UMD =   A0 * /A1               ; directly addressed, any size
       + /A1 * /SIZ1            ; word aligned, size is word or line
       + SIZ1 * SIZ0            ; enable every byte for long word size
       + /SIZ1 * /SIZ0          ; enable every byte for line size
/LMD =   /A0 * /A1              ; directly addressed, any size
       + /SIZ1 * /SIZ0          ; enable every byte for long word size
       + SIZ1 * SIZ0            ; enable every byte for line size
/LLD =    A0 * /A1              ; directly addressed, any size
       + /A1 * /SIZ1            ; word aligned, word or line size
       + SIZ1 * SIZ0            ; enable every byte for long word size
       + /SIZ1 * /SIZ0          ; enable every byte for line size
```

Figure 7-4. Byte Enable Signal Generation and PAL Equation

A brief summary of the bus signal encodings for each access type is listed in Table 7-2. Additional information on the encodings for the M68040 signals can be found in **Section 5 Signal Description.**

Table 7-2. Summary of Access Types versus Bus Signal Encodings

Bus Signal	Data Cache Push Access	Normal Data/Code Access	Table Search Access	MOVE16 Access	Alternate Access	Interrupt Acknowledge	Breakpoint Acknowledge
A31–A0	Access Address	Access Address	Entry Address	Access Address	Access Address	$FFFFFFFF	$00000000
UPA1, UPA0	$0	MMU Source[1]	$0	MMU Source[1]	$0	$0	$0
SIZ1, SIZ0	L/Line	B/W/L/Line	Long Word	Line	B/W/L	Byte	Byte
TT1, TT0	$0	$0	$0	$1	$2	$3	$3
TM4–TM2	$0	$1,2,5, or 6	$3 or 4	$1 or 5	Function Code	Int. Level $1–7	$0
TLN1, TLN0	Cache Set Entry	Cache Set Entry[2]	Undefined	Undefined	Undefined	Undefined	Undefined
R/$\overline{W}$	Write	Read/Write	Read/Write	Read/Write	Read/Write	Read	Read
$\overline{LOCK}$ $\overline{LOCKE}$	Negated	Asserted/ Negated[3]	Asserted/ Negated[3]	Negated	Negated	Negated	Negated
$\overline{CIOUT}$	Negated	MMU Source[1]	Negated	MMU Source[1]	Asserted	Negated	Negated

NOTES
1. The UPA1, UPA0, and $\overline{CIOUT}$ signals are determined by the U1, U0 data and CM bit fields, respectively, corresponding to the access address.
2. The TLNx signals are defined only for normal push accesses and normal data line read accesses.
3. The $\overline{LOCK}$ signal is asserted during TAS, CAS, and CAS2 operand accesses and for some table search update sequences. $\overline{LOCKE}$ is asserted for the last transfer of each locked sequence of transfers.
4. Refer to **Section 5 Signal Description** for definitions of the TMx signal encodings for normal, MOVE16, and alternate accesses.

7.3 MISALIGNED OPERANDS

All M68040 data formats can be located in memory on any byte boundary. A byte operand is properly aligned at any address; a word operand is misaligned at an odd address; and a long word is misaligned at an address that is not evenly divisible by 4. However, since operands can reside at any byte boundary, they can be misaligned. Although the M68040 does not enforce any alignment restrictions for data operands (including PC relative data addressing), some performance degradation occurs when additional bus cycles are required for long-word or word operands that are misaligned. For maximum performance, data items should be aligned on their natural boundaries. All instruction words and extension words must reside on word boundaries. Attempting to prefetch an instruction word at an odd address causes an address error exception. Refer to **Section 8 Exception Processing** for details on address error exceptions.

The M68040 data memory unit converts misaligned operand accesses that are noncachable to a sequence of aligned accesses. These aligned accesses are then sent to the bus controller for completion, always resulting in aligned bus transfers. Misaligned operand accesses that miss in the data cache are cachable and are not aligned before line filling. Refer to **Section 4 Instruction and Data Caches** for details on line fill and the data cache.

Figure 7-5 illustrates the transfer of a long-word operand from an odd address requiring more than one bus cycle. For the first transfer or bus cycle, the SIZx signals specify a byte transfer, and the byte offset is $1. The slave device supplies the byte and acknowledges the data transfer. When the processor starts the second cycle, the SIZx signals specify a word transfer with a byte offset of $2. The next two bytes are transferred during this cycle. The processor then initiates the third cycle, with the SIZEx signals indicating a byte transfer. The byte offset is now $0; the port supplies the final byte and the operation is complete. This example is similar to the one illustrated in Figure 7-6 except that the operand is word sized and the transfer requires only two bus cycles. Figure 7-7 illustrates a functional timing diagram for a misaligned long-word read transfer.

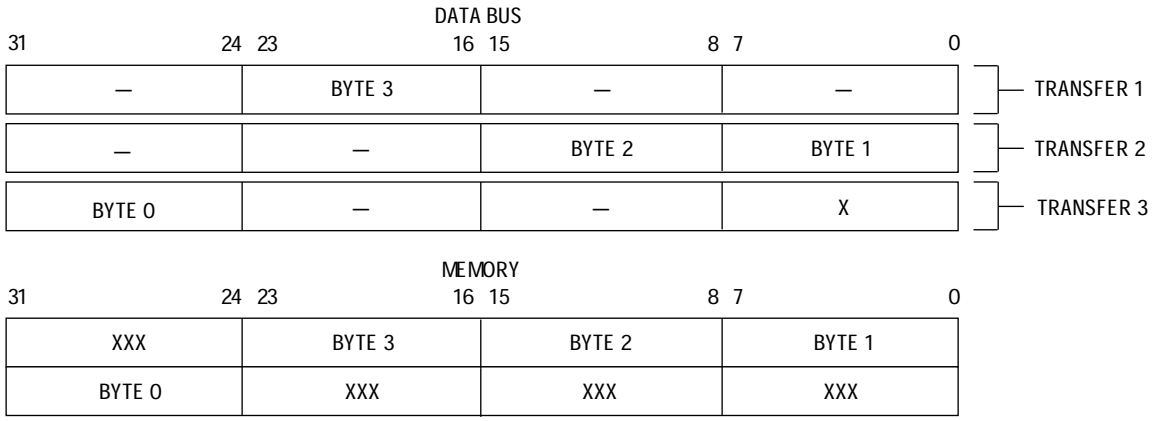


Figure 7-5. Example of a Misaligned Long-Word Transfer

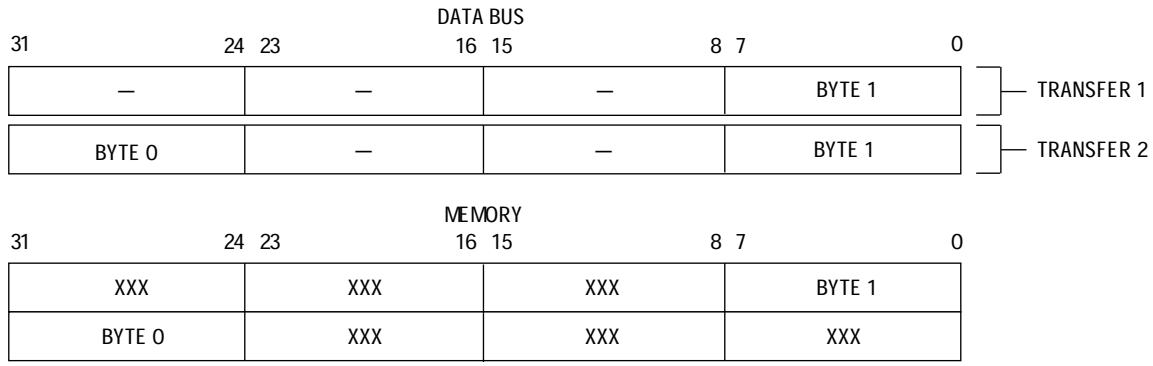


Figure 7-6. Example of a Misaligned Word Transfer

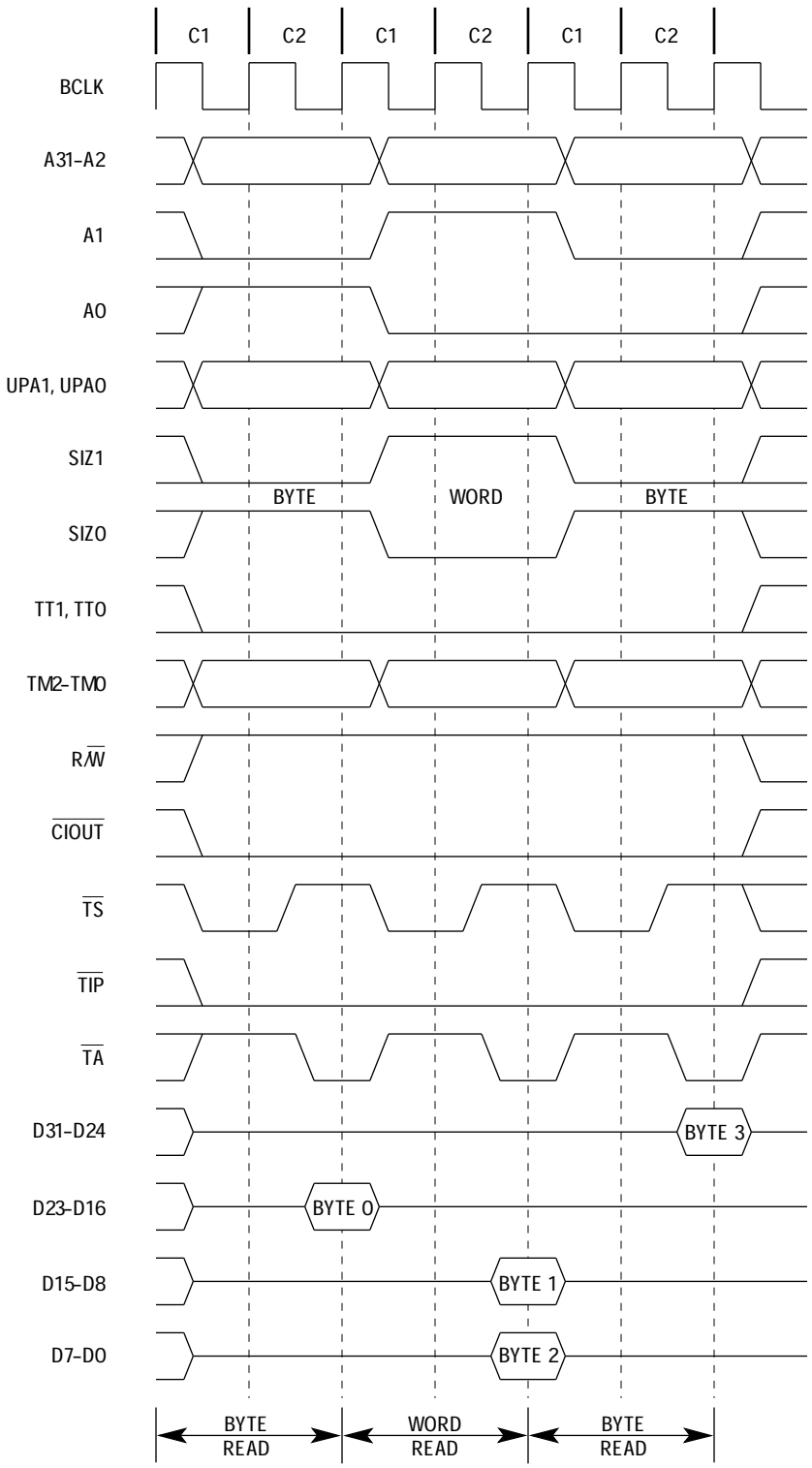


Figure 7-7. Misaligned Long-Word Read Transfer Timing

The combination of operand size and alignment determines the number of bus cycles required to perform a particular memory access. Table 7-3 lists the number of bus cycles required for different operand sizes with all possible alignment conditions for read and write cycles. The table confirms that alignment significantly affects bus cycle throughput for noncachable accesses. For example, in Figure 7-5 the misaligned long-word operand took three bus cycles because the byte offset = $1. If the byte offset = $0, then it would have taken one bus cycle. The M68040 system designer and programmer should account for these effects, particularly in time-critical applications.

Table 7-3. Memory Alignment Influence on Noncachable and Write-Through Bus Cycles

Transfer Size	Number of Bus Cycles			
	$0*	$1*	$2*	$3*
Instruction	1	N/A	N/A	N/A
Byte Operand	1	1	1	1
Word Operand	1	2	1	2
Long-Word Operand	1	3	2	3

*Where the byte offset (A1 and A0) equals this encoding.

The processor always prefetches instructions by reading a long word from a half-line address (A2–A0 = $0), regardless of alignment. When the required instruction begins at the second long word, the processor attempts to fetch the entire half-line (two long words) although the second long word contains the required instruction.

7.4 PROCESSOR DATA TRANSFERS

The transfer of data between the processor and other devices involves the address bus, data bus, and control signals. The address and data buses are normally parallel, nonmultiplexed buses, supporting byte, word, long-word, and line (16-byte) bus cycles. 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 that do not support bursting can burst-inhibit the first long word of 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 BCLK signal. The M68040 moves data on the bus by issuing control signals and using a handshake protocol to ensure correct data movement. The following paragraphs describe the bus cycles for byte, word, long-word, and line read, write, and read-modify-write transfers.

7.4.1 Byte, Word, and Long-Word Read Transfers

During a read transfer, the processor receives data from a memory or peripheral device. Since the data read for a byte, word, or long-word access is not placed in either of the internal caches by definition, the processor ignores the level on the transfer cache inhibit ($\overline{TCI}$) signal when latching the data. The bus controller performs byte, word, and long-word read transfers for the following cases:

- Accesses to a disabled cache.
- Accesses to a memory page that is specified noncachable.
- Accesses that are implicitly noncachable (read-modify-write accesses and accesses to an alternate logical address space via the MOVES instruction).
- Accesses that do not allocate in the data cache on a read miss (table searches, exception vector fetches, and exception stack deallocation for an RTE instruction).
- The first transfer of a line read is terminated with transfer burst inhibit ($\overline{TBI}$), forcing completion of the line access using three additional long-word read transfers.

Figure 7-8 is a flowchart for byte, word, and long-word read transfers. Bus operations are similar for each case and vary only with the size indicated and the portion of the data bus used for the transfer. Figure 7-9 is a functional timing diagram for byte, word, and long-word read transfers.

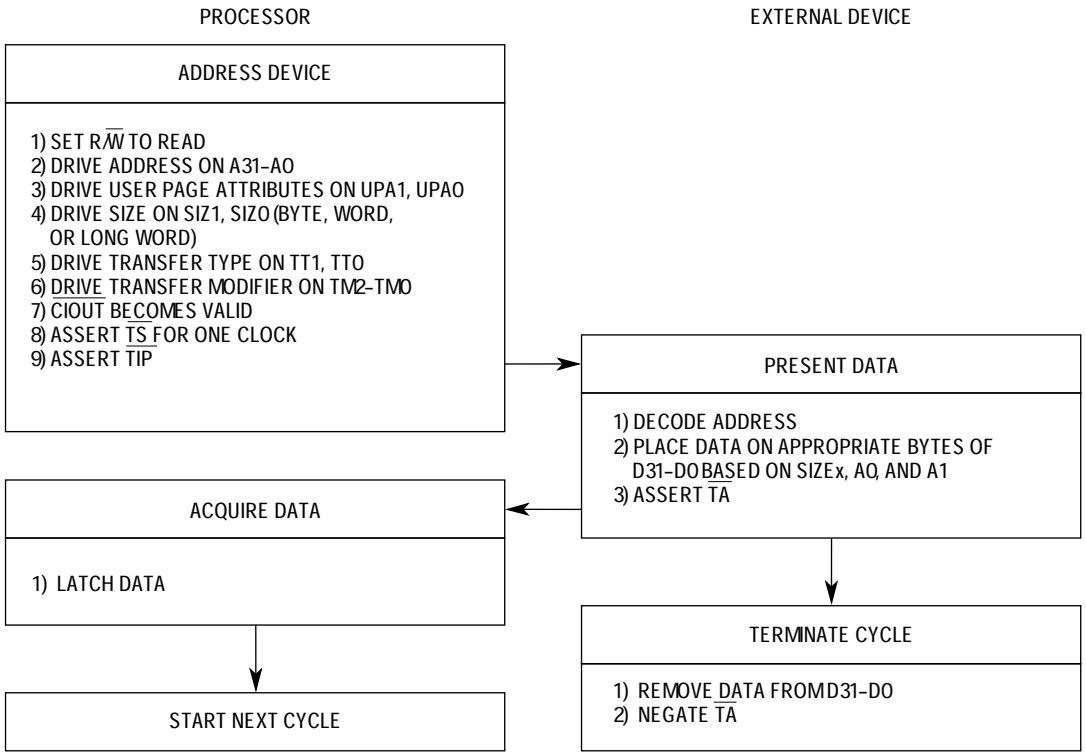


Figure 7-8. Byte, Word, and Long-Word Read Transfer Flowchart

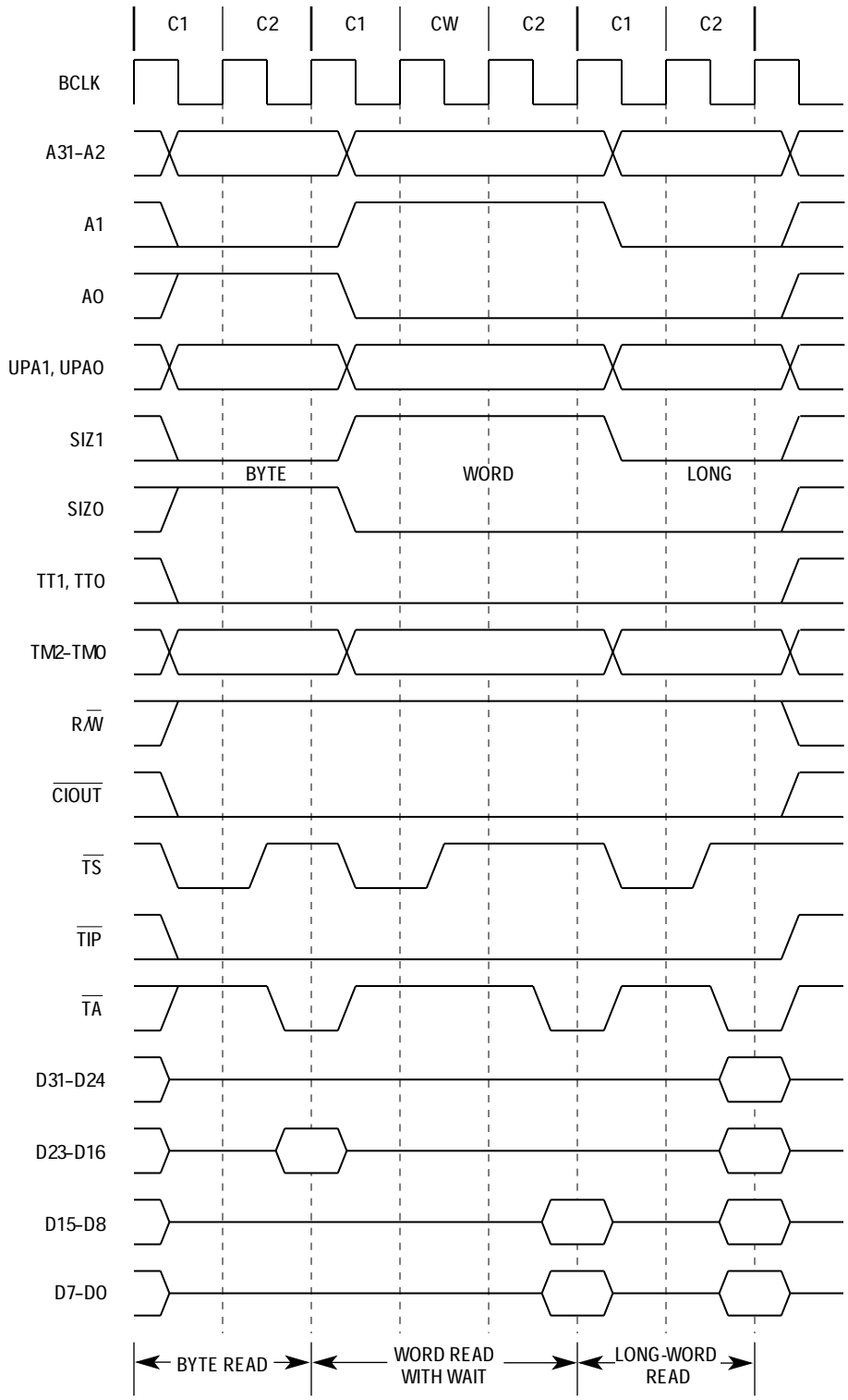


Figure 7-9. Byte, Word, and Long-Word Read Transfer Timing

Clock 1 (C1)

The read cycle starts in C1. During the first half of C1, the processor places valid values on the address bus and transfer attributes. For user and supervisor mode accesses, which the corresponding memory unit translates, the user-programmable attribute signals (UPAx) are driven with the values from the matching user bits (U1 and U0). The transfer type (TTx) and transfer modifier (TMx) signals identify the specific access type. The read/write (R/$\overline{W}$) signal is driven high for a read cycle. Cache inhibit out ($\overline{CIOUT}$) is asserted since the access is identified as noncachable. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for information on the M68040 and MC68LC040 memory units and **Appendix B MC68EC040** for information on the MC68EC040 memory unit.

The processor asserts transfer start ($\overline{TS}$) during C1 to indicate the beginning of a bus cycle. If not already asserted from a previous bus cycle, the transfer in progress ($\overline{TIP}$) signal is also asserted at this time to indicate that a bus cycle is active.

Clock 2 (C2)

During the first half of the clock after C1, the processor negates $\overline{TS}$. The selected peripheral device uses R/$\overline{W}$, SIZ1, SIZ0, A1, and A0 to place its information on the data bus. With the exception of the R/$\overline{W}$ signal, these signals also select any or all of the operand bytes (D31–D24, D23–D16, D15–D8, and D7–D0). If the first clock after C1 is not a wait state (CW), then the selected peripheral device asserts the transfer acknowledge ($\overline{TA}$) signal.

At the end of the first clock cycle after C1, the processor samples the level of $\overline{TA}$ and latches the current value on the data bus; the bus cycle terminates, and the data is passed to the processor's appropriate memory unit if $\overline{TA}$ is asserted. If $\overline{TA}$ is not recognized asserted at the end of the clock cycle, the processor ignores the data and inserts a wait state instead of terminating the transfer. The processor continues to sample $\overline{TA}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized asserted. The data is then passed to the processor's appropriate memory unit.

When the processor recognizes $\overline{TA}$ at the end of a clock and terminates the bus cycle, $\overline{TIP}$ remains asserted if the processor is ready to begin another bus cycle. Otherwise, the processor negates $\overline{TIP}$ during the first half of the next clock.

7.4.2 Line Read Transfer

The processor uses line read transfers to access a 16-byte operand for a MOVE16 instruction and to support cache line filling. A line read accesses a block of four long words, aligned to a 16-byte memory boundary, by supplying a starting address that points to one of the long words and requiring the memory device to sequentially drive each long word on the data bus. The selected device must internally increment A3 and A2 of the supplied address for each transfer, causing the address to wrap around at the end of the block. The address and transfer attributes supplied by the processor remain stable during the transfers, and the selected device terminates each transfer by driving the long word on

the data bus and asserting $\overline{TA}$. A line transfer performed in this manner with a single address is referred to as a line burst transfer.

The M68040 also supports burst-inhibited line transfers for memory devices that are unable to support bursting. For this type of bus cycle, the selected device supplies the first long word pointed to by the processor address and asserts transfer burst inhibit ($\overline{TBI}$) with $\overline{TA}$ for the first transfer of the line access. The processor responds by terminating the line burst transfer and accessing the remainder of the line, using three long-word read bus cycles. Although the selected device can then treat the line transfer as four, independent, long-word bus cycles, the bus controller still handles the four transfers as a single line transfer and does not allow other unrelated processor accesses or bus arbitration to intervene between the transfers. $\overline{TBI}$ is ignored after the first long-word transfer.

Line reads to support cache line filling can be cache inhibited by asserting transfer cache inhibit ($\overline{TCI}$) with $\overline{TA}$ for the first long-word transfer of the line. The assertion of $\overline{TCI}$ does not affect completion of the line transfer, but the bus controller latches and passes it to the memory controller for use. $\overline{TCI}$ is ignored after the first long-word transfer of a line burst transfer and during the three long-word bus cycles for a burst-inhibited line transfer.

The address placed on the address bus by the processor for line transfers does not necessarily point to the most significant byte of each long word because for a line read, A1 and A0 are copied from the original operand address supplied to the memory unit by the integer unit. These two bits are also unchanged for the three long-word bus cycles for a burst-inhibited line transfer. The selected device should ignore A1 and A0 for long-word and line read transfers.

The address of an instruction fetch will always be aligned to a half-line boundary ($XXXXXXX0 or $XXXXXXX8); therefore, compilers should attempt to locate branch targets on half-line boundaries to minimize branch stalls. For example, if the target of a branch is a two-word instruction located at $1000000C, the following burst sequence will occur upon a cache miss: $10000008, $1000000C, $10000000, then $10000004. The internal pipeline of the M68040 stalls until the second access of the burst (the address of the instruction to be executed) has completed. Figures 7-10 and 7-11 illustrate a flowchart and functional timing diagram for a line read bus transfer.

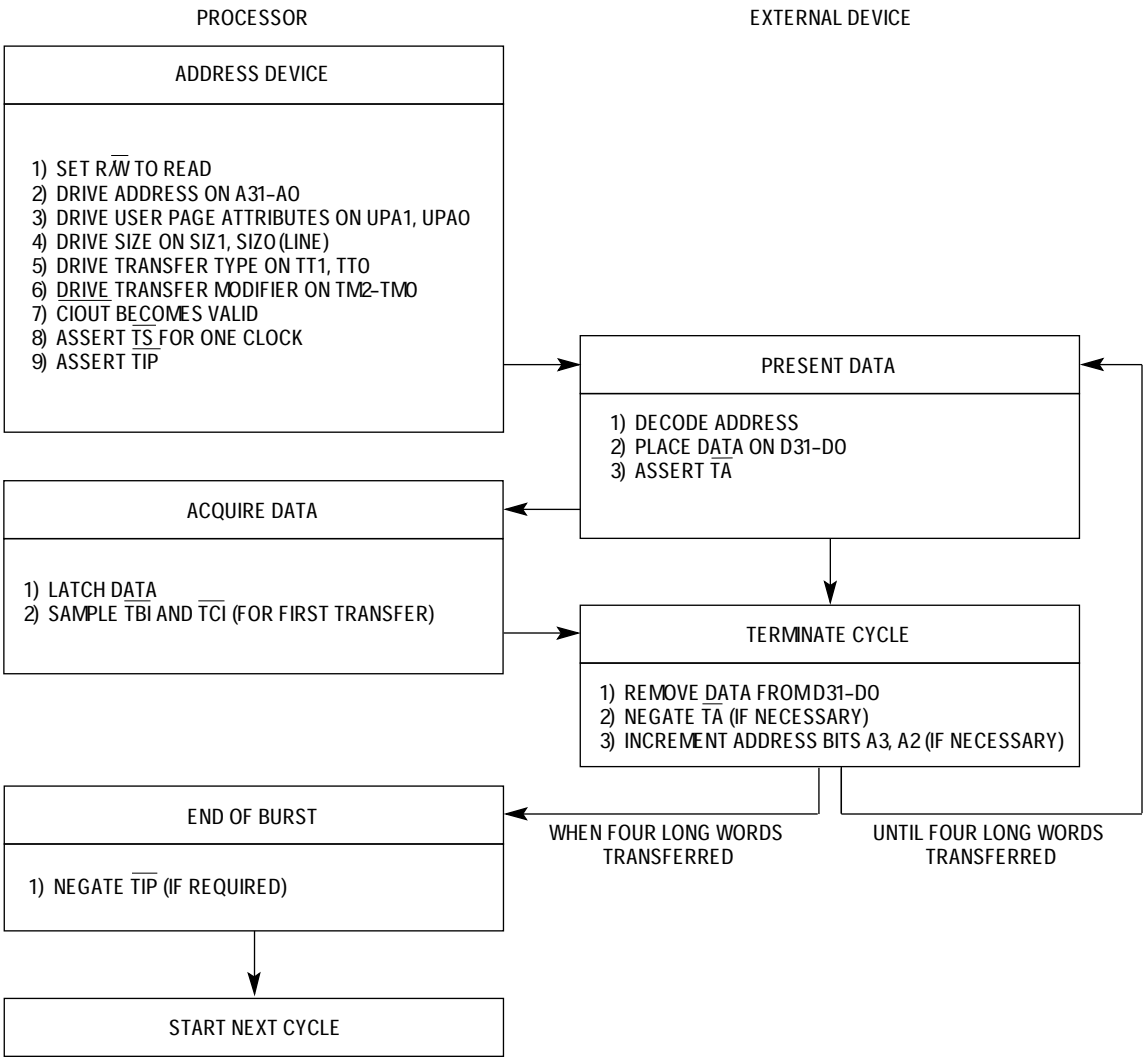
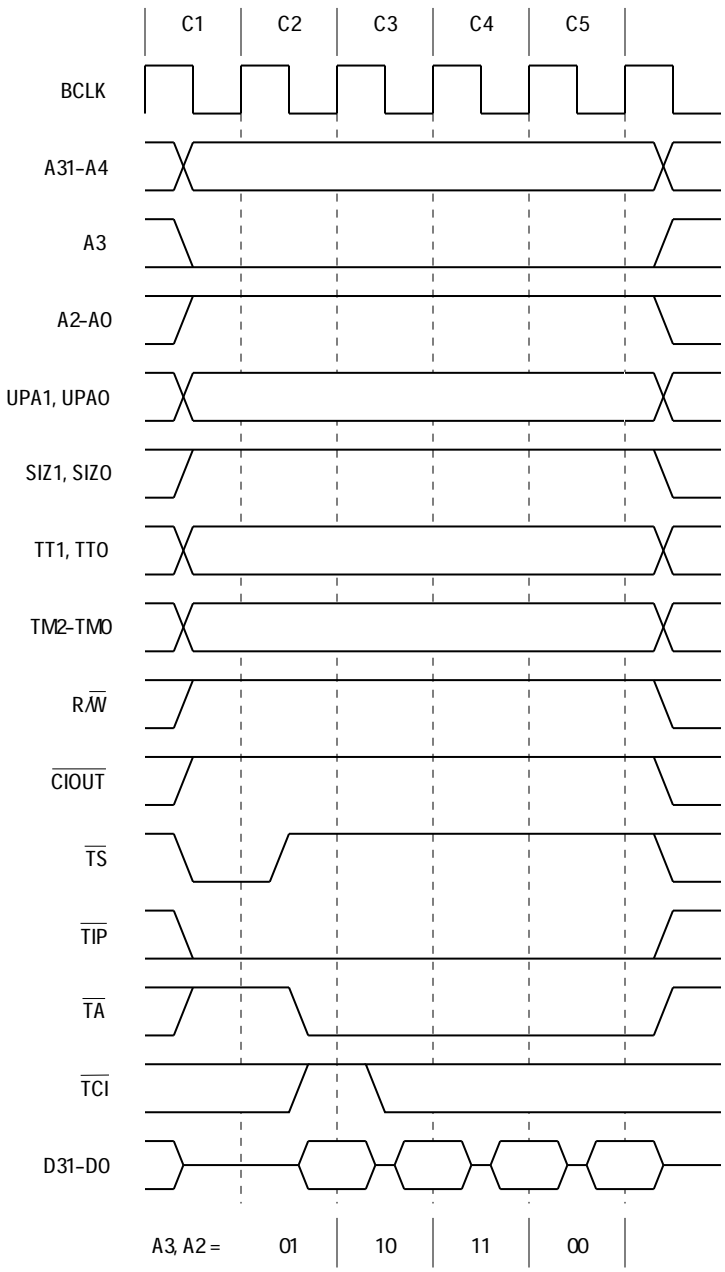


Figure 7-10. Line Read Transfer Flowchart



NOTE: The selected device increments the value of A3 and A2.

Figure 7-11. Line Read Transfer Timing

Clock 1 (C1)

The line read cycle starts in C1. During the first half of C1, the processor places valid values on the address bus and transfer attributes. For user and supervisor mode accesses that are translated by the corresponding memory unit, the UPAx signals are driven with the values from the matching U1 and U0 bits. The TTx and TMx signals identify the specific access type. The R/$\overline{W}$ signal is driven high for a read cycle, and the size signals (SIZx) indicate line size. $\overline{CIOUT}$ is asserted for a MOVE16 operand read if the access is identified as noncachable. Refer to **Section 3 Memory Management Unit**

(Except MC68EC040 and MC68EC040V) for information on the M68040 and MC68LC040 memory units and **Appendix B MC68EC040** for information on the MC68EC040 memory unit.

The processor asserts $\overline{TS}$ during C1 to indicate the beginning of a bus cycle. If not already asserted from a previous bus cycle, $\overline{TIP}$ is also asserted at this time to indicate that a bus cycle is active.

Clock 2 (C2)

During the first half of the first clock after C1, the processor negates $\overline{TS}$. The selected device uses R/$\overline{W}$, SIZ1, and SIZ0 to place the data on the data bus. (The first transfer must supply the long word at the corresponding long-word boundary.) Concurrently, the selected device asserts $\overline{TA}$ and either negates or asserts $\overline{TBI}$ to indicate it can or cannot support a burst transfer. At the end of the first clock cycle after C1, the processor samples the level of $\overline{TA}$, $\overline{TBI}$, and $\overline{TCI}$ and latches the current value on the data bus. If $\overline{TA}$ is asserted, the transfer terminates and the data is passed to the appropriate memory unit. If $\overline{TA}$ is not recognized asserted, the processor ignores the data and inserts wait states instead of terminating the transfer. The processor continues to sample $\overline{TA}$, $\overline{TBI}$, and $\overline{TCI}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized asserted. The latched data and the level on $\overline{TCI}$ are then passed to the appropriate memory unit.

If $\overline{TBI}$ was negated with $\overline{TA}$, the processor continues the cycle with C3. Otherwise, if $\overline{TBI}$ was asserted, the line transfer is burst inhibited, and the processor reads the remaining three long words using long-word read bus cycles. The processor increments A3 and A2 for each read, and the new address is placed on the address bus for each bus cycle. Refer to **7.4.1 Byte, Word, and Long-Word Read Transfers** for information on long-word reads. If no wait states are generated, a burst-inhibited line read completes in eight clocks instead of the five required for a burst read.

Clock 3 (C3)

The processor holds the address and transfer attribute signals constant during C3. The selected device must increment A3 and A2 to reference the next long word to transfer, place the data on the data bus, and assert $\overline{TA}$. At the end of C3, the processor samples the level of $\overline{TA}$ and latches the current value on the data bus. If $\overline{TA}$ is asserted, the transfer terminates, and the second long word of data is passed to the appropriate memory unit. If $\overline{TA}$ is not recognized asserted at the end of C3, the processor ignores the latched data and inserts wait states instead of terminating the transfer. The processor continues to sample $\overline{TA}$ on successive rising edges of BCLK until it is recognized. The latched data is then passed to the appropriate memory unit.

Clock 4 (C4)

This clock is identical to C3 except that once $\overline{TA}$ is recognized asserted, the latched value corresponds to the third long word of data for the burst.

Clock 5 (C5)

This clock is identical to C3 except that once $\overline{TA}$ is recognized, the latched value corresponds to the third long word of data for the burst. After the processor recognizes the last $\overline{TA}$ assertion and terminates the line read bus cycle, $\overline{TIP}$ remains asserted if the processor is ready to begin another bus cycle. Otherwise, the processor negates $\overline{TIP}$ during the first half of the next clock.

Figures 7-12 and 7-13 illustrate a flowchart and functional timing diagram for a burst-inhibited line read bus cycle.

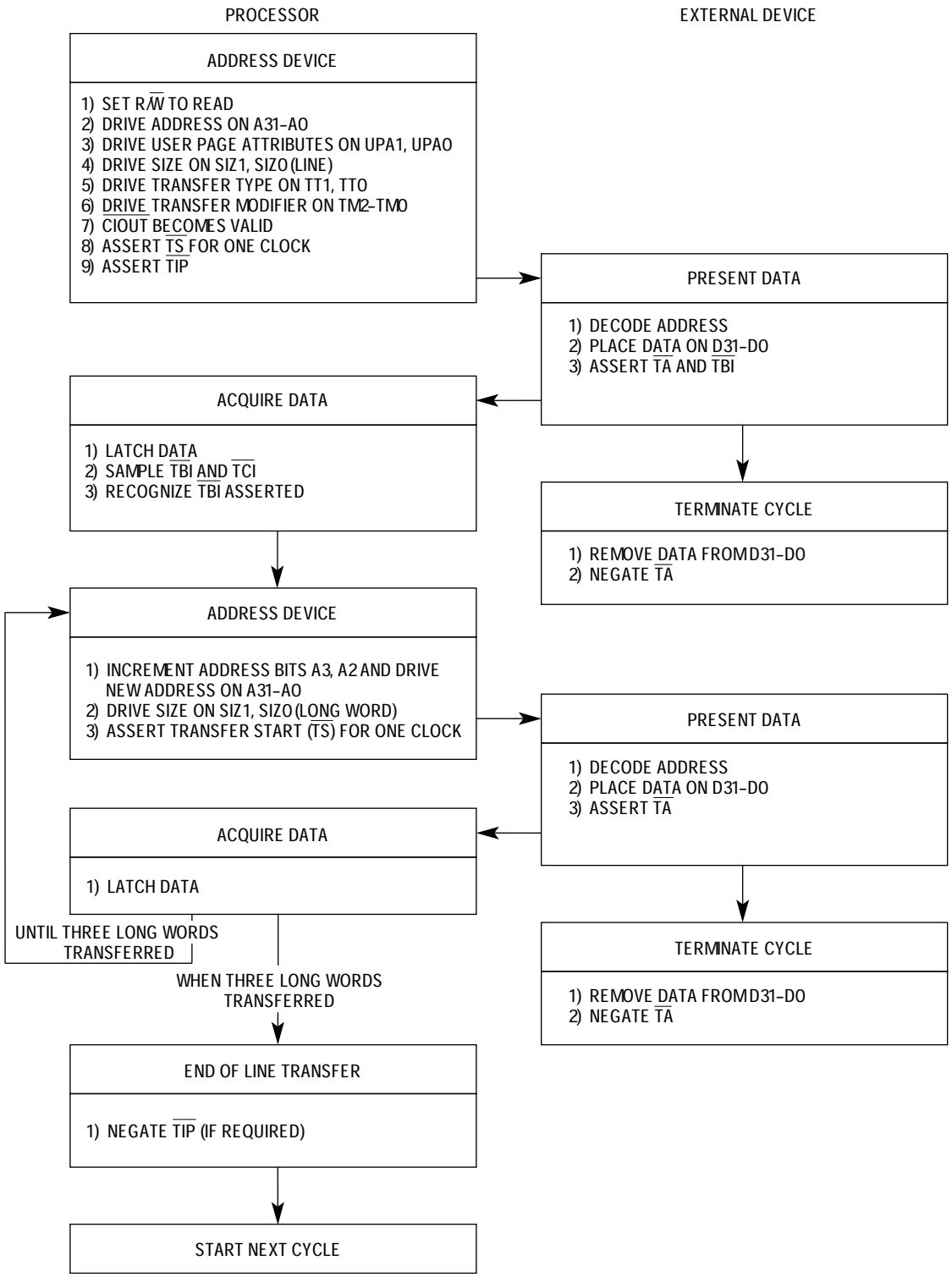


Figure 7-12. Burst-Inhibited Line Read Transfer Flowchart

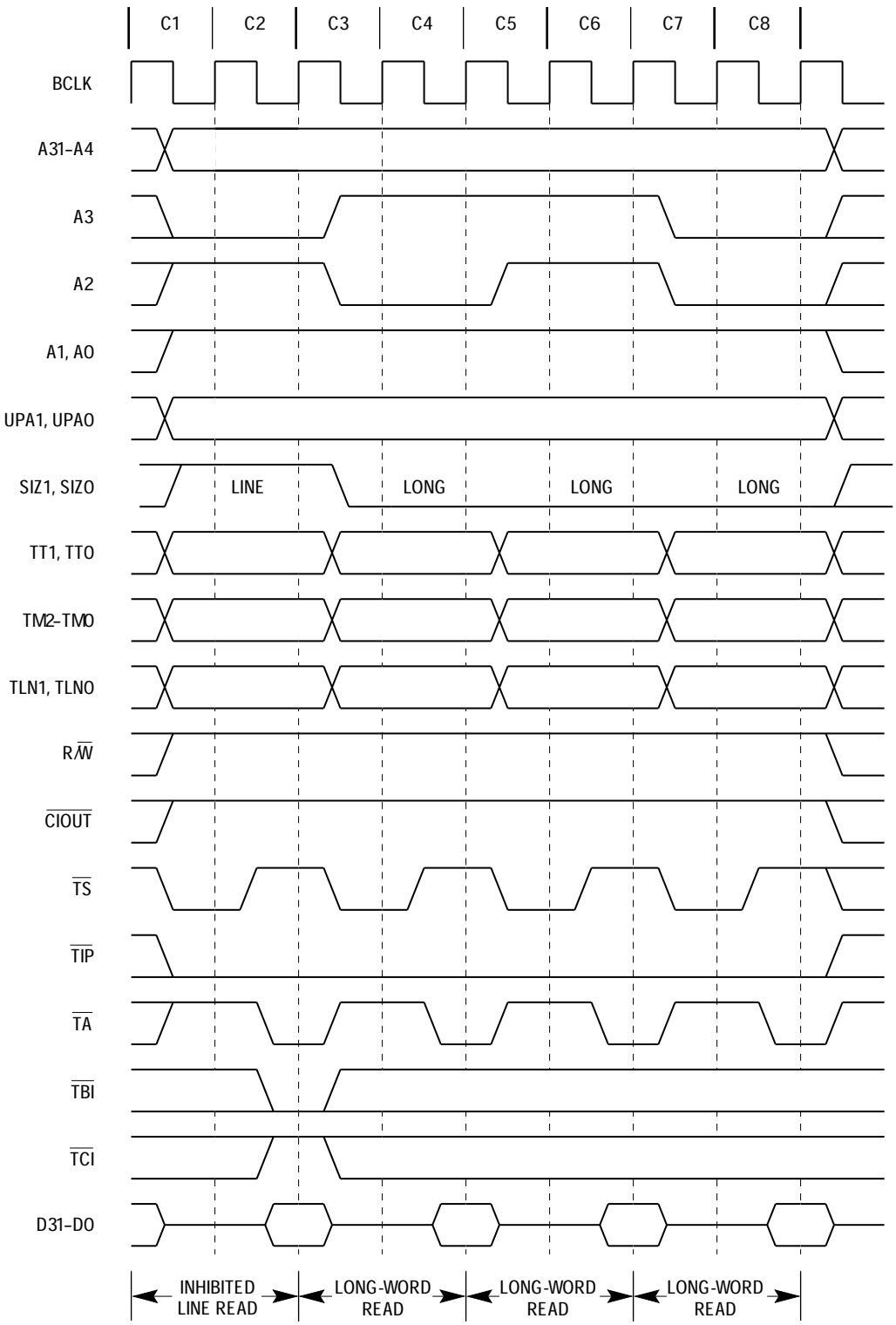


Figure 7-13. Burst-Inhibited Line Read Transfer Timing

7.4.3 Byte, Word, and Long-Word Write Transfers

During a write transfer, the processor transfers data to a memory or peripheral device. The level on the $\overline{TCI}$ signal is ignored by the processor during all write cycles. The bus controller performs byte, word, and long-word write transfers for the following cases:

- Accesses to a disabled cache.
- Accesses to a memory page that is specified noncachable.
- Accesses that are implicitly noncachable (read-modify-write accesses and accesses to an alternate logical address space via the MOVES instruction).
- Writes to write-through pages.
- Accesses that do not allocate in the data cache on a write miss (table updates and exception stacking).
- The first transfer of a line write is terminated with $\overline{TBI}$, forcing completion of the line access using three additional long-word write transfers.
- Cache line pushes for lines containing a single dirty long word.

Figures 7-14 and 7-15 illustrate a flowchart and functional timing diagram for byte, word, and long-word write bus transfers.

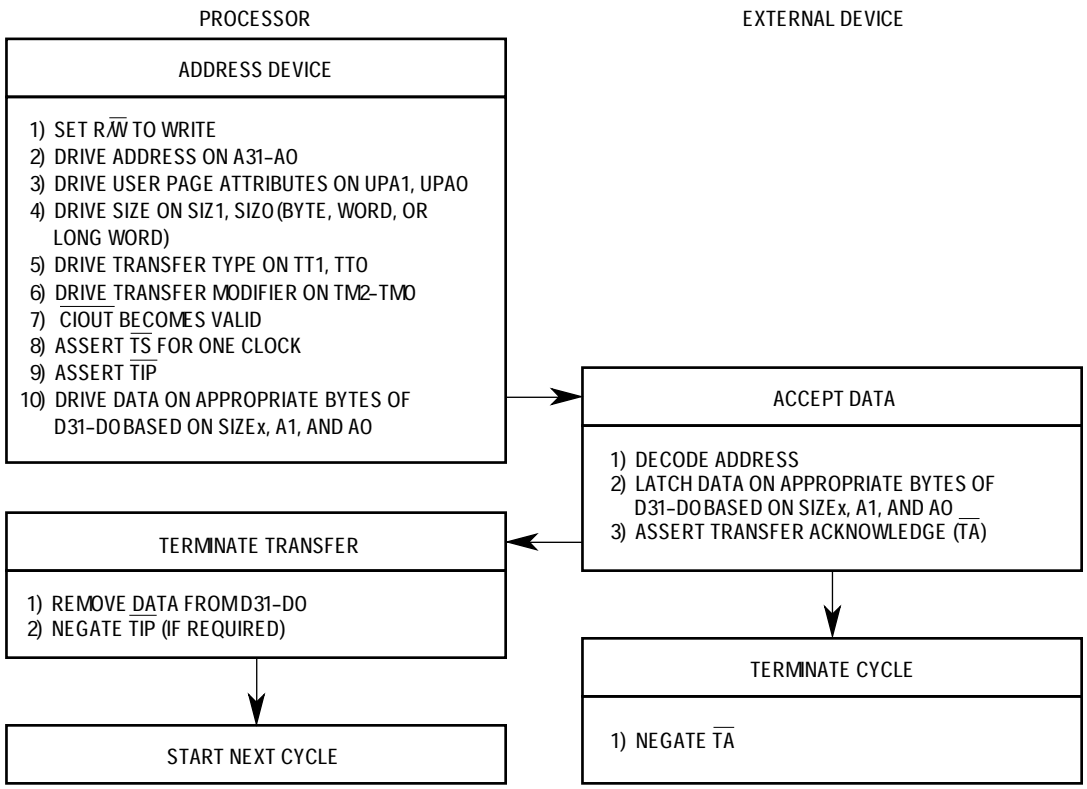


Figure 7-14. Byte, Word, and Long-Word Write Transfer Flowchart

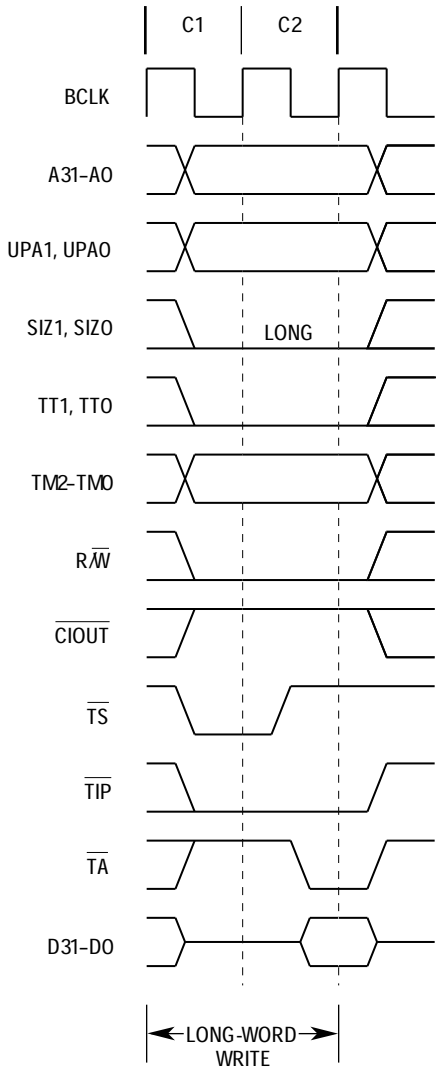


Figure 7-15. Long-Word Write Transfer Timing

Clock 1 (C1)

The write cycle starts in C1. During the first half of C1, the processor places valid values on the address bus and transfer attributes. For user and supervisor mode accesses, which the corresponding memory unit translates, the UPAx signals are driven with the values from the U1 and U0 bits for the area. The TTx and TMx signals identify the specific access type. The R/$\overline{W}$ signal is driven low for a write cycle. $\overline{CIOUT}$ is asserted if the access is identified as noncachable or if the access references an alternate address space. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for information on the M68040 and MC68LC040 memory units and **Appendix B MC68EC040** for information on the MC68EC040 memory unit.

The processor asserts $\overline{TS}$ during C1 to indicate the beginning of a bus cycle. If not already asserted from a previous bus cycle, the $\overline{TIP}$ signal is also asserted at this time to indicate that a bus cycle is active.

Clock 2 (C2)

During the first half of the clock after C1, the processor negates $\overline{TS}$ and drives the appropriate bytes of the data bus with the data to be written. All other bytes are driven with undefined values. The selected device uses R/$\overline{W}$, SIZ1, SIZ0, A1, A0, and $\overline{CIOUT}$ to latch only the required information on the data bus. With the exception of R/$\overline{W}$ and $\overline{CIOUT}$, these signals also select any or all of the bytes (D31–D24, D23–D16, D15–D8, and D7–D0). If the first clock after C1 is not a wait state, then the selected peripheral device asserts the $\overline{TA}$ signal.

At the end of the first clock cycle after C1, the processor samples the level of $\overline{TA}$, terminating the bus cycle if $\overline{TA}$ is asserted. If $\overline{TA}$ is not recognized asserted at the end of the clock cycle, the processor ignores the data and inserts a wait state instead of terminating the transfer. The processor continues to sample $\overline{TA}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized asserted. The data bus then three-states and the bus cycle ends.

When the processor recognizes $\overline{TA}$ at the clock edge and terminates the bus cycle, $\overline{TIP}$ remains asserted if the processor is ready to begin another bus cycle. Otherwise, the processor negates $\overline{TIP}$ during the first half of the next clock. The processor also three-states the data bus during the first half of the next clock following termination of the write transfer.

7.4.4 Line Write Transfers

The processor uses line write bus cycles to access a 16-byte operand for a MOVE16 instruction and to support cache line pushes. Both burst and burst-inhibited transfers are supported. Figures 7-16 and 7-17 illustrate a flowchart and functional timing diagram for a line write bus cycle.

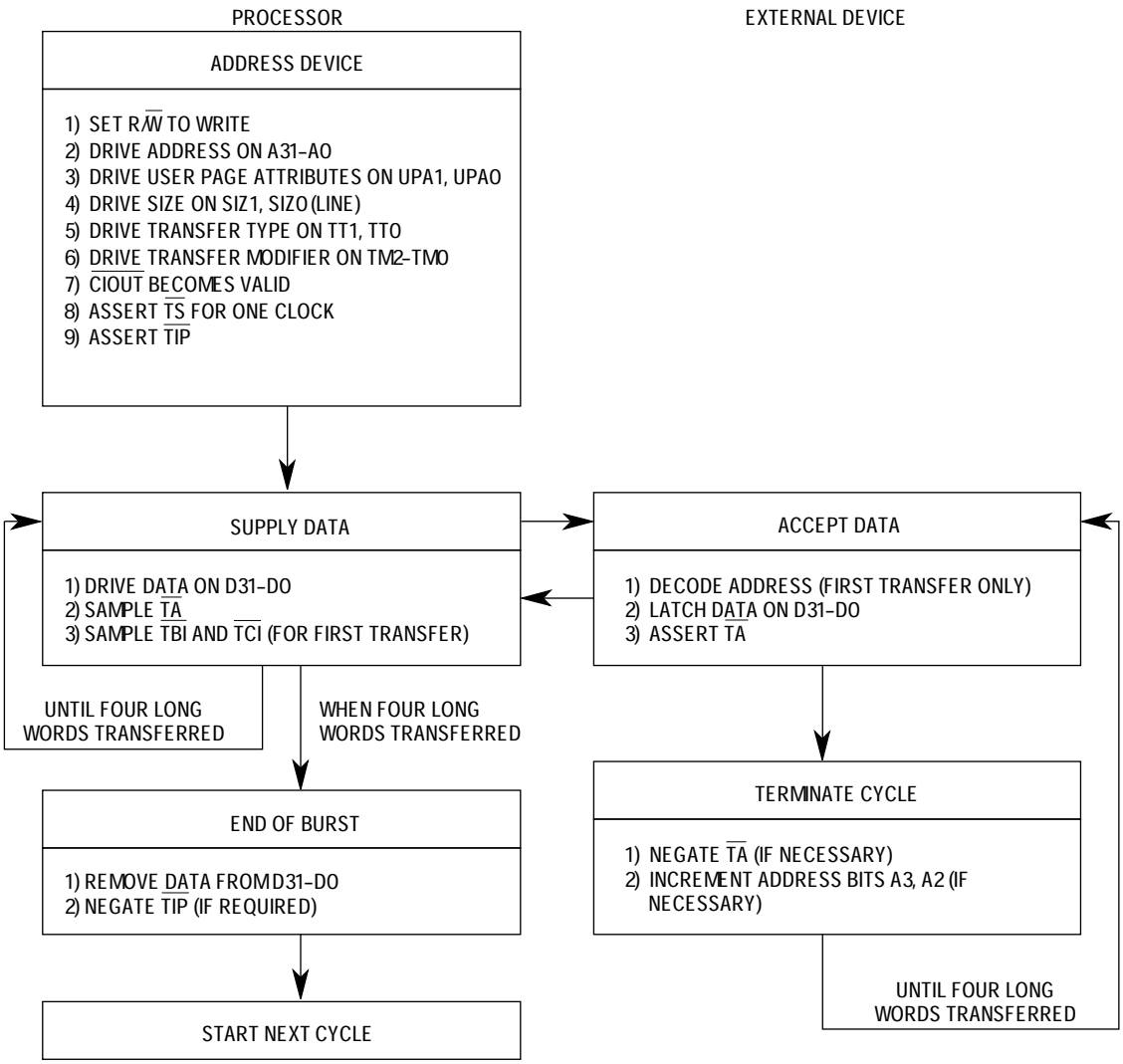
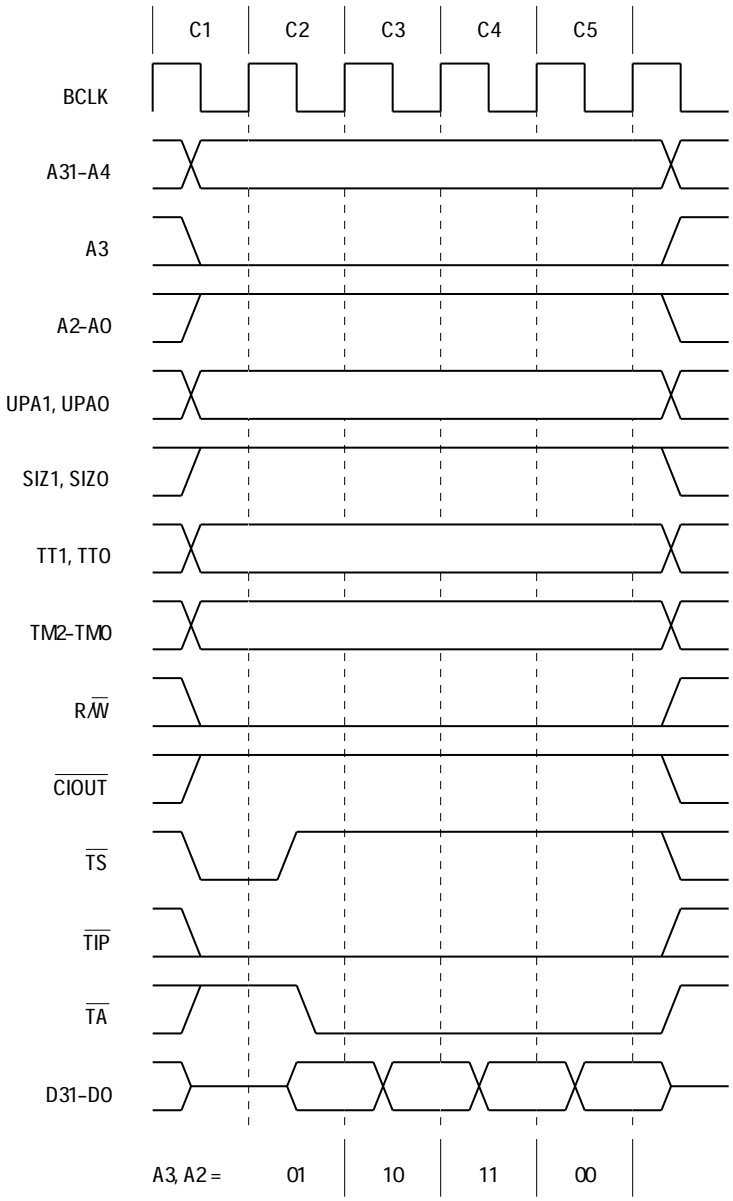


Figure 7-16. Line Write Transfer Flowchart



NOTE: The selected device increments the value of A3 and A2.

Figure 7-17. Line Write Transfer Timing

Clock 1 (C1)

The line write cycle starts in C1. During the first half of C1, the processor places valid values on the address bus and transfer attributes. For user and supervisor mode accesses that are translated by the corresponding memory unit, UPAx signals are driven with the values from the matching U1 and U0 bits. The TTx and TMx signals identify the specific access type. The R/$\overline{W}$ signal is driven low for a write cycle, and SIZ1 and SIZ0 indicate line size. $\overline{CIOUT}$ is asserted for a MOVE16 operand read if the access is identified as noncachable. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for information on the M68040 and

MC68LC040 memory units and **Appendix B MC68EC040** for information on the MC68EC040 memory unit.

The processor asserts $\overline{TS}$ during C1 to indicate the beginning of a bus cycle. If not already asserted from a previous bus cycle, the $\overline{TIP}$ signal is also asserted at this time to indicate that a bus cycle is active.

Clock 2 (C2)

During the first half of the first clock after C1, the processor negates $\overline{TS}$ and drives the data bus with the data to be written. The selected device uses R/$\overline{W}$, SIZ1, and SIZ0 to latch the data on the data bus. Concurrently, the selected device asserts $\overline{TA}$ and either negates or asserts $\overline{TBI}$ to indicate it can or cannot support a burst transfer. At the end of the first clock after C1, the processor samples the level of $\overline{TA}$ and $\overline{TBI}$. If $\overline{TA}$ is asserted, the transfer terminates. If $\overline{TA}$ is not recognized asserted, the processor inserts wait states instead of terminating the transfer. The processor continues to sample $\overline{TA}$ and $\overline{TBI}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized asserted.

If $\overline{TBI}$ was negated with $\overline{TA}$, the processor continues the cycle with C3. Otherwise, if $\overline{TBI}$ was asserted, the line transfer is burst inhibited, and the processor writes the remaining three long words using long-word write bus cycles. Only in this case does the processor increment A3 and A2 for each write, and the new address is placed on the address bus for each bus cycle. Refer to **7.4.3 Byte, Word, and Long-Word Write Transfers** for information on long-word writes. If no waits states are generated, a burst-inhibited line write completes in eight clocks instead of the five required for a burst write.

Clock 3 (C3)

The processor drives the second long word of data on the data bus and holds the address and transfer attribute signals constant during C3. The selected device increments A3 and A2 to reference the next long word, latches this data from the data bus, and asserts $\overline{TA}$. At the end of C3, the processor samples the level of $\overline{TA}$; if $\overline{TA}$ is asserted, the transfer terminates. If $\overline{TA}$ is not recognized asserted at the end of C3, the processor inserts wait states instead of terminating the transfer. The processor continues to sample $\overline{TA}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized asserted.

Clock 4 (C4)

This clock is identical to C3 except that the value driven on the data bus corresponds to the third long word of data for the burst.

Clock 5 (C5)

This clock is identical to C3 except that the value driven on the data bus corresponds to the fourth long word of data for the burst. After the processor recognizes the last $\overline{TA}$ assertion and terminates the line write bus cycle, $\overline{TIP}$ remains asserted if the processor is ready to begin another bus cycle. Otherwise, the processor negates $\overline{TIP}$ during the first half of the next clock. The processor also three-states the data bus during the first half of the next clock following termination of the write cycle.

7.4.5 Read-Modify-Write Transfers (Locked Transfers)

The read-modify-write transfer performs a read, conditionally modifies the data in the processor, and writes the data out to memory. In the M68040, this operation can be indivisible, providing semaphore capabilities for multiprocessor systems. During the entire read-modify-write sequence, the M68040 asserts the $\overline{\text{LOCK}}$ signal to indicate that an indivisible operation is occurring and asserts the $\overline{\text{LOCKE}}$ signal for the last transfer to indicate completion of the locked sequence. The external arbiter can use the $\overline{\text{LOCK}}$ and $\overline{\text{LOCKE}}$ signals to prevent arbitration of the bus during locked processor sequences. External bus arbitrations can use $\overline{\text{LOCKE}}$ to support bus arbitration between consecutive read-modify-write cycles. A read-modify-write operation is treated as noncachable. If the access hits in the data cache, it invalidates a matching valid entry and pushes a matching dirty entry. The read-modify-write transfer begins after the line push (if required) is complete; however, $\overline{\text{LOCK}}$ may assert during the line push bus cycle.

The TAS, CAS, and CAS2 instructions are the only M68040 instructions that utilize read-modify-write transfers. Some page descriptor updates during translation table searches also use read-modify-write transfers. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for information about table searches.

The read-modify-write transfer for the CAS and CAS2 instructions in the M68040 differs from those used by previous members of the M68000 family. If an operand does not match one of these instructions, the M68040 still executes a single write transfer to terminate the locked sequence with $\overline{\text{LOCKE}}$ asserted. For the CAS instruction, the value read from memory is written back; for the CAS2 instruction, the second operand read is written back. Figure 7-18 illustrates a functional timing diagram for a TAS instruction read-modify-write bus transfer.

Clock 1 (C1)

The read cycle starts in C1. During the first half of C1, the processor places valid values on the address bus and transfer attributes. $\overline{\text{LOCK}}$ is asserted to identify a locked read-modify-write bus cycle. For user and supervisor mode accesses, which the corresponding memory unit translates, the UPAx signals are driven with the values from the matching U1 and U0 bits. The TTx and TMx signals identify the specific access type. R/$\overline{\text{W}}$ is driven high for a read cycle. $\overline{\text{CIOUT}}$ is asserted if the access is identified as noncachable. The processor asserts $\overline{\text{TS}}$ during C1 to indicate the beginning of a bus cycle. If not already asserted from a previous bus cycle, the $\overline{\text{TIP}}$ signal is also asserted at this time to indicate that a bus cycle is active. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for information on the M68040 and MC68LC040 memory units and **Appendix B MC68EC040** for information on the MC68EC040 memory unit.

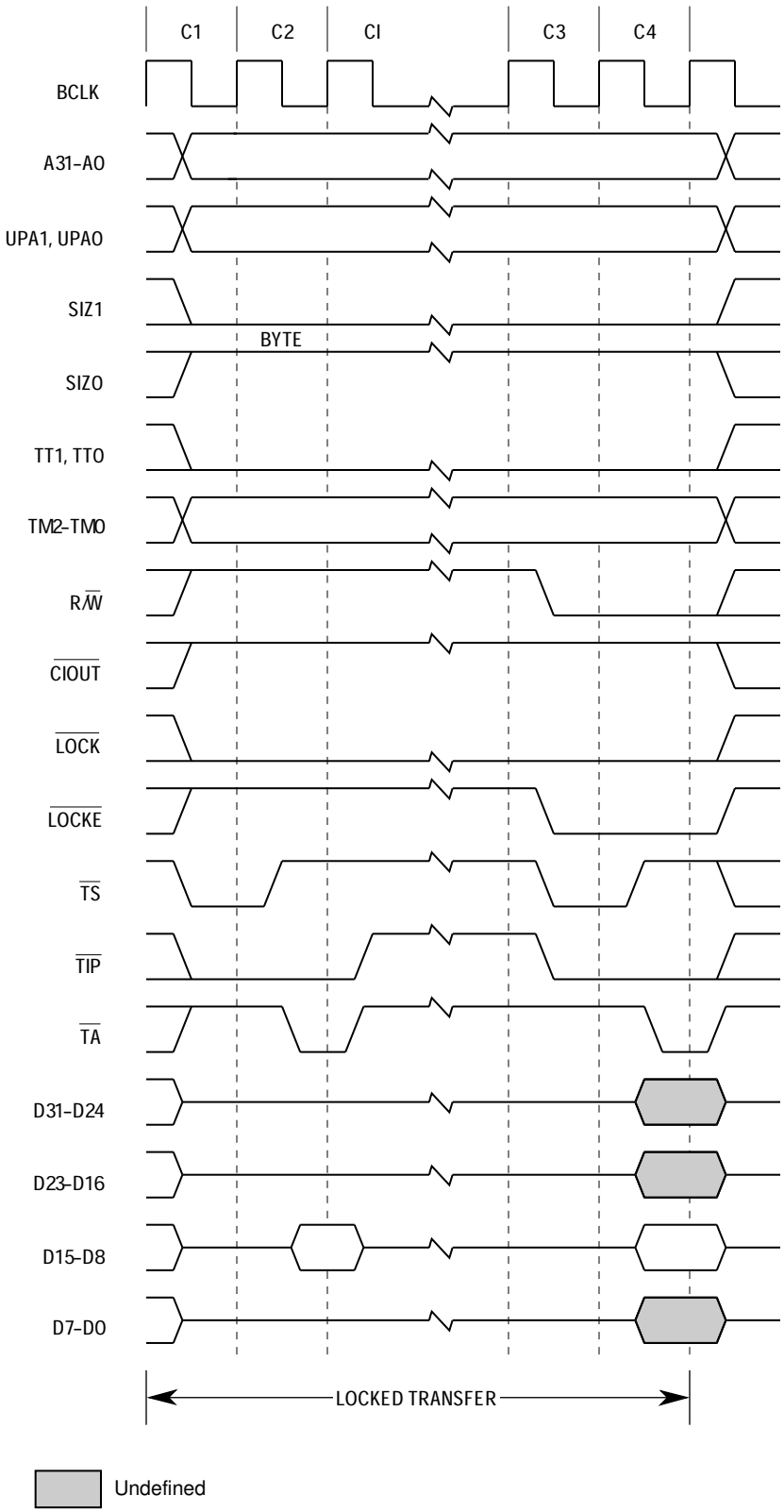


Figure 7-18. Locked Transfer for TAS Instruction Timing

Clock 2 (C2)

During the first half of the first clock cycle after C1, the processor negates $\overline{TS}$**.** The selected device uses R/$\overline{W}$, SIZ1, SIZ0, A1, and A0 to place its information on the data bus. With the exception of R/$\overline{W}$, these signals also select any or all of the bytes (D24–D31, D16–D23, D15–D8, and D7–D0). Concurrently, the selected device asserts $\overline{TA}$. At the end of the first clock cycle after C1, the processor samples the level of $\overline{TA}$ and latches the current value on the data bus. If $\overline{TA}$ is asserted, the read transfer terminates, and the latched data is passed to the appropriate memory unit. If $\overline{TA}$ is not recognized asserted, the processor ignores the data and appends a wait state instead of terminating the transfer. The processor continues to sample $\overline{TA}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized as asserted. The latched data is then passed to the appropriate memory unit. If more than one read cycle is required to read in the operand(s), C1 and C2 are repeated accordingly.

When the processor recognizes $\overline{TA}$ at the end of the last read transfer for the locked bus cycle, it negates $\overline{TIP}$ during the first half of the next clock.

Clock Idle (CI)

The processor does not assert any new control signals during the idle clock states, but it may begin the modify portion of the cycle at this time. The R/$\overline{W}$ signal remains in the read mode until C3 to prevent bus conflicts with the preceding read portion of the cycle; the data bus is not driven until C4.

Clock 3 (C3)

During the first half of C3, the processor places valid values on the address bus and transfer attributes and drives R/$\overline{W}$ low for a write cycle. The processor asserts $\overline{TS}$ to indicate the beginning of a bus cycle. The $\overline{TIP}$ signal is also asserted at this time to indicate that a bus cycle is active.

$\overline{LOCKE}$ is asserted during C3 for the last write transfer of the locked sequence. If multiple write transfers are required for misaligned operands or multiple operands, $\overline{LOCKE}$ is asserted only for the final write transfer. The external arbiter can use this indication to distinguish between two back-to-back locked bus cycles and allow arbitration between them.

Clock 4 (C4)

During the first half of C4, the processor negates $\overline{TS}$ and drives the appropriate bytes of the data bus with the data to be written. All other bytes are driven with undefined values. The selected device uses R/$\overline{W}$**,** SIZ1, SIZ0, A1, and A0 to latch the information on the data bus. Any or all of the bytes (D31–D24, D23–D16, D15–D8, and D7–D0) are selected by SIZ1, SIZ0, A1, and A0. Concurrently, the selected device asserts $\overline{TA}$. At the end of C4, the processor samples the level of $\overline{TA}$; if $\overline{TA}$ is asserted, the bus cycle terminates. If $\overline{TA}$ is not recognized asserted at the end of C4, the processor appends a wait state instead of terminating the transfer. The processor continues to sample the $\overline{TA}$ signal on successive rising edges of BCLK until it is recognized asserted.

When the processor recognizes $\overline{TA}$ at the end of a clock, the bus cycle is terminated, but $\overline{TIP}$ remains asserted if the processor is ready to begin another bus cycle. Otherwise, the processor negates $\overline{TIP}$ during the first half of the next clock. The processor also three-states the data bus during the first half of the next clock following termination of the write cycle. When the last write transfer is terminated, $\overline{LOCKE}$ is negated. The processor also negates $\overline{LOCK}$ if the next bus cycle is not a read-modify-write.

7.5 ACKNOWLEDGE BUS CYCLES

Bus transfers with transfer type signals TT1 and TT0 = $3 are classified as acknowledge bus cycles. The following paragraphs describe interrupt acknowledge and breakpoint acknowledge bus cycles that use this encoding.

7.5.1 Interrupt Acknowledge Bus Cycles

When a peripheral device requires the services of the M68040 or is ready to send information that the processor requires, it can 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 ($\overline{IPL2}$–$\overline{IPL0}$) to signal an interrupt condition to the processor and to specify the priority level for the condition. Refer to **Section 8 Exception Processing** for a discussion on the $\overline{IPLx}$ levels and $\overline{IPEND}$.

The status register (SR) of the M68040 contains an interrupt priority mask (I2–I0 bits). 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 a pending interrupt. $\overline{IPL2}$–$\overline{IPL0}$ must maintain the interrupt request level until the M68040 acknowledges the interrupt to guarantee that the interrupt is recognized. The M68040 continuously samples $\overline{IPL2}$–$\overline{IPL0}$ on consecutive rising edges of BCLK 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 can be recognized. Figure 7-19 is a flowchart of the procedure for making an interrupt pending.

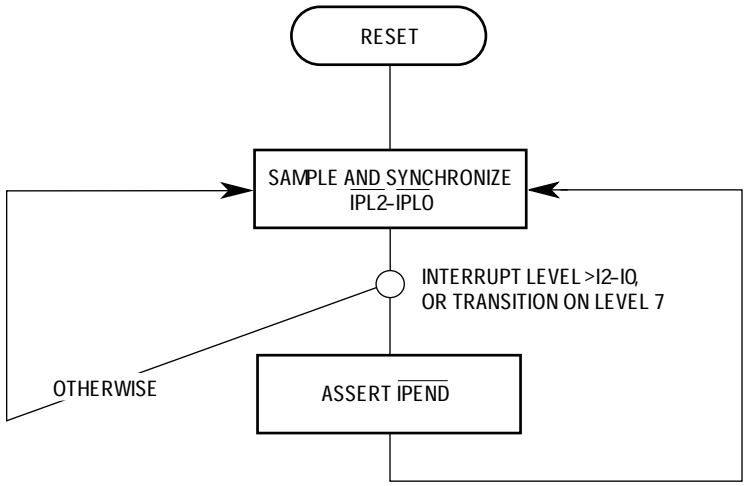


Figure 7-19. Interrupt Pending Procedure

The M68040 asserts IPEND when an interrupt request is pending. Figure 7-20 illustrates the assertion of IPEND relative to the assertion of an interrupt level on the IPLx signals. IPEND signals external devices that an interrupt exception will be taken at an upcoming instruction boundary (following any higher priority exception). The IPEND signal negates after the processor recognizes the internal interrupt acknowledge and can precede the external interrupt acknowledge bus cycle.

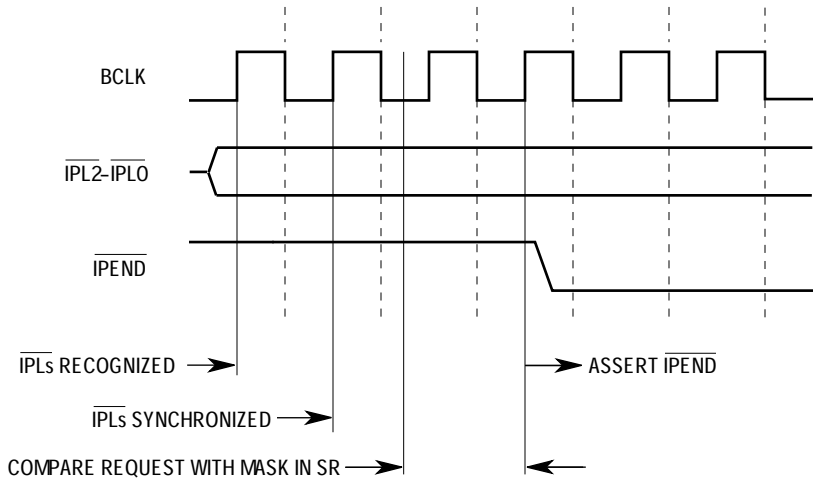


Figure 7-20. Assertion of IPEND

The M68040 takes an interrupt exception for a pending interrupt within one instruction boundary after processing any other pending exception with a higher priority. Thus, the M68040 executes at least one instruction in an interrupt exception handler before recognizing another interrupt request. The following paragraphs describe the various kinds of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing. Table 7-4 provides a summary of the possible interrupt acknowledge terminations and the exception processing results.

Table 7-4. Interrupt Acknowledge Termination Summary

			Termination Condition
High	High	Don't Care	Insert Waits
High	Low	Don't Care	Take Spurious Interrupt Exception
Low	High	High	Latch Vector Number on D7–D0 and Take Interrupt Exception
Low	High	Low	Take Autovectored Interrupt Exception
Low	Low	Don't Care	Retry Interrupt Acknowledge Cycle

7.5.1.1 INTERRUPT ACKNOWLEDGE BUS CYCLE (TERMINATED NORMALLY). When the M68040 processes an interrupt exception, it performs an interrupt acknowledge bus cycle to obtain the vector number that contains the starting location of the interrupt exception handler. Some interrupting devices have programmable vector registers that contain the interrupt vectors for the exception handlers they use. Other interrupting conditions or devices cannot supply a vector number and use the autovector bus cycle described in **7.5.1.2 Autovector Interrupt Acknowledge Bus Cycle**.

The interrupt acknowledge bus cycle is a read transfer. It differs from a normal read cycle in the following respects:

1. TT1 and TT0 = $3 to indicate an acknowledged bus cycle.
2. Address signals A31–A0 are set to all ones ($FFFFFFFF).
3. TM2–TM0 are set to the interrupt request level (the inverted values of $\overline{IPL2}$–$\overline{IPL0}$).

The responding device places the vector number on the data bus during the interrupt acknowledge bus cycle, and the cycle is terminated normally with $\overline{TA}$. Figures 7-21 and 7-22 illustrate a flowchart and functional timing diagram for an interrupt acknowledge cycle terminated with $\overline{TA}$.

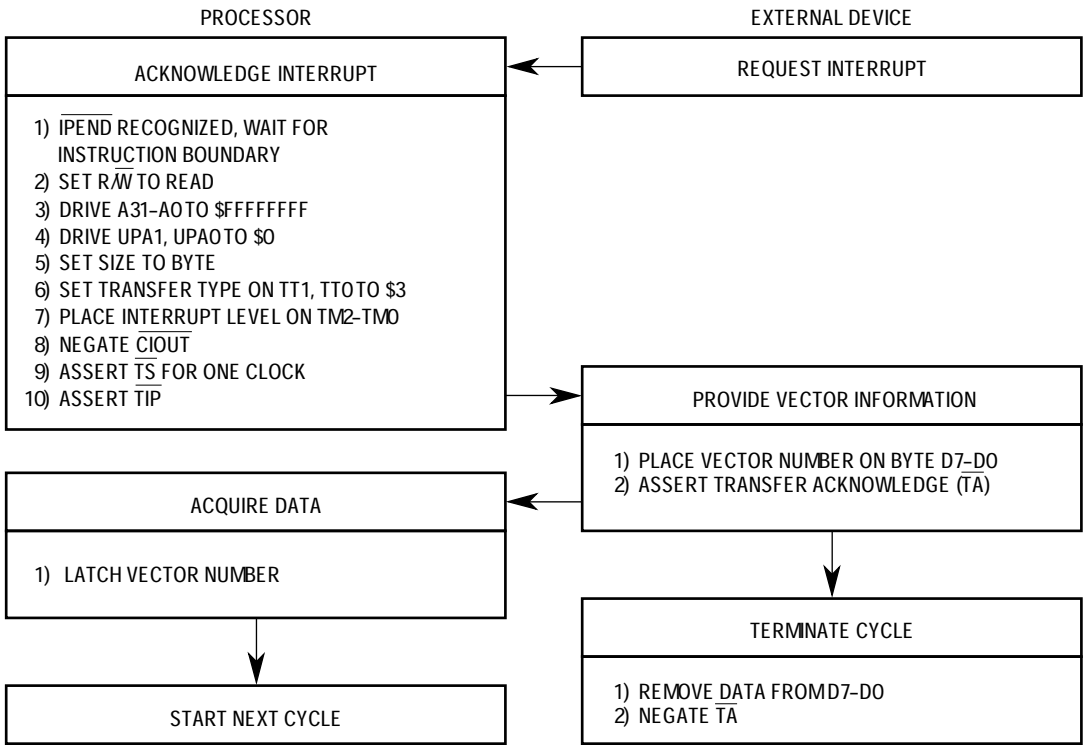


Figure 7-21. Interrupt Acknowledge Bus Cycle Flowchart

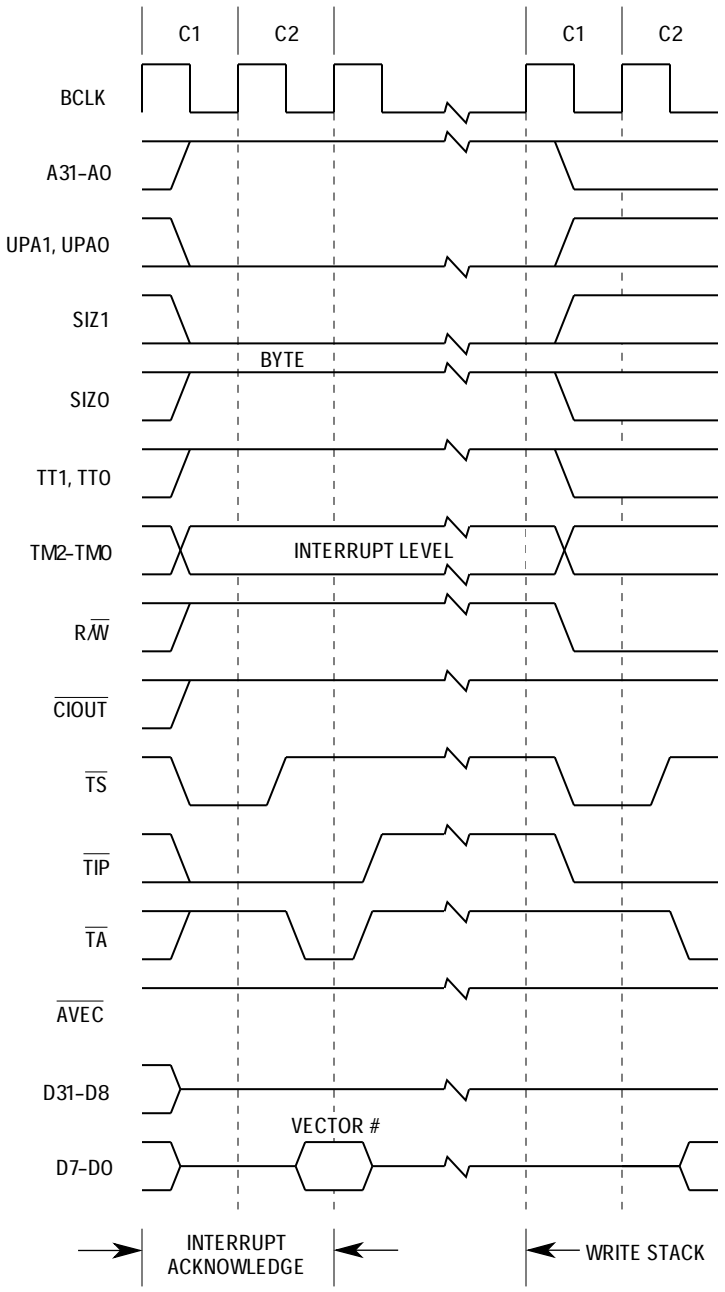


Figure 7-22. Interrupt Acknowledge Bus Cycle Timing

7.5.1.2 AUTOVECTOR INTERRUPT ACKNOWLEDGE BUS CYCLE. When the interrupting device cannot supply a vector number, it requests an automatically generated vector (autovector). Instead of placing a vector number on the data bus and asserting $\overline{TA}$, the device asserts the autovector ($\overline{AVEC}$) signal with $\overline{TA}$ to terminate the cycle. $\overline{AVEC}$ is only sampled with $\overline{TA}$ asserted. $\overline{AVEC}$ can be grounded if all interrupt requests are autovectored.

The vector number supplied in an autovector operation is derived from the interrupt priority level of the current interrupt. When the $\overline{AVEC}$ signal is asserted with $\overline{TA}$ during an interrupt acknowledge bus cycle, the M68040 ignores the state of the data bus and internally

generates the vector number, which is the sum of the interrupt priority level plus 24 ($18). There are seven distinct autovectors that can be used, corresponding to the seven levels of interrupts available with $\overline{IPL2}$–$\overline{IPL0}$ signals. Figure 7-23 illustrates a functional timing diagram for an autovector operation.

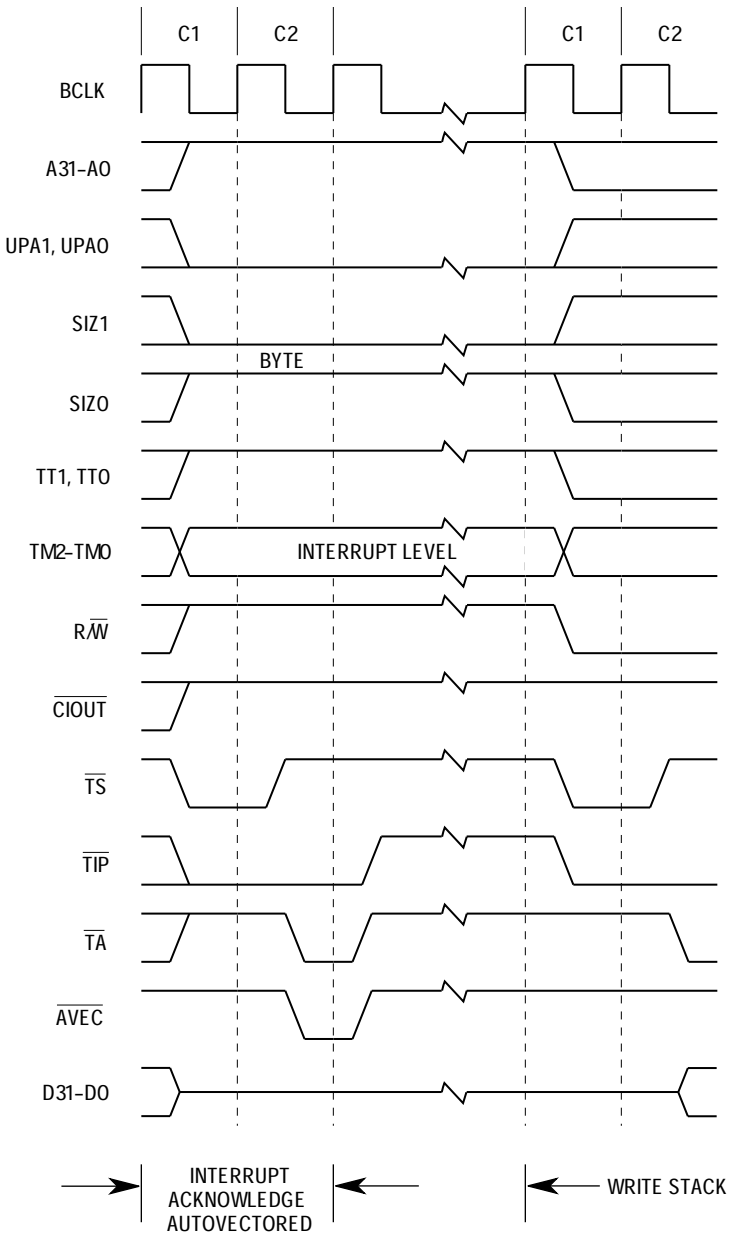


Figure 7-23. Autovector Interrupt Acknowledge Bus Cycle Timing

7.5.1.3 SPURIOUS INTERRUPT ACKNOWLEDGE BUS CYCLE. When a device does not respond to an interrupt acknowledge bus cycle with $\overline{TA}$, or $\overline{AVEC}$ and $\overline{TA}$, the external logic typically returns the transfer error acknowledge signal ($\overline{TEA}$). In this case, the M68040 automatically generates the spurious interrupt vector number 24 ($18) instead of the interrupt vector number. If $\overline{TA}$ and $\overline{TEA}$ are both asserted, the processor retries the cycle.

7.5.2 Breakpoint Interrupt Acknowledge Bus Cycle

The execution of a breakpoint instruction (BKPT) generates the breakpoint interrupt acknowledge bus cycle. An acknowledged access is indicated with TT1 and TT0 = $3, address A31–A0 = $00000000, and TM2–TM0 = $0. When the external device terminates the cycle with either $\overline{TA}$ or $\overline{TEA}$, the processor takes an illegal instruction exception. Figures 7-24 and 7-25 illustrate a flowchart and functional timing diagram for a breakpoint interrupt acknowledge transfer.

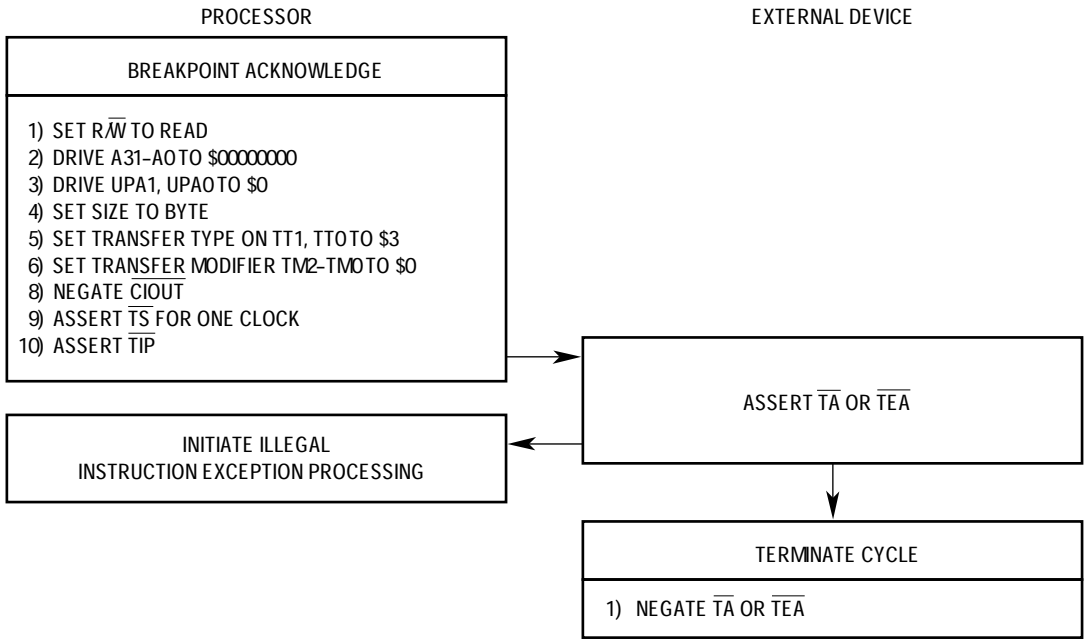


Figure 7-24. Breakpoint Interrupt Acknowledge Bus Cycle Flowchart

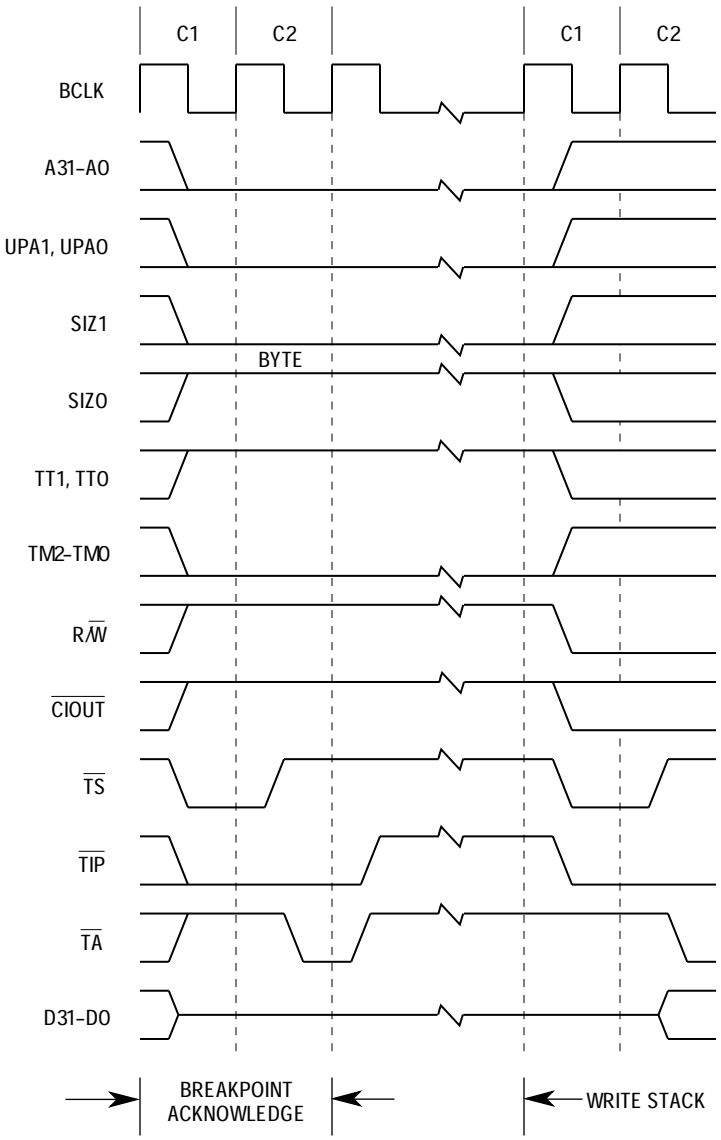


Figure 7-25. Breakpoint Interrupt Acknowledge Bus Cycle Timing

7.6 BUS EXCEPTION CONTROL CYCLES

The M68040 bus architecture requires assertion of $\overline{TA}$ from an external device to signal that a bus cycle is complete. $\overline{TA}$ is not asserted in the following cases:

- The external device does not respond.
- No interrupt vector is provided.
- Various other application-dependent errors occur.

External circuitry can provide $\overline{TEA}$ when no device responds by asserting $\overline{TA}$ within an appropriate period of time after the processor begins the bus cycle. This allows the cycle to terminate and the processor to enter exception processing for the error condition. $\overline{TEA}$ can also be asserted in combination with $\overline{TA}$ to cause a retry of a bus cycle in error.

To properly control termination of a bus cycle for a bus error or retry condition, $\overline{TA}$ and $\overline{TEA}$ must be asserted and negated for the same rising edge of BCLK. Table 7-5 lists the control signal combinations and the resulting bus cycle terminations. Bus error and retry terminations during burst cycles operate as described in **7.4.2 Line Read Transfers** and **7.4.4 Line Write Transfers**.

Table 7-5. and Assertion Results

Case No.			Result
1	High	Low	Bus Error—Terminate and Take Bus Error Exception, Possibly Deferred
2	Low	Low	Retry Operation—Terminate and Retry
3	Low	High	Normal Cycle Terminate and Continue
4	High	High	Insert Wait States

7.6.1 Bus Errors

The system hardware can use the $\overline{TEA}$ signal to abort the current bus cycle when a fault is detected. A bus error is recognized during a bus cycle when $\overline{TA}$ is negated and $\overline{TEA}$ is asserted. When the processor recognizes a bus error condition for an access, the access is terminated immediately. A line access that has $\overline{TEA}$ asserted for one of the four long-word transfers aborts without completing the remaining transfers, regardless of whether the line transfer uses a burst or burst-inhibited access.

When $\overline{TEA}$ is asserted to terminate a bus cycle, the M68040 can enter access error exception processing immediately following the bus cycle, or it can defer processing the exception. The instruction prefetch mechanism requests instruction words from the instruction memory unit before it is ready to execute them. If a bus error occurs on an instruction fetch, the processor does not take the exception until it attempts to use the instruction. Should an intervening instruction cause a branch or should a task switch occur, the access error exception for the unused access does not occur. Similarly, if a bus error is detected on the second, third, or fourth long-word transfer for a line read access, an access error exception is taken only if the execution unit is specifically requesting that long word. Otherwise, the line is not placed in the cache, and the processor repeats the line access when another access references the line. If a misaligned operand spans two long words in a line, a bus error on either the first or second transfer for the line causes exception processing to begin immediately. A bus error termination for any write accesses or for read accesses that reference data specifically requested by the execution unit causes the processor to begin exception processing immediately. Refer to **Section 8 Exception Processing** for details of access error exception processing.

When a bus error terminates an access, the contents of the corresponding cache can be affected in different ways, depending on the type of access. For a cache line read to replace a valid instruction or data cache line, the cache line being filled is invalidated before the bus cycle begins and remains invalid if the replacement line access is terminated with a bus error. If a dirty data cache line is being replaced and a bus error occurs during the replacement line read, the dirty line is restored from an internal push

buffer into the cache to eliminate an unnecessary push access. If a bus error occurs during a data cache push, the corresponding cache line remains valid (with the new line data) if the line push follows a replacement line read, or is invalidated if a CPUSH instruction explicitly forces the push. Write accesses to memory pages specified as write-through by the data memory unit update the corresponding cache line before accessing memory. If a bus error occurs during a memory access, the cache line remains valid with the new data. Figure 7-26 illustrates a functional timing diagram of a bus error on a word write access causing an access error exception. Figure 7-27 illustrates a functional timing diagram of a bus error on a line read access that does not cause an access error exception.

A physical bus error during an FSAVE instruction results in corruption of the floating-point state frame. This is not a serious limitation since, prior to writing the stack frame, the M68040 ensures that the pages required for the floating-point state frame are resident. Therefore, only a physical bus error can cause an access error during the stacking of the state frame. In a normal application, writes caused by the processor should not result in a physical bus error since the logical address space has already been translated and allocated. Since there should be no parity errors caused by processor write accesses, only spurious assertions of the $\overline{\text{TEA}}$ pin can cause physical bus errors. Furthermore, because FSAVE instructions usually place the state frame on the system stack, the occurrence of a physical bus error when using the system stack indicates a serious hardware error.

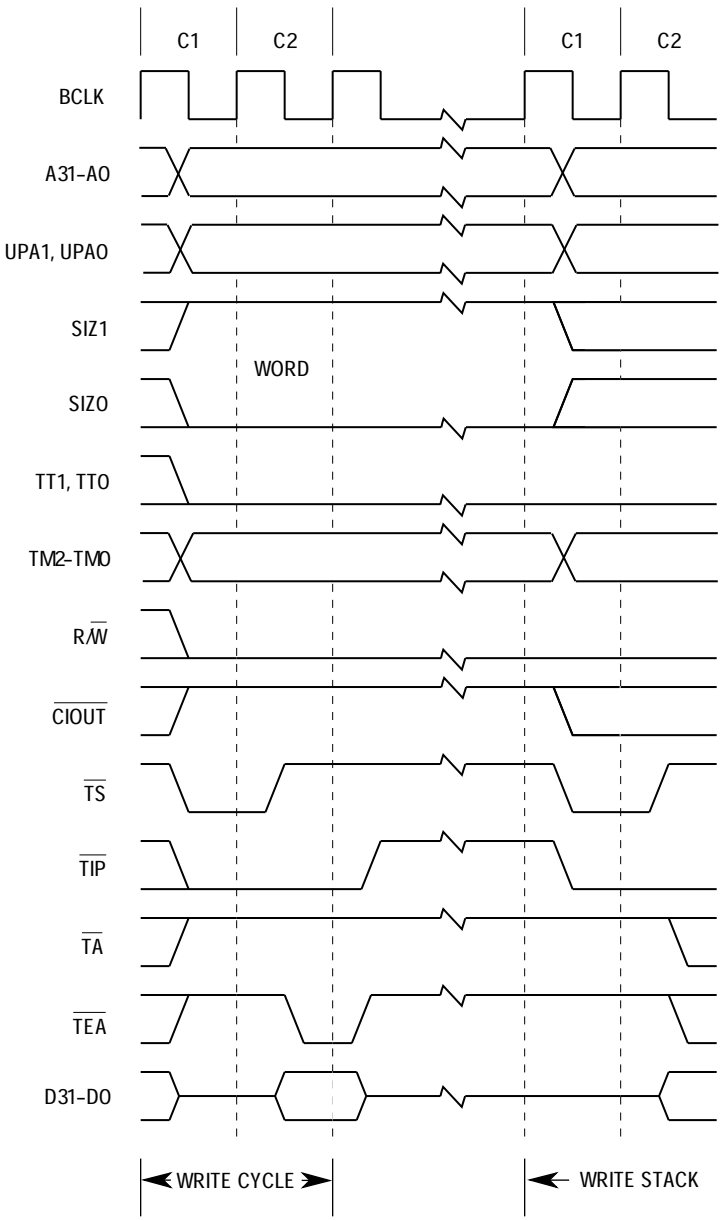
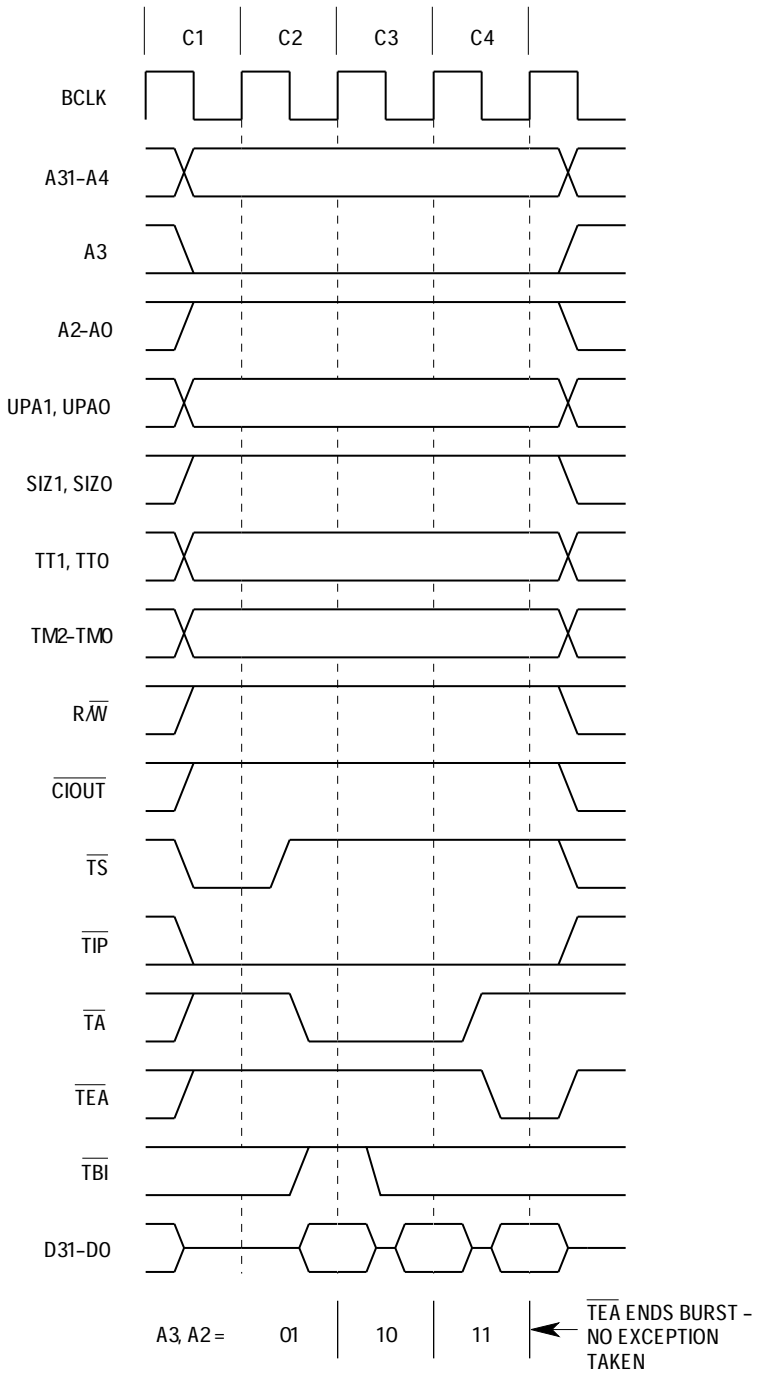


Figure 7-26. Word Write Access Terminated with Timing



NOTE: The selected device increments the value on A3 and A2.

Figure 7-27. Line Read Access Terminated with Timing

7.6.2 Retry Operation

When an external device asserts both the $\overline{TA}$ and $\overline{TEA}$ signals during a bus cycle, the processor enters the retry sequence. The processor terminates the bus cycle and immediately retries the cycle using the same access information (address and transfer attributes). However, if the bus cycle was a cache push operation, the bus is arbitrated away from the M68040 before the retry operation, and a snoop during the arbitration invalidates the cache push, then the processor does not use the same access information. Figure 7-28 illustrates a functional timing diagram for a retry of a read bus transfer.

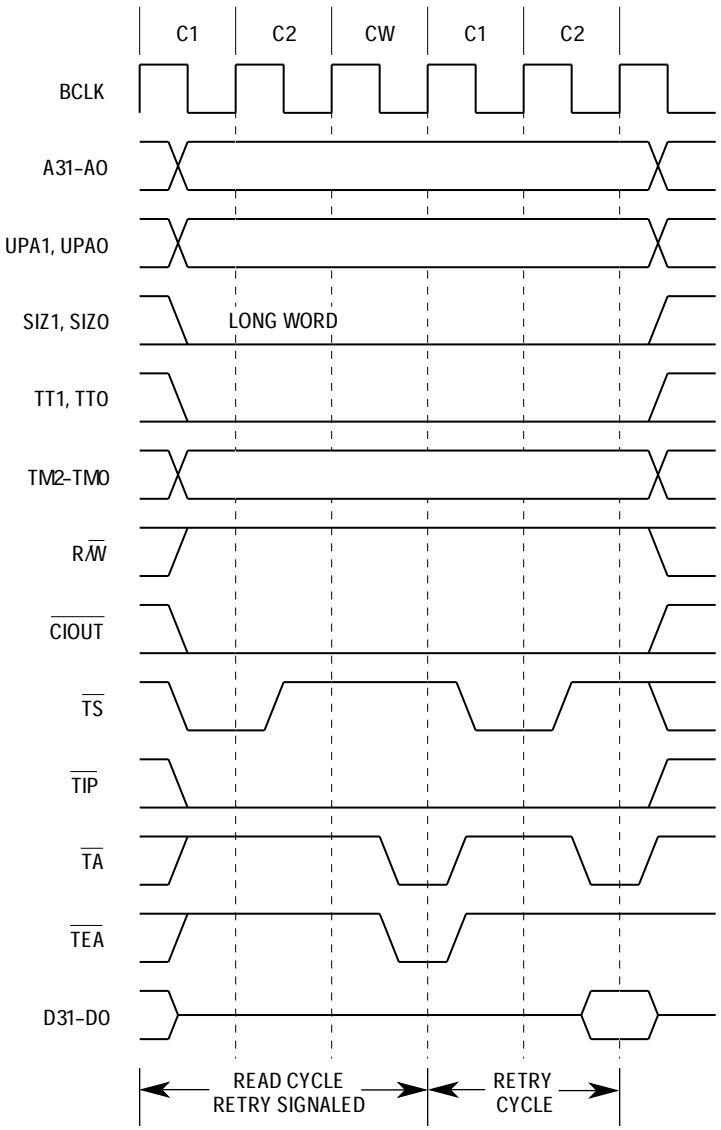


Figure 7-28. Retry Read Transfer Timing

The processor retries any read or write cycles of a read-modify-write transfer separately; $\overline{LOCK}$ remains asserted during the entire retry sequence. If the last bus cycle of a locked access is retried, $\overline{LOCKE}$ remains asserted through the retry of the write cycle.

On the initial cycle of a line transfer, a retry causes the processor to retry the bus cycle as illustrated in Figure 7-29. However, the processor recognizes a retry signaled during the second, third, or fourth cycle of a line as a bus error and causes the processor to abort the line transfer. A burst-inhibited line transfer can only be retried on the initial transfer. A burst-inhibited line transfer aborts if a retry is signaled for any of the three long-word transfers used to complete the line transfer. Negating the bus grant ($\overline{BG}$) signal on the M68040 while asserting both $\overline{TA}$ and $\overline{TEA}$ provides a relinquish and retry operation for any bus cycle that can be retried (see Figure 7-31).

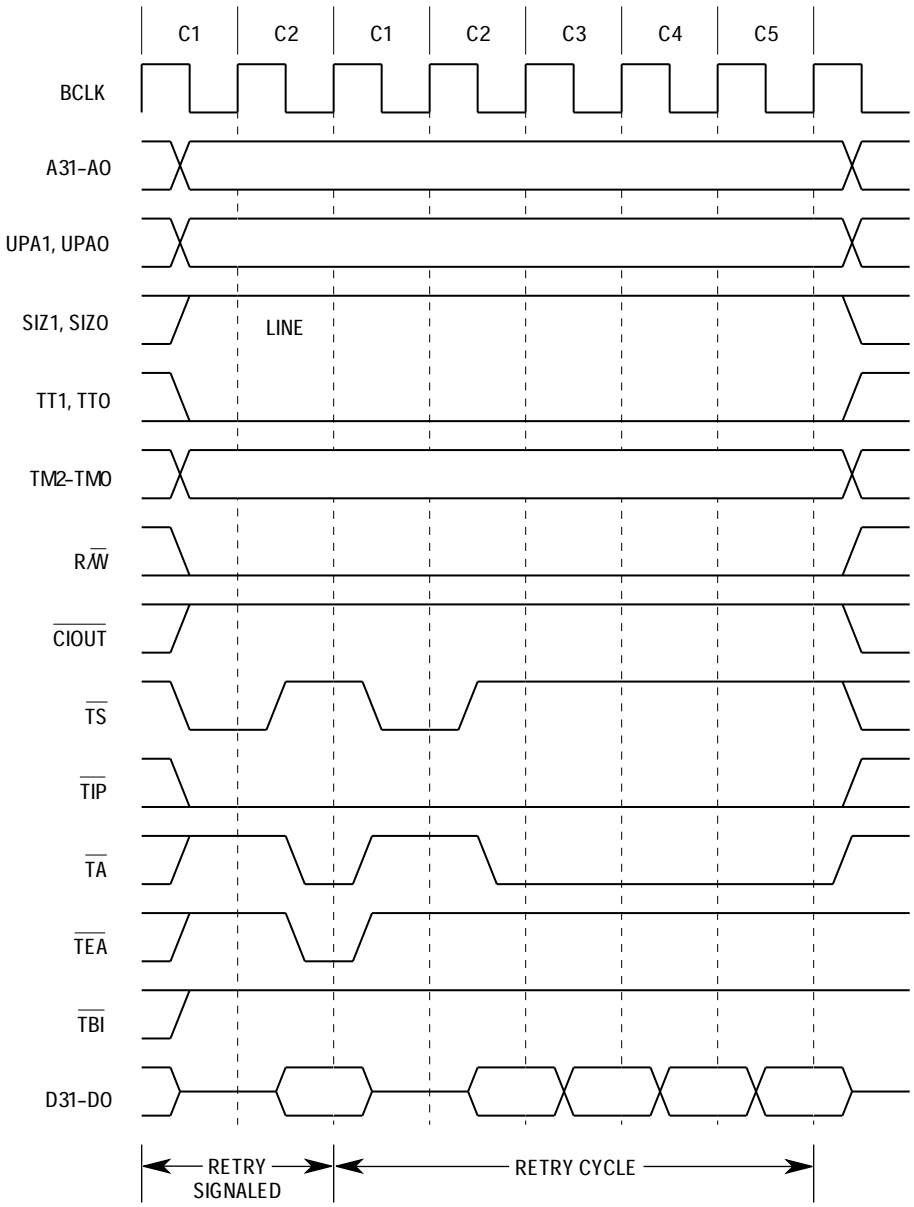


Figure 7-29. Retry Operation on Line Write

7.6.3 Double Bus Fault

A double bus fault occurs when an access or address error occurs during the exception processing sequence—e.g., the processor attempts to stack several words containing information about the state of the machine while processing an access error exception. If a bus error occurs during the stacking operation, the second error is considered a double bus fault.

The M68040 indicates a double bus fault condition by continuously driving PST3–PST0 with an encoded value of $5 until the processor is reset. Only an external reset operation can restart a halted processor. While the processor is halted, negating $\overline{BR}$ and forcing all outputs to a high-impedance state releases the external bus.

A second access or address error that occurs during execution of an exception handler or later, does not cause a double bus fault. A bus cycle that is retried does not constitute a bus error or contribute to a double bus fault. The processor continues to retry the same bus cycle as long as external hardware requests it.

7.7 BUS SYNCHRONIZATION

The M68040 integer unit generates access requests to the instruction and data memory units to support integer and floating-point operations. Both the <ea> fetch and write-back stages of the integer unit pipeline perform accesses to the data memory unit, with effective address fetches assigned a higher priority. This priority allows data read and write accesses to occur out of order, with a memory write access potentially delayed for many clocks while allowing read accesses generated by later instructions to complete. The processor detects a read access that references earlier data waiting to be written (address collisions) and allows the corresponding write access to complete. A given sequence of read accesses or write accesses is completed in order, and reordering only occurs with writes relative to reads. Figure 2-1 in **Section 2 Integer Unit** illustrates the integer pipeline stages.

Besides address collisions, the instruction restart model used for exception processing in the M68040 causes another potential problem. After the operand fetch for an instruction, an exception that causes the instruction to be aborted can occur, resulting in another access for the operand after the instruction restarts. For example, an exception could occur after a read access of an I/O device's status register. The exception causes the instruction to be aborted and the register to be read again. If the first read accesses clears the status bits, the status information is lost, and the instruction obtains incorrect data.

Designating the memory page containing the address of the device as serialized noncachable prevents multiple out-of-order accesses to devices sensitive to such accesses. When the data memory unit detects an attempt to read an operand from a page designated as serialized noncachable, it allows all pending write accesses to complete before beginning the external read access. The definition of a page as noncachable versus serialized noncachable only affects read accesses. When a write operation reaches the integer unit's write-back stage, all previous instructions have completed. When a read access to a serialized noncachable page begins, only a bus error exception

on the operand read itself can cause the instruction to be aborted, preventing multiple reads. It is important to note that when memory accesses are serialized noncachable, FMOVE will cause two identical writes to the same location to occur if the next instruction prefetch receives a bus error.

Since write cycles can be deferred indefinitely, many subsequent instructions can be executed, resulting in seemingly nonsequential instruction execution. When this action is not desired and the system depends on sequential execution following bus activity, the NOP instruction can be used. The NOP instruction forces instruction and bus synchronization because it freezes instruction execution until all pending bus cycles have completed.

A write operation of control information to an external register in which the external hardware attempts to control program execution based on the data that is written with the conditional assertion of $\overline{\text{TEA}}$ is one situation where the NOP instruction can be used to prevent multiple executions. If the data cache is enabled and the write cycle results in a hit in the data cache, the cache is updated. That data, in turn, may be used in a subsequent instruction before the external write cycle completes. Since the M68040 cannot process the bus error until the end of the bus cycle, the external hardware cannot successfully interrupt program execution. To prevent a subsequent instruction from executing until the external cycle completes, the NOP instruction can be inserted after the instruction causing the write. In this case, access error exception processing proceeds immediately after the write before subsequent instructions are executed. This is an irregular situation, and the use of the NOP instruction for this purpose is not required by most systems.

Note that the NOP instruction can also be used to force access serialization by placing NOP before the instruction that reads an I/O device. This practice eliminates the need to specify the entire page as serialized noncachable but does not prevent the instruction from being aborted by an exception condition.

7.8 BUS ARBITRATION AND EXAMPLES

The bus design of the M68040 provides for one bus master at a time, either the M68040 or an external device. More than one device having the capability to control the bus can be attached to the bus. An external arbiter prioritizes requests and determines which device is granted access to the bus. Bus arbitration is the protocol by which the processor or an external device becomes the bus master. When the M68040 is the bus master, it uses the bus to read instructions and data not contained in its internal caches from memory and to write data to memory. When an alternate bus master owns the bus, the M68040 is able to monitor the alternate bus master's transfer and intervene when necessary to maintain cache coherency. This capability is discussed in more detail in **7.9 Bus Snooping Operation.**

Unlike earlier members of the M68000 family, the M68040 implements an arbitration method in which an external arbiter controls bus arbitration and the processor acts as a slave device requesting ownership of the bus from the arbiter. Since the user defines the functionality of the external arbiter, it can be configured to support any desired priority scheme. For systems in which the processor is the only possible bus master, the bus can

be continuously granted to the processor, and no arbiter is needed. Systems that include several devices that can become bus masters require an arbiter to assign priorities to these devices so that, when two or more devices simultaneously attempt to become the bus master, the one having the highest priority becomes the bus master first.

7.8.1 Bus Arbitration

The M68040 bus controller generates bus requests to the external arbiter in response to internal requests from the instruction and data memory units. The M68040 performs bus arbitration using the bus request ($\overline{BR}$), bus grant ($\overline{BG}$), and bus busy ($\overline{BB}$) signals. The arbitration protocol, which allows arbitration to overlap with bus activity, requires a single idle clock to prevent bus contention when transferring bus ownership between bus masters. The bus arbitration unit in the M68040 operates synchronously and transitions between states on the rising edge of BLCK.

The M68040 requests the bus from the external bus arbiter by asserting $\overline{BR}$ whenever an internal bus request is pending. The processor continues to assert $\overline{BR}$ for as long as it requires the bus. The processor negates $\overline{BR}$ at any time without regard to the status of $\overline{BG}$ and $\overline{BB}$. If the bus is granted to the processor when an internal bus request is generated, $\overline{BR}$ is asserted simultaneously with transfer start ($\overline{TS}$), allowing the access to begin immediately. The processor always drives $\overline{BR}$, and $\overline{BR}$ cannot be wire-ORed with other devices.

The external arbiter asserts $\overline{BG}$ to indicate to the processor that it has been granted the bus. If $\overline{BG}$ is negated while a bus cycle is in progress, the processor relinquishes the bus at the completion of the bus cycle. To guarantee that the bus is relinquished, $\overline{BG}$ must be negated prior to the rising edge of the BCLK in which the last $\overline{TA}$ or $\overline{TEA}$ is asserted. Note that the bus controller considers the four bus transfers for a burst-inhibited line transfer to be a single bus cycle and does not relinquish the bus until completion of the fourth transfer. The read and write portions of a locked read-modify-write sequence are divisible in the M68040, allowing the bus to be arbitrated away during the locked sequence. For system applications that do not allow locked sequences to be broken, the arbiter can use $\overline{LOCK}$ to detect locked accesses and prevent the negation of $\overline{BG}$ to the processor during these sequences. The processor also provides the $\overline{LOCKE}$ signal to indicate the last write cycle of a locked sequence, allowing arbitration between back-to-back locked sequences. See **7.4.5 Read-Modify-Write Transfers (Locked Transfers)** for a detailed description of read-modify-write transfers.

When the bus has been granted to the processor in response to the assertion of $\overline{BR}$, one of two situations can occur. In the first situation, the processor monitors $\overline{BB}$ to determine when the bus cycle of the alternate bus master is complete. After the alternate bus master negates $\overline{BB}$, the processor asserts $\overline{BB}$ to indicate explicit bus ownership and begins the bus cycle by asserting $\overline{TS}$. The processor continues to assert $\overline{BB}$ until the external arbiter negates $\overline{BG}$, after which $\overline{BB}$ is first negated at the completion of the bus cycle, then forced to a high-impedance state. As long as $\overline{BG}$ is asserted, $\overline{BB}$ remains asserted to indicate the bus is owned, and the processor continuously drives the bus signals. The processor negates $\overline{BR}$ when there are no pending accesses to allow the external arbiter to grant the bus to the alternate bus master if necessary.

In the second situation, the processor samples $\overline{BB}$ until the external bus arbiter negates $\overline{BB}$. The processor drives its output pins with undetermined values and three-states $\overline{BB}$, but does not perform a bus cycle. This procedure, called implicit ownership of the bus, occurs when the processor is granted the bus but there are no pending bus cycles. If an internal access request is generated, the processor assumes explicit ownership of the bus and immediately begins an access, simultaneously asserting $\overline{BB}$, $\overline{BR}$, $\overline{TIP}$, and $\overline{TS}$. If the external arbiter keeps $\overline{BG}$ asserted after completion of the bus cycle, the processor keeps $\overline{BB}$ asserted and drives the bus with undefined values, causing the processor to park. In this case, because $\overline{BB}$ remains asserted until the external arbiter negates $\overline{BG}$, the processor must assert $\overline{BR}$, $\overline{TIP}$, and $\overline{TS}$ simultaneously to enter an active bus cycle. When it completes the active bus cycle and the external arbiter has not negated $\overline{BG}$, the processor goes back into park, negating $\overline{BR}$, $\overline{TIP}$, and $\overline{TS}$. As long as $\overline{BG}$ is asserted, the processor oscillates between park and active bus cycles.

The M68040 can be in any one of five bus arbitration states during bus operation: idle, snoop, implicit ownership, park, and active bus cycle. There are two characteristics that determine these five states: whether the three-state logic determines if the M68040 drives the bus and how the M68040 drives $\overline{BB}$. If neither the processor nor the external bus arbiter asserts $\overline{BB}$, then an external pullup resistor drives $\overline{BB}$ high to negate it. Note that the relationship between the internal $\overline{BR}$ and the external $\overline{BR}$ is best described as a synchronous delay off BCLK.

The idle state occurs when the M68040 does not have ownership of the bus and is not in the process of snooping an access. In the idle state, $\overline{BB}$ is negated and the M68040 does not drive the bus. The snoop state is similar to the idle state in that the M68040 does not have ownership of the bus. The snoop state differs from the idle state in that the M68040 is ready to service snooped transfers. Otherwise, the status of $\overline{BB}$ and the bus is identical.

The implicit ownership state indicates that the M68040 owns the bus. The M68040 explicitly owns the bus when it runs a bus cycle immediately after being granted the bus. If the processor has completed at least one bus cycle and no internal transfers are pending, the processor drives the bus with undefined values, entering the park state. In either case, $\overline{BG}$ remains asserted. The simultaneous assertion of $\overline{BR}$, $\overline{TIP}$, and $\overline{TS}$ allows the processor to leave the park state and enter the active bus cycle state.

Figure 7-30 is a bus arbitration state diagram illustrating the relationship of these five states with an example of an external bus arbiter circuit. Table 7-6 lists the five states and the conditions that indicate them.

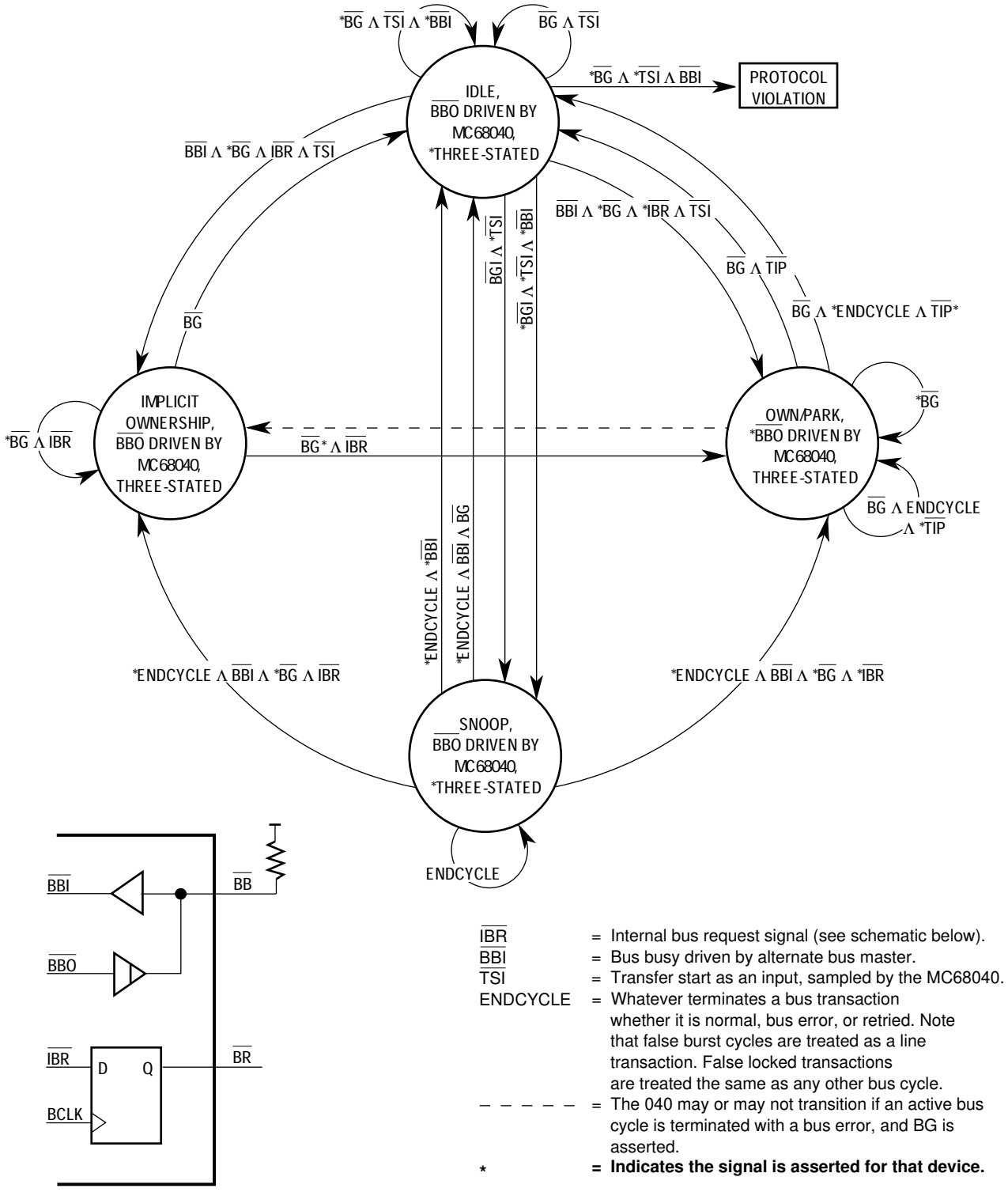


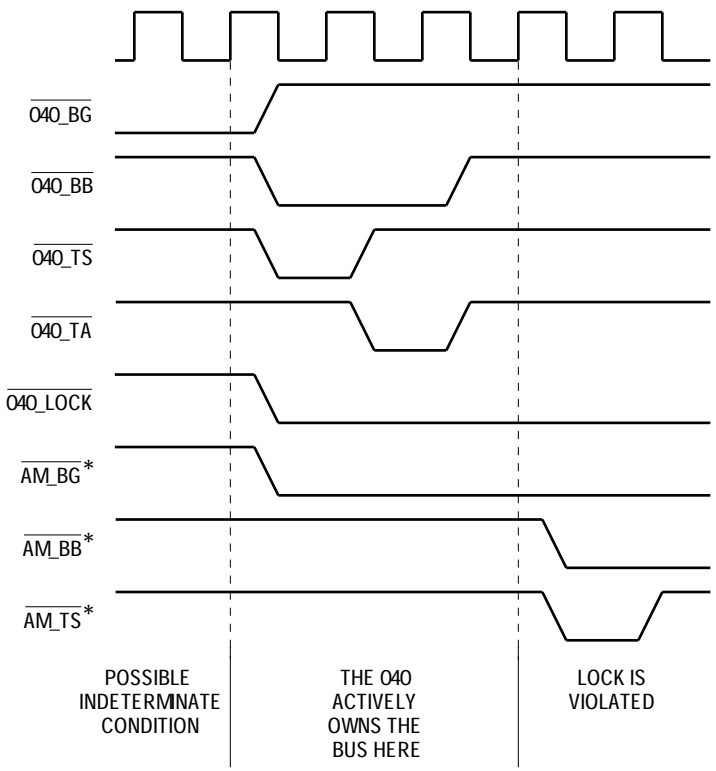
Figure 7-30. M68040 Internal Interpretation State Diagram and External Bus Arbiter Circuit

Table 7-6. M68040 Bus Arbitration States

		State	Conditions
Negated	Negated	Idle	M68040 three-states $\overline{BB}$; arbiter negates $\overline{BG}$; bus is not driven.
Negated	Asserted	Implicit Ownership	M68040 three-states $\overline{BB}$; arbiter asserts $\overline{BG}$; bus is driven with undefined values.
Asserted	Negated	Active Bus Cycle	M68040 asserts $\overline{BB}$; arbiter asserts $\overline{BG}$; bus is driven with defined values; $\overline{TIP}$ is asserted.
Asserted	Asserted	Park	M68040 asserts $\overline{BB}$; arbiter asserts $\overline{BG}$; bus is driven with undefined values; $\overline{TIP}$ is asserted.
Asserted	Asserted	Alternate Bus Master Ownership and Snooped	M68040 three-states $\overline{BB}$; arbiter asserts $\overline{BG}$; M68040 does not drive the bus.

The M68040 can be in the active bus cycle, park, or implicit ownership states when $\overline{BG}$ is negated. Depending on the state the processor is in when $\overline{BG}$ is negated, uncertain conditions can occur. The only guaranteed time that the processor relinquishes the bus is when $\overline{BG}$ is negated prior to the rising edge of BCLK in which the last $\overline{TA}$ or $\overline{TEA}$ is asserted and the processor is in the active bus cycle state. However, if the processor is in either the active bus cycle, park, or implicit ownership states and $\overline{BG}$ is negated at the same time or after the last $\overline{TA}$ or $\overline{TEA}$ is asserted, then from the standpoint of the external bus arbiter, the next action that the processor takes is undetermined because the processor can internally decide to perform another active bus cycle (indeterminate condition).

External bus arbiters must consider this indeterminate condition when negating $\overline{BG}$ and must be designed to examine the state of $\overline{BB}$ immediately after negating $\overline{BG}$ to determine whether or not the processor will run another bus cycle. A somewhat dangerous situation exists when the processor begins a locked transfer after the bus has been granted to the alternate bus master, causing the alternate bus master to perform a bus transfer during a locked sequence. To correct this situation, the external bus arbiter must be able to recognize the possible indeterminate condition and reassert $\overline{BG}$ to the processor when the processor begins a locked sequence. The indeterminate condition is most significant when dealing with systems that cannot allow locked transfers to be broken. Figure 7-31 illustrates an example of an error condition that is a consequence of the interaction between the indeterminate condition and a locked transfer. External bus arbiters must be designed so that all bus grants to all bus masters be nagated for at least one rising edge of BCLK between bus tenures; preventing bus conflicts resulting from the above conditions.



*AM indicates the alternate bus master.

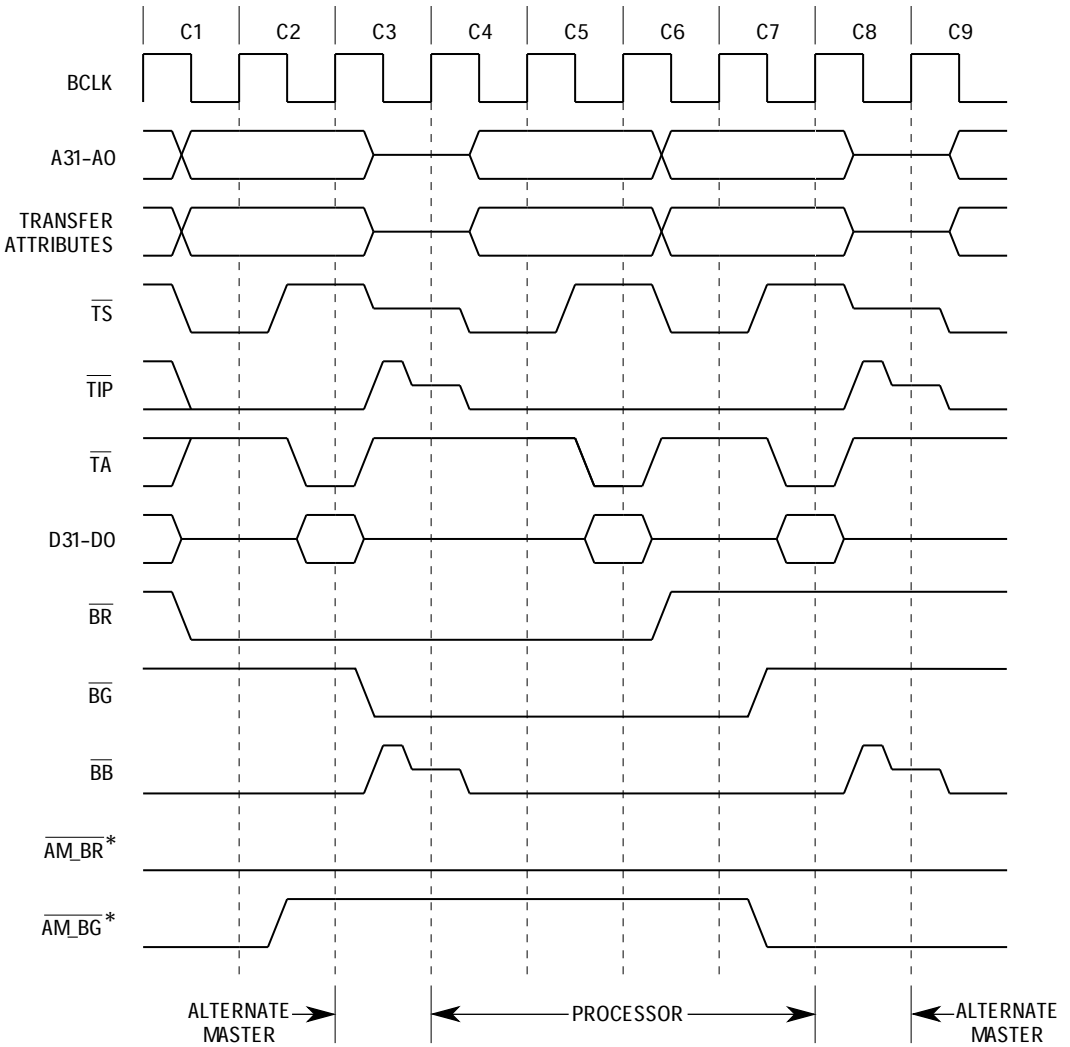
Figure 7-31. Lock Violation Example

In addition to the indeterminate condition, the external arbiter's design needs to include the function of $\overline{BR}$. For example, in certain cases associated with conditional branches, the M68040 can assert $\overline{BR}$ to request the bus from an alternate bus master, then negate $\overline{BR}$ without using the bus, regardless of whether or not the external arbiter eventually asserts $\overline{BG}$. This situation happens when the M68040 attempts to prefetch an instruction for a conditional branch. To achieve maximum performance, the processor prefetches the instructions of both paths for a conditional branch. If the conditional branch results in a branch-not-taken, the previously issued branch-taken prefetch is then terminated since the prefetch is no longer needed. In an attempt to save time, the M68040 negates $\overline{BR}$. If $\overline{BG}$ takes too long to assert, the M68040 enters a disregard request condition.

The $\overline{BR}$ signal can be reasserted immediately for a different pending bus request, or it can stay negated indefinitely. If an external bus arbiter is designed to wait for the M68040 to assert $\overline{BB}$ before proceeding, then the system experiences an extended period of time in which bus arbitration is locked. Motorola recommends that an external bus arbiter not assume that there is a direct relationship between $\overline{BR}$ and $\overline{BB}$ or $\overline{BR}$ and $\overline{BG}$ signals.

Figure 7-32 illustrates an example of the processor requesting the bus from the external bus arbiter. During C1, the M68040 asserts $\overline{BR}$ to request the bus from the arbiter, which negates the alternate bus master's $\overline{BG}$ signal and grants the bus to the processor by asserting $\overline{BG}$ during C3. During C3, the alternate bus master completes its current access and relinquishes the bus by three-stating all bus signals. Typically, the $\overline{BB}$ and $\overline{TIP}$ signals

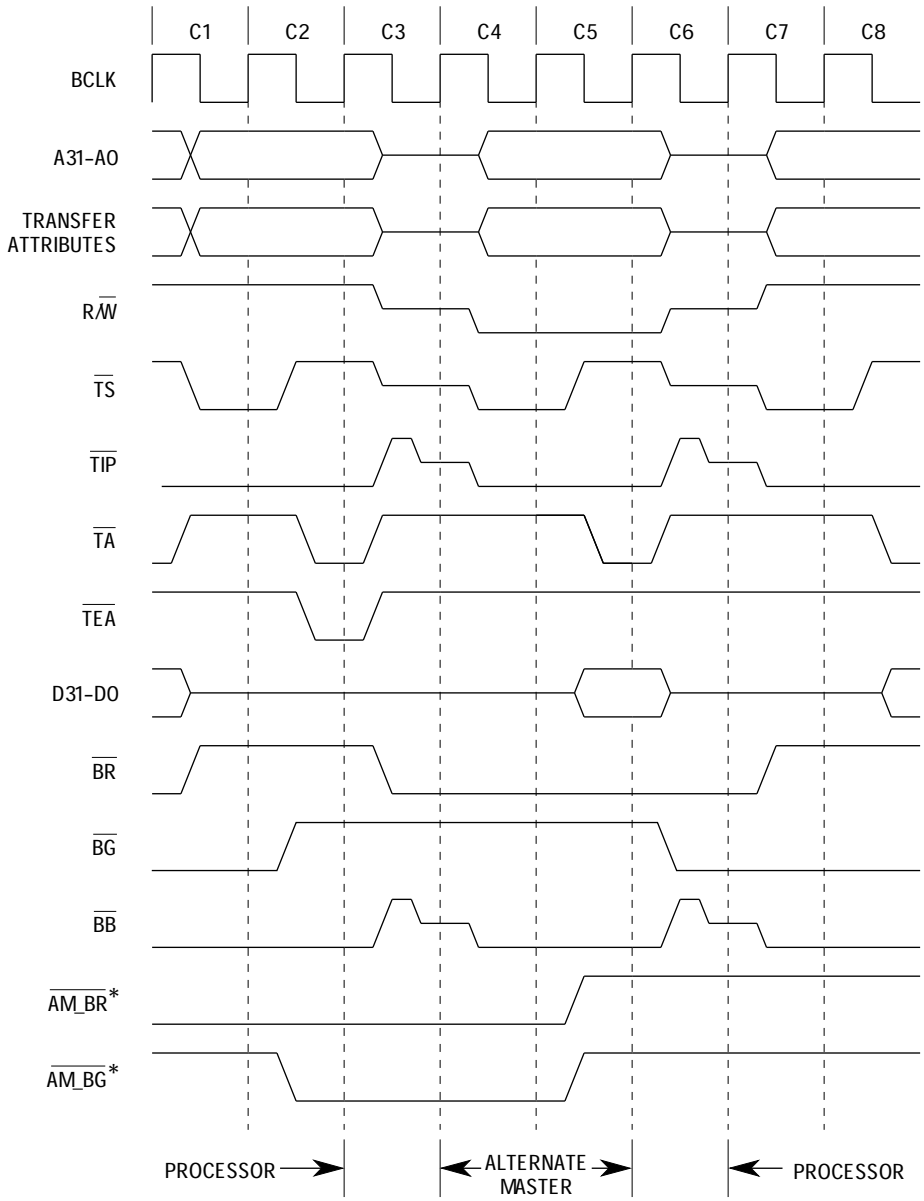
require a pullup resistor to maintain a logic-one level between bus master tenures. The alternate bus master should negate these signals before three-stating to minimize rise time of the signals and ensure that the processor recognizes the correct level on the next BCLK rising edge. At the end of C3, the processor recognizes the bus grant and bus idle conditions ($\overline{BG}$ asserted and $\overline{BB}$ negated) and assumes ownership of the bus by asserting $\overline{BB}$ and immediately beginning a bus cycle during C4. During C6, the processor begins the second bus cycle for the misaligned operand and negates $\overline{BR}$ since no other accesses are pending. During C7, the external bus arbiter grants the bus back to the alternate bus master that is waiting for the processor to relinquish the bus. The processor negates $\overline{BB}$ and $\overline{TIP}$ before three-stating these and all other bus signals during C8. Finally, the alternate bus master recognizes the bus grant and idle conditions at the end of C8 and is able to resume bus activity during C9.



*AM indicates the alternate bus master.

Figure 7-32. Processor Bus Request Timing

Figure 7-33 illustrates a functional timing diagram for an arbitration of a relinquish and retry operation. Figure 7-34 is a functional timing diagram for implicit ownership of the bus. In Figure 7-33, the processor read access that begins in C1 is terminated at the end of C2 with a retry request and $\overline{BG}$ negated, forcing the processor to relinquish the bus and allow the alternate master to access the bus. Note that the processor reasserts $\overline{BR}$ during C3 since the original access is pending again. After alternate bus master ownership, the bus is granted to the processor to allow it to retry the access beginning in C7.



*AM indicates the alternate bus master.

Figure 7-33. Arbitration During Relinquish and Retry Timing

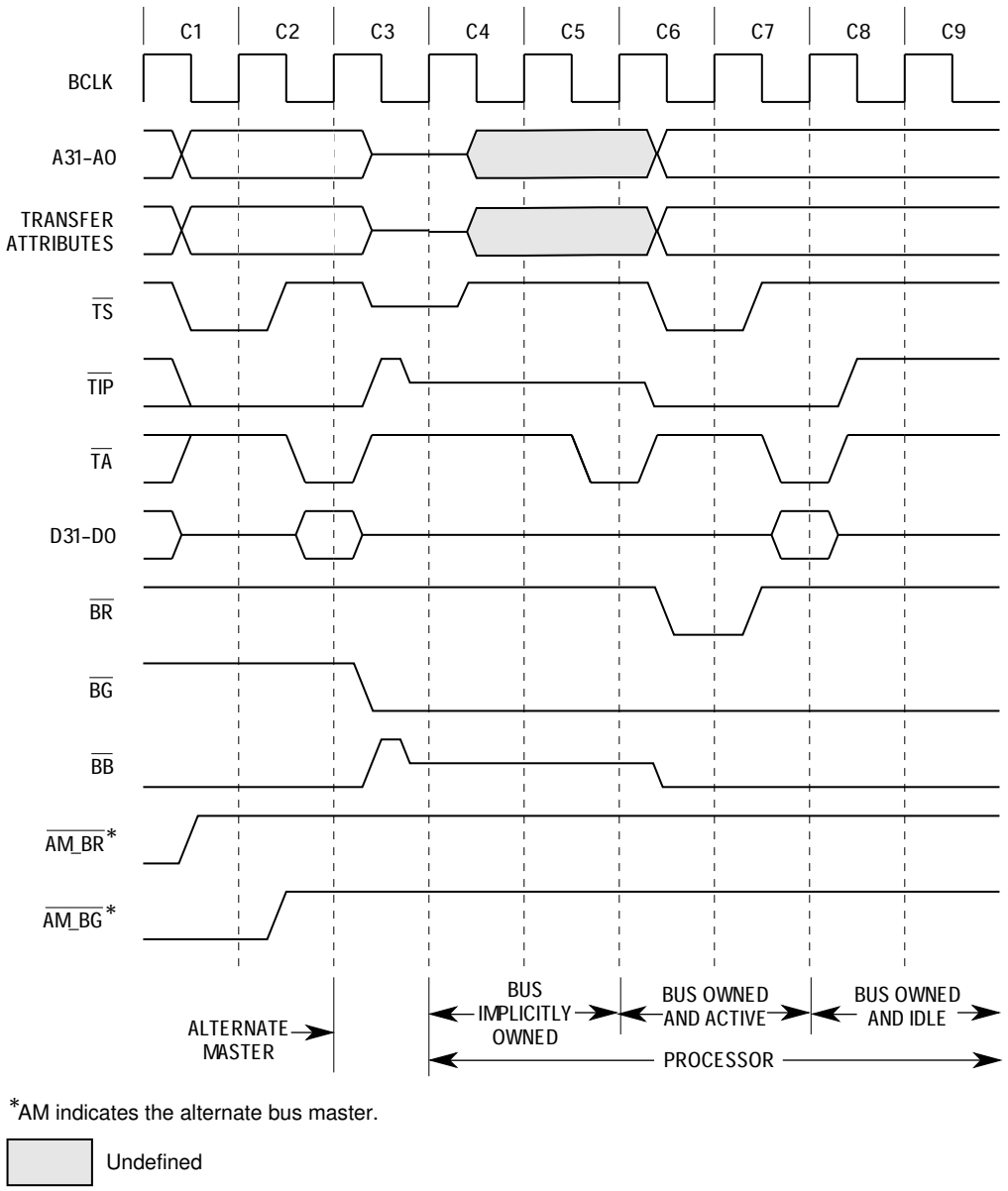


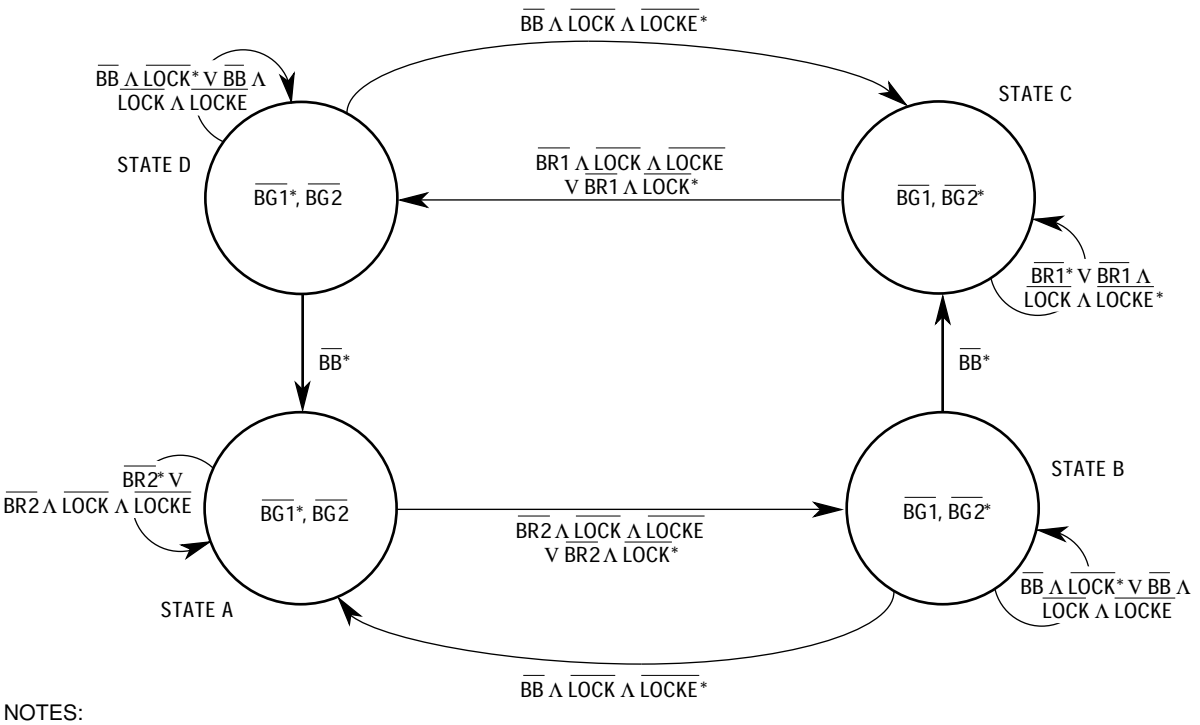
Figure 7-34. Implicit Bus Ownership Arbitration Timing

7.8.2 Bus Arbitration Examples

The following paragraphs illustrate the behavior of the M68040 bus arbitration scheme and provide examples of how an external bus arbiter can be designed to keep the integrity of locked bus operations. The examples include the previously mentioned indeterminate and disregard request conditions.

7.8.2.1 DUAL M68040 FAIRNESS ARBITRATION. The following state diagram illustrates a fairness algorithm using two MC68040s and assigning the least priority to the processor that owns the bus. If both processors keep their respective $\overline{BR}$ signals asserted, bus ownership alternates between the two processors so that each processor can run at least one bus cycle during its tenure. Each processor is allowed to own the bus without relinquishing it to maintain the integrity of locked transfers. This example also illustrates

how the $\overline{\text{LOCKE}}$ signal can be used to end a locked sequence and to yield the bus one bus cycle earlier than is normally possible. Figure 7-35 illustrates the state diagram of a hypothetical external arbiter design.



NOTES:

1. Because this example uses two MC68040s, 1 and 2 refer to the processor and its signals.
2. *Indicates the signal is asserted for that device.

Figure 7-35. Dual M68040 Fairness Arbitration State Diagram

Assuming that processor 1 currently owns the bus, the external arbiter is in state A. If processor 2 asserts $\overline{\text{BR2}}$, then processor 1 behaves in one of three ways:

1. If processor 1 is currently in the middle of a nonlocked bus access, then the external arbiter proceeds to state B, in which $\overline{\text{BG1}}$ is negated and $\overline{\text{BG2}}$ is asserted. The external arbiter then proceeds to state C only when $\overline{\text{BB}}$ is negated, signifying the end of the bus cycle.
2. If processor 1 is currently in the middle of a locked bus access, then the external arbiter stays in state A until $\overline{\text{LOCKE}}$ is asserted. Once $\overline{\text{LOCKE}}$ is asserted, the external arbiter enters state B, in which $\overline{\text{BG1}}$ is negated and $\overline{\text{BG2}}$ is asserted. The external arbiter proceeds to state C once $\overline{\text{BB}}$ is negated, signifying the end of the bus cycle.
3. If processor 1 is in one of the three boundary conditions, then the external arbiter proceeds to state B. During state B, the external arbiter checks for the possibility of a newly initiated locked bus access. If it detects a locked bus cycle, it returns the bus to processor 1 by entering state A. Note that even though processor 1 recognizes $\overline{\text{BG1}}$ is asserted, it does not take the bus because processor 1 asserts $\overline{\text{BB}}$ whenever the boundary condition results in processor 1 performing another bus cycle. The external arbiter stays in state A until $\overline{\text{LOCKE}}$ is asserted, then proceeds to state B to

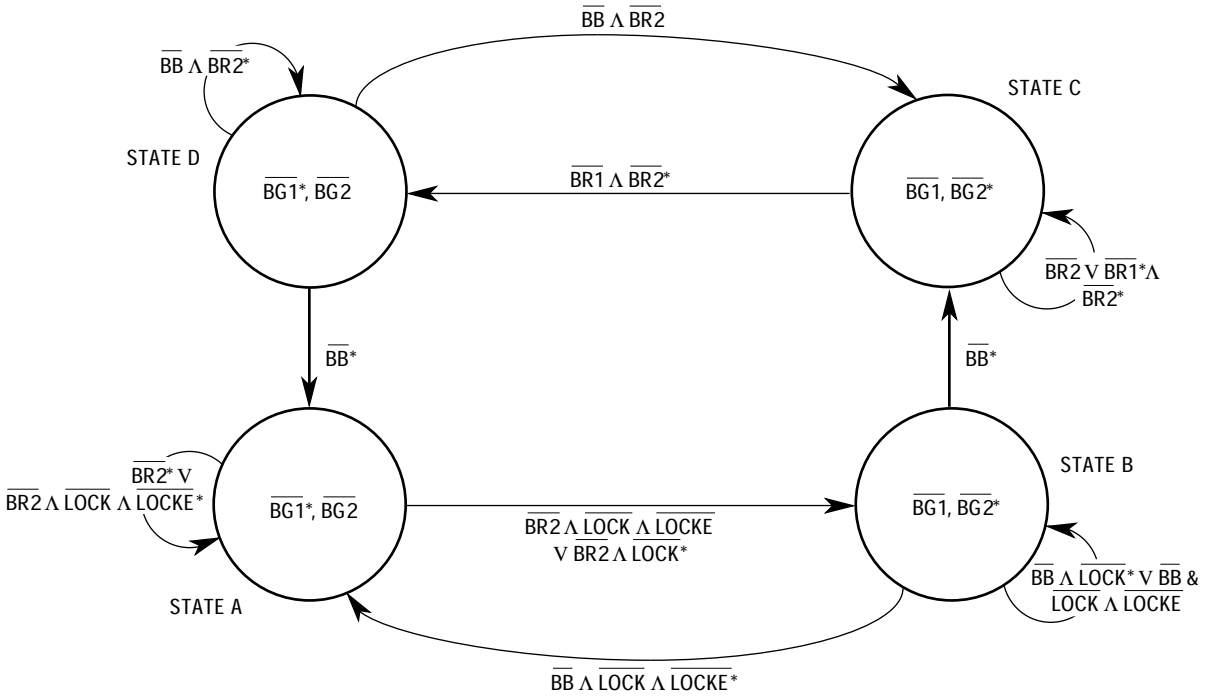
give the bus to processor 2. The arbiter remains in state B until $\overline{BB}$ is negated, signifying the end of the bus cycle.

Once state C is reached, depending on whether or not processor 2 asserts $\overline{BR2}$ and then negates $\overline{BR2}$ because of a disregard request condition, processor 1 may or may not actively begin a bus cycle. If no other bus requests are pending by the time state C is reached, processor 2 is in the implicit ownership state. If processor 1 asserts $\overline{BR1}$, then it is possible for state C to persist for only one clock. In this case, processor 2 does not have a chance to run any active bus cycles.

A null bus cycle tenure is better than having the external bus arbiter wait for processor 2 to perform at least one bus cycle before returning bus ownership to processor 1, even though this appears to be a waste of bus arbitration overhead. Note that once processor 2 enters the disregard request condition, processor 2 reasserts $\overline{BR}$ anywhere from one clock to an undetermined number of clocks before running another bus cycle. Waiting for processor 2 to run a bus cycle can result in a temporary bus arbitration lockup.

This bus arbitration scheme is restricted if the system supports the relinquish and retry operation that can occur for the last write cycle of a locked transfer. In this case, $\overline{LOCKE}$ cannot be used. Assuming that $\overline{LOCKE}$ is always negated excludes the need for $\overline{LOCKE}$ in an arbitration similar to this example. The reason for this restriction is that the external bus arbiter gives up the bus to the other processor once $\overline{LOCKE}$ is asserted. If a relinquish and retry operation were to occur, then the next bus cycle would be from the other processor violating the integrity of the locked transfer.

7.8.2.2 DUAL M68040 PRIORITIZED ARBITRATION. This example is very similar to the dual M68040 fairness arbitration example, except that one processor is assigned higher priority over the other. Processor 2 can own the bus only if there are no processor 1 pending requests. It is important to note that when the processor asserts the $\overline{LOCK}$ signal, it also asserts $\overline{BR1}$. This implementation replaces $\overline{LOCK}$ with $\overline{BR}$ because $\overline{BR}$ is more demanding than using $\overline{LOCK}$. Only when processor 2 is in the middle of a locked operation does it have higher priority than processor 1. Similar to the M68040 fairness arbitration example, the restriction on using $\overline{LOCKE}$ applies to this example. Figure 7-36 illustrates the state diagram for dual M68040 prioritized arbitration.

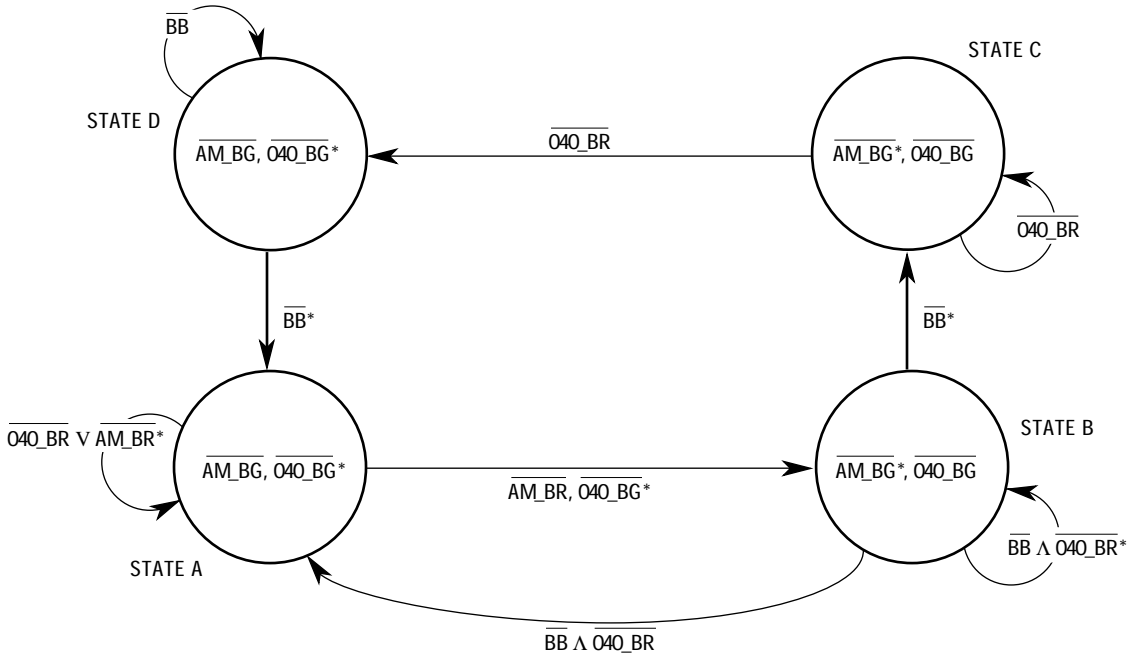


NOTES:

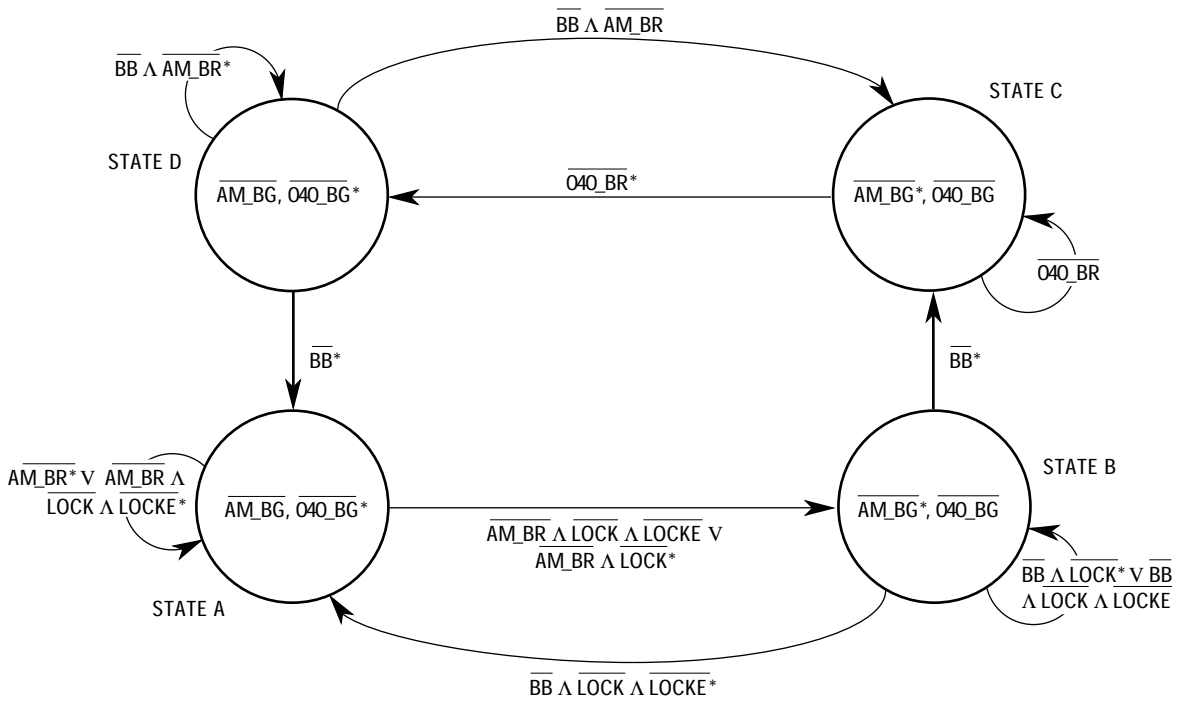
1. Because this example uses two MC68040s, 1 or 2 refers to the processor and its signals.
2. *Indicates the signal is asserted for that device.

Figure 7-36. Dual M68040 Prioritized Arbitration State Diagram

7.8.2.3 M68040 SYNCHRONOUS DMA ARBITRATION. Figure 7-37 illustrates a system with an M68040 and a synchronous direct memory access (DMA) that contains an M68040 interface. Figure 7-37(a) illustrates that the DMA owning the bus only when the M68040 has no pending requests, and Figure 7-37(b) illustrates the DMA having higher priority than the M68040 causing the M68040 to yield the bus to the DMA at any time except when the M68040 is performing a locked bus operation. In either case, the M68040 is the default bus master; if there are no pending requests from either device, the external arbiter gives the bus to the M68040. Similar to the M68040 fairness arbitration example, the restriction on using $\overline{\text{LOCKE}}$ applies to this example.



(a) MC68040 High Priorty, Default Bus Master



* Indicates the signal is asserted for that device.

(b) MC68040 Low-Priorty, Default Bus Master

Figure 7-37. M68040 Synchronous DMA Arbitration

7.8.2.4 M68040 ASYNCHRONOUS DMA ARBITRATION. Figure 7-38 illustrates a sample synchronizer circuit. Figure 7-39 illustrates how an M68040 can be implemented to simulate an MC68030. The synchronizer circuit has an output indicating whether or not a signal has been asserted for at least two consecutive rising edges of BCLK. If the synchronizer circuit indicates that the input has not been stable for at least two clocks, then the processor and alternate bus master stay in the current state. Figure 7-37(a) duplicates the MC68030 implementation of the bus arbitration circuitry in which the M68040 is allowed to yield the bus only after the indeterminate condition has been eliminated. Figure 7-37(b) is similar to the MC68030 implementation except that the DMA device has lower priority and can only perform transfers when the M68040 is in the idle state. In either case, the M68040 is the default bus master; therefore, if there are no pending requests from either device, the external bus arbiter gives the bus to the M68040.

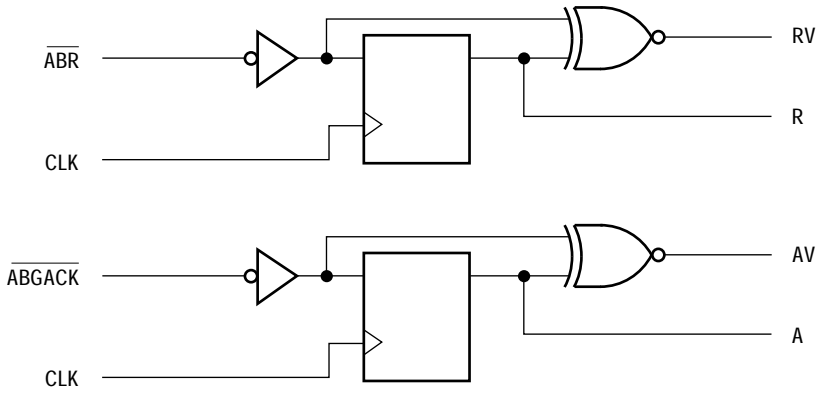
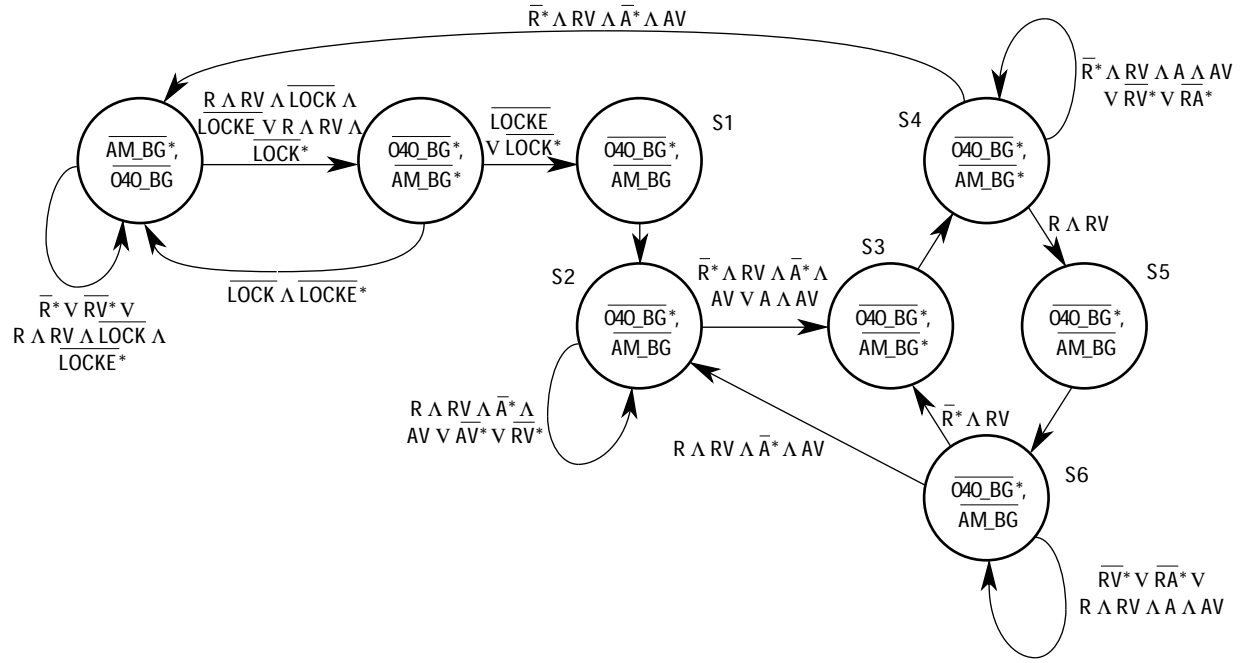
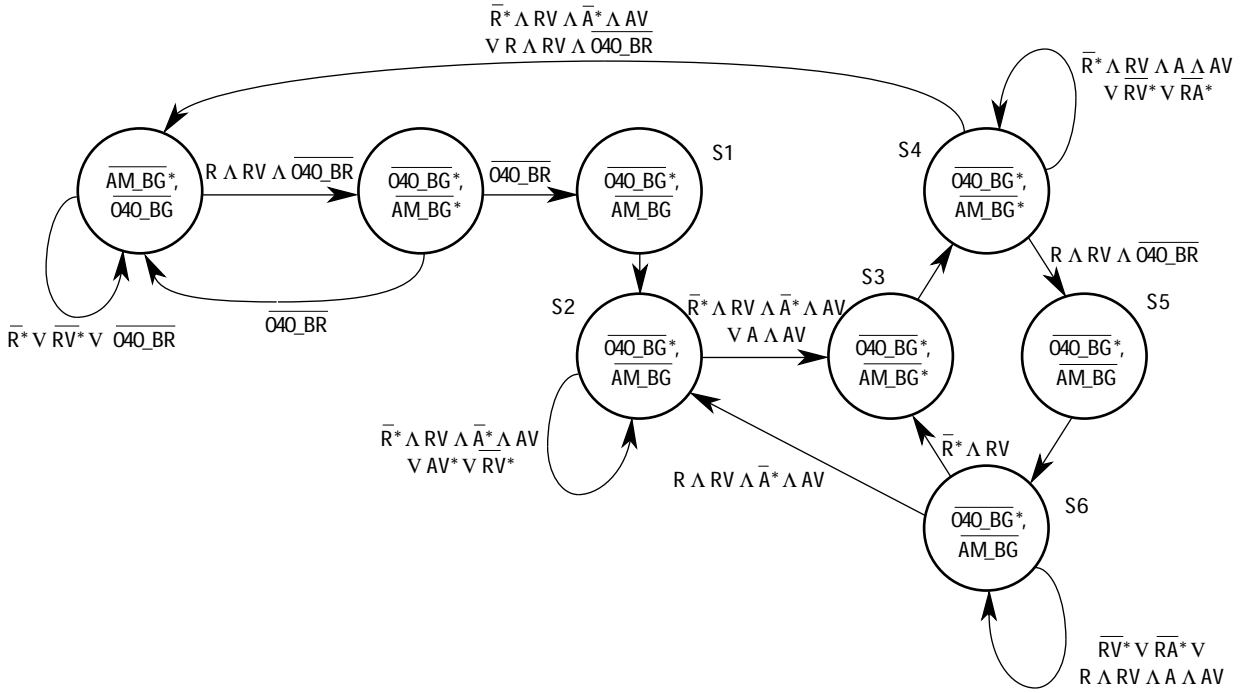


Figure 7-38. Sample Synchronizer Circuit



(a) MC68040 Low-Priorty, Default Bus Master



NOTES:
1. It is assumed that the asynchronous device takes the bus only after TIP or the MC68040's BB is negated.
2. *Indicates the signal is asserted for that device.

(b) MC68040 High-Priorty, Default Bus Master

Figure 7-39. M68040 Asynchronous DMA Arbitration

7.9 BUS SNOOPING OPERATION

When required, the M68040 can monitor alternate bus master transfers and intervene in the access to maintain cache coherency. The encoding of the SCx signals generated by the alternate bus master for each bus cycle controls the process of bus monitoring and intervention called snooping. Only byte, word, long-word, and line bus transfers can be snooped. Refer to **Section 4 Instruction and Data Caches** for SCx encodings.

When the M68040 recognizes that an alternate bus master has asserted $\overline{TS}$, the processor latches the level on the byte offset, SIZx, TMx, and R/$\overline{W}$ signals during the rising edge of BCLK for which $\overline{TS}$ is first asserted. The processor then evaluates the SCx and TTx signals to determine the type of access (TTx = $0 or $1), if it is snoopable, and, if so, how it should be snooped. If snooping is enabled for the access, the processor inhibits memory from responding by continuing to assert the memory inhibit signal ($\overline{MI}$) while checking the internal caches for matching lines. During the snooped bus cycle, the M68040 ignores all $\overline{TA}$ assertions while $\overline{MI}$ is asserted. Unless the data cache contains a dirty line corresponding to the access and the requested snoop operation indicates sink data for a write or source data for a read, $\overline{MI}$ is negated, and memory is allowed to respond and complete the access. Otherwise, the processor continues to intervene in the access by keeping $\overline{MI}$ asserted and responding to the alternate bus master as a slave device. The processor monitors the levels of $\overline{TA}$, $\overline{TEA}$, and $\overline{TBI}$ to detect normal, bus error, retry, and burst-inhibited terminations. Note that for alternate bus master burst-inhibited line transfers, the M68040 snoops each of the four resulting long-word transfers. If snooping is disabled, $\overline{MI}$ is negated, and the M68040 counts the appropriate number of $\overline{TA}$ or $\overline{TEA}$ assertions before proceeding. For example, if the SIZx signals are pulled high, the M68040 requires four $\overline{TA}$ assertions, one $\overline{TEA}$ assertion, or one retry termination before proceeding.

As a bus master, the M68040 can be configured to request snooping operations on a page-by-page basis. The UPAx signals are connected to the SCx inputs of the snooping processors. Appropriately programming the user attribute bits in the corresponding page descriptor selects the required snooping operation for a page. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for details on configuring the caching mode and user attribute bits for each memory page for the M68040 and MC68LC040, and refer to **Appendix B MC68EC040** for the MC68EC040.

In a system with multiple bus masters, the memory unit must wait for each snooping bus master to negate $\overline{MI}$ before responding to an access. A termination signal asserted before the negation of $\overline{MI}$ leads to undefined operation and must be avoided at all costs. Also, if the system contains multiple caching masters, then each master must access shared data using write-through pages that allow writes to the data to be snooped by other masters. The copyback caching mode is typically used for data local to a processor because in a multimaster caching system only one master at a time can access a given page of copyback data. The copyback caching mode also prevents multiple snooping processors from intervening in a specific access.

7.9.1 Snoop-Inhibited Cycle

For alternate bus master accesses in which the SCx signal encodings indicate that snooping is inhibited (SCx = $0), the M68040 immediately negates $\overline{MI}$ and allows memory to respond to the access. Snoop-inhibited alternate bus master accesses do not affect performance of the processor since no cache lookups are required. Figure 7-40 illustrates an example of a snoop-inhibited operation in which an alternate bus master is granted the bus for an access. No matter what the values are on the SCx and TTx signals, $\overline{MI}$ is asserted between bus cycles. Because $\overline{MI}$ is asserted while a cache lookup is performed, snooping inherently degrades system performance.

$\overline{MI}$ is asserted from the last $\overline{TA}$ of the current bus cycle if the M68040 owns the bus and loses it (see Figure 7-40). If an alternate bus master has the bus and loses it, there are two different resulting cases. Usually, an idle clock occurs between the alternate bus master's cycle and the MC68040's cycle. If so, $\overline{MI}$ is asserted during the idle clock and negated from the same edge that the M68040 asserts the $\overline{TS}$ signal (see Figure 7-40). If there is no idle clock, $\overline{MI}$ is not asserted. $\overline{MI}$ is asserted during and after reset until the first bus cycle of the M68040. Even though snoop is inhibited, all $\overline{TA}$ or $\overline{TEA}$ assertions while $\overline{MI}$ is asserted are ignored. If a line snoop is started, the M68040 still requires four $\overline{TA}$ assertions.

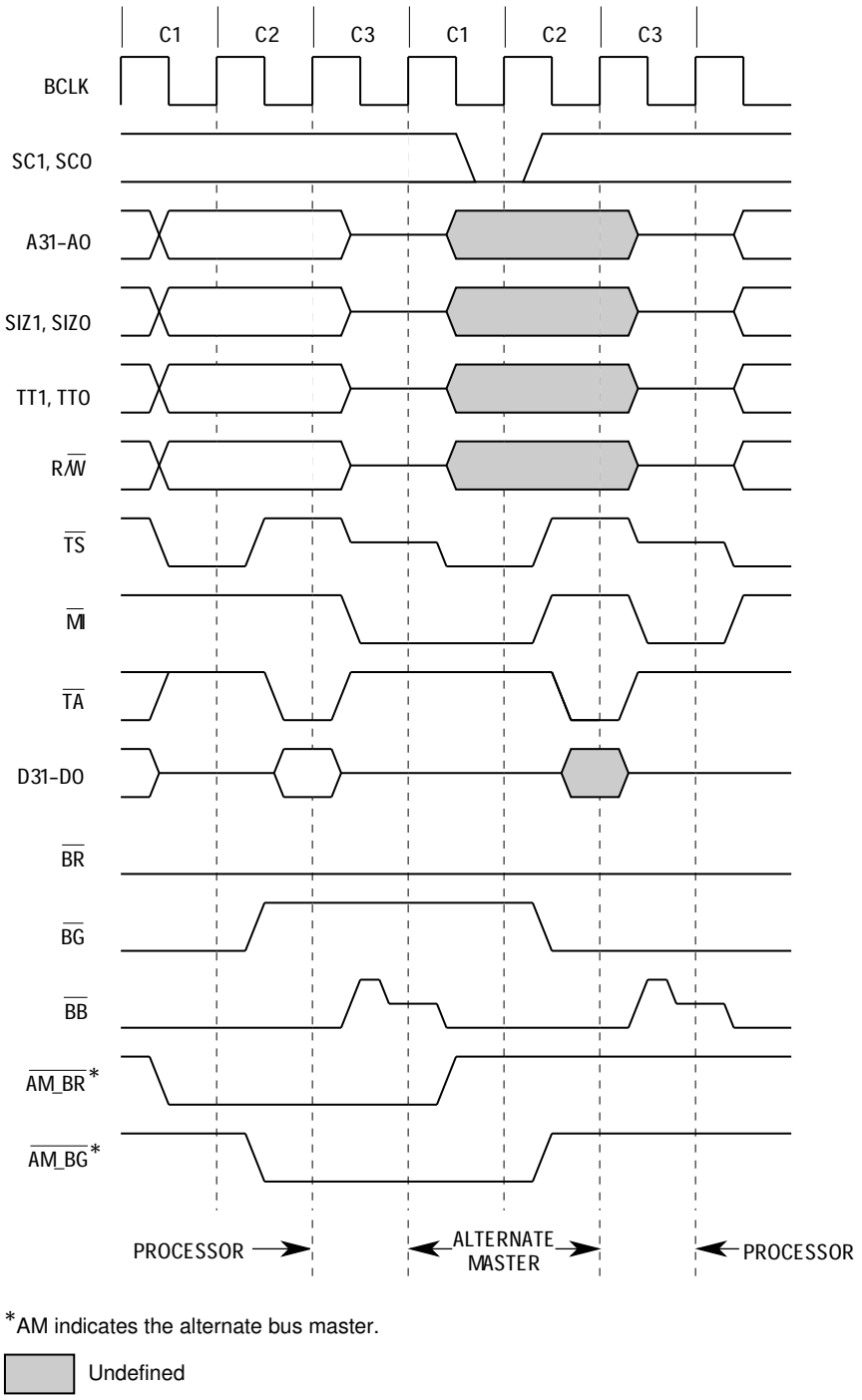


Figure 7-40. Snoop-Inhibited Bus Cycle

7.9.2 Snoop-Enabled Cycle (No Intervention Required)

For alternate bus master accesses in which SCx = $1 or $2, indicating that snooping is enabled, the M68040 continues to assert $\overline{MI}$ while checking for a matching cache line. If intervention in the alternate bus master access is not required, $\overline{MI}$ is then negated, and memory is allowed to respond and complete the access. Figure 7-41 illustrates an example of snooping in which memory is allowed to respond. Best-case timing is

illustrated, which results in a memory access having the equivalent of two wait states. Variations in the timing required by snooping logic to access the caches can delay the negation of $\overline{MI}$ by up to two additional clocks. External logic must ensure that the termination signals negate at all rising BCLK edges in which $\overline{MI}$ is asserted. Otherwise, if one of the termination signals is asserted, either the M68040 ignores all termination signals, reading them as negated, or the M68040 exhibits improper operation.

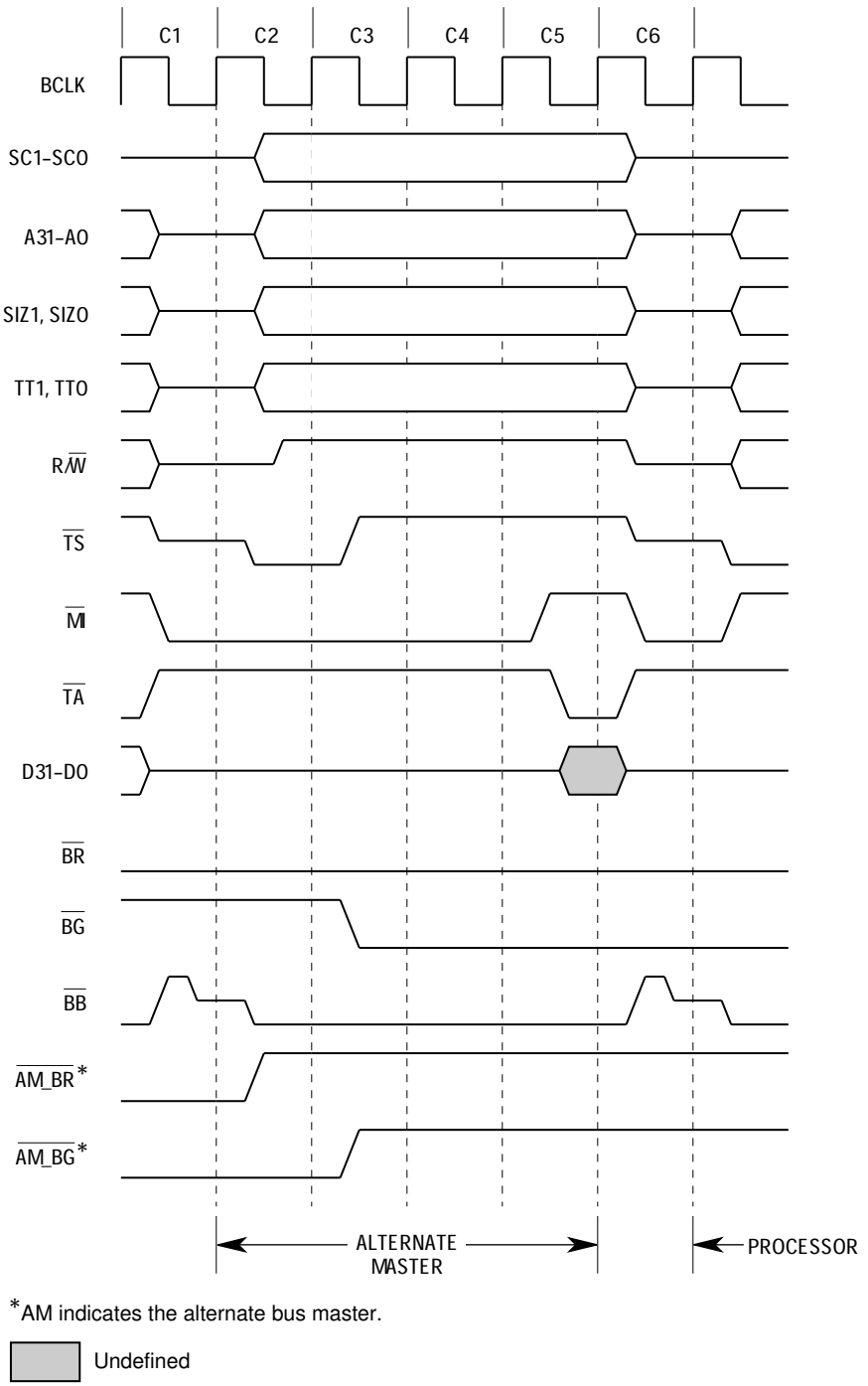


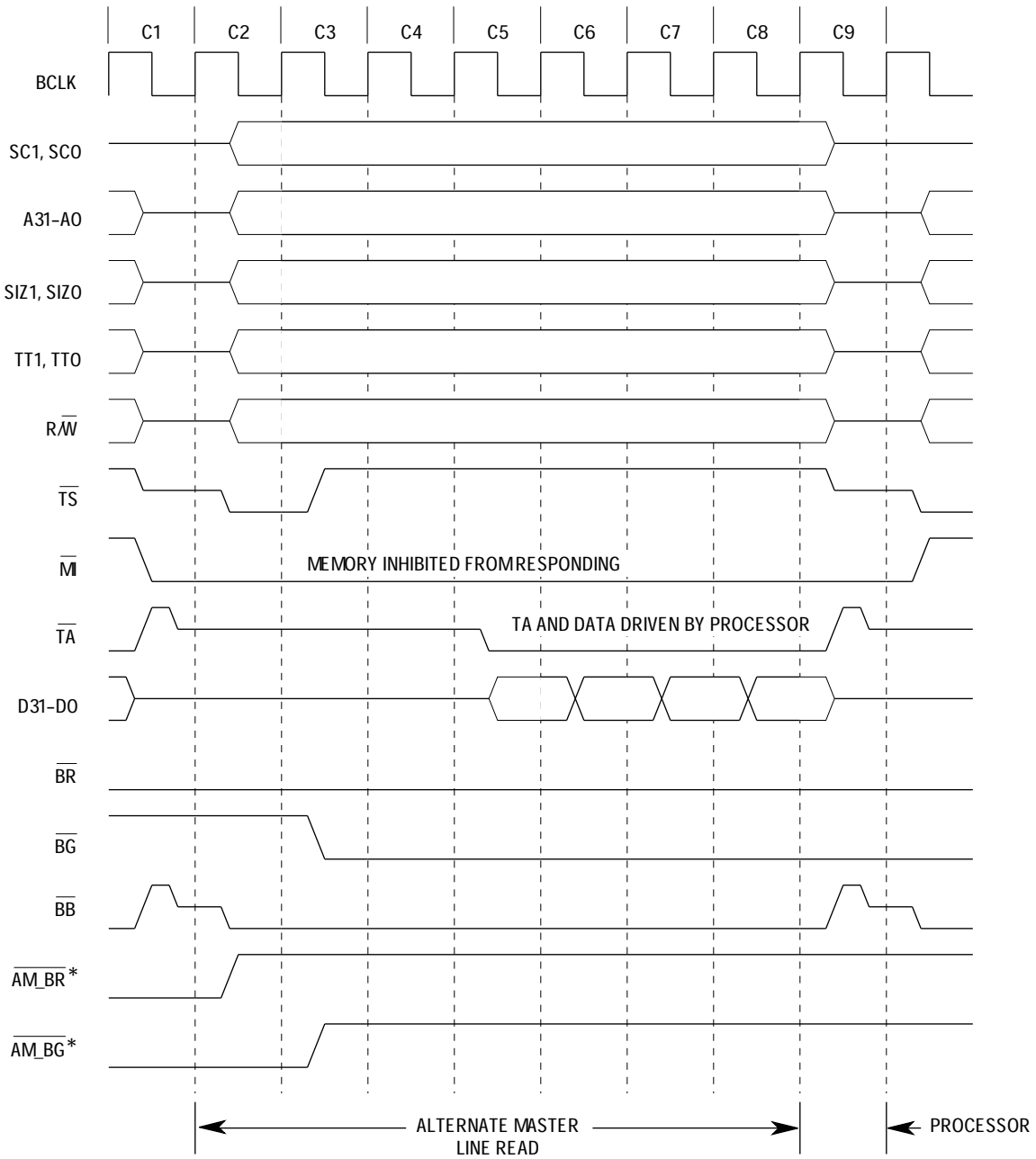
Figure 7-41. Snoop Access with Memory Response

7.9.3 Snoop Read Cycle (Intervention Required)

If snooping is enabled for a read access and the corresponding data cache line contains dirty data, the M68040 inhibits memory and responds to the access as a slave device to supply the requested read data. Intervention in a byte, word, or long-word access is independent of which long-word entry in the cache line is dirty. Figure 7-42 illustrates an alternate bus master line read that hits a dirty line in the M68040 data cache. The processor asserts $\overline{TA}$ to acknowledge the transfer of data to the alternate bus master, and the data bus is driven with the four long words of data for the line. The timing illustrated is for a best-case response time. Variations in the timing required by snooping logic to access the caches can delay the assertion of $\overline{TA}$ by up to two additional clocks.

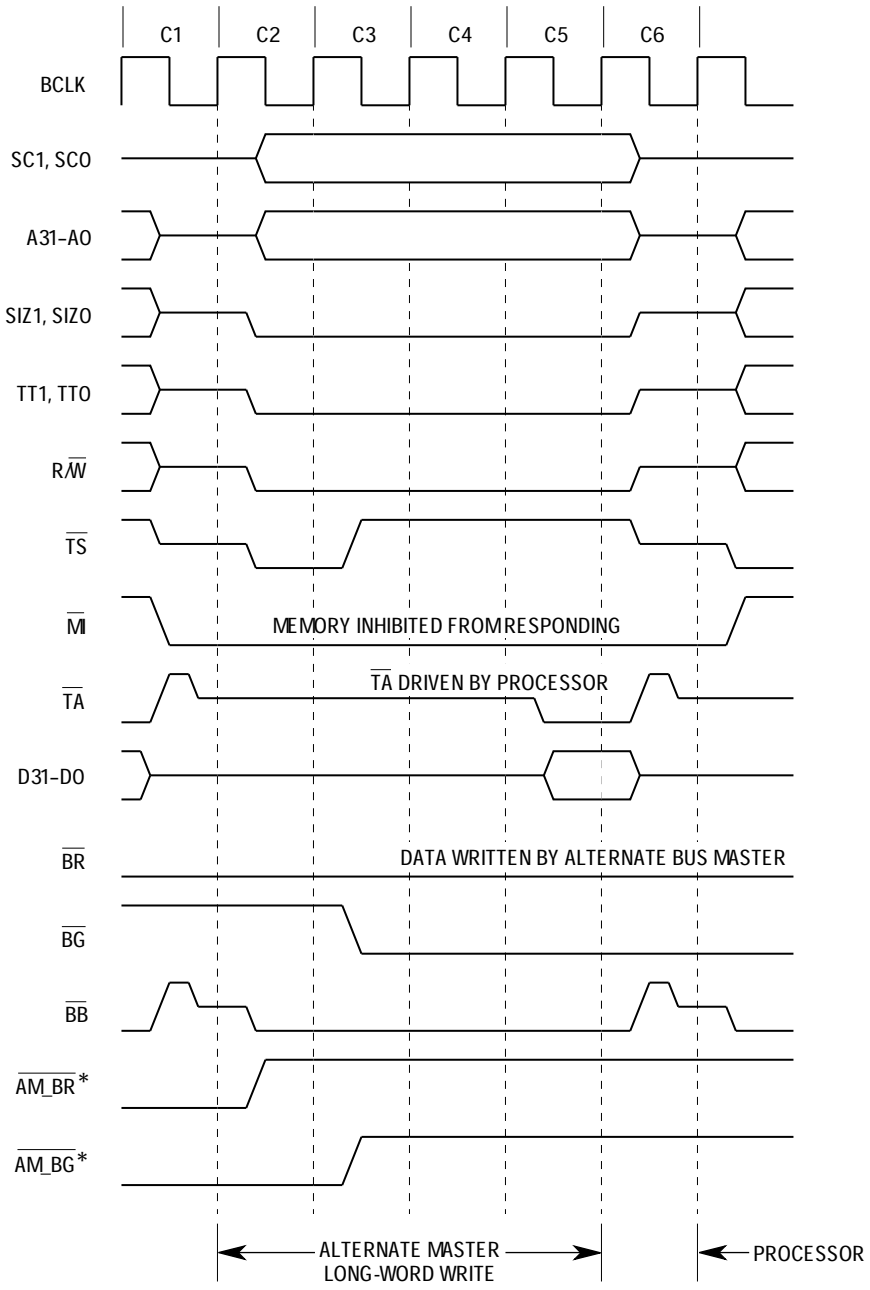
7.9.4 Snoop Write Cycle (Intervention Required)

If snooping with sink data is enabled for a byte, word, or long-word write access and the corresponding data cache line contains dirty data, the M68040 inhibits memory and responds to the access as a slave device to read the data from the bus and update the data cache line. The dirty bit is set for the long word changed in the cache line. Figure 7-43 illustrates a long-word write by an alternate bus master that hits a dirty line in the M68040 data cache. The processor asserts $\overline{TA}$ to acknowledge the transfer of data from the alternate master, and the processor reads the value on the data bus. The timing illustrated is for a best-case response time. Variations in the timing required by snooping logic to access the caches can delay the assertion of $\overline{TA}$ by up to two additional clocks.



*AM indicates the alternate bus master.

Figure 7-42. Snooped Line Read, Memory Inhibited



*AM indicates the alternate bus master.

Figure 7-43. Snooped Long-Word Write, Memory Inhibited

7.10 RESET OPERATION

An external device asserts the reset input signal ($\overline{RSTI}$) to reset the processor. When power is applied to the system, external circuitry should assert $\overline{RSTI}$ for a minimum of 10 BCLK cycles after V_{CC} is within tolerance. Figure 7-44 is a functional timing diagram of the power-on reset operation, illustrating the relationships among V_{CC}, $\overline{RSTI}$, mode selects, and bus signals. The BCLK and PCLK clock signals are required to be stable by the time V_{CC} reaches the minimum operating specification. The V_{IH} levels of the clocks

should not exceed V_{CC} while it is ramping up. $\overline{RSTI}$ is internally synchronized for two BCLKS before being used and must meet the specified setup and hold times to BCLK (specifications #51 and #52 in **Section 11 MC68040 Electrical and Thermal Characteristics**) only if recognition by a specific BCLK rising edge is required. $\overline{MI}$ is asserted while the M68040 is in reset.

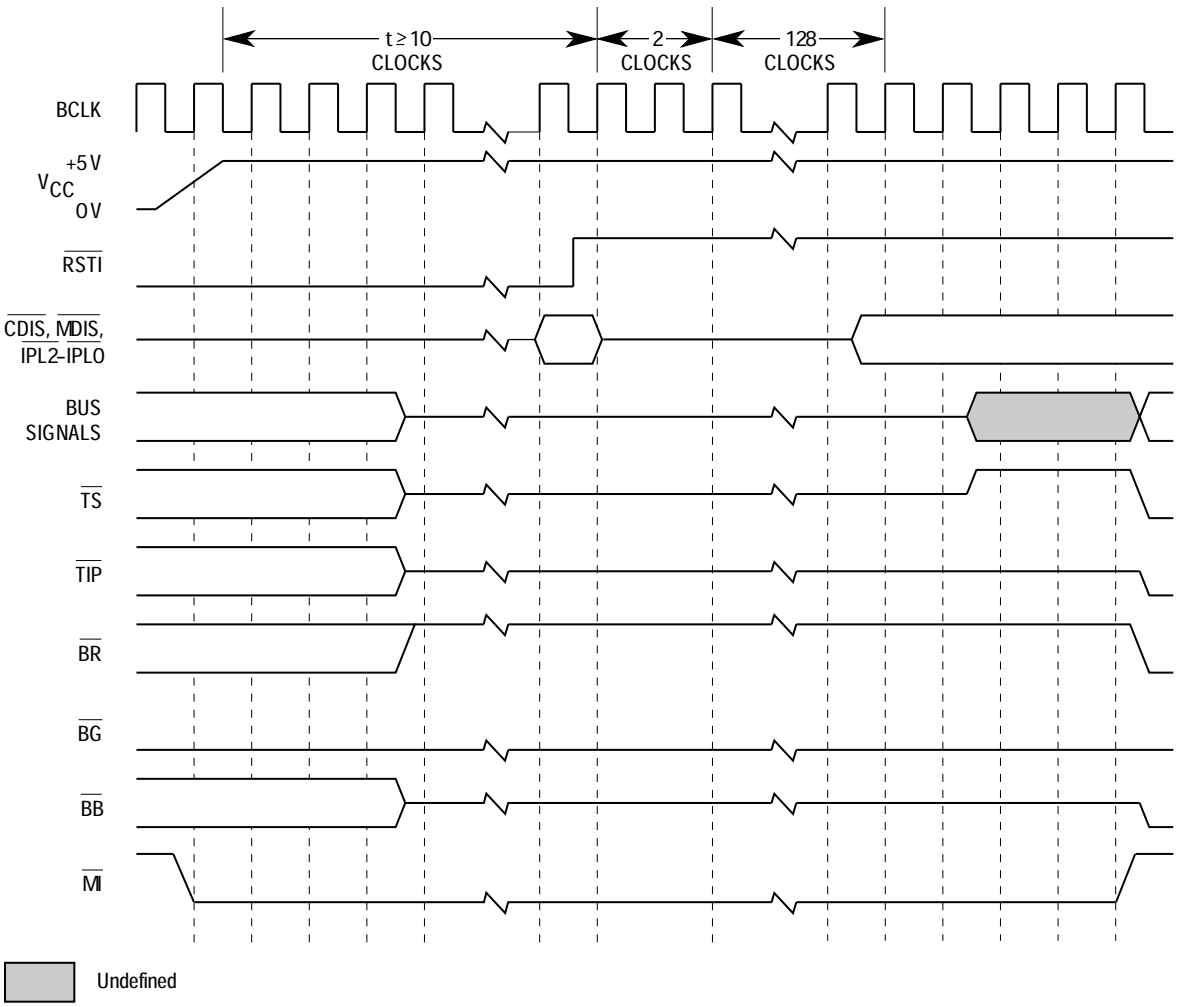


Figure 7-44. Initial Power-On Reset Timing

Once $\overline{RSTI}$ negates, the processor is internally held in reset for another 128 clock cycles. During the reset period, all signals that can be, are three-stated, and the rest are driven to their inactive state. Once the internal reset signal negates, all bus signals continue to remain in a high-impedance state until the processor is granted the bus. Afterwards, the first bus cycle for reset exception processing begins. In Figure 7-44 the processor assumes implicit bus ownership before the first bus cycle begins. The levels on $\overline{CDIS}$, $\overline{MDIS}$, and $\overline{IPL2}$–$\overline{IPL0}$ are used to selectively enable the special modes of operation when $\overline{RSTI}$ is negated. These signals should be driven to their normal levels before the end of the 128-clock internal reset period.

For processor resets after the initial power-on reset, $\overline{\text{RSTI}}$ should be asserted for at least 10 clock periods. Figure 7-45 illustrates timings associated with a reset when the processor is executing bus cycles. Note that $\overline{\text{BB}}$ and $\overline{\text{TIP}}$ (and $\overline{\text{TA}}$ if driven during a snooped access) are negated before transitioning to a three-state level.

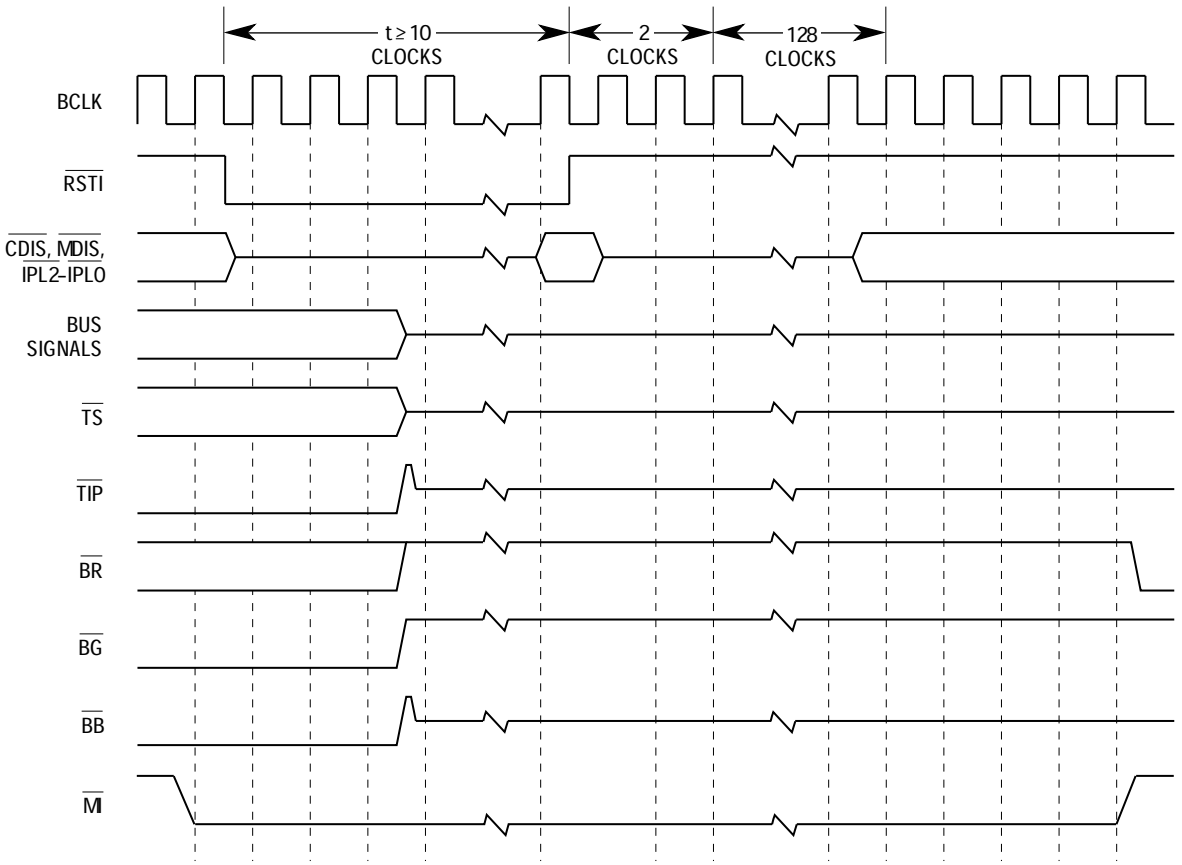


Figure 7-45. Normal Reset Timing

Resetting the processor causes any bus cycle in progress to terminate as if $\overline{\text{TA}}$ or $\overline{\text{TEA}}$ had been asserted. In addition, the processor initializes registers appropriately for a reset exception. **Section 8 Exception Processing** describes exception processing. When a RESET instruction is executed, the processor drives the reset out ($\overline{\text{RSTO}}$) 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 $\overline{\text{RSTO}}$ signal are reset at the completion of the RESET instruction. An $\overline{\text{RSTI}}$ signal that is asserted to the processor during execution of a RESET instruction immediately resets the processor and causes the $\overline{\text{RSTO}}$ signal to negate. $\overline{\text{RSTO}}$ can be logically ANDed with the external signal driving $\overline{\text{RSTI}}$ to derive a system reset signal that is asserted for both an external processor reset and execution of a RESET instruction.

7.11 SPECIAL MODES OF OPERATION

The MC68LC040 and MC68EC040 do not support the following three modes of operation, which for the M68040 are selectively enabled during processor reset and remain in effect until the next processor reset. Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for differences in the special modes of operation for the MC68LC040 and MC68EC040.

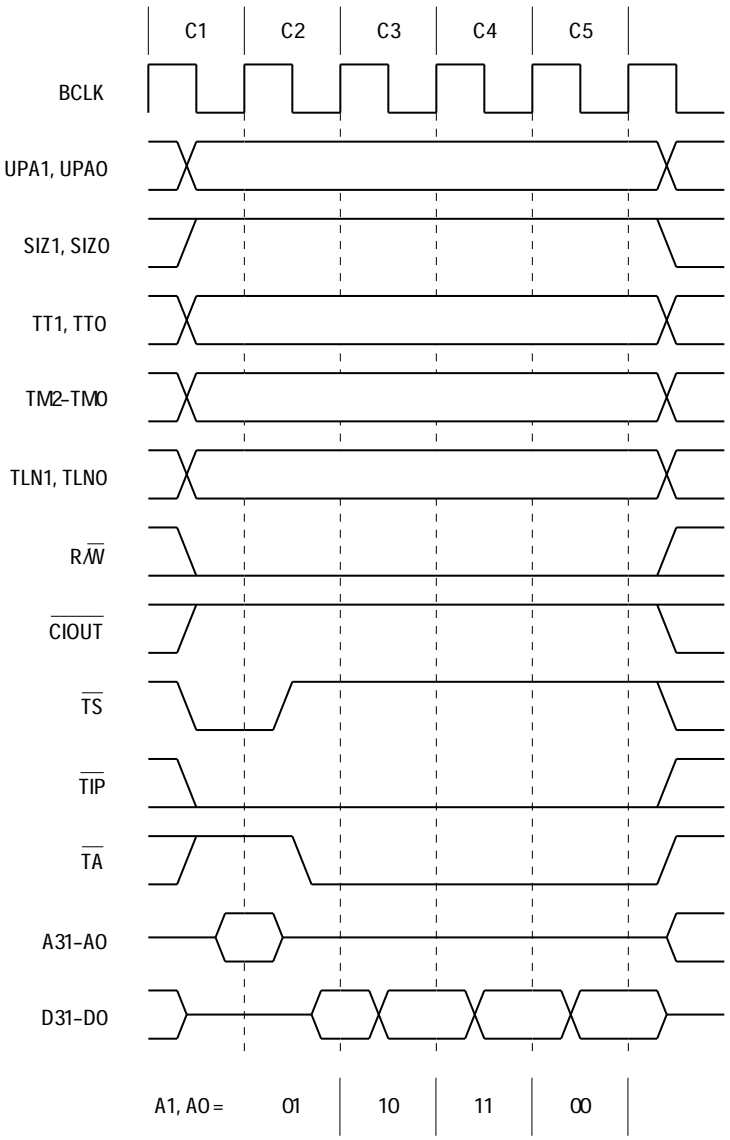
7.11.1 Output Buffer Impedance Selection

All output drivers in the M68040 can be configured to operate in either a large buffer mode (low-impedance driver) or small buffer mode (high-impedance driver). Large buffers have a nominal output impedance of 6 Ω for both high and low drive, resulting in minimum output delays. Signal traces driven by large buffers usually require transmission line effects to be considered in their design, including the use of signal termination. Small buffers have a nominal impedance of 25 Ω for high and low drive, resulting in longer output delays and less critical board-design requirements. Refer to **Section 11 MC68040 Electrical and Thermal Characteristics** for further information on electrical specifications, buffer characteristics, and transmission line design examples. The output drivers are configured in three groups. Each group of signals is configured depending on the corresponding $\overline{\text{IPLx}}$ signal level during processor reset (see Table 5-5).

7.11.2 Multiplexed Bus Mode

The multiplexed bus mode changes the timing of the three-state control logic for the address and data buses to support generation of a multiplexed address/data bus. When the M68040 is operating in this mode, the address and data bus signals can be hardwired together to form a single 32-bit bus, with address and data information time-multiplexed on the bus. This configuration minimizes the number of pins required to interface to peripheral devices without requiring additional discrete multiplexing logic. This mode is enabled during a processor reset by a logic zero on the $\overline{\text{CDIS}}$ signal.

Figure 7-46 illustrates a line write with multiplexed bus mode enabled. The address bus drivers are enabled during C1 and disabled during C2. Later in C2, the data bus drivers are enabled to drive the data bus with the data to be written. The address bus is only driven for the BCLK rising edge at the start of each bus cycle.



NOTE: The selected device increments the value of A3 and A2.

Figure 7-46. Multiplexed Address and Data Bus (Line Write)

7.11.3 Data Latch Enable Mode

The data latch enable (DLE) mode allows read data to be latched by the assertion of the DLE signal instead of by the BCLK rising edge at the end of each transfer. In some applications, this mode can reduce the number of clocks required to perform line burst reads. A logic zero on the $\overline{\text{MDIS}}$ enables this mode during a processor reset.

Figure 7-47 illustrates a conceptual block diagram of the logic used to latch the read data bus in DLE mode. The DLE signal controls transparent latch A, which allows data to be latched before the rising edge of BCLK. Latch A operates transparently when DLE is negated and latches the level on the data bus when DLE is asserted. Note that the DLE signal only controls latching of the read data and does not affect termination of the bus

transfer. Edge-triggered latch B is clocked by the rising edge of BCLK and latches the data from latch A for use by internal logic.

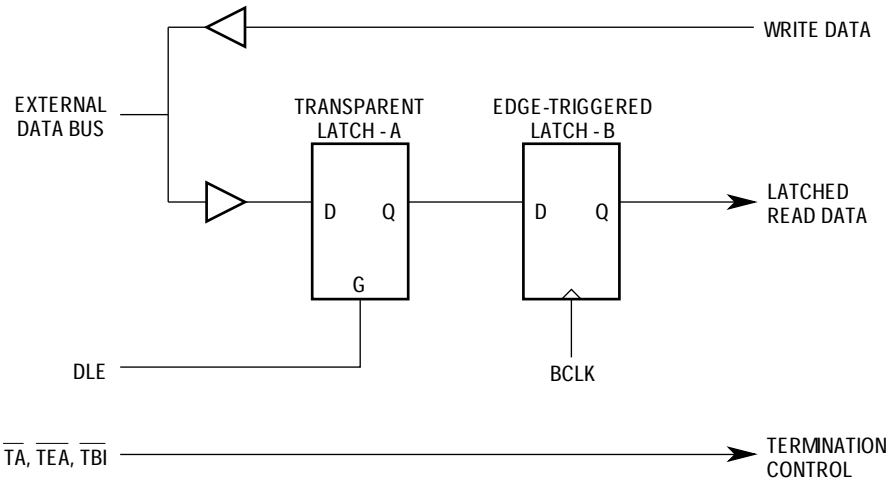


Figure 7-47. DLE Mode Block Diagram

Figure 7-48 illustrates the data read timing for both normal operation and DLE mode. During normal operation (i.e., DLE mode disabled), latch A is always transparent, and by the rising edge of BCLK, read data is latched. Data must meet setup and hold time specifications #15 and #16 in this case. When the DLE mode is enabled, the data can be latched by the rising edge of BCLK or the falling edge of DLE, depending on the timing for DLE.

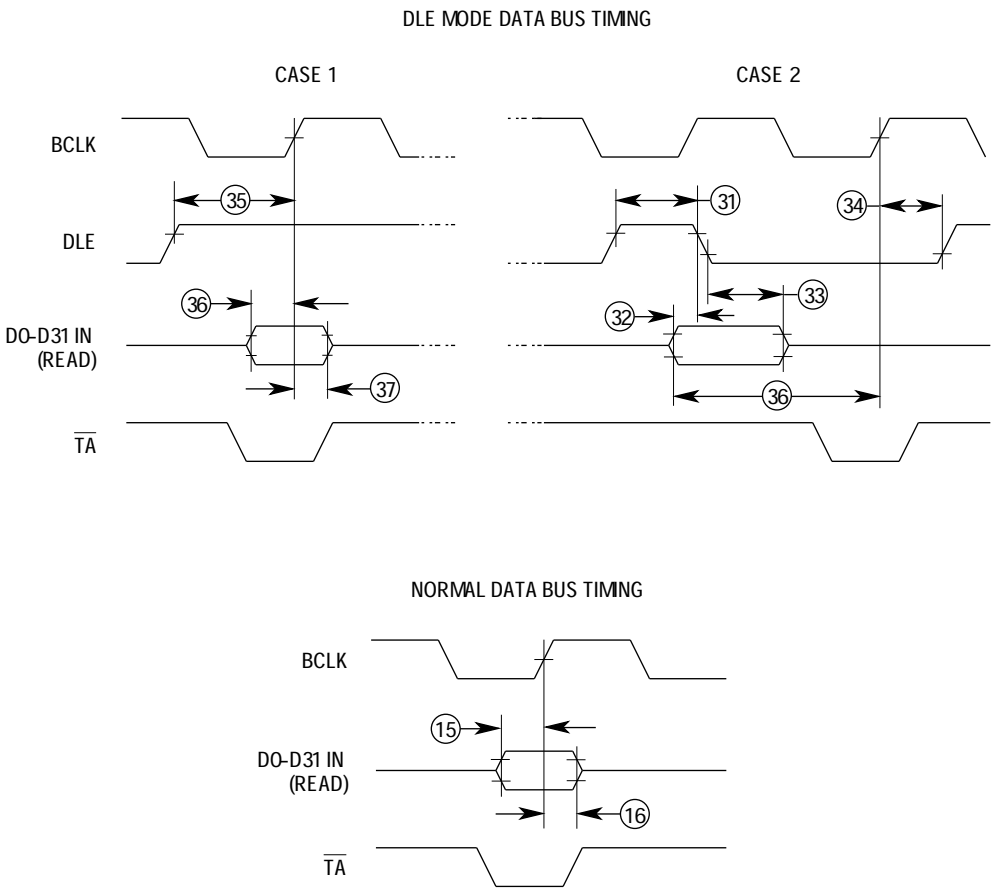


Figure 7-48. DLE versus Normal Data Read Timing

Case 1

If DLE is negated and meets setup time specification #35 to the rising edge of BCLK when the bus read is terminated, latch A is transparent, and the read data must meet setup and hold time specifications #36 and #37 to the rising edge of BCLK. Read timing is similar to normal timing for this case.

Case 2

If DLE is asserted, the data bus levels are latched and held internally. D31–D0 must meet setup and hold time specifications #32 and #33 to the falling edge of DLE, and can transition to a new level once DLE is asserted. D31–D0 must still meet setup time specification #36 to BCLK, but not hold time specification #37, since the data is internally held valid as long as DLE remains asserted low.

SECTION 8
EXCEPTION PROCESSING

Exception processing is the activity performed by the processor in preparing to execute a special routine for any condition that causes an exception. In particular, exception processing does not include execution of the routine itself. This section describes the processing for each type of integer unit exception, exception priorities, the return from an exception, and bus fault recovery. This section also describes the formats of the exception stack frames. For details on floating-point exceptions refer to **Section 9 Floating-Point Unit (MC68040 Only)**.

NOTE

For the MC68040V, MC68LC040, MC68EC040, and MC68EC040V ignore all references to floating-point, including any instructions that begin with an “F”. Also, for the MC68EC040 and MC68EC040V ignore all references to the memory management unit (MMU) and the instructions PFLUSH and PTEST. The functionality of the MC68040 transparent translation register has been changed in the MC68EC040 and MC68EC040V to the access control registers (ACR). Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for details.

8.1 EXCEPTION PROCESSING OVERVIEW

Exception processing is the transition from the normal processing of a program to the processing required for any special internal or external condition that preempts normal processing. External conditions that cause exceptions are interrupts from external devices, bus errors, and resets. Internal conditions that cause exceptions are instructions, address errors, and tracing. For example, the TRAP, TRAPcc, FTRAPcc, CHK, RTE, DIV, and FDIV instructions can generate exceptions as part of their normal execution. In addition, illegal instructions, unimplemented floating-point instructions and data types, and privilege violations cause exceptions. Exception processing uses an exception vector table and an exception stack frame. The following paragraphs describe the vector table and a generalized exception stack frame.

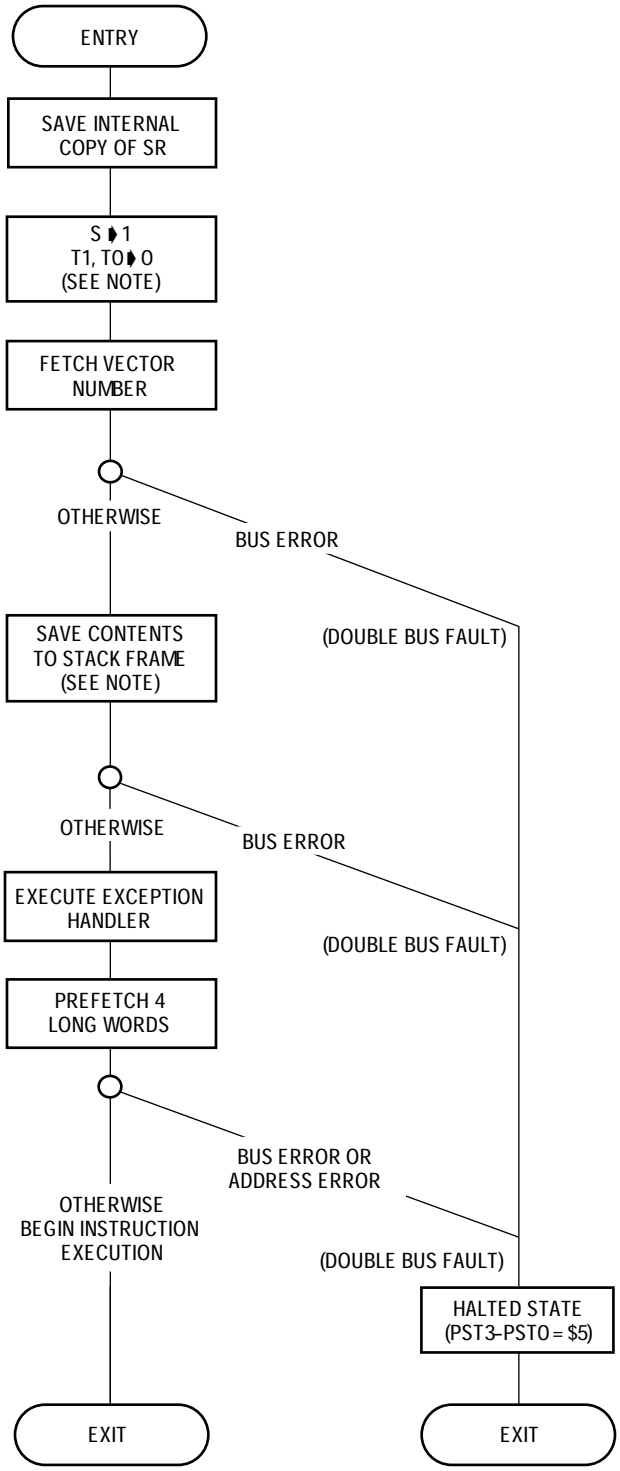
The M68040 uses a restart exception processing model to minimize interrupt and instruction latency and to reduce the size of the stack frame (compared to the frame required for a continuation model). Exceptions are recognized at each instruction boundary in the execute stage of the integer pipeline and force later instructions that have not yet reached the execute stage to be aborted. Instructions that cannot 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. Figure 8-1 illustrates a general flowchart for the steps taken by the processor during exception processing.

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

During the second step, the processor determines the vector number for the exception. For interrupts, the processor performs an interrupt acknowledge bus 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.



NOTE: These blocks vary for reset and interrupt exceptions.

Figure 8-1. General Exception Processing Flowchart

The third step is to save the current processor contents for all exceptions other than reset. The processor creates one of five exception stack frame formats on the active supervisor stack and fills it with information appropriate for the type of exception. Other information can 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 M-bit of the SR is set, the processor clears the M-bit and builds a second stack frame on the interrupt stack. Figure 8-2 illustrates the general form of the exception stack frame.

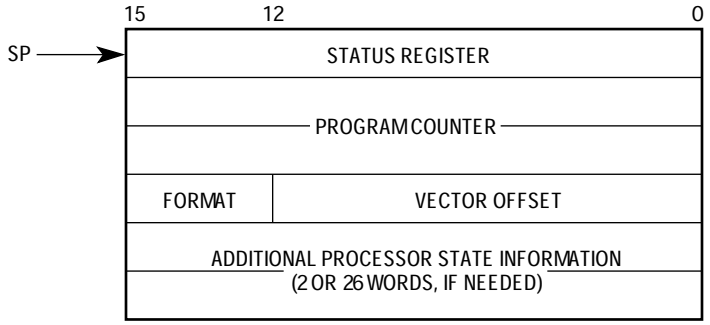


Figure 8-2. General Form of Exception Stack Frame

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 (PC) (and the interrupt stack pointer (ISP) for the reset exception) from the exception vector table entry. After prefetching the first four long words to fill the instruction pipe, the processor resumes normal processing at the address in the PC. 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.

All exception vectors are located in the 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 exception vector table, the exception vector table can be located anywhere in memory; it can even be dynamically relocated for each task that an operating system executes.

The M68040 supports a 1024-byte vector table containing 256 exception vectors (see Table 8-1). Motorola defines the first 64 vectors and reserves the other 192 vectors for user-defined interrupt vectors. External devices can use vectors reserved for internal purposes at the discretion of the system designer. External devices can also supply vector numbers for some exceptions. External devices that cannot supply vector numbers use the autovector capability, which allows the M68040 to automatically generate a vector number.

Table 8-1. 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	FTRAPcc, TRAPcc, TRAPV Instructions
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 M68040
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	Floating-Point Exception Vectors (see Note)
56	0E0	Defined for MC68030 and MC68851, not used by M68040
57	0E4	Defined for MC68851, not used by M68040
58	0E8	Defined for MC68851, not used by M68040
59–63	0EC–0FC	(Unassigned, Reserved)
64–255	100–3FC	User Defined Vectors (192)

NOTE: Refer to **Section 9 Floating-Point Unit (MC68040 Only)**.

8.2 INTEGER UNIT EXCEPTIONS

The following paragraphs describe the external interrupt exceptions and the different types of exceptions generated internally by the M68040 integer unit. The following exceptions are discussed:

- Access Fault
- Address Error
- Instruction Trap
- Illegal and Unimplemented Instructions
- Privilege Violation

- Trace
- Format Error
- Breakpoint Instruction
- Interrupt
- Reset

8.2.1 Access Fault Exception

An access fault exception occurs when a data or instruction prefetch access faults due to either an external bus error or an internal access fault. Both types of access faults are treated identically and the access fault exception handler or a status bit in the access fault stack frame distinguishes them. An access fault exception may or may not be taken immediately, depending on whether the faulted access specifically references data required by the execution unit or whether there are any other exceptions that can occur, allowing the execution pipeline to idle.

An external access fault (bus error) occurs when external logic aborts a bus cycle and asserts the $\overline{\text{TEA}}$ input signal. A bus error on a data write access always results in an access fault exception, causing the processor to begin 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 that references data in these transfers.

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

Processor accesses for either data or instructions can result in internal access faults. Internal access faults must be corrected to complete execution of the current context. Four types of internal access faults can occur:

1. Push transfer faults occur when the execution unit is idle, the integer unit pipeline is frozen, the instruction and data cache requests are cancelled (however, writes are not lost), and pending writes are stacked.
2. Data access faults occur when the bus controller and the execution unit are idle. A data access fault freezes the pipeline and cancels any pending instruction cache accesses. Pending writes are stacked because the data cache is deadlocked until stacking transfers are initiated.

3. Instruction access faults occur when the PC section is deadlocked because of the faulted data or another prefetch is required, the copyback stage is empty, and the data cache and bus controller are idle. Since instruction access faults are reset, they can be ignored.
4. An internal access fault also occurs when the data or instruction MMU detects that a successful address translation is not possible because the page is write protected, supervisor only, or nonresident. Furthermore, when an address translation cache (ATC) miss occurs, the processor searches the translation tables in memory for the mapping, and then retries the access. If a valid translation for the logical address is not available due to a problem encountered during the table search, an internal access fault occurs when the aborted access is retried. The problem encountered could be either an invalid descriptor or the assertion of the $\overline{\text{TEA}}$ signal during a bus cycle used to access the translation tables. A miss in the ATC causes the processor to automatically initiate a table search but does not cause an internal access fault unless one of the three previous conditions is encountered. However, this is not true if the memory management unit (MMU) is disabled.

When an exception is detected, all parts of the execution unit either remain or are forced to idle, at which time the highest priority exception is taken. Restarting the instruction or a user-defined supervisor cleanup exception handler routine regenerates lower priority exceptions on the return from exception handling. Internal access faults and bus errors are reported after all other pending integer instructions complete execution. If an exception is generated during completion of the earlier instructions, the pending instruction fault is cleared, and the new exception is serviced first. The processor restarts the pending prefetch after completing exception handling for the earlier instructions and takes a bus error exception if the access faults again. For data access faults, the processor aborts current instruction execution. If a data access fault is detected, the processor waits for the current instruction prefetch bus cycle to complete, then begins exception processing immediately.

As illustrated in Figure 8-1, the processor begins exception processing for an access fault 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 access fault vector. It saves the vector offset, PC, and internal copy of the SR on the stack. The saved PC value is the logical address of the instruction executing at the time the fault was detected. This instruction is not necessarily the one that initiated the bus cycle since the processor overlaps execution of instructions. It also saves information to allow continuation after a fault during a MOVEM instruction and to support other pending exceptions. The faulted address and pending write-back information is saved. The information saved on the stack is sufficient to identify the cause of the bus error, complete pending write-backs, and recover from the error. The exception handler must complete the pending write-backs. Up to three write-backs can be pending for push errors and data access errors.

If a bus error occurs during the exception processing for an access fault, 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 encoding $5. 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 or internal access fault. Memory pages allocated to the stack that are higher in memory than the current stack pointer can be nonresident since an RTE or FRESTORE instruction can check for residency and trap before restoring the state.

A special case exists for systems that allow arbitration of the processor bus during locked transfer sequences. If the arbiter can signal a bus error of a locked translation table update due to an improperly broken lock, any pages touched by exception stack operations must have the U-bit set in the corresponding page descriptor to prevent the occurrence of the locked access during translation table searches.

8.2.2 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 have completed. If the completion of these instructions generates another exception, the address error exception is deferred, and the new exception is serviced. After exception processing for the address error exception commences, the sequence is the same as an access fault exception, except that the vector number is 3 and the vector offset in the stack frame refers to the address error vector. The stack frame is generated containing the address of the instruction that caused the address error and the address itself (A0 is cleared). If an address error occurs during the exception processing for a bus error, address error, or reset, a double bus fault occurs.

8.2.3 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, FTRAPcc, TRAPV, CHK, and CHK2 instructions force exceptions if the user program detects an error, which can 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.

As illustrated in Figure 8-1, when a trap exception occurs, the processor internally copies the 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 FTRAPcc, 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 is prefetched.

8.2.4 Illegal Instruction and Unimplemented Instruction Exceptions

An illegal instruction exception corresponds to vector number 4, and occurs when the processor attempts to execute an illegal instruction. An illegal instruction is an instruction that contains any bit pattern that does not correspond to the bit pattern of a valid M68040 instruction. An illegal instruction exception is also taken after a breakpoint acknowledge bus cycle is terminated, either by the assertion of the transfer acknowledge ($\overline{TA}$) or the transfer error acknowledge ($\overline{TEA}$) signal. An illegal instruction exception can also be a MOVEC instruction with an undefined register specification field in the first extension word.

Instruction word patterns with bits 15–12 equal to $A do not correspond to legal instructions for the M68040 and are treated as unimplemented instructions. $A word patterns are referred to as an unimplemented instruction 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. For instruction word patterns with bits 15–12 equal to $F refer to **Section 9 Floating-Point Unit (MC68040 Only)**.

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 T1 and T0, disabling further tracing. The processor generates the vector number, either 4 or 10, 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 exception handling routine to adjust the stacked PC if the instruction is emulated in software or is to be skipped on return from the exception handler.

8.2.5 Privilege Violation Exception

To provide system security, some instructions are privileged. An attempt to execute one of the following privileged instructions while in the user mode causes a privilege violation exception:

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

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. As illustrated in Figure 8-1, the processor copies the SR, enters the supervisor mode, and clears the trace bits. The processor generates vector number 8, saves the privilege violation vector offset, 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.

8.2.6 Trace Exception

To aid in program development, the M68000 family includes an instruction-by-instruction tracing capability. The M68040 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 the instruction completes execution, allowing a debugging program to monitor execution of a program.

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 trace exception processing is complete. If an instruction does not complete due to an access fault or address error exception, trace exception processing is deferred until after execution of the suspended instruction is resumed. If an interrupt is pending at the completion of an instruction, trace exception processing occurs before 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.

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. T1 and T0 bit = $1 causes an instruction that forces a change of flow to take a trace exception. The following instructions cause a trace exception to be taken when trace on change of flow is enabled.

ANDI to SR	CAS2	FBcc (Taken)	JMP	MOVES	RTD
Bcc (Taken)	CINV	FDBcc (Always)	JSR	NOP	RTE
BRA	CPUSH	FMOVEM	MOVE to SR	ORI to SR	RTR
BSR	DBcc (Taken)	FRESTORE	MOVE USP	PFLUSH	RTS
CAS	EORI to SR	FSAVE	MOVEC	PTEST	STOP

Instructions that increment the PC normally do not take the trace exception. This mode also includes SR manipulations because the processor must prefetch instruction words again to fill the pipeline any time an instruction that modifies the SR is executed. Table 8-2 lists the different trace modes.

Table 8-2. Tracing Control

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

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 the instruction is not executed. This is of particular importance to an instruction emulation routine that performs the instruction function, adjusts the stacked PC 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 also be emulated for the trace exception handler to account for the emulated instruction.

Trace exception processing starts at the end of normal processing for the traced instruction and before the start of the next instruction. As illustrated in Figure 8-1, the processor makes an internal copy of the SR, and enters the supervisor mode. It also clears the T1 and T0 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 internal 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.

When the STOP instruction is traced, the processor never enters the stopped condition. A STOP instruction that begins execution with the trace bits equal to $3 forces a trace exception after it loads the SR. Upon return from the trace exception handler, execution continues with the instruction following the STOP instruction, and the processor never enters the stopped condition.

8.2.7 Format Error Exception

Just as the processor checks for valid prefetched instructions, it also performs some checks of data values for control operations. The RTE instruction checks the validity of the stack format code. For floating-point unit (FPU) state frames, the FRESTORE instruction compares the internal version number of the processor to that contained in the state frame (refer to **Section 9 Floating-Point Unit (MC68040 Only)**). This check ensures that the processor can correctly interpret internal FPU state information from the state frame. If any of these checks determine that the format of the data is improper, the instruction generates a format error exception. This exception saves a 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.

8.2.8 Breakpoint Instruction Exception

To use the M68040 in a hardware emulator, the processor must provide a means of inserting breakpoints in the emulator code and performing appropriate operations at each breakpoint. Inserting an illegal instruction at the breakpoint and detecting the illegal instruction exception from its vector location can achieve this. However, since the VBR allows arbitrary relocation of exception vectors, the exception address cannot reliably identify a breakpoint. Consequently, the processor provides a breakpoint capability with a set of breakpoint exceptions, $4848–$484F.

When the M68040 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 7 Bus Operation** for a description of the breakpoint acknowledge cycle. After external hardware terminates the bus cycle with either $\overline{TA}$ or $\overline{TEA}$, the processor performs illegal instruction exception processing.

8.2.9 Interrupt Exception

When a peripheral device requires the services of the M68040 or is ready to send information that the processor requires, it can signal the processor to take an interrupt exception using the active-low $\overline{IPL2}$–$\overline{IPL0}$ signals. The three signals encode a value of 0–7 ($\overline{IPL0}$ is the least significant bit). High levels on all three signals correspond to no interrupt requested (level 0). Values 1–7 specify one of seven levels of interrupts, with level 7 having the highest priority. Table 8-3 lists the interrupt levels, the states of $\overline{IPL2}$–$\overline{IPL0}$ that define each level, and the SR interrupt mask value that allows an interrupt at each level.

Table 8-3. Interrupt Levels and Mask Values

Requested Interrupt Level	Control Line Status			Interrupt Mask Level Required for Recognition
	$\overline{IPL2}$	$\overline{IPL1}$	$\overline{IPL0}$	
0	High	High	High	No Interrupt Requested
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

When an interrupt request has a priority higher than the value in the interrupt priority mask of the SR (bits 10–8), the processor makes the request a pending interrupt. Priority level 7, the nonmaskable interrupt, is a special case. Level 7 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 7, regardless of the value in the mask. Figure 8-3 shows two examples of interrupt recognitions, one for level 6 and one for level 7. When the M68040 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 6 interrupts and lower level interrupts are masked. Provided no instruction that lowers the mask value is executed, the external request can be lowered to level 3 and then raised back to level 6 and a second level 6 interrupt is not processed. However, if the M68040 is handling a level 7 interrupt (SR mask set to level 7) and the external request is lowered to level 3 and than raised back to level 7, a second level 7 interrupt is processed. The second level 7 interrupt is processed because the level 7 interrupt is transition sensitive. A level comparison also generates a level 7 interrupt if the request level and mask level are at 7 and the priority mask is then set to a lower level (with the MOVE to SR or RTE instruction, for example). The level 6 interrupt request and mask level example in Figure 8-3 is the same as for all interrupt levels except 7.

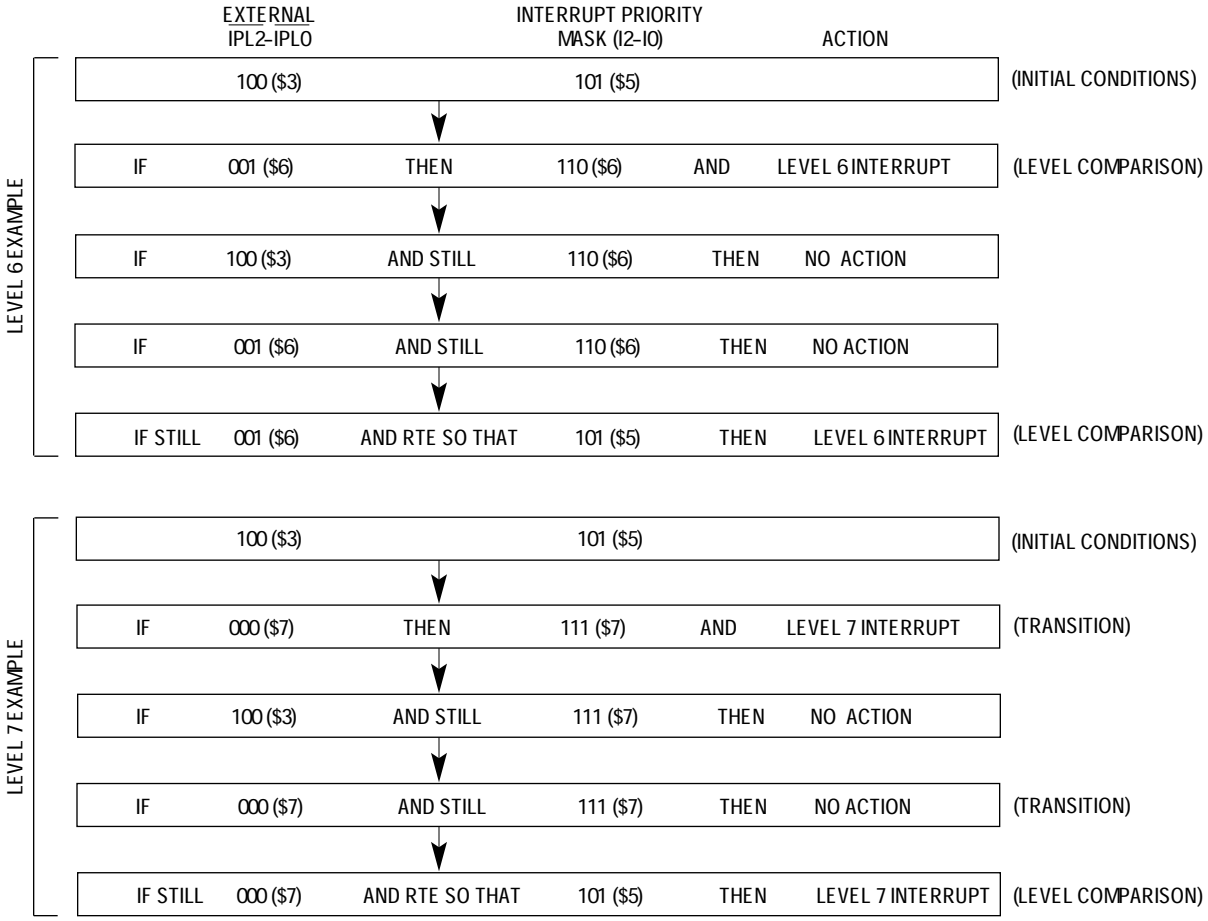


Figure 8-3. Interrupt Recognition Examples

Note that a mask value of 6 and a mask value of 7 both inhibit request levels of 1–6 from being recognized. In addition, neither masks a transition to an interrupt request level of 7. The only difference between mask values of 6 and 7 occurs when the interrupt request level is 7 and the mask value is 7. If the mask value is lowered to 6, a second level 7 interrupt is recognized.

External circuitry can chain or otherwise merge signals from devices at each level, allowing an unlimited number of devices to interrupt the processor. When several devices are connected to the same interrupt level, each device should hold its interrupt priority level constant until its corresponding interrupt acknowledge bus cycle ensures that all requests are processed. Refer to **Section 7 Bus Operation** for details on the interrupt acknowledge cycle.

Figure 8-4 illustrates a flowchart for interrupt exception processing. 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 ($\overline{\text{AVEC}}$) must be asserted. In this case, the M68040 uses an internally generated autovector, which is one of vector numbers 25–31, that corresponds to the interrupt level number (see Table 8-1). 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 has the S-bit set, the M-bit clear, and the interrupt mask level set to the new interrupt level. 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 acknowledge operation for 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.

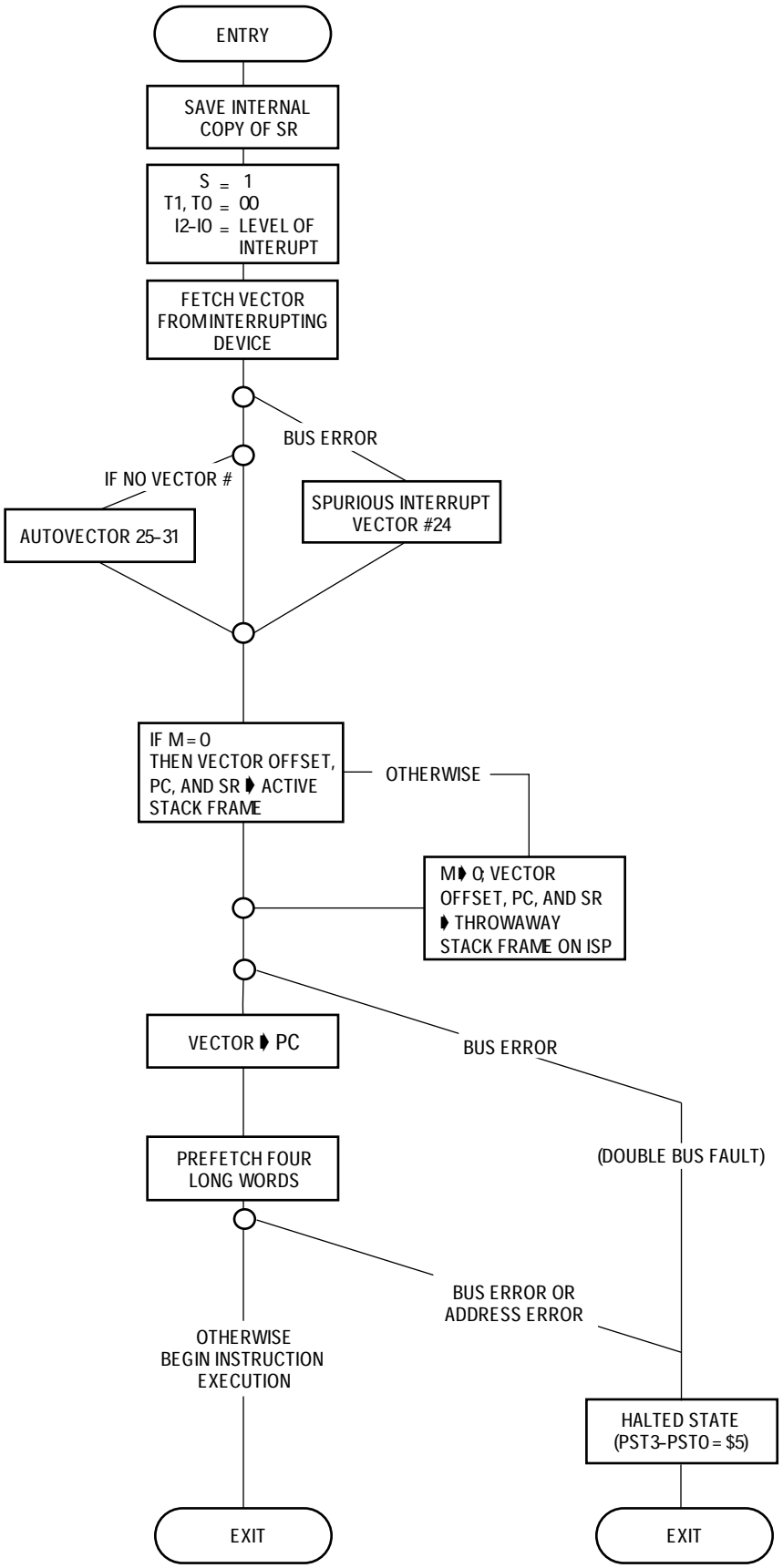


Figure 8-4. Interrupt Exception Processing Flowchart

8.2.10 Reset Exception

Asserting 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. Reset also aborts any processing in progress when $\overline{\text{RSTI}}$ is recognized; processing cannot be recovered. Figure 8-5 is a flowchart of the reset exception processing.

The reset exception places the processor in the interrupt mode of the supervisor privilege mode by setting the S-bit and clearing the M-bit and disables tracing by clearing the T1 and T0 bits in the SR. This exception also sets the processor's interrupt priority mask in the SR to the highest level, level 7. Next the VBR is initialized to zero ($00000000), and the enable bits in the cache control register (CACR) for the on-chip caches are cleared. The reset exception also clears the enable bit but does not affect page size in the translation control registers. It clears the enable bit in each of the four transparent translation registers. An interrupt acknowledge bus cycle is begun to generate a vector number. This vector number references the reset exception vector (two long words, vector numbers 0 and 1) at offset zero in the supervisor address space. The first long word is loaded into the interrupt stack pointer, and the second long word is loaded into the PC. Reset exception processing concludes with the prefetch of the first four long words beginning at the memory location pointed to by the PC.

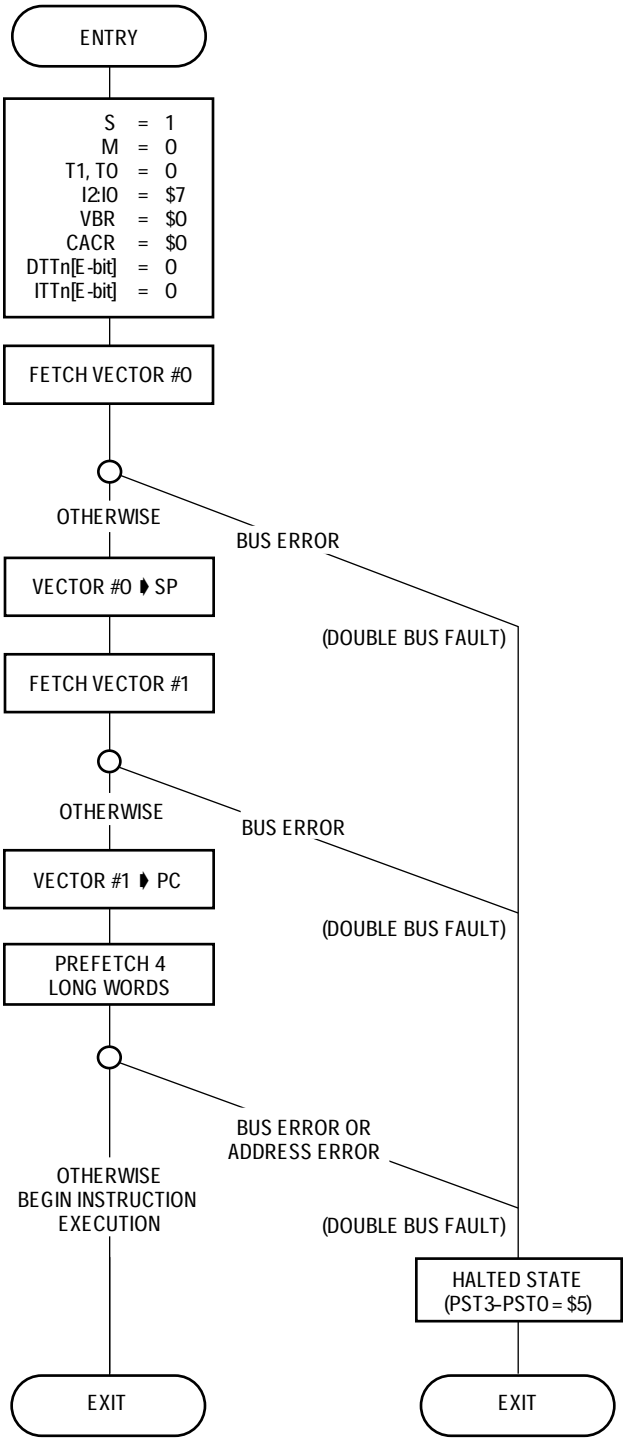


Figure 8-5. Reset Exception Processing Flowchart

After the initial instruction is prefetched, program execution begins at the address in the PC. The reset exception does not flush the ATCs or invalidate entries in the instruction or data caches; it does not save the value of either the PC or the SR. If an access fault or address error occurs during the exception processing sequence for a reset, a double bus fault is generated. The processor halts, and the processor status (PST3–PST0) signals indicate $5. Execution of the reset instruction does not cause a reset exception, or affect

any internal registers, but it does cause the M68040 to assert the reset out ($\overline{\text{RSTO}}$) signal, resetting all external devices.

8.3 EXCEPTION PRIORITIES

When several exceptions occur simultaneously, they are processed according to a fixed priority. Table 8-4 lists the exceptions, grouped by characteristics. Each group has a priority, from 0–7, with 0 as the highest priority.

Table 8-4. 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 (ATC Fault or Bus Error)	Aborts current instructions; can have pending trace, floating-point post-instruction, or unimplemented floating-point instruction exceptions.
2	Floating-Point Pre-Instruction*	Exception processing begins before current floating-point instruction is executed. Instruction is restarted on return from exception.
3	BKPT #n, CHK, CHK2, Divide by Zero, FTRAPcc, RTE, TRAP#n, TRAPV	Exception processing is part of instruction execution.
	Illegal Instruction, Unimplemented A- and F-Line, Privilege Violation	Exception processing begins before instruction is executed.
	Unimplemented Floating-Point Instruction*	Exception processing begins after memory operands are fetched and before instruction is executed.
4	Floating-Point Post-Instruction*	Only reported for FMOVE to memory. Exception processing begins when FMOVE instruction and previous exception processing have completed.
5	Address Error	Reported after all previous instructions and associated exceptions have completed.
6	Trace	Exception processing begins when current instruction or previous exception processing has completed.
7	Instruction Access Error (ATC Fault or Bus Error)	Reported after all previous instructions and associated exceptions have completed.
8	Interrupt	Exception processing begins when current instruction or previous exception processing has completed.

* Refer to **Section 9 Floating-Point Unit (MC68040 Only)** for details concerning floating-point instructions.

The method used to process exceptions in the M68040 is significantly different from that used in earlier members of the M68000 processor family due to the restart exception model. 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 restarts. Note that the reset operation clears all other exceptions except in the following circumstances:

- As soon as the M68040 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 format $7 stack frame that contains status information that can indicate a pending trace, floating-point post-instruction, 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.
- If an access error, trace, and one of the two (mutually exclusive) floating-point exceptions occur simultaneously, the pending floating-point exception is indicated in the access error stack and the trace exception flag is undefined. The exception handler for the floating-point exception must check the trace bits on the stack and call the trace handler directly (after adjusting the stack frame to match the format for the trace exception).
- If a trace exception is pending at the same time an exception priority level 3 or floating-point post-instruction exception is pending, the trace exception is not reported, and the exception handler for the other exception condition must check for the trace condition.

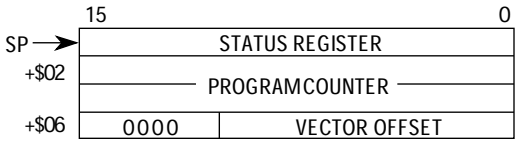
8.4 RETURN FROM EXCEPTIONS

After the processor has completed executing the exception handlers for all pending exceptions, the processor resumes normal instruction execution at the address in the processor's vector table for the last exception processed. Once the exception handler has completed execution, if possible the processor must return the system context as it was prior to the exception using the RTE instruction. (If the internal data of the exception stack frames are manipulated, M68040 may enter into an undefined state; this applies specifically to the SSW on the access error stack frame.)

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. If during restoration, a stack frame has an odd address PC and an SR that indicates user trace mode enabled, then an address error is taken. The SR stacked for the address error has the SR S-bit set. For previous members of the M68000 family the S-bit is clear. When the M68040 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 format of stack frame for any type of exception. The following paragraphs discuss in detail each stack frame format.

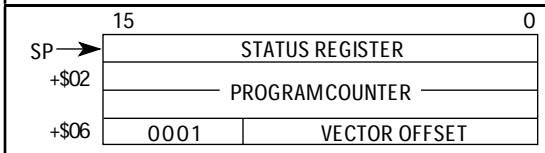
8.4.1 Four-Word Stack Frame (Format $0)

If a four-word stack frame is on the active stack and an RTE instruction is encountered, 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.

Stack Frames	Exception Types	Stacked PC Points To
 FOUR-WORD STACK FRAME–FORMAT $0	• Interrupt • Format Error • TRAP #N • Illegal Instruction • A-Line Instruction • F-Line Instruction • Privilege Violation • Floating-Point Pre-Instruction	• Next Instruction • RTE or RESTORE Instruction • Next Instruction • Illegal Instruction • A-Line Instruction • F-Line Instruction • First Word of Instruction Causing Privilege Violation • Floating-Point Pre-Instruction Exception

8.4.2 Four-Word Throwaway Stack Frame (Format $1)

If a four-word throwaway stack frame is on the active stack and an RTE instruction is encountered, the processor increments the active stack pointer by eight, updates the SR with the value read from the stack, and then begins RTE processing again, as illustrated in Figure 8-6. 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-bit and M-bit are set. In that case, there is a normal four-word frame on the master stack. However, the second frame can be any format (even another throwaway frame) and can reside on any of the three system stacks.

Stack Frames	Exception Types	Stacked PC Points To
 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 as on master stack.

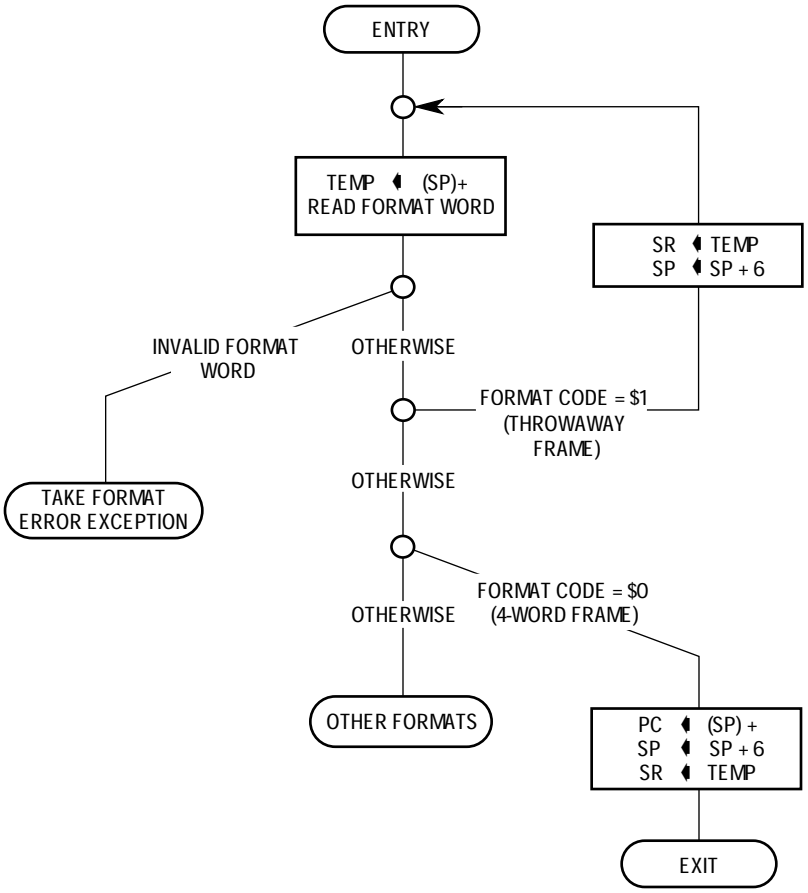


Figure 8-6. Flowchart of RTE Instruction for Throwaway Four-Word Frame

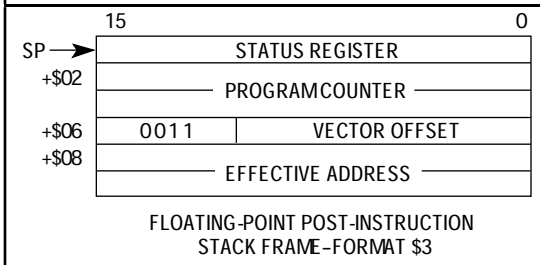
8.4.3 Six-Word Stack Frame (Format $2)

If a six-word throwaway stack frame is on the active stack and an RTE instruction is encountered, the processor restores the SR and PC values from the stack, increments the active supervisor stack pointer by $C, and resumes normal instruction execution.

Stack Frames	Exception Types	Stacked PC Points To
15 0 SP → STATUS REGISTER +$02 PROGRAM COUNTER +$06 0010 VECTOR OFFSET +$08 ADDRESS SIX-WORD STACK FRAME–FORMAT $2	• CHK, CHK2, TRAPcc, FTRAPcc, TRAPV, Trace, or Zero Divide • Unimplemented Floating-Point Instruction • Address Error	• Next Instruction: address is the address of the instruction that caused the exception. • Next Instruction: address is the calculated <ea> for the floating-point instruction. • Instruction that caused the address error, address is the reference address – 1.

8.4.4 Floating-Point Post-Instruction Stack Frame (Format $3)

The processor restores the SR and PC values from the stack and increments the active supervisor stack pointer by $C. If another floating-point post-instruction exception is pending, exception processing begins immediately for the new exception; otherwise, the processor resumes normal instruction execution.

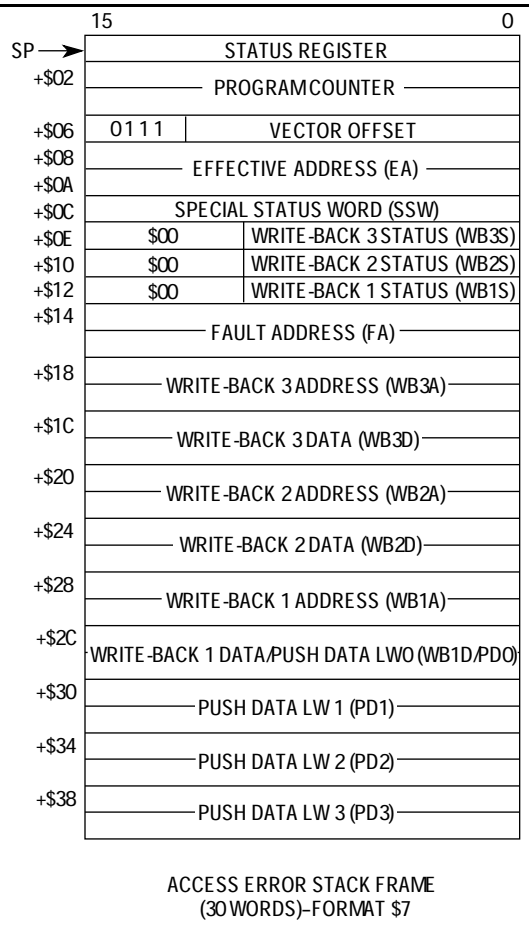
Stack Frames	Exception Types	Stacked PC Points To
 FLOATING-POINT POST-INSTRUCTION STACK FRAME–FORMAT $3	• Floating-Point Post-Instruction	• Next Instruction: <ea> is the calculated effective address for the floating-point instruction.

8.4.5 Eight-Word Stack Frame (Format $4)

The MC68040V, MC68LC040, MC68EC040, and MC68EC040V use this stack frame for unimplemented floating-point instructions. The MC68040 does not generate or recognize this format stack frame. Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for further details about this stack frame.

8.4.6 Access Error Stack Frame (Format $7)

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 write-backs that the access error exception handler must complete. The following paragraphs describe in detail the format for this frame and how the processor uses it when returning from exception processing.

Stack Frames	Exception Types	Stacked PC Points To
	• Data or Instruction Access Fault (ATC Fault or Bus Error)	• Next Instruction

8.4.6.1 EFFECTIVE ADDRESS. The effective address contains address information when one of the continuation flags CM, CT, CU, or CP in the SSW is set.

8.4.6.2 SPECIAL STATUS WORD (SSW). The SSW information indicates whether an access to the instruction stream or the data stream (or both) caused the fault and contains status information for the faulted access. Figure 8-7 illustrates the SSW format.

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

Figure 8-7. Special Status Word Format

CP—Continuation of Floating-Point Post-Instruction Exception Pending

CP is set for an access error with a floating-point post-instruction exception pending. 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 effective address field contains the address of the instruction being traced. The RTE fetches the appropriate floating-point post-instruction exception vector.

When a post-instruction exception occurs during tracing, the post-instruction exception takes precedence. CP is set, and CT = 0 and can be traced. The kernel must check for a trace condition using the stacked SR. The effective address field contains the calculated effective address determined by the effective address field of the floating-point instruction that caused the post-instruction exception.

CU—Continuation of 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 CP flag except the RTE fetches the F-line exception vector. The effective address field contains the calculated effective address determined by the effective address field of the unimplemented instruction.

When an unimplemented floating-point instruction is traced, the unimplemented exception takes precedence, CU is set, and CT = 0. The kernel must check for a trace condition using the stacked SR. If this condition is true, create the required stack frame and jump directly to the trace handler.

CT—Continuation of Trace Exception Pending

CT is set for an access error with a pending trace exception. Operation is the same as for the CP flag. When RTE is executed with CT set, the M68040 will move the words on the stack an offset of $00–$0B from the current SP to offset $30–$3B, adjusting the stack pointer by +$30. The M68040 changes the stack frame format to $2 before fetching the trace exception vector and jumping directly to trace exception handling. This stack adjustment creates the stack frame that normally would have been created for the trace exception had the pending access not encountered a bus error.

CM—Continuation of MOVEM Instruction Execution Pending

CM is set if a data access encounters a bus error for a MOVEM. Since the MOVEM operation can write over the memory location or registers used to calculate the effective address, the M68040 internally saves the effective address after calculation. When MOVEM encounters a bus error, a stack frame is created with CM set, and the effective address field contains the calculated effective address for the instruction. When RTE is executed, MOVEM restarts using the effective address on the stack (instead of repeating the effective address calculate operation) if the address mode is PC relative (mode = 111, register = 010 or 011) or indirect with index (mode = 110).

MA—Misaligned Access

MA is set if an ATC fault occurs for second-page access that spans two pages in memory.

ATC—ATC Fault

This bit is set for an ATC fault due to a nonresident entry (bus error during table search or invalid descriptor encountered) or privilege violation (write protected or supervisor only). It is cleared for a bus-errored instruction, data, or cache line-push access.

LK—Locked Transfer (Read-Modify-Write)

This bit is set if a fault occurred on a locked transfer; it is cleared otherwise.

RW—Read/Write

This bit is set if a fault occurred on a read transfer; it is cleared otherwise.

X—Undefined

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 encounters a bus error, the SIZE field in the resulting stack frame indicates the size of the original read generated by the execution unit.

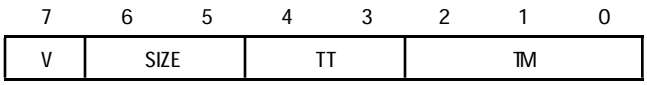
TT—Transfer Type

This field defines the TT1–TT0 signal encodings for the faulted transfer.

TM—Transfer Modifier

This field defines the TM2–TM0 signal encodings for the faulted transfer.

8.4.6.3 WRITE-BACK STATUS. These fields contain status information for the three possible write-backs that could be pending after the faulted access (see Figure 8-8). For a data cache line-push fault or a MOVE16 write fault, WB1S is zero (invalid).



TM—Transfer Modifier
TT—Transfer Type
SIZE—Transfer Size
V—Valid Write (write-back pending if set)

Figure 8-8. Write-Back Status Format

8.4.6.4 FAULT ADDRESS. The fault address (FA) is the initial address for the access that faulted. The FA is a physical address only for cache pushes and a logical address for all other cases. For a misaligned access that 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.

8.4.6.5 WRITE-BACK ADDRESS AND WRITE-BACK DATA. Write-back addresses (WB3A, WB2A, and WB1A) are memory pointers that indicate where to place the write-

back data (WB3D, WB2D, and WB1D). WB3A and WB3D correspond to the temporary holding register in the integer unit (WB3). WB2A and WB2D correspond to the temporary holding register in the data memory unit (WB2) prior to address translation. WB1A and WB1D correspond to the temporary holding register in the bus controller (WB1), which determines the external address and data bus bit patterns. Refer to **Section 2 Integer Unit** for details on the operation of the integer unit pipeline.

The write-back data in WB3D and WB2D is register aligned with byte and word data contained in the least significant byte and word, respectively, of the field. Write-back 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 8-5 lists the data alignment for each combination of data format and A1 and A0.

Table 8-5. Write-Back Data Alignment

	Address		Data Alignment	
Data Format	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 Word	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 in 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 write-back address.

8.4.6.6 PUSH DATA. The push data field contains an image of the cache line that needs to be pushed to memory.

8.4.6.7 ACCESS ERROR STACK FRAME RETURN FROM EXCEPTION. For the access error stack frame (format $7), the processor restores the SR and PC values from the stack and checks the four continuation status bits in the SSW on the stack. If these bits are not set, the processor increments the active supervisor stack pointer by 30 words and resumes normal instruction execution. If the MOVEM continuation bit is set, the processor restores the calculated effective address from the stack frame, increments the active supervisor stack pointer by 30 words, and restarts the MOVEM instruction at a point after the effective address 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, unimplemented floating-point instruction, or floating-point post-instruction exception, the processor restores the calculated effective address from the stack frame, increments the active supervisor stack pointer by 30 words, 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 the access error exception handler sets multiple bits, 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 error stack frame below the frame that it was attempting to use. The illegal stack frame remains intact, so that the exception handler can examine or repair the illegal frame. In a multiprocessor system, the illegal frame can be left so that, when appropriate, another processor of a different type can use it.

The bus error exception handler can identify bus error exceptions due to instruction faults by examining the TM field in the SSW of the access error stack frame. For user and supervisor instruction faults, the TM field contains $2 and $6, respectively (see Figure 8-7). Since the processor allows all pending accesses to complete before reporting an instruction fault, the stack frame for an instruction fault will not contain any pending write-backs. The ATC bit of the SSW is used to distinguish between ATC faults and physical bus errors, and the FA field contains the logical address of the instruction prefetch. For ATC faults, the exception handler can execute a PTEST instruction (using the FA and TM fields from the SSW) to determine the specific cause of the address translation failure. After the handler corrects the cause of the fault, it executes an RTE instruction to restart execution of the instruction that contained the faulted prefetch.

For an address error fault, the processor saves a format $2 exception stack frame on the stack. This stack frame contains the PC pointing to the instruction that caused the address error as well as the actual address referenced by the instruction. Note that bit 0 of the referenced address is cleared on the stack frame. Address error faults must be repaired in software.

For a fault due to a data ATC fault or bus error, pending write-backs are also saved on the access error stack frame and must be completed by the exception handler. For the faulted access, the fault address in the FA field combined with the transfer attribute information from the SSW can be used to identify the cause of the fault. In identifying the fault, the system programmer should be aware that the data memory unit considers the read portion of read-modify-write transfers (for TAS, CAS, CAS2, and some translation table updates) a write. This prevents both read and write accesses from occurring unless all pages touched by the instruction or table update are write enabled.

All accesses other than instruction prefetches go through the data memory unit, and the M68040 treats the instruction and data address spaces as a single merged address space (the exception is the presence of separate transparent translation registers). The function codes for accesses such as PC relative operand addressing and MOVES transfers to function codes $2 and $6 (user and supervisor instruction spaces in the MC68000) are converted to data references to go through the data memory unit, and appear in the TM field of the access error stack frame as data references.

After the fault is corrected, any pending write-backs on the stack frame must be completed. The write-back status fields should be checked for possible write-backs, which the exception handler should complete in the following order: write-back 1, write-back 2, and write-back 3. For a push fault, the push must be completed first, followed by two potential write-backs. Completion of write-back 1 should not generate another access error since this write-back corresponds to the faulted access that has been corrected by the handler. However, write-backs 2 and 3 can cause another bus error exception when the handler attempts to write to memory and should be checked before attempting the write to prevent nesting of exceptions if required by the operating system. The following general bus fault examples indicate the resulting contents of the access error stack frame fields:

1. All Read Access Errors (SSW–RW = $1, TT = $0, TM = $1 or $5)—The FA field contains the logical address of the fault. The WB1S and WB2S fields are zero, and only WB3S can indicate an additional write-back.
2. Cache Push Physical Bus Error (SSW–RW = $0, TT = $0, TM = $0)—The assertion of $\overline{\text{TEA}}$ causes this error when a cache push bus cycle is in progress. The FA field contains the physical address of the fault, and the WB1S field is ignored. All four long words of the data for a push are contained in LW3–LW0 regardless of the size of the transfer. The size of the transfer is indicated in the SIZE field of the SSW and can be either a line or long word. If a line is indicated, all four long words need to be pushed out. If a long word is indicated, all four long words can be written out, or bits 3 and 2 of the FA field can be evaluated to indicate which long words need to be written out to memory ($3, $2, $1, and $0 indicate LW3, LW2, LW1, and LW0, respectively). The WB2S and WB3S fields indicate up to two additional write-backs. If WB2S is valid and if it indicates a MOVE16 instruction, no data should be written out for that write-back slot.
3. Normal Write Physical Bus Error (SSW–RW = $0, TT = $0, TM = $1 or $5)—The assertion of $\overline{\text{TEA}}$ causes this error when a normal write bus cycle is in progress. The FA field contains the logical address of the fault, and the WB1S field indicates that it is valid. The FA and WB1A are equivalent. The WB2S and WB3S fields indicate up to two additional write-backs.
4. MOVE16 Write Physical Bus Error (SSW–RW = $0, TT = $1)—The assertion of $\overline{\text{TEA}}$ causes this error during the write portion of a MOVE16 instruction. The FA field contains the logical address of the fault, and the WB1S field indicates that it is valid. All four long words are contained in LW3–LW0 and must be written out before using FA. Software must ensure that address bits 1 and 0 are both clear if regular move instruction are to be used to write out to the destination.
5. Page Fault (SSW–RW = $0, WB1S–V = $0)—The FA field contains the physical address of the faulted instruction, WB1S = 0, and WB2S indicates that it is valid. Only WB3S can indicate an additional write-back. If WB2S indicates a MOVE16 instruction and if the MOVE16 instruction is used to read from a peripheral that cannot tolerate double reads, then software must write the data contained in PD3–PD0 out to memory and increment the stacked PC to take it beyond the MOVE16 instruction that caused the page fault. Otherwise, if the MOVE16 instruction is allowed to be restarted, another read from the peripheral would occur. If double reads can be tolerated, simply do no write-backs and allow instruction to restart. This is the only case in which the action to be taken depends on whether or not a double read can be tolerated.

Table 8-6 lists the possible combinations of write-backs and the proper way to handle them. The SSW_RW column indicates a read or write cycle; the SSW_PUSH column indicates whether the fault is for a push (TT = 00 and TM = 000). The WB1S, WB2S, and WB3S columns list the respective field's V-bit and indicate a MOVE16 transfer type (TT = 01). The easy cleanup data written column lists the stack's field to be written out to memory if the user is not concerned with retouching peripherals. The hard cleanup action column lists the action to be taken if the peripherals cannot be retouched by MOVE16 (if different from easy cleanup). Note that if a push access error is reported and the size is long word, all four long words, PD0–PD3, are still valid for the line. The exception handler can either write PD0–PD3 using the fault address with bits 3–0 cleared or write the PD corresponding to bits 3–2 of the address (e.g., address $0000000C corresponds to PD3). Note that a MOVE16 is never reported in the WB3S. The SIZE field of WB3S is never a line.

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, or floating-point post-instruction exception, the RTE adjusts the stack to match the pending exception and immediately begins exception processing, without requiring the exception to reoccur.

Table 8-6. Access Error Stack Frame Combinations

Main Case	SSW_RW	SSW_PUSH	WB1S		WB2S		WB3S	Easy Cleanup Data Written	Hard Cleanup Action
			1V	1M16	2V	2M16	3V		
All Read Access Errors	1[a]	No	0	X	0	X	0	None	(Note b)
	1[a]	No	0	X	0	X	1	WB3D	
	All other read cases are not possible.								
Cache Push Physical Bus Error[c]	0	Yes	0	X	0	X	0	PD3–0	(Note b)
	0	Yes	0	X	0	X	1	PD3–0, WB3D	
	0	Yes	0	X	1	0	0	PD3–0, WB2D	
	0	Yes	0	X	1	0	1	PD3–0, WB2D, WB3D	
	0	Yes	0	X	1	1	0	PD3–0, ~WB2D[d]	
Normal Write Physical bus Error	0	No	1	0	0	X	0	WB1D	(Note b)
	0	No	1	0	0	X	1	WB1D, WB3D	
	0	No	1	0	1	0	0	WB1D, WB2D	
	0	No	1	0	1	0	1	WB1D, WB2D, WB3D	
	0	No	1	0	1	1	0	WB1D, ~WB2D[d]	
MOVE16 Write Physical Bus Error	0	No	1	1	0	X	1	PD3–0, WB3D	(Note b)
	0	No	1	1	0	X	0	PD3–0	
	0	No	1	1	1	0	0	PD3–0, WB2D	
	0	No	1	1	1	0	1	PD3–0, WB2D, WB3D	
	0	No	1	1	1	1	0	PD3–0, ~WB2D[d]	
Write Page Fault	0	No	0	X	1	0	0	WB2D	Write PD3–0 and skip[e].
	0	No	0	X	1	0	1	WB2D, WB3D	
	0	No	0	X	1	1	0	~WB2D[d]	
Impossible Write Cases	0	Yes	1	X	X	X	X	(Note f)	—
	0	Don't Care	X	X	X	1	1	(Note g)	

NOTES:

a. The data memory unit stage is tied up until the bus controller passes the read back through the data memory unit and to the execution stage in the integer unit. Therefore, no pending write is possible in WB1 or WB2. WB3 could hold a pending write that was deferred due to operand read or was generated after the read.

b. If any kind of access error is reported and if a MOVE16 write is pending in the WB2 stage, then that MOVE16 read must hit in the cache so the MOVE16 can be safely restarted since it has not caused bus cycles that could retouch peripherals.

c. A cache push physical bus error is normally considered a fatal error. For these cases, the FA field is a physical address, not a logical address as in the other cases.

d. Indicates that the data should not be written even though the V-bit for it is set (WB2 corresponds to a MOVE16 write).

e. The exception handler must alter the stacked PC to point past the MOVE16 and predecrement and postincrement address registers.

f. 1V must be 0 for push exceptions.

g. The execution stage does not post a write until the MOVE16 is in the integer unit.

SECTION 9
FLOATING-POINT UNIT (MC68040 ONLY)

NOTE

This section does not apply to the MC68040V, MC68LC040, MC68EC040, or MC68EC040V. Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for details.

Floating-point math refers to numeric calculations with a variable decimal point location. It is distinguished from integer math, which deals only with whole numbers and fixed decimal point locations. Historically, general-purpose microprocessors have had to depend on add-on coprocessors and accelerators such as the MC68881/MC68882 for fast floating-point capabilities. The MC68040 features a built-in floating-point unit (FPU). Consolidating this important function on chip speeds up the overall processing and eliminates some interfacing overhead required for external accelerators. The MC68040 FPU operates in parallel with the integer unit (IU). The FPU does the numeric calculation while the IU moves on to other tasks. Like the IU, the FPU has its own three-stage pipeline overlapping operations such as integer to floating-point conversion, instruction execution, and write-back. When used with the M68040FPSP, the MC68040 FPU is fully compliant with IEEE floating-point standards.

9.1 FLOATING-POINT UNIT PIPELINE

Integer data from memory (memory to register) requires a pass through the FPU pipeline, converting the data to the extended-precision format for the FPU to use. The result of this conversion is presented to the conversion stage of the FPU pipeline where the desired operation begins, starting a second pass through the pipeline. The IU is then released to execute other instructions once the data has been transferred to the FPU.

Floating-point data to memory (register to memory) requires a complete pass through the FPU pipeline, converting the data from the extended-precision format to an integer data format. Register-to-memory instructions are normally handled entirely by the conversion stage of the pipeline where the data move to memory operation completes. The IU is not released until it has received the converted data (during the last conversion unit cycle).

Like the IU, the FPU has been optimized for the most frequently used instructions and data types to provide the highest possible performance. To boost performance further, the FMOVE instruction concurrently executes with arithmetic calculations and executes completely transparent to the user. Instructions can execute nonsequentially as long as there are no register dependencies. Refer to **Section 10 Instruction Timings** for details on floating-point timings.

The MC68040 FPU is compatible with the MC68881/MC68882. The MC68040 performs basic math functions such as floating-point addition and multiplication directly on dedicated circuitry and performs transcendental functions such as sine and cosine calculations by means of software routines. Motorola offers the M68040FPSP, a software package providing these routines. The software functions are compatible with the MC68881/MC68882, refer to **Appendix E Floating-Point Emulation (M68040FPSP)**.

9.2 FLOATING-POINT USER PROGRAMMING MODEL

Figure 9-1 illustrates the floating-point portion of the user programming model. The following paragraphs describe the FPU portion of the user programming model for the MC68040. The model, which is identical to the programming model for the MC68881/MC68882 floating-point coprocessors, consists of the following registers:

- Eight 80-Bit Floating-Point Data Registers (FP7–FP0)
- 16-Bit Floating-Point Control Register (FPCR)
- 32-Bit Floating-Point Status Register (FPSR)
- 32-Bit Floating-Point Instruction Address Register (FPIAR)

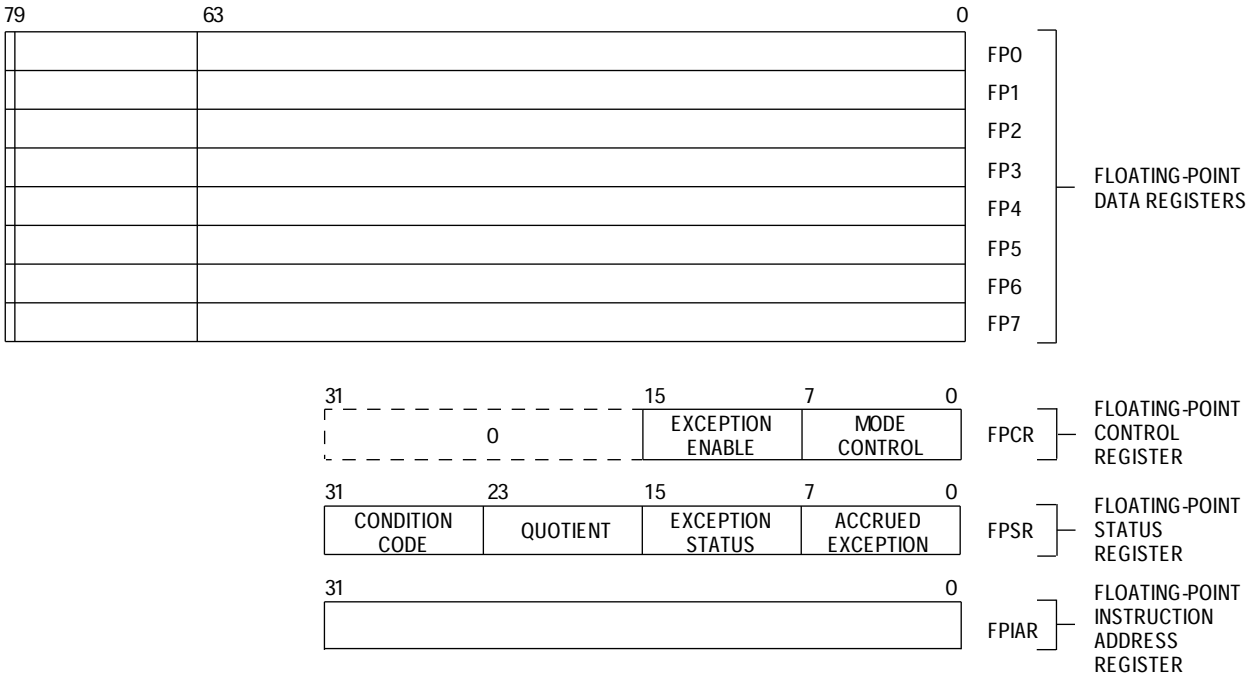


Figure 9-1. Floating-Point User Programming Model

9.2.1 Floating-Point Data Registers (FP7–FP0)

The floating-point data registers are analogous to the integer data registers of the M68000 family. The floating-point data registers always contain extended-precision numbers. All external operands, regardless of the data format, are converted to extended-precision values before being used in any calculation or stored in a floating-point data register. A

reset or a restore operation of the null state sets FP7–FP0 to positive, nonsignaling not-a-numbers (NANs).

9.2.2 Floating-Point Control Register (FPCR)

The FPCR (see Figure 9-2) contains an exception enable (ENABLE) byte that enables or disables traps for each class of floating-point exceptions and a mode control (MODE) byte that sets the user-selectable modes. The user can read or write to the FPCR. Motorola reserves bits 31–16 for future definition; these bits are always read as zero and are ignored during write operations. The reset function or a restore operation of the null state clears the FPCR. When cleared, this register provides the IEEE 754 standard defaults.

9.2.2.1 EXCEPTION ENABLE BYTE. Each bit of the ENABLE byte (see Figure 9-2) corresponds to a floating-point exception class. The user can separately enable traps for each class of floating-point exceptions.

9.2.2.2 MODE CONTROL BYTE. The MODE byte (see Figure 9-2) controls the user-selectable rounding modes and precisions. Zeros in this byte select the IEEE 754 standard defaults. The rounding mode (RND) specifies how inexact results are rounded, and the rounding precision (PREC) selects the boundary for rounding the mantissa.

The processor supports four rounding modes specified by the IEEE 754 standard. These modes are: round to nearest (RN), round toward zero (RZ), round toward plus infinity (RP), and round toward minus infinity (RM). The RP and RM modes are directed rounding modes that are useful in interval arithmetic. Rounding is accomplished through the intermediate result. Single-precision results are rounded to a 24-bit boundary; double-precision results are rounded to a 53-bit boundary; and extended-precision results are rounded to a 64-bit boundary. Table 9-1 lists the encodings for the FPCR.

Table 9-1. Floating-Point Control Register Encodings

Rounding Mode (RND Field)	Encoding		Rounding Precision (PREC Field)
To Nearest (RN)	0	0	Extend (X)
Toward Zero (RZ)	0	1	Single (S)
Toward Minus Infinity (RM)	1	0	Double (D)
Toward Plus Infinity (RP)	1	1	Undefined

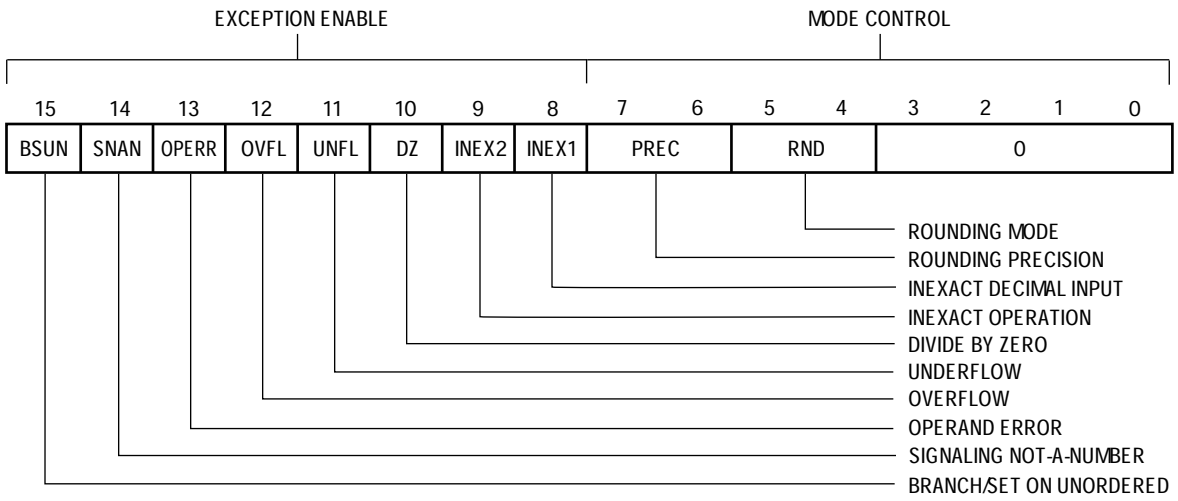


Figure 9-2. Floating-Point Control Register

9.2.3 Floating-Point Status Register (FPSR)

The FPSR (see Figure 9-1) contains a floating-point condition code (FPCC) byte, a quotient byte, a floating-point exception status byte (EXC), and a floating-point accrued exception byte (AEXC). The user can read or write to all bits in the FPSR. Execution of most floating-point instructions modifies this register. The reset function or a restore operation of the null state clears the FPSR. Floating-point conditional operations are not guaranteed if the FPSR is written directly, because the FPSR is only valid as a result of a floating-point instruction.

9.2.3.1 FLOATING-POINT CONDITION CODE BYTE. The FPCC byte (see Figure 9-3) contains four condition code bits that are set at the end of all arithmetic instructions involving the floating-point data registers. These bits are sign of mantissa (N), zero (Z), infinity (I), and NAN. The FMOVE FPm,<ea>, FMOVEM FPm, and FMOVE FPCR instructions do not affect the FPCC.

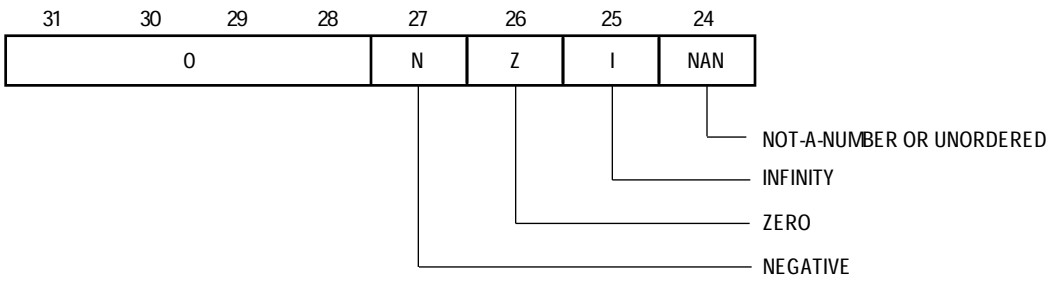


Figure 9-3. FPSR Condition Code Byte

To aid programmers of floating-point subroutine libraries, the MC68040 implements the four FPCC bits in hardware instead of only implementing the four IEEE conditions. An instruction derives the IEEE conditions when needed. For example, the programmers of a complex arithmetic multiply subroutine usually prefer to handle special data types such as

zeros, infinities, or NANs separately from normal data types. The floating-point condition codes allow users to efficiently detect and handle these special values.

9.2.3.2 QUOTIENT BYTE. The quotient byte (see Figure 9-4) provides compatibility with the MC68881/MC68882 FPU. This byte contains the seven least significant bits of the unsigned quotient as well as the sign of the entire quotient.

The quotient bits can be used in argument reduction for transcendentals and other functions. For example, seven bits are more than enough to determine the quadrant of a circle in which an operand resides. The quotient field (bits 22–16) remains set until the user clears it.

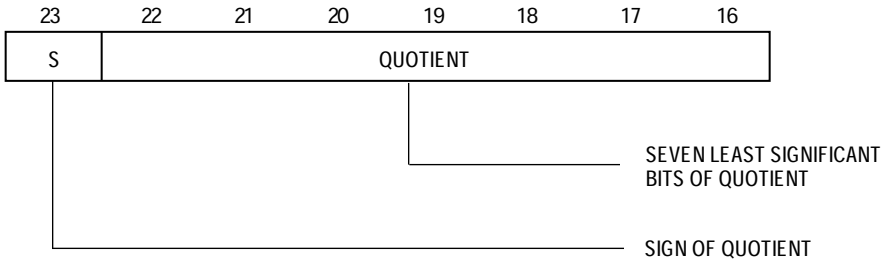


Figure 9-4. FPSR Quotient Byte

9.2.3.3 EXCEPTION STATUS BYTE. The EXC byte (see Figure 9-5) contains a bit for each floating-point exception that can occur during the most recent arithmetic instruction or move operation. The start of most operations clears this byte; however, operations that cannot generate floating-point exceptions do not clear this byte. An exception handler can use this byte to determine which floating-point exception(s) caused a trap.

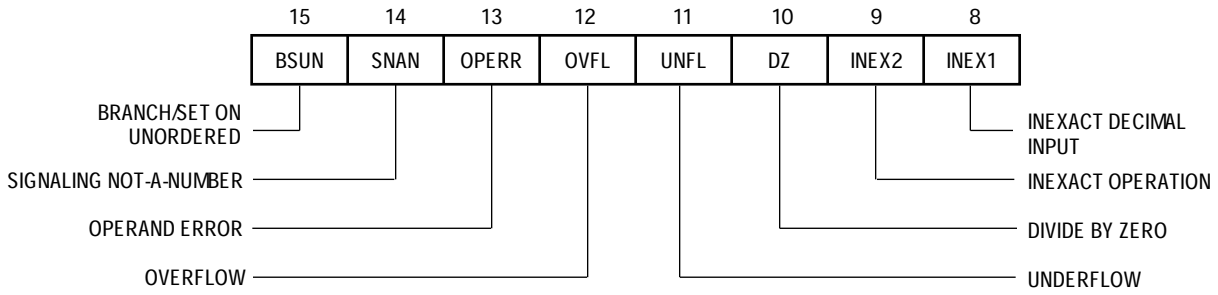


Figure 9-5. FPSR Exception Status Byte

9.2.3.4 ACCRUED EXCEPTION (AEXC) BYTE. The AEXC byte contains five exception bits (see Figure 9-6) that the IEEE 754 standard requires for exception disabled operations. These exceptions are logical combinations of the bits in the EXC byte. The AEXC byte contains the history of all floating-point exceptions that have occurred since the user last cleared the AEXC byte. In normal operations, only the user clears this byte by writing to the FPSR; however, a reset or a restore operation of the null state can also clear the AEXC byte.

Many users elect to disable traps for all or part of the floating-point exception classes. The AEXC byte makes it unnecessary to poll the EXC byte after each floating-point instruction. At the end of most operations (FMOVEM and FMOVE excluded), the bits in the EXC byte are logically combined to form an AEXC value that is logically ORed into the existing AEXC byte. This operation creates sticky floating-point exception bits in the AEXC byte that the user needs to poll only once (i.e., at the end of a series of floating-point operations). A sticky bit is one that remains set until the user clears it.

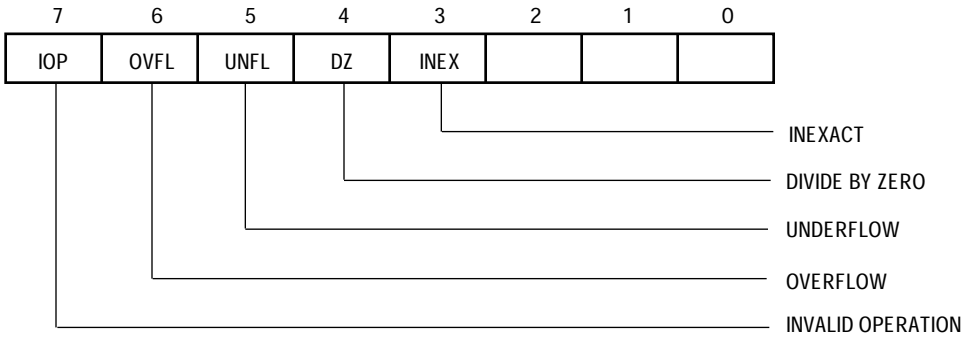


Figure 9-6. FPSR Accrued Exception Byte

Setting or clearing the AEXC bits neither causes nor prevents an exception. The following equations show the comparative relationship between the EXC byte and AEXC byte. Comparing the current value in the AEXC bit with a combination of bits in the EXC byte derives a new value in the corresponding AEXC bit. These equations apply to setting the AEXC bits at the end of each operation affecting the AEXC byte:

New AEXC Bit	= Old AEXC Bit	V	EXC Bits
IOP	= IOP	V	(SNAN V OPERR)
OVFL	= OVFL	V	(OVFL)
UNFL	= UNFL	V	(UNFL Λ INEX2)
DZ	= DZ	V	(DZ)
INEX	= INEX	V	(INEX1 V INEX2 V OVFL)

9.2.4 Floating-Point Instruction Address Register (FPIAR)

For the subset of the floating-point instructions that generate exception traps, the FPU loads the 32-bit FPIAR with the logical address of the instruction before executing the instruction. Because the IU can execute instructions while the FPU executes floating-point instructions and, the FPU can concurrently execute two floating-point instructions the PC value stacked by the MC68040 in response to a floating-point exception handler cannot point to the offending instruction. Therefore, a floating-point exception handler uses the address in the FPIAR to locate a floating-point instruction that has caused an exception. Since the FMOVE to/from the FPCR, FPSR, or FPIAR and FMOVEM instructions cannot generate floating-point exceptions, these instructions do not modify the FPIAR. However,

they can be used to read the FPIAR in an exception handler without changing the previous value. A reset or a restore operation of the null state clears the FPIAR.

9.3 FLOATING-POINT DATA FORMATS AND DATA TYPES

The M68000 floating-point model (MC68881, MC68882, MC68040) supports the following data formats: single precision, double precision, extended precision, and packed decimal. The M68000 floating-point model supports the following data types: normalized, zeros, infinities, denormalized numbers, and NANs. The MC68040 supports part of the M68000 floating-point model in hardware. Table 9-2 lists the data formats and data types supported by the MC68040. Tables 9-3 through 9-6 summarize the floating-point data formats and data types details. For further information on the data formats and data types, refer to the M68000UM/AD, *M68000 Family Programmer's Reference Manual*.

Table 9-2. MC68040 FPU Data Formats and Data Types

Number Types	Data Formats						
	Single-Precision Real	Double-Precision Real	Extended-Precision Real	Packed-Decimal Real	Byte Integer	Word Integer	Long-Word Integer
Normalized	*	*	*	†	*	*	*
Zero	*	*	*	†	*	*	*
Infinity	*	*	*	†			
NAN	*	*	*	†			
Denormalized	†	†	†	†			
Unnormalized			†	†			

*Data Format/Type Supported by On-Chip MC68040 FPU Hardware
†Data Format/Type Supported by Software (MC68040FPSP)

Table 9-3. Single-Precision Real Format Summary

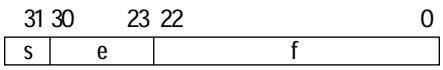
Data Format	
	
Field Size In Bits	
Sign (s)	1
Biased Exponent (e)	8
Fraction (f)	23
Total	32
Interpretation of Sign	
Positive Fraction	s = 0
Negative Fraction	s = 1
Normalized Numbers	
Bias of Biased Exponent	+127 ($7F)
Range of Biased Exponent	0 < e < 255 ($FF)
Range of Fraction	Zero or Nonzero
Fraction	1.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{e-127} \times 1.f$
Denormalized Numbers	
Biased Exponent Format Minimum	0 ($00)
Bias of Biased Exponent	+126 ($7E)
Range of Fraction	Nonzero
Fraction	0.f
Relation to Representation of Real Numbers	$(-1)^s \times 2^{-126} \times 0.f$
Signed Zeros	
Biased Exponent Format Minimum	0 ($00)
Fraction	0.f = 0.0
Signed Infinities	
Biased Exponent Format Maximum	255 ($FF)
Fraction	0.f = 0.0
NANs	
Sign	Don't Care
Biased Exponent Format Maximum	255 ($FF)
Fraction	Nonzero
Representation of Fraction Nonsignaling Signaling Nonzero Bit Pattern Created by User Fraction When Created by FPCP	 1xxxx…xxxx 0xxxx…xxxx xxxxx…xxxx 11111…1111
Approximate Ranges	
Maximum Positive Normalized	3.4×10^{38}
Minimum Positive Normalized	1.2×10^{-38}
Minimum Positive Denormalized	1.4×10^{-45}

Table 9-4. Double-Precision Real Format Summary

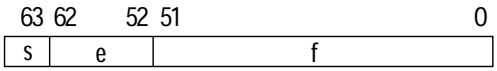
Data Format	
	
Field Size (in Bits)	
Sign (s)	1
Biased Exponent (e)	11
Fraction (f)	52
Total	64
Interpretation of Sign	
Positive Fraction	s = 0
Negative Fraction	s = 1
Normalized Numbers	
Bias of Biased Exponent	+1023 ($3FF)
Range of Biased Exponent	0 < e < 2047 ($7FF)
Range of Fraction	Zero or Nonzero
Fraction	1.f
Relation to Representation of Real Numbers	$(-1)^{s} \times 2^{e-1023} \times 1.f$
Denormalized Numbers	
Biased Exponent Format Minimum	0 ($000)
Bias of Biased Exponent	+1022 ($3FE)
Range of Fraction	Nonzero
Fraction	0.f
Relation to Representation of Real Numbers	$(-1)^{s} \times 2^{-1022} \times 0.f$
Signed Zeros	
Biased Exponent Format Minimum	0 ($00)
Fraction (Mantissa/Significand)	0.f = 0.0
Signed Infinities	
Biased Exponent Format Maximum	2047 ($7FF)
Fraction	0.f = 0.0
NANs	
Sign	0 or 1
Biased Exponent Format Maximum	255 ($7FF)
Fraction	Nonzero
Representation of Fraction Nonsignaling Signaling Nonzero Bit Pattern Created by User Fraction When Created by FPCP	 1xxxx…xxxx 0xxxx…xxxx xxxxx…xxxx 11111…1111
Approximate Ranges	
Maximum Positive Normalized	1.8×10^{308}
Minimum Positive Normalized	2.2×10^{-308}
Minimum Positive Denormalized	4.9×10^{-324}

Table 9-5. Extended-Precision Real Format Summary

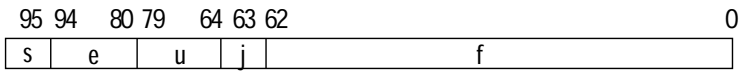
Data Format	
	
Field Size (in Bits)	
Sign (s)	1
Biased Exponent (e)	15
Zero, Reserved (u)	16
Explicit Integer Bit (j)	1
Mantissa (f)	63
Total	96
Interpretation of Unused Bits	
Input	Don't Care
Output	All Zeros
Interpretation of Sign	
Positive Mantissa	s = 0
Negative Mantissa	s = 1
Normalized Numbers	
Bias of Biased Exponent	+16383 ($3FFF)
Range of Biased Exponent	0 < = e < 32767 ($7FFF)
Explicit Integer Bit	1
Range of Mantissa	Zero or Nonzero
Mantissa (Explicit Integer Bit and Fraction)	1.f
Relation to Representation of Real Numbers	$(-1)^{s} \times 2^{e-16383} \times 1.f$
Denormalized Numbers	
Biased Exponent Format Minimum	0 ($0000)
Bias of Biased Exponent	+16383 ($3FFF)
Explicit Integer Bit	0
Range of Mantissa	Nonzero
Mantissa (Explicit Integer Bit and Fraction)	0.f
Relation to Representation of Real Numbers	$(-1)^{s} \times 2^{-16383} \times 0.f$
Signed Zeros	
Biased Exponent Format Minimum	0 ($0000)
Mantissa (Explicit Integer Bit and Fraction)	0.0
Signed Infinities	
Biased Exponent Format Maximum	32767 ($7FFF)
Explicit Integer Bit	Don't Care
Mantissa (Explicit Integer Bit and Fraction)	x.000…0000

Table 9-5. Extended-Precision Real Format Summary (Continued)

NANs	
Sign	Don't Care
Explicit Integer Bit	Don't Care
Biased Exponent Format Maximum	32767 ($7FFF)
Mantissa	Nonzero
Representation of Mantissa Nonsignaling Signaling Nonzero Bit Pattern Created by User Mantissa When Created by FPCP	 x.1xxxx…xxxx x.0xxxx…xxxx x.xxxxx…xxxx 1.11111…1111
Approximate Ranges	
Maximum Positive Normalized	1.2×10^{4932}
Minimum Positive Normalized	1.7×10^{-4932}
Minimum Positive Denormalized	3.7×10^{-4951}

Table 9-6. Packed Decimal Real Format Summary

Data Type	SM	SE	Y	Y	3-Digit Exponent	1-Digit Integer	16-Digit Fraction
±Infinity	0/1	1	1	1	$FFF	$XXXX	$00…00
±NAN	0/1	1	1	1	$FFF	$XXXX	Nonzero
±SNAN	0/1	1	1	1	$FFF	$XXXX	Nonzero
+Zero	0	0/1	X	X	$000–$999	$XXX0	$00…00
–Zero	1	0/1	X	X	$000–$999	$XXX0	$00…00
+In-Range	0	0/1	X	X	$000–$999	$XXX0–$XXX9	$00…01–$99…99
–In-Range	1	0/1	X	X	$000–$999	$XXX0–$XXX9	$00…01–$99…99

9.4 COMPUTATIONAL ACCURACY

Whenever an attempt is made to represent a real number in a binary format of finite precision, there is a possibility that the number can not be represented exactly. This is commonly referred to as a round-off error. Furthermore, when two inexact numbers are used in a calculation, the error present in each number is reflected, and possibly aggravated, in the result. All FPU calculations use an intermediate result. When the MC68040 performs an operation, the calculation is carried out using extended-precision inputs, and the intermediate result is calculated as if to produce infinite precision. After the calculation is complete, the intermediate result is rounded to the selected precision and stored in the destination.

The FPCR encodings provide emulation for devices that only support single and double precision. The execution speed of all instructions is the same whether using single- or double-precision rounding. When using these two forced rounding precisions, the

MC68040 produces the same results as any other device that conforms to the IEEE 754 standard but does not support extended precision. The results are the same when performing the same operation in extended precision and storing the results in single- or double-precision format.

The FPU performs all floating-point internal operations in extended precision. It supports mixed-mode arithmetic by converting single- and double-precision operands to extended-precision values before performing the specified operation. The FPU converts all memory data formats to extended-precision before using it in a floating-point operation or loading it in a floating-point data register. The FPU also converts extended-precision data formats in a floating-point data register to any data format and either stores it in a memory destination or in an integer data register.

If the external operand is a denormalized number, the number is normalized before an operation is performed. However, an external denormalized number moved into a floating-point data register is stored as a denormalized number.

If an external operand is an unnormalized number, the number is normalized before it is used in an arithmetic operation. If the external operand is an unnormalized zero (i.e., with a mantissa of all zeros), the number is converted to a normalized zero before the specified operation is performed. The regular use of unnormalized inputs not only defeats the purpose of the IEEE 754 standard, but also can produce gross inaccuracies in the results.

9.4.1 Intermediate Result

Figure 9-7 illustrates the intermediate result format. The intermediate result's exponent for some dyadic operations (i.e., multiply and divide) can easily overflow or underflow the 15-bit exponent of the destination floating-point register. To simplify the overflow and underflow detection, intermediate results in the FPU maintain a 16-bit, twos-complement integer exponent. Detection of an overflow or underflow intermediate result always converts the 16-bit exponent into a 15-bit biased exponent before being stored in a floating-point data register. The FPU internally maintains the 67-bit mantissa for rounding purposes. The mantissa is always rounded to 64 bits (or less, depending on the selected rounding precision) before it is stored in a floating-point data register.

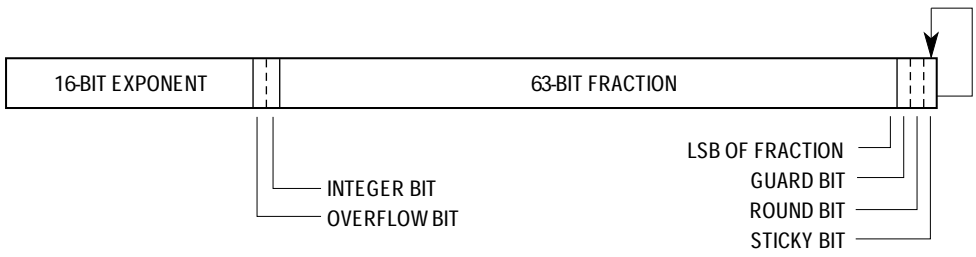


Figure 9-7. Intermediate Result Format

If the destination is a floating-point data register, the result is in the extended-precision format and is rounded to the precision specified by the FPSR PREC bits before being stored. All mantissa bits beyond the selected precision are zero. If the single- or double-

precision mode is selected, the exponent value is in the correct range even if it is stored in extended-precision format. If the destination is a memory location, the FPSR PREC bits are ignored. In this case, a number in the extended-precision format is taken from the source floating-point data register, rounded to the destination format precision, and then written to memory.

Depending on the selected rounding mode or destination data format in effect, the location of the least significant bit of the mantissa and the locations of the guard, round, and sticky bits in the 67-bit intermediate result mantissa varies. The guard and round bits are always calculated exactly. The sticky bit is used to create the illusion of an infinitely wide intermediate result. As the arrow illustrates in Figure 9-7, the sticky bit is the logical OR of all the bits in the infinitely precise result to the right of the round bit. During the calculation stage of an arithmetic operation, any non-zero bits generated that are to the right of the round bit set the sticky bit to one. Because of the sticky bit, the rounded intermediate result for all required IEEE arithmetic operations in the RN mode is in error by no more than one-half unit in the last place.

9.4.2 Rounding The Result

Range control is the process of rounding the mantissa of the intermediate result to the specified precision and checking the 16-bit intermediate exponent to ensure that it is within the representable range of the selected rounding-precision format. Range control ensures correct emulation of a device that only supports single- or double-precision arithmetic. If the intermediate result's exponent exceeds the range of the selected precision, the exponent value appropriate for an underflow or overflow is stored as the result in the 16-bit extended-precision format exponent. For example, if the data format and rounding mode is single-precision RM and the result of an arithmetic operation overflows the magnitude of the single-precision format, the largest normalized single-precision value is stored as an extended-precision number in the destination floating-point data register (i.e., an unbiased 15-bit exponent of $00FF and a mantissa of $FFFFFF0000000000). If an infinity is the appropriate result for an underflow or overflow, the infinity value for the destination data format is stored as the result (i.e., an exponent with the maximum value and a mantissa of zero).

Figure 9-8 illustrates the algorithm that is used to round an intermediate result to the selected rounding precision and destination data format. If the destination is a floating-point data register, either the selected rounding precision specified by the FPCR PREC bits or by the instruction itself determines the rounding boundary. For example, FSADD and FDADD specify single- and double-precision rounding regardless of the precision specified in the FPCR PREC bits. If the destination is external memory or an integer data register, the destination data format determines the rounding boundary. If the rounded result of an operation is not exact, then the INEX2 bit is set in the FPSR EXC byte.

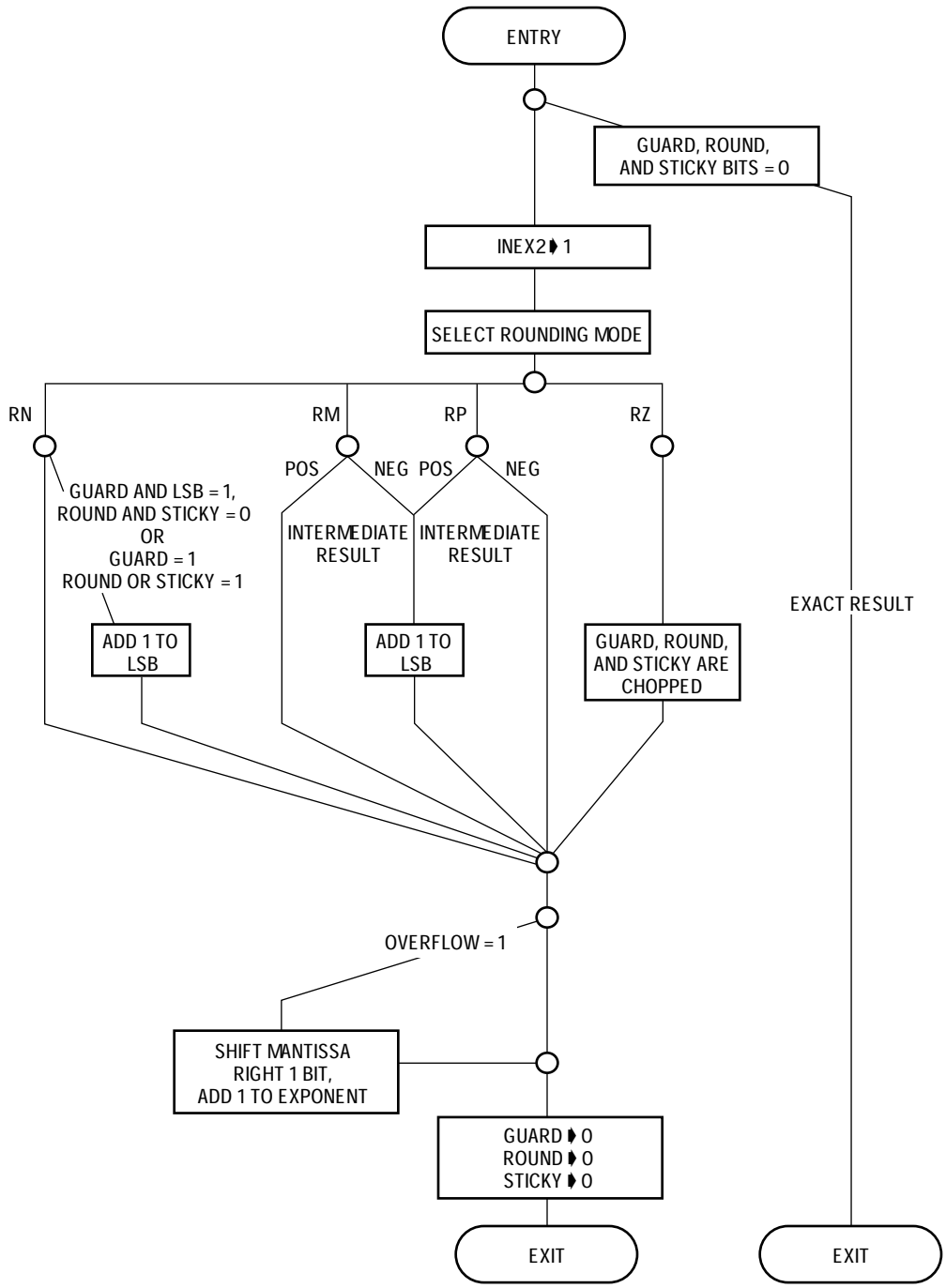


Figure 9-8. Rounding Algorithm Flowchart

The three additional bits beyond the extended-precision format, the difference between the intermediate result's 67-bit mantissa and the stored result's 64-bit mantissa, allow the FPU to perform all calculations as though it were performing calculations using a float engine with infinite bit precision. The result is always correct for the specified destination's data format before performing rounding (unless an overflow or underflow error occurs). The specified rounding operation then produces a number that is as close as possible to the infinitely precise intermediate value and still representable in the selected precision.

The following tie-case example illustrates how the 67-bit mantissa allows the FPU to meet the error bound of the IEEE specification:

Result	Integer	63-Bit Fraction	Guard	Round	Sticky
Intermediate	x	xxx…x00	1	0	0
Rounded-to-Nearest	x	xxx…x00	0	0	0

The least significant bit of the rounded result does not increment even though the guard bit is set in the intermediate result. The IEEE 754 standard specifies that tie cases should be handled in this manner. If the destination data format is extended and there is a difference between the infinitely precise intermediate result and the round-to-nearest result, the relative difference is 2^{-64} (the value of the guard bit). This error is equal to one-half of the least significant bit's value and is the worst case error that can be introduced when using the RN mode. Thus, the term one-half unit in the last place correctly identifies the error bound for this operation. This error specification is the relative error present in the result; the absolute error bound is equal to $2^{exponent} \times 2^{-64}$. The following example illustrates the error bound for the other rounding modes:

Result	Integer	63-Bit Fraction	Guard	Round	Sticky
Intermediate	x	xxx…x00	1	1	1
Rounded-to-Nearest	x	xxx…x00	0	0	0

The difference between the infinitely precise result and the rounded result is $2^{-64} + 2^{-65} + 2^{-66}$, which is slightly less than 2^{-63} (the value of the least significant bit). Thus, the error bound for this operation is not more than one unit in the last place. For all arithmetic operations, the FPU meets these error bounds, providing accurate and repeatable results.

9.5 POSTPROCESSING OPERATION

Most operations end with a postprocessing step. The FPU provides two steps in postprocessing. First, the condition code bits in the FPSR are set or cleared at the end of each arithmetic operation or move operation to a single floating-point data register. The condition code bits are consistently set based on the result of the operation. Second, the FPU supports 32 conditional tests that allow floating-point conditional instructions to test floating-point conditions in exactly the same way as the integer conditional instructions test the integer condition codes. The combination of consistently set condition code bits and the simple programming of conditional instructions gives the MC68040 a very flexible, high-performance method of altering program flow based on floating-point results. While reading the summary for each instruction, it should be assumed that an instruction performs postprocessing unless the summary specifically states that the instruction does not do so. The following paragraphs describe postprocessing in detail.

9.5.1 Underflow, Round, Overflow

During the calculation of an arithmetic result, the FPU arithmetic logic unit (ALU) has more precision and range than the 80-bit extended-precision format. However, the final result of these operations is an extended-precision floating-point value. In some cases, an intermediate result becomes either smaller or larger than can be represented in extended precision. Also, the operation can generate a larger exponent or more bits of precision than can be represented in the chosen rounding precision. For these reasons, every arithmetic instruction ends by rounding the result and checking for overflow and underflow.

At the completion of an arithmetic operation, the intermediate result is checked to see if it is too small to be represented as a normalized number in the selected precision. If so, the UNFL-bit is set in the FPSR EXC byte. The MC68040 then takes a nonmaskable underflow exception and executes the M68040FPSP underflow exception handler, denormalizing the result. Denormalizing a number causes a loss of accuracy, but a zero is not returned unless absolutely necessary. If a number has grossly underflowed, the M68040FPSP returns a zero or the smallest denormalized number with the correct sign, depending on the rounding mode in effect.

If no underflow occurs, the intermediate result is rounded according to the user-selected rounding precision and rounding mode. After rounding, the INEX2-bit of the FPSR EXC byte is set accordingly. Finally, the magnitude of the result is checked to see if it is too large to be represented in the current rounding precision. If so, the OVFL-bit of the FPSR EXC byte is set. The M68040FPSP returns a correctly signed infinity or a correctly signed largest normalized number, depending on the rounding mode in effect.

9.5.2 Conditional Testing

Unlike the integer arithmetic condition codes, an instruction either always sets the floating-point condition codes in the same way or it does not change them at all. Therefore, the instruction descriptions do not include floating-point condition code settings. The following paragraphs describe how floating-point condition codes are set for all instructions that modify condition codes. Refer to **9.2.3.1 Floating-Point Condition Code Byte** for a description of the FPCC byte.

The condition code bits differ slightly from the integer condition codes. Unlike the operation-type-dependent integer condition codes, examining the result at the end of the operation sets or clears the floating-point condition codes accordingly. The M68000 family integer condition codes bits N and Z have this characteristic, but the V and C bits are set differently for different instructions. The data type of the operation's result determines how the four condition code bits are set. Table 9-7 lists the condition code bit setting for each data type. The MC68040 generates only eight of the 16 possible combinations. Loading the FPCC with one of the other combinations and executing a conditional instruction can produce an unexpected branch condition.

Table 9-7. Floating-Point Condition Code Encodings

Data Type	N	Z	I	NAN
+ Normalized or Denormalized	0	0	0	0
– Normalized or Denormalized	1	0	0	0
+ 0	0	1	0	0
– 0	1	1	0	0
+ Infinity	0	0	1	0
– Infinity	1	0	1	0
+ NAN	0	0	0	1
– NAN	1	0	0	1

The inclusion of the NAN data type in the IEEE floating-point number system requires each conditional test to include the NAN condition code bit in its Boolean equation. Because a comparison of a NAN with any other data type is unordered (i.e., it is impossible to determine if a NAN is bigger or smaller than an in-range number), the compare instruction sets the NAN condition code bit when an unordered compare is attempted. All arithmetic instructions also set the FPCC NAN bit if the result of an operation is a NAN. The conditional instructions interpret the NAN condition code bit equal to one as the unordered condition.

The IEEE 754 standard defines four conditions: equal to (EQ), greater than (GT), less than (LT), and unordered (UN). In addition, the standard only requires the generation of the condition codes as a result of a floating-point compare operation. The FPU tests for these conditions and 28 others at the end of any operation affecting the condition codes. For purposes of the floating-point conditional branch, set byte on condition, decrement and branch on condition, and trap on condition instructions, the MC68040 logically combines the four FPCC bits to form 32 conditional tests. The 32 conditional tests are separated into two groups—16 that cause an exception if an unordered condition is present when the conditional test is attempted, IEEE nonaware tests, and 16 that do not cause an exception, IEEE aware tests. The set of IEEE nonaware tests is best used:

- when porting a program from a system that does not support the IEEE 754 standard to a conforming system or
- when generating high-level language code that does not support IEEE floating-point concepts (i.e., the unordered condition).

An unordered condition occurs when one or both of the operands in a floating-point compare operation is a NAN. The inclusion of the unordered condition in floating-point branches destroys the familiar trichotomy relationship (greater than, equal, less than) that exists for integers. For example, the opposite of floating-point branch greater than (FBGT) is not floating-point branch less than or equal (FBLE). Rather, the opposite condition is floating-point branch not greater than (FBNGT). If the result of the previous instruction was unordered, FBNGT is true; whereas, both FBGT and FBLE would be false since unordered fails both of these tests (and sets BSUN). Compiler programmers should be

particularly careful of the lack of trichotomy in the floating-point branches since it is common for compilers to invert the sense of conditions.

When using the IEEE nonaware tests, the user receives a BSUN exception whenever a branch is attempted and the NAN condition code bit is set, unless the branch is an FBEQ or an FBNE. If the BSUN exception is enabled in the FPCR, the exception causes another exception. Therefore, the IEEE nonaware program is interrupted if an unexpected condition occurs. Compilers and programmers who are knowledgeable of the IEEE 754 standard should use the IEEE aware tests in programs that contain ordered and unordered conditions. Since the ordered or unordered attribute is explicitly included in the conditional test, the BSUN bit is not set in the FPSR EXC byte when the unordered condition occurs. Table 9-8 summarizes the conditional mnemonics, definitions, equations, predicates, and whether the BSUN bit is set in the FPSR EXC byte for the 32 floating-point conditional tests. The equation column lists the combination of FPCC bits for each test in the form of an equation.

Table 9-8. Floating-Point Conditional Tests

Mnemonic	Definition	Equation	Predicate	BSUN Bit Set
IEEE Nonaware Tests				
EQ	Equal	Z	000001	No
NE	Not Equal	$\overline{Z}$	001110	No
GT	Greater Than	$\overline{NAN \vee Z \vee N}$	010010	Yes
NGT	Not Greater Than	$NAN \vee Z \vee N$	011101	Yes
GE	Greater Than or Equal	$Z \vee (\overline{NAN \vee N})$	010011	Yes
NGE	Not Greater Than or Equal	$NAN \vee (N \wedge Z)$	011100	Yes
LT	Less Than	$N \wedge (\overline{NAN \vee Z})$	010100	Yes
NLT	Not Less Than	$NAN \vee (Z \vee \overline{N})$	011011	Yes
LE	Less Than or Equal	$Z \vee (N \wedge \overline{NAN})$	010101	Yes
NLE	Not Less Than or Equal	$NAN \vee (\overline{N \vee Z})$	011010	Yes
GL	Greater or Less Than	$\overline{NAN \vee Z}$	010110	Yes
NGL	Not Greater or Less Than	$NAN \vee Z$	011001	Yes
GLE	Greater, Less, or Equal	$\overline{NAN}$	010111	Yes
NGLE	Not Greater, Less, or Equal	NAN	011000	Yes
IEEE Aware Tests				
EQ	Equal	Z	000001	No
NE	Not Equal	$\overline{Z}$	001110	No
OGT	Ordered Greater Than	$\overline{NAN \vee Z \vee N}$	000010	No
ULE	Unordered or Less or Equal	$NAN \vee Z \vee N$	001101	No
OGE	Ordered Greater Than or Equal	$Z \vee (\overline{NAN \vee N})$	000011	No
ULT	Unordered or Less Than	$NAN \vee (N \wedge \overline{Z})$	001100	No
OLT	Ordered Less Than	$N \wedge (\overline{NAN \vee Z})$	000100	No
UGE	Unordered or Greater or Equal	$NAN \vee Z \vee N$	001011	No
OLE	Ordered Less Than or Equal	$Z \vee (N \wedge \overline{NAN})$	000101	No
UGT	Unordered or Greater Than	$NAN \vee (\overline{N \vee Z})$	001010	No
OGL	Ordered Greater or Less Than	$\overline{NAN \vee Z}$	000110	No
UEQ	Unordered or Equal	$NAN \vee Z$	001001	No
OR	Ordered	$\overline{NAN}$	000111	No
UN	Unordered	NAN	001000	No
Miscellaneous Tests				
F	False	False	000000	No
T	True	True	001111	No
SF	Signaling False	False	010000	Yes
ST	Signaling True	True	011111	Yes
SEQ	Signaling Equal	Z	010001	Yes
SNE	Signaling Not Equal	$\overline{Z}$	011110	Yes

NOTE: All condition codes with an overbar indicate cleared bits; all other bits are set.

9.6 FLOATING-POINT EXCEPTIONS

There are two classes of floating-point-related exceptions: nonarithmetic floating-point exceptions and arithmetic floating-point exceptions. The latter relates to the handling of arithmetic exceptions caused by floating-point activity, and the former includes unimplemented floating-point instructions and unsupported data types not related to the handling of arithmetic exceptions. Format error and FTRAPcc exceptions may seem to be floating-point related, but are considered IU exceptions (see **Section 8 Exception Processing**). The following sections detail floating-point exceptions and how the MC68040 and M68040FPSP handle them. Table 9-9 lists the vector numbers related to floating-point exceptions.

Table 9-9. Floating-Point Exception Vectors

Vector Number	Vector Offset (Hex)	Assignment
11	02C	Floating-Point Unimplemented Instruction (also used for F-line instruction)
48	0C0	Floating-Point Branch or Set on Unordered Condition
49	0C4	Floating-Point Inexact Result
50	0C8	Floating-Point Divide by Zero
51	0CC	Floating-Point Underflow
52	0D0	Floating-Point Operand Error
53	0D4	Floating-Point Overflow
54	0D8	Floating-Point SNAN
55	0DC	Floating-Point Unimplemented Data Type

The following paragraphs detail nonarithmetic floating-point exceptions.

9.6.1 Unimplemented Floating-Point Instructions

F-line instructions are instruction word patterns with bits 15–12 that have an $F encoding, causing F-line exceptions. These instructions are termed unimplemented floating-point instructions and cause an unimplemented floating-point exception. The MC68040 recognizes some F-line instructions, such as the FMUL and CPUSH, which do not cause F-line exceptions. There are some F-line instructions that the MC68040 recognizes as valid MC68881/MC68882 floating-point instruction patterns, but as floating-point instructions that the processor cannot complete in hardware. Table 9-10 lists the floating-point instructions that are unimplemented and therefore cause an unimplemented instruction exception.

If the processor encounters an F-line instruction and the instruction patterns do not match either of the above two cases, the processor takes an F-line illegal exception. F-line illegal exceptions are discussed further in **Section 8 Exception Processing**. The processor generates an exception with vector number 11 and pushes a four-word stack frame format $0 on the system stack. An illegal instruction exception is also reported when a breakpoint acknowledge bus cycle is run and terminated with either a transfer acknowledge ($\overline{TA}$) or transfer error acknowledge ($\overline{TEA}$) signal. Since the unimplemented floating-point

exception and the F-line illegal instruction share the same vector, the exception handler uses the stack frame format ($0 or $2) to distinguish between the two.

Table 9-10. Unimplemented Instructions

Monadic Operations	
FACOS	FINTRZ
FASIN	FLOG10
FATAN	FLOGN
FATANH	FLOGNP1
FCOS	FMOVECR
FCOSH	FSIN
FETOX	FSINCOS
FETOXM1	FSINH
FGETEXP	FTAN
FGETMAN	FTANH
FINT	FTENTOX
FTWOTOX	—
Dyadic Operations	
FMOD	FREM
FSCALE	—

When an unimplemented floating-point instruction is encountered, the processor waits for all previous floating-point instructions to complete execution. Pending exceptions are taken and handled prior to the execution of the unimplemented instruction.

Next, the instruction is partially decoded to allow fetching of the memory source operand, if required. When the operand fetch begins, all other read accesses for previous instructions are complete, and only the execution and write-back of results for previous integer instructions remains to be completed. If an access error (bus error) occurs in fetching the operand or in completing any other access before beginning the operand fetch, the unimplemented instruction is restarted after the processor returns from exception handling for the error. Refer to **Section 8 Exception Processing** for more information on access errors.

The fetched source operand is passed to the FPU, which converts the operand to extended precision and saves the intermediate result. If the operand is an unsupported data type (denormalized, unnormalized, or packed decimal real), the unimplemented floating-point exception takes precedence, and the floating-point instruction emulation routine must detect the unsupported data type.

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 format $2 stack frame and saves the vector offset, PC, internal copy of the SR, and calculated

effective address in the stack frame. The saved PC value is the logical 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 processor's exception vector table, pushes the format $2 stack frame on the system stack, and begins execution of the exception handler after prefetching instructions to fill the pipeline. The exception handler emulates the unimplemented floating-point instruction in software, maintaining user-object-code compatibility. Refer to **Section 8 Exception Processing** for details about exception vectors and format $2 stack frames.

The F-line exception handler checks for the format $2 stack frame to distinguish an unimplemented floating-point instruction from other F-line unimplemented instructions. When the exception handler for unimplemented floating-point instructions executes an FSAVE, a 26-word unimplemented instruction state frame is created (see Figure 9-10). At this point, an FSAVE instruction yields the information as listed in Table 9-16. Note that unless the instruction specifies a packed decimal real source, the state frame contains both operands (if required). For packed decimal real data format, the second operand is in the designated format of the destination floating-point data register.

The exception handler uses the information provided in the state frame to determine the instruction that it needs to emulate and the input operands to that instruction. Once the instruction has been emulated and the result is reached, the exception handler moves the result into the appropriate destination floating-point data register, discards the unimplemented instruction state frame, and returns to normal instruction flow using the RTE instruction. The limitation to this approach is that no floating-point arithmetic exceptions can be reported at the end of the emulated instruction.

The M68040FPSP not only emulates the instruction, but in addition, it ensures that if any floating-point arithmetic exceptional conditions arise from the emulation of the unimplemented instruction and if the corresponding floating-point arithmetic exception is enabled, the M68040FPSP manipulates the stack and restores the stack back into the FPU in the desired exceptional state. This effectively imitates the action of the MC68040 implemented instructions since the exception is not reported until the next floating-point instruction is encountered. This manipulation of the stack is rather complicated and is beyond the scope of this manual. Motorola recommends that the user utilize the M68040FPSP if a full exception-reporting model is required. Motorola does not provide any printed documentation other than what is embedded in the source code of the M68040FPSP.

9.6.2 Unsupported Floating-Point Data Types

An unsupported data type exception occurs when either operand to an implemented floating-point instruction is denormalized (for single-, double-, and extended-precision operands), unnormalized (for extended-precision operands), or either the source or destination data format is packed decimal real. These data types are unimplemented in the MC68040 and must be emulated in software.

NOTE

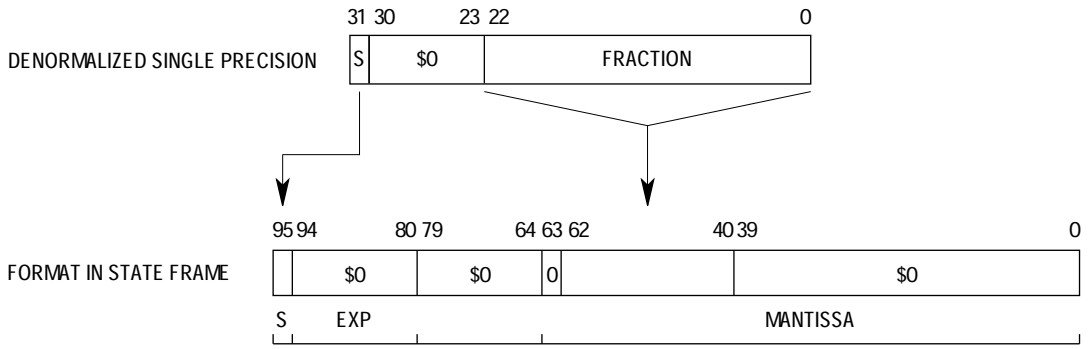
In this manual, all references to the unsupported floating-point data types also refer to the unimplemented data types in the M68040FPSP.

When the processor encounters an unsupported data type, the procedure taken is identical to that used when an unimplemented instruction is taken. Unsupported data types with operands that have opclass 010 or 000 (register-to-register or memory-to-register) instructions cause a pre-instruction exception. When an unsupported data type is detected for opclass 011 (register-to-memory) instructions, a post-instruction exception is generated immediately. A format $0 (for the pre-instruction exception) or format $3 (for the post-instruction exception) stack frame is saved, and vector number 55 is fetched. A denormalized value generated as the result of a floating-point operation generates a nonmaskable underflow exception instead of an unsupported data type exception.

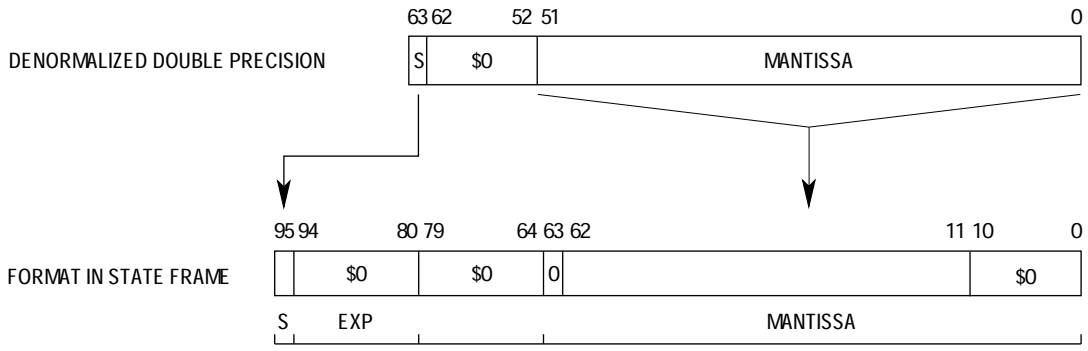
Table 9-16 lists the floating-point state frame fields for unsupported data type exceptions resulting from the execution of opclass 010 or 000 (register-to-register or memory-to-register) instructions, and opclass 011 (register-to-memory) instructions defined for the use by the supervisor exception handler.

A denormalized or unnormalized extended-precision source or destination operand is copied directly without modification to ETEMP or FPTEMP fields in the floating-point state frame. If a packed decimal real source operand is specified, the upper 32 bits of the operand are copied to the FPTEMP field, and the lower 64 bits are copied to ETEMP. The destination operand in this case remains in the destination floating-point register, and can be either denormalized or unnormalized. Figure 9-9 illustrates denormalized single- (a) and double-precision (b) operands stored in ETEMP field.

The exception handler uses the floating-point state frame information to determine which operand (or operands) is the unsupported data type and which instruction attempted to use the offending operand. The exception handler must provide the routines needed to complete the instruction and to store that instruction to the proper destination, whether it be in a floating-point data register, integer data register, or external memory. Once the destination is written, the floating-point state frame is discarded, and normal execution is resumed by using the RTE instruction. This approach does not report floating-point arithmetic exceptions that may have been generated. Motorola recommends that the user utilize the M68040FPSP if a full exception-reporting model is required. Motorola does not provide any printed documentation other than what is embedded in the source code of the M68040FPSP.



(a) Single Precision



(b) Double Precision

Figure 9-9. Format of Denormalized Operand in State Frame

9.7 FLOATING-POINT ARITHMETIC EXCEPTIONS

The following eight user floating-point arithmetic exceptions are listed in order of priority. The MC68040 generates the first seven exceptions in hardware and the eighth only in software.

- Branch/Set on Unordered (BSUN)
- Signaling Not-A-Number (SNAN)
- Operand Error (OPERR)
- Overflow (OVFL)
- Underflow (UNFL)
- Divide by Zero (DZ)
- Inexact 2 (INEX2)
- Inexact 1 (INEX1)

INEX1 exception is the condition that exists when a packed decimal operand cannot be converted exactly to the extended-precision format in the current rounding mode. Since

the MC68040 does not directly support packed decimal real operands, the processor never sets INEX1 bit in the FPSR EXC byte, but provides it as a latch so that emulation software can report the exception.

A floating-point arithmetic exception is taken in one of two situations. The first situation occurs when the user program enables an arithmetic exception by setting a bit in the FPCR ENABLE byte and the corresponding bit in the FPSR EXC byte matches the bit in the FPCR ENABLE byte as a result of program execution; this is referred to as maskable exception conditions. A user write operation to the FPSR, which sets a bit in the EXC byte, does not cause an exception to be taken, regardless of the value in the ENABLE byte. When a user writes to the ENABLE byte that enables a class of floating-point exceptions, a previously generated floating-point exception does not cause an exception to be taken, regardless of the value in the FPSR EXC byte. The user can clear a bit in the FPCR ENABLE byte, disabling each corresponding exception.

The second situation occurs when the processor encounters a nonmaskable SNAN, OPERR, OVFL, and UNFL condition; this is referred to as nonmaskable exception conditions. This allows a supervisor exception handler to correct a defaulting result generated by the MC68040 that is different from the result generated by an MC68881/MC68882 executing the same code. After correcting the result, the supervisor exception handler calls a user-defined exception handler if the exception has been enabled in the FPCR ENABLE byte or returns to the main program flow if the exception is disabled.

A single instruction execution can generate dual and triple exceptions. When multiple exceptions occur with exceptions enabled for more than one exception class, the highest priority exception is reported; the lower priority exceptions are never reported or taken. The previous list of arithmetic floating-point exceptions is in order of priority. The bits of the ENABLE byte are organized in decreasing priority, with bit 15 being the highest and bit 8 the lowest. The exception handler must check for multiple exceptions. The address of the exception handler is derived from the vector number corresponding to the exception. The following is a list of multiple instruction exceptions that can occur:

- SNAN and INEX1
- OPERR and INEX2
- OPERR and INEX1
- OVFL and INEX2 and/or INEX1
- UNFL and INEX2 and/or INEX1

9.7.1 Branch/Set On Unordered (BSUN)

The BSUN exception is the result of performing an IEEE nonaware conditional test associated with the FBcc, FDBcc, FTRAPcc, and FScc instructions when an unordered condition is present. Refer to **9.5.2 Conditional Testing** for information on conditional tests.

If a floating-point exception is pending from a previous floating-point instruction, a pre-instruction exception is taken. After the appropriate exception handler is executed, the conditional instruction is restarted. When the FPU pipeline is idle (all previous floating-point instructions have completed) and no exceptions are pending, the processor evaluates the conditional predicate and checks for a BSUN exception before executing the conditional instruction.

9.7.1.1 MASKABLE EXCEPTION CONDITIONS. A BSUN exception occurs if the conditional predicate is one of the IEEE nonaware branches and the FPCC NAN bit is set. When the processor detects this condition, it sets the BSUN bit in the FPSR EXC byte.

a. If the user BSUN exception handler is disabled, the floating-point condition is evaluated as if it were the equivalent IEEE aware conditional predicate. No exceptions are taken.
b. If the user BSUN exception handler is enabled, the processor takes a floating-point pre-instruction exception. A $0 stack frame is saved, and vector number 48 is generated to access the BSUN exception vector. The BSUN entry in the processor's vector table points to the M68040FPSP BSUN exception handler.

For MC68881/MC68882 compatibility, the M68040FPSP updates the FPIAR by copying the PC value in the pre-instruction stack frame to the FPIAR. The M68040FPSP BSUN exception handler restores the FPU to its exceptional state, cleans up the stack to the state prior to the M68040FPSP BSUN exception handler's execution, and continues instruction execution at the user BSUN exception handler. No parameters are passed to the user BSUN exception handler since the M68040FPSP BSUN exception handler provides the illusion that it never existed.

The user BSUN exception handler must execute an FSAVE as its first floating-point instruction. FSAVE allows other floating-point instructions to execute without reporting the BSUN exception again, although none of the state frame values are useful in the execution of the user BSUN exception handler. The BSUN exception is unique in that the exception is taken before the conditional predicate is evaluated. If the user BSUN exception handler does not set the PC to the instruction following the one that caused BSUN exception when returning, the exception is executed again. Therefore, it is the responsibility of the user BSUN exception handler to prevent the conditional instruction from taking the BSUN exception again. There are four ways to prevent taking the exception again:

1. Incrementing the stored PC in the stack bypasses the conditional instruction. This technique applies to situations where a fall-through is desired. Note that accurate calculation of the PC increment requires detailed knowledge of the size of the conditional instruction being bypassed.
2. Clearing the NAN bit prevents the exception from being taken again. However, this alone cannot deterministically control the result's indication (true or false) that would be returned when the conditional instruction reexecutes.
3. Disabling the BSUN bit also prevents the exception from being taken again. Like the second method, this method cannot control the result indication (true or false) that would be returned when the conditional instruction reexecutes.

4. Examining the conditional predicate and setting the FPCC NAN bit accordingly prevents the exception from being taken again. This technique gives the most control since it is possible to pre-determine the direction of program flow. Bit 7 of the F-line operation word indicates where the conditional predicate is located. If bit 7 is set, the conditional predicate is the lower six bits of the F-line operation word. Otherwise, the conditional predicate is the lower six bits of the instruction word, which immediately follows the F-line operation word. Using the conditional predicate and the table for IEEE nonaware test in **9.5.2 Conditional Testing**, the condition codes can be set to return a known result indication when the conditional instruction is reexecuted.

Prior to exiting the user BSUN exception handler, the exception handler discards the floating-point state frame.

9.7.1.2 NONMASKABLE EXCEPTION CONDITIONS. There are no conditions.

9.7.2 Signaling Not-a-Number (SNAN)

An SNAN is used as an escape mechanism for a user-defined, non-IEEE data type. The processor never creates an SNAN as a result of an operation; a NAN created by an operand error exception is always a nonsignaling NAN. When an operand is an SNAN involved in an arithmetic instruction, the SNAN bit is set in the FPSR EXC byte. Since the FMOVEM, FMOVE FPCR, and FSAVE instructions do not modify the status bits, they cannot generate exceptions. Therefore, these instructions are useful for manipulating SNANs.

9.7.2.1 MASKABLE EXCEPTION CONDITIONS. When an SNAN is encountered, if the destination is a floating-point data register or is in memory (or an integer data register) and the format is single, double, or extended precision, the SNAN is maskable and may or may not take an exception.

a. If the user SNAN exception is disabled, the processor clears the SNAN bit in the NAN data format and the resulting nonsignaling NAN is transferred to the destination. No bits other than the SNAN bit of the NAN data format are modified, although the input NAN is truncated if necessary. Instruction execution continues without taking any exceptions.
b. If the user SNAN exception handler is enabled, the processor posts an exception and another floating-point instruction is eventually encountered; a pre-instruction exception is reported at that time. The SNAN entry in the processor's vector table points to the M68040FPSP SNAN exception handler. Once the M68040FPSP SNAN exception handler recognizes the operand error as a maskable condition, it does not modify the destination or pass control to the user SNAN exception handler.

9.7.2.2 NONMASKABLE EXCEPTION CONDITIONS. When an SNAN is encountered, if the destination is either in memory or an integer data register and the format is byte, word, or long word, a nonmaskable post-instruction exception occurs and is taken immediately. The SNAN entry in the processor's vector table points to the M68040FPSP SNAN exception handler.

The M68040FPSP SNAN exception handler checks to see if the instruction is an FMOVE to byte, word, or long word. If one of these conditions is met, the M68040FPSP SNAN exception handler stores the most significant 8, 16, or 32 bits, respectively, of the SNAN mantissa, with the SNAN bit set, to the destination. Next, it determines whether or not the user SNAN exception is enabled.

a. If the user SNAN exception is disabled, the M68040FPSP SNAN exception handler checks for an INEX1 or INEX2 exception condition and determines whether or not it needs to go to the user INEX exception handler. If not, the M68040FPSP returns to normal instruction execution. Otherwise, the M68040FPSP SNAN exception handler restores the FPU to its exceptional state, cleans up the stack to the conditions prior to execution, and continues instruction execution at the user INEX exception handler. No parameters are passed to the user INEX exception handler since the M68040FPSP SNAN exception handler provides the illusion that it never existed.

b. If the user SNAN exception handler is enabled, the M68040FPSP SNAN exception handler checks to see if the destination is a floating-point data register or in memory (or an integer data register) with single-, double-, or extended-precision format. If so, the M68040FPSP SNAN exception handler determines which input operand is the SNAN, sets the SNAN bit in the NAN data format, and transfers the resulting nonsignaling NAN to the destination. Once the destination has been written, the M68040FPSP SNAN exception handler restores the FPU to its exceptional state, cleans up the stack to the conditions prior to its execution, and continues instruction execution at the user SNAN exception handler. No parameters are passed to the user SNAN exception handler since the M68040FPSP SNAN exception handler provides the illusion that it never existed.

The user SNAN exception handler must execute an FSAVE as the first floating-point instruction. Table 9-16 lists the floating-point state frame fields for SNAN pre-instruction exceptions resulting from the execution of opclass 010 or 000 (register-to-register or memory-to-register) instructions, and for SNAN post-instruction exceptions resulting from the execution of opclass 011 (register-to-memory) instructions defined for the use by the supervisor exception handler. A source or destination SNAN is stored in ETEMP or FPTEMP, respectively, with its SNAN bit set.

The user SNAN exception handler can overwrite the result to the specified destination. The exception handler must be aware that it is possible for an INEX1 exceptional condition to co-exist with an SNAN exception. Since the SNAN exception has higher priority, the INEX1 exception is hidden, and it becomes the responsibility of the SNAN exception handler to detect and correct this if desired. To return to normal execution, the state frame is discarded prior to execution of the RTE of the user-defined exception handler.

9.7.3 Operand Error

The operand error exception encompasses problems arising in a variety of operations, including those errors not frequent or important enough to merit a specific exceptional condition. Basically, an operand error occurs when an operation has no mathematical interpretation for the given operands. Table 9-11 lists the possible operand errors, both native and not native to the MC68040, which the M68040FPSP unimplemented instruction

exception handler can report. When an operand error occurs, the OPERR bit is set in the FPSR EXC byte.

Table 9-11. Possible Operand Errors Exceptions

Instruction	Condition Causing Operand Error
Native to MC68040	
FADD	(+inf) + (–inf) or (–inf) + (+inf)
FDIV	0 ÷ 0 or inf ÷ inf
FMOVE to B,W,or L	Integer overflow where the source is nonsignaling NAN or +infinity.
FMUL	One operand is 0 and other is +inf.
FSQRT	Source < 0 or ±inf.
FSUB	(+inf) – (+inf) or (–inf) – (–inf)
Nonnative to MC68040	
FACOS	Source is ±inf, > +1, or < –1
FASIN	Source is ±inf, > +1, or < –1
FATANH	Source is > +1 or < –1
FCOS	Source is ±inf
FGETEXP	Source is ±inf
FGETMAN	Source is ±inf
FLOG10	Source is < 0
FLOG2	Source is < 0
FLOGN	Source is < 0
FLOGNP1	Source is ≤ 1
FMOD	Floating-point data register is ±inf or source is 0, other operand is not a NAN
FMOVE to P	Source exponent > 999 (decimal) or k-Factor > 17
FREM	Floating-point data register is ±inf or source is 0, other operand is not a NAN
FSCALE	Source is ±inf
FSGLDIV	0 ÷ 0 or inf ÷ inf
FSGLMUL	One operand is 0, other operand is inf
FSIN	Source is ±inf
FSINCOS	Source is ±inf
FTAN	Source is ±inf

9.7.3.1 MASKABLE EXCEPTION CONDITIONS. All conditions apply as listed in Table 9-11, with the exception of the FMOVE to byte, word, or long-word case.

a. If the user OPERR exception handler is disabled, an extended-precision nonsignaling NAN with all mantissa bits set is stored in the destination floating-point data register. No exceptions are reported, and instruction execution proceeds normally.

b. If the user OPERR exception handler is enabled and the destination floating-point data register is not modified, an OPERR exception is posted. The next floating-point instruction that is encountered takes a pre-instruction exception. The OPERR entry in the processor's vector table points to the M68040FPSP OPERR exception handler. Once the M68040FPSP OPERR exception handler recognizes the operand error as a maskable condition, it does not modify the destination or pass control to the user OPERR exception handler.

9.7.3.2 NONMASKABLE EXCEPTION CONDITIONS. If an FMOVE to byte, word, or long word has a source operand that is too large to be represented in the specified destination integer format (integer overflow, NAN, infinity) or if the source operand is equal to the largest negative integer representable in the specified destination integer format (erroneous MC68040 condition), the processor immediately takes a post-instruction exception. Instruction execution continues at the M68040FPSP OPERR exception handler.

If the M68040FPSP determines a nonmaskable erroneous MC68040 condition caused the exception, it stores the largest negative integer representable in the given destination integer format (-2^7 for byte, -2^{15} for word, and -2^{31} for long word). The M68040FPSP OPERR exception handler then returns the processor to normal processing. If an integer overflow or an FMOVE to byte, word, or long word with a source of infinity causes the exception, then the destination is written with the largest positive or negative integer that can be represented in the given format. If an FMOVE to byte of word or long word with a source of NAN causes the exception, then the most significant 8, 16, or 32 bits, respectively, are written to the destination. Next, the M68040FPSP OPERR exception handler checks to see if the user OPERR exception handler is enabled.

a. If the user OPERR exception handler is disabled, an exception-causing INEX1 or INEX2 condition exists, and the user INEX exception handler is enabled. The M68040FPSP OPERR exception handler restores the FPU to its exceptional state, cleans up the stack to the conditions prior to execution, and continues instruction execution at the user INEX exception handler. No parameters are passed to the user INEX exception handler since the M68040FPSP OPERR exception handler provides the illusion that it never existed. Otherwise, the M68040FPSP OPERR exception handler returns the processor to normal processing.
b. If the user OPERR exception handler is enabled and the destination is a floating-point data register, then the M68040FPSP exception handler does not modify the register. The M68040FPSP OPERR exception handler restores the FPU to its exceptional state, cleans up the stack to the conditions prior to execution, and continues instruction execution at the user OPERR exception handler. No parameters are passed to the user OPERR exception handler since the M68040FPSP OPERR exception handler provides the illusion that it never existed.

The user OPERR exception handler must execute an FSAVE as its first floating-point instruction. Table 9-16 lists the floating-point state frame fields for OPERR exceptions resulting from the execution of opclass 010 or 000 (register-to-register or memory-to-register) instructions and opclass 011 (register-to-memory) instructions defined for the use by the supervisor exception handler.

The CMDREG1B field of the floating-point state frame can be used to determine the instruction that caused of the OPERR exception. Note that CMDREG1B could be any of the instructions listed in Table 9-11. If the destination is a floating-point data register, this exception handler needs to supply the contents. If the destination is memory, the effective address is supplied in the format $3 stack frame. If the destination is an integer data register, the FPIAR points to the F-line instruction word that contains the integer data register number. To exit the user OPERR exception handler, the saved floating-point frame need not be restored and can be discarded prior to execution of the RTE instruction.

9.7.4 Overflow

An overflow exception is detected for arithmetic operations in which the destination is a floating-point data register or memory when the intermediate result's exponent is greater than or equal to the maximum exponent value of the selected rounding precision. Overflow can only occur when the destination is in the single-, double-, or extended-precision format; all other data format overflows are handled as operand errors. At the end of any operation that could potentially overflow, the intermediate result is checked for underflow, rounded, and then checked for overflow before it is stored to the destination. If overflow occurs, the OVFL bit is set in the FPSR EXC byte.

Even if the intermediate result is small enough to be represented as an extended-precision number, an overflow can occur. The intermediate result is rounded to the selected precision, and the rounded result is stored in the extended-precision format. If the magnitude of the intermediate result exceeds the range of the selected rounding precision format, an overflow occurs.

9.7.4.1 MASKABLE EXCEPTION CONDITIONS. There are no conditions.

9.7.4.2 NONMASKABLE EXCEPTION CONDITIONS. When the OVFL bit is set in the FPSR EXC byte as a result of a floating-point instruction, the processor always takes a nonmaskable overflow exception. If the destination is a floating-point data register, then the register is not affected, and either a pre-instruction or a post-instruction exception is reported. If the destination is a memory or integer data register, an undefined result is stored, and a post-instruction exception is taken immediately. Execution begins at the M68040FPSP OVFL exception handler.

The values defined in Table 9-12 are stored in the destination based on the rounding mode defined in the FPCR MODE byte. The M68040FPSP OVFL exception handler rounds the result according to the rounding precision defined in the FPCR MODE byte if the destination is a floating-point data register. If the destination is in memory or an integer data register, then the rounding precision in the FPCR MODE byte is ignored, and the given destination format defines the rounding precision. If the instruction has a forced rounding precision (e.g., FSADD, FDMUL), the instruction defines the rounding precision. The M68040FPSP OVFL exception handler then checks to see if the user OVFL exception handler is enabled.

Table 9-12. Overflow Rounding Mode Values

Rounding Mode	Result
RN	Infinity, with the sign of the intermediate result.
RZ	Largest magnitude number, with the sign of the intermediate result.
RM	For positive overflow, largest positive number; for negative overflow, infinity.
RP	For positive overflow, infinity; for negative overflow, largest negative number.

a. If the user OVFL exception handler is disabled, the M68040FPSP OVFL exception handler checks for an INEX1 or INEX2 exception condition with the user INEX exception handler enabled. If not, the processor returns to normal instruction flow. Otherwise, the M68040FPSP OVFL exception handler restores the FPU to its exceptional state, cleans up the stack to the conditions prior its execution, and continues instruction execution at the user INEX exception handler. No parameters are passed to the user INEX exception handler since the M68040FPSP OVFL exception handler provides the illusion that it never existed. Otherwise, the M68040FPSP OVFL exception handler returns the processor to normal processing.

b. If the user OVFL exception handler is enabled, the M68040FPSP OVFL restores the FPU to its exceptional state, cleans up the stack to the conditions prior to execution, and continues instruction execution at the user OVFL exception handler. No parameters are passed to the user OVFL exception handler since the M68040FPSP OVFL exception handler provides the illusion that it never existed.

The user OVFL exception handler must execute an FSAVE as its first floating-point instruction. The destination contains the rounding mode values listed in Table 9-12, and the user OVFL exception handler can choose to modify these values. The E3 and E1 bits of the floating-point state frame are examined to determine which fields on the floating-point state frame are valid. E3 always takes precedence and must be serviced first. Table 9-16 lists the floating-point state frame fields for OVFL exceptions with E3 set or with E3 clear and E1 set. Note that it is possible for an FADD, FSUB, FMUL, and FDIV to report a post-instruction exception, although these instructions normally generate a pre-instruction exception. The following example illustrates the reason why a post-instruction exception is generated.

```
FADD      FP2,FP0       ; this instruction generates an overflow exception
FMOVE     FP0, <ea>     ; this instruction is executing when overflow occurs
```

In this example, assume that the FMOVE instruction starts once the FADD instruction generates an overflow. Given the register dependency on FP0, the destination of the FADD instruction, FP0 needs to be resolved prior to FMOVE instruction execution. For this example, there is no choice but to have the FADD instruction report a post-instruction exception immediately. Note that for this case, even though the T-bit of the floating-point state frame is set, (post-instruction exception), it does not imply an FMOVE OUT instruction. Therefore, the effective address field in the format $3 stack frame is invalid.

The FMOVE OUT instruction generates a post-instruction exception. For this case, the effective address field in the format $3 stack frame points to the destination memory location. If the destination is an integer data register, the FPIAR points to the F-line word

of the offending instruction, and the F-line word contains the integer data register number. If the M68040FPSP unimplemented instruction exception handler is used, there can be some other cases in which an overflow is reported.

In addition to normal overflow, the exponential instructions can generate results that catastrophically overflow the 16-bit exponent used for intermediate results. For these instructions (FETOX, FTENTOX, FTWOTOX, FSINH, and FCOSH), the intermediate result found in either FPTEMP or WBTEMP fields of the floating-point state frame are invalid. If an INEX2 or INEX1 exceptional condition exists and the user INEX exception handler is enabled, it is the responsibility of the user OVFL exception handler to look for this situation.

The user OVFL exception handler examines the E3 bit of the floating-point state frame to exit from this exception handler. If the E3 bit is set, it must be cleared prior to restoring the floating-point frame through the FRESTORE instruction. If the E3 bit is clear and the E1 bit is set, the floating-point state frame is discarded. The RTE instruction must be executed to return to normal instruction flow.

9.7.5 Underflow

An underflow exception occurs when the intermediate result of an arithmetic operation is too small to be represented as a normalized number in a floating-point data register or memory using the selected rounding precision. An arithmetic operation is too small when the intermediate result exponent is less than or equal to the minimum exponent value of the selected rounding precision. Underflow is not detected for intermediate result exponents that are equal to the extended-precision minimum exponent since the explicit integer part bit permits representation of normalized numbers with a minimum extended-precision exponent. Underflow can only occur when the destination format is single, double, or extended precision. When the destination format is byte, word, or long word, the conversion underflows to zero without causing either an underflow or an operand error. At the end of any operation that could potentially underflow, the intermediate result is checked for underflow, rounded, and checked for overflow before it is stored at the destination. If an underflow occurs, the UNFL bit is set in the FPSR EXC byte.

Even if the intermediate result is large enough to be represented as an extended-precision number, an underflow can occur. The intermediate result is rounded to the selected precision, and the rounded result is stored in extended-precision format. If the magnitude of the intermediate result is too small to be represented in the selected rounding precision, an underflow occurs.

The IEEE 754 standard defines two causes of an underflow: 1) when the absolute value of the number is less than the minimum number that can be represented by a normalized number in a specific data format; 2) when loss of accuracy occurs while attempting to calculate such a number (a loss of accuracy also causes an inexact exception). The IEEE 754 standard specifies that if the underflow exception is disabled, an underflow should only be signaled when both of these cases are satisfied (i.e., the result is too small to be represented with a given format and there is a loss of accuracy during calculation of the

final result). If the exception is enabled, the underflow should be signaled any time a very small result is produced, regardless of whether accuracy is lost in calculating it.

The processor UNFL bit in the FPSR AEXC byte implements the IEEE exception disabled definition since it is only set when a very small number is generated and accuracy has been lost when calculating that number. The UNFL bit in the FPCR EXC byte implements the IEEE exception enabled definition since it is set any time a tiny number is generated.

9.7.5.1 MASKABLE EXCEPTION CONDITIONS. There are no conditions.

9.7.5.2 NONMASKABLE EXCEPTION CONDITIONS. When the UNFL bit of the FPSR is set, the processor always takes an exception regardless of whether or not the user UNFL exception handler is enabled. If the destination is a floating-point data register, the register is not affected, and either a pre-instruction or a post-instruction exception is reported. If the destination is a memory or integer data register, then an undefined result is stored, and a post-instruction exception is taken immediately. Exception processing begins with the M68040FPSP UNFL exception handler.

The M68040FPSP UNFL exception handler stores the result in the destination as either a denormalized number or zero. Shifting the mantissa of the intermediate result to the right while incrementing the exponent until it is equal to the denormalized exponent value for the destination format accomplishes denormalization. The denormalized intermediate result is rounded to the selected rounding precision if the destination is a floating-point data register or rounded to the destination format in the case of an FMOVE OUT instruction. For the instructions with forced rounding precision (e.g., FSADD and FDMUL), the destination is rounded using the precision defined by the instruction.

If in the process of denormalizing the intermediate result, all of the most significant bits are shifted off to the right, the selected rounding mode determines the value to be stored at the destination, Table 9-13 lists these values. Once the result is stored in the destination, the M68040FPSP UNFL exception handler checks to see if the user UNFL exception handler is enabled.

Table 9-13. Underflow Rounding Mode Values

Rounding Mode	Result
RN	Zero, with the sign of the intermediate result.
RZ	Zero, with the sign of the intermediate result.
RM	For positive overflow, + zero; for negative underflow, smallest denormalized negative number.
RP	For positive overflow, smallest denormalized positive number; for negative underflow, –zero.

a. If the user UNFL exception handler is disabled, the M68040FPSP UNFL exception handler checks for an INEX1 or INEX2 exception condition with the user INEX exception handler enabled. If not, the processor returns to normal instruction flow. Otherwise, the M68040FPSP UNFL exception handler restores the FPU to its exceptional state, cleans up the stack to the conditions prior to execution, and continues instruction execution at the user INEX exception handler. No parameters are passed to the user INEX exception handler since the M68040FPSP UNFL exception handler provides the illusion that it never existed. Otherwise, the M68040FPSP UNFL exception handler returns the processor to normal processing.

b. If the user UNFL exception handler is enabled, the M68040FPSP UNFL exception handler restores the FPU to its exceptional state, cleans up the stack to the conditions prior to execution, and continues instruction execution at the user UNFL exception handler. Once the M68040FPSP UNFL exception handler recognizes the operand error as a maskable condition, it does not modify the destination or pass control to the user UNFL exception handler.

The user UNFL exception handler must execute an FSAVE as its first floating-point instruction. At this point, the destination contains the rounding mode values listed in Table 9-13, and the user UNFL exception handler can choose to modify these values. The E3 and E1 bits of the floating-point state frame need to be examined to determine which fields on the floating-point state frame are valid. E3 always takes precedence and must always be serviced first. Table 9-16 lists the floating-point state frame fields for OVFL exceptions with E3 set or with E3 clear and E1 set. It is possible for an FADD, FSUB, FMUL, and FDIV to report a post-instruction exception, although these instructions normally generate a pre-instruction exception. The following example illustrates why a post-instruction exception is generated.

```
FADD      FP2,FP0       ; this instruction generates an underflow exception
FMOVE     FP0, <ea>     ; this instruction is executing when underflow occurs
```

In this example, assume that the FMOVE instruction starts once the FADD instruction generates an underflow. Given the register dependency on FP0, the destination of the FADD instruction, FP0 needs to be resolved prior to the FMOVE instruction execution. For this example, there is no choice but to have the FADD instruction report a post-instruction exception immediately. Note that for this case, even though the T-bit of the floating-point state frame is set (post-instruction exception), it does not imply an FMOVE OUT instruction. Therefore, the effective address field in the format $3 stack frame is invalid.

The FMOVE OUT instruction generates a post-instruction exception. For this case, the effective address field in the format $3 stack frame points to the destination memory location. If the destination is an integer data register, the FPIAR points to the F-line word of the offending instruction, and the F-line word contains the integer data register number. If the M68040FPSP unimplemented instruction exception handler is used, there can be some other cases in which an underflow is reported. If an INEX2 or INEX1 exceptional condition exists and the user INEX exception handler is enabled, it is the responsibility of the user UNFL exception handler to look for this situation.

The user UNFL exception handler examines the E3 bit of the floating-point state frame to exit from this exception handler. If the E3 bit is set, it must be cleared prior to restoring the floating-point frame through the FRESTORE instruction. If the E3 bit is clear and the E1 bit

is set, the floating-point frame is discarded. The RTE instruction must be executed to return to normal instruction flow.

9.7.6 Divide by Zero

This exception happens when a zero divisor occurs for a divide instruction or when a transcendental function is asymptotic with infinity as the asymptote. Table 9-14 lists the instructions that can cause the divide by zero exception. Note that only the FDIV and FSGLDIV instructions are native to the MC68040. The other conditions occur only if the M68040FPSP is used. When a divide by zero is detected, the DZ bit is set in the FPSR EXC byte. The divide by zero exception only has maskable exceptional conditions; therefore, no M68040FPSP intervention is needed. An exception is taken only if the DZ bit is set in FPSR EXC byte and the corresponding bit in the FPCR ENABLE byte is set.

a. If the user divide by zero exception handler is enabled, an infinity with the sign set to the exclusive OR of the signs of the input operands is stored in the destination floating-point data register. No exception is taken.

b. If the user divide by zero exception handler is disabled, the destination floating-point data register is not modified, and the exception is reported as a pre-instruction exception when the next floating-point instruction is attempted. The divide by zero entry in the processor's vector table points to the user divide by zero exception handler.

Table 9-14. Possible Divide by Zero Exceptions

Instruction	Operand Value
FDIV	Source operand = 0 and floating-point data register is not a NAN
FLOG10	Source operand = 0
FLOG2	Source operand = 0
FLOGN	Source operand = 0
FTAN	Source operand is an odd multiple of $\pm\pi \div 2$
FSGLDIV	Source operand = 0 and floating-point data register is not a NAN

An FSAVE must be the first instruction of the user divide by zero exception handler. The user divide by zero exception handler must generate a result to store in the destination. To assist the exception handler in this function, the processor supplies the information listed in Table 9-16, which lists the floating-point state frame fields for divide by zero exceptions that are defined for supervisor exception handler use. To exit the user divide by zero exception handler, the saved floating-point frame is discarded, and an RTE returns the processor to normal processing.

9.7.7 Inexact Result

The processor provides two inexact bits in the FPSR EXC byte to help distinguish between inexact results generated by emulated decimal input (INEX1 exceptions) and other inexact results (INEX2 exceptions). These two bits are useful in instructions where both types of inexact results can occur (e.g., FDIV.P #7E-1,FP3). In this case, the packed decimal to extended-precision conversion of the immediate source operand causes an

inexact error to occur that is signaled as INEX1 exception. Furthermore, the subsequent divide could also produce an inexact result and cause INEX2 to be set in the FPCR EXC byte. Note that only one inexact exception vector number is generated by the processor. If either of the two inexact exceptions is enabled, the processor fetches the inexact exception vector, and the user INEX exception handler is initiated. INEX refers to both exceptions in the following paragraphs.

The INEX2 exception is the condition that exists when any operation, except the input of a packed decimal number, creates a floating-point intermediate result whose infinitely precise mantissa has too many significant bits to be represented exactly in the selected rounding precision or in the destination data format. If this condition occurs, the INEX2 bit is set in the FPSR EXC byte, and the infinitely precise result is rounded. Table 9-15 lists these rounding mode values.

Table 9-15. Divide by Zero Rounding Mode Values

Rounding Mode	Result
RN	The representable value nearest to the infinitely precise intermediate value is the result. If the two nearest representable values are equally near (a tie), then the one with the least significant bit equal to zero (even) is the result. This is sometimes referred to as “round nearest, even.”
RZ	The result is the value closest to and no greater in magnitude than the infinitely precise intermediate result. This is sometimes referred to as the “chip mode,” since the effect is to clear the bits to the right of the rounding point.
RM	The result is the value closest to and no greater than the infinitely precise intermediate result (possibly minus infinity).
RP	The result is the value closest to and no less than the infinitely precise intermediate result (possibly plus infinity).

The INEX1 and INEX2 exceptions are always maskable. Therefore, any INEX exception goes directly to the user INEX exception handler. The M68040FPSP does not provide any special handling for the INEX exception. When an INEX2 or INEX1 bit in the FPSR EXC byte is set, the processor stores the rounded result (listed in Table 9-15), to the destination. The FPCR MODE byte determines the rounding mode, and the PREC byte determines the rounding precision if the destination is a floating-point data register. Otherwise, if the destination is memory or an integer data register, the destination format determines the rounding precision. If one of the instructions has a forced precision, the instruction determines the rounding precision. If the INEX2 or INEX1 condition exists and if the corresponding INEX bit in the FPCR ENABLE byte is set, then the user INEX exception handler is taken.

a. If the user INEX exception handler is disabled, result is rounded and normal processing continues.
b. If the user INEX exception handler is enabled, the exception is taken. The INEX entry in the processor’s vector table points to the user INEX exception handler.

The user INEX exception handler must execute an FSAVE as its first floating-point instruction. At this point, the destination contains the rounding mode values as listed in

Table 9-15, and the user INEX exception handler can choose to modify these values. The E3 and E1 of the floating-point state frame bits need to be examined to determine which fields in the floating-point state frame are valid. E3 always takes precedence and must always be serviced first. Table 9-16 lists the floating-point state frame fields for INEX exceptions with E3 set or with E3 clear and E1 set. It is possible for an FADD, FSUB, FMUL, and FDIV to report a post-instruction exception, although these instructions normally generate a pre-instruction exception. The following example shows why a post-instruction exception is generated.

```
FADD      FP2,FP0       ; this instruction generates an inexact exception
FMOVE     FP0, <ea>     ; this instruction is executing when inexact occurs
```

For this example, assume that the FMOVE instruction starts once the FADD instruction generates an underflow. Given the register dependency on FP0, the destination of the FADD instruction, FP0 needs to be resolved prior to the FMOVE instruction execution. For this example, there is no choice but to have the FADD instruction report a post-instruction exception immediately. Note that for this case, even though the T-bit of the floating-point state frame is set (post instruction exception), it does not imply an FMOVE OUT instruction. Therefore, the effective address field in the format $3 stack frame is invalid.

The FMOVE OUT instruction generates a post-instruction exception. For this case, the effective address field in the format $3 stack frame points to the destination memory location. If the destination is an integer data register, the FPIAR points to the F-line word of the offending instruction, and the F-line word contains the integer data register number. If the MC68040FPSP unimplemented instruction exception handler is used, there can be some other cases in which an inexact exception is reported.

The user INEX exception handler examines the E3 bit of the floating-point state frame to exit from this exception handler. If the E3 bit is set, it must be cleared prior to restoring the floating-point frame via the FRESTORE instruction. If the E3 bit is clear and the E1 bit is set, the floating-point frame is discarded. The RTE instruction must be executed to return to normal instruction flow.

NOTE

The IEEE 754 standard specifies that inexactness should be signaled on overflow as well as for rounding. The processor implements this via the INEX bit in the FPSR AEXC byte. However, the standard also indicates that the inexact exception should be taken if an overflow occurs with the OVFL bit disabled and the INEX bit enabled in the FPSR AEXC byte. Therefore, the processor takes the inexact exception if this combination of conditions occurs, even though the INEX1 or INEX2 bit may not be set in the FPSR EXC byte. In this case, the INEX bit is set in the FPSR AEXC byte, and the OVFL bit is set in both the FPSR EXC and AEXC bytes.

9.8 FLOATING-POINT STATE FRAMES

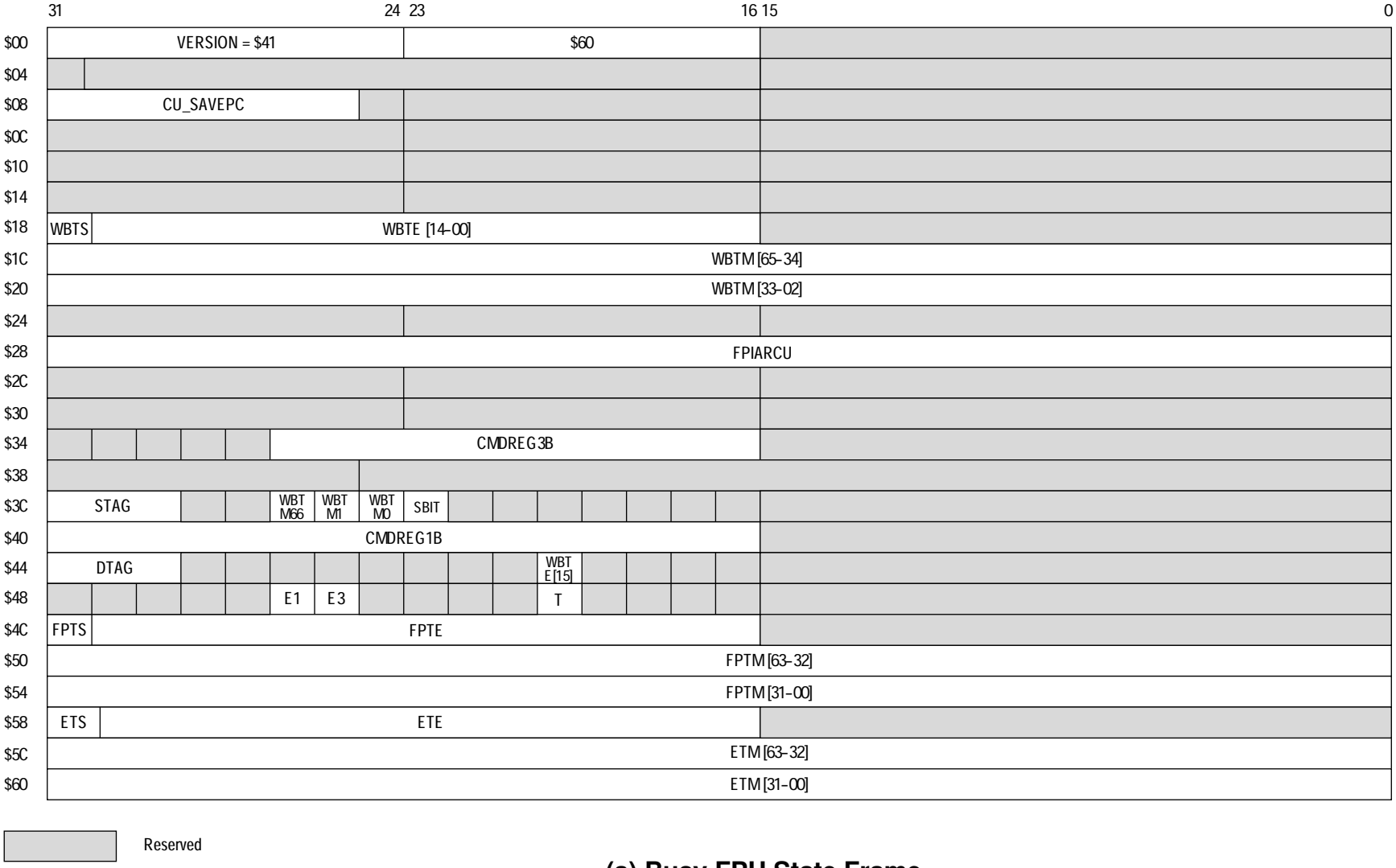
All floating-point arithmetic exception handlers must have FSAVE as the first floating-point instruction; any other floating-point instruction causes another exception to be reported. Once the FSAVE instruction has executed, the exception handler should use only the FMOVEM instruction to read or write to the floating-point data registers since FMOVEM cannot generate further exceptions or change the FPCR.

The FPU executes an FSAVE instruction to save the current floating-point internal state for context switches and floating-point exception handling. When an FSAVE is executed, the processor waits until the FPU either completes execution of all current instructions or is unable to perform further processing due to a pending exception that must be serviced. Any exceptions generated during this time are not reported and are saved in the resulting busy state frame.

Four state frames can be generated as a result of an FSAVE instruction: busy, null, idle, and unimplemented floating-point instruction. When an unimplemented floating-point exception occurs, the FSAVE generates a 26-word unimplemented instruction state frame. When an unsupported data type exception occurs, the FSAVE generates a 50-word busy state frame. All floating-point arithmetic exceptions causes the FSAVE to generate either the 26-word unimplemented instruction state frame or the 50-word busy state frame. For a hardware reset or an FRESTORE of a null state frame, the FSAVE instruction generates a null state frame. This null state frame is generated until the first nonconditional floating-point instruction is executed (conditionals include FNOP, FBcc, FDBcc, FScc, and FTRAPcc). Floating-point conditional instructions do not set an internal flag, which changes the state frame from null to idle. If these instructions are the only ones executed after a reset or an FRESTORE of a null state frame, then when FSAVE is executed, it stacks a null state frame instead of an idle state frame. Note that this function is different from that of the MC68881 and MC68882, and software must be aware of this difference if compatibility with the MC68881 and MC68882 is desired. Once a nonconditional floating-point instruction is executed, an FSAVE generates an idle state frame. The idle state frame is generated whenever the FPU has no exceptions pending. An idle state frame is saved if no exceptions are pending and at least one instruction has been executed since the last hardware reset or FRESTORE of a null state frame. A 26-word unimplemented floating-point instruction state frame is saved if the last instruction was an unimplemented floating-point instruction. Figure 9-10 illustrates each of these state frames, followed by definitions for each of the fields listed in alphabetical order.

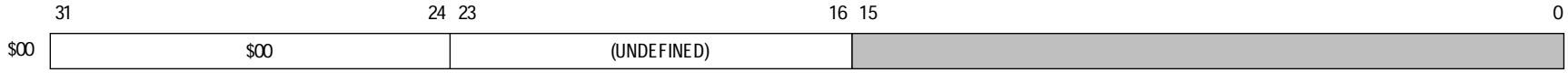
NOTE

The notation [XX–XX] indicates the length of the field but does not indicate the field's actual location. [XX, XX–XX] indicates that one bit of the field is located separately or termed differently from the other bits. This notation is for convenience of explanation only. For example, WBTM [65–34] indicates that WBTM is 32 bits long and gives a reference to each bit in WBTM without giving its actual location in the state frame. For the actual locations refer to Figure 9-10.

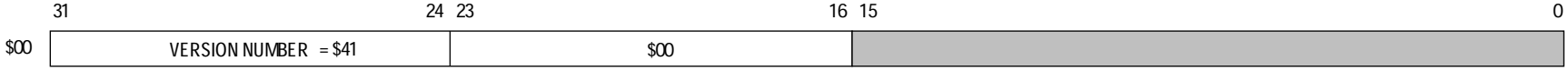


(a) Busy FPU State Frame

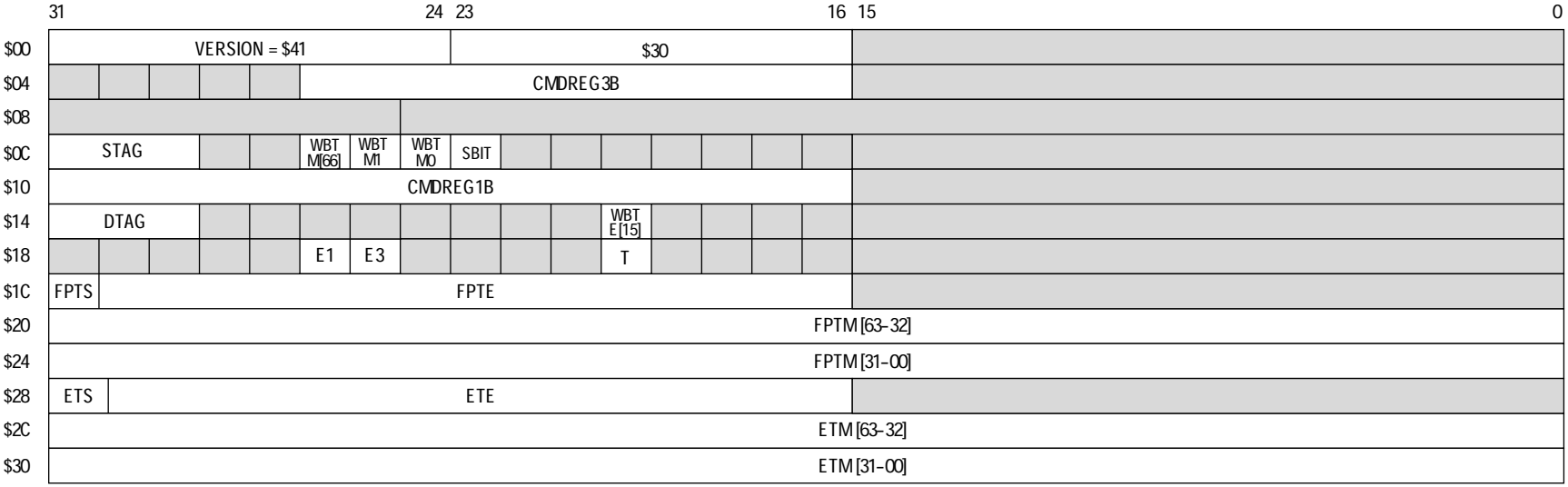
Figure 9-10. MC68040 Floating-Point State Frames (Sheet 1 of 2)



(b) Null FPU State Frame



(c) Idle FPU State Frame



(d) Unimplemented Instruction FPU State Frame

Figure 9-10. MC68040 Floating-Point State Frames (Sheet 2 of 2)

CMDREG1B—This field contains the command word of the exceptional floating-point instruction for an E1 exception, which is an exception detected by the conversion unit (CU) in the floating-point pipeline (see Figure 9-1). For FSQRT, CMDREG1B [6–0] are mapped from $4 for the instruction to $5 in CMDREG1B. All other instructions map directly.

CMDREG3B—This field contains the encoded instruction command word for an E3 exception, which is an exception detected by the write-back unit (WB) in the floating-point pipeline (see Figure 9-1). Figure 9-11 details the bit mapping between CMDREG1B and CMDREG3B. For FSQRT, bits CMDREG1B [6–0] are changed from $4 for the instruction to $5 for CMDREG1B, and therefore map to $21 for CMDREG3B.

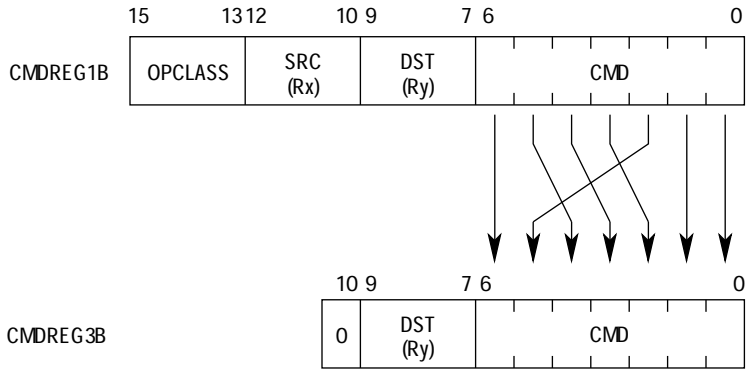


Figure 9-11. Mapping of Command Bits for CMDREG3B Field

CU_SAVEPC—This field contains the PC for the FPU pipeline's conversion unit.

E1—If set, this bit indicates that an exception has been detected by the conversion unit pipeline stage. All exception types are possible. The exception handler first checks for an E3 exception and processes it before checking and processing an E1 exception. The E1 exception is processed if the E1 bit is set. For the unimplemented instruction state frame, the source operand's unsupported data type is packed if the E1 bit is set.

E3—If set, this bit indicates that an exception has been detected by the WB pipeline stage. Only OVFL, UNFL, and INEX2 exceptions on opclass 010 or 000 (register to register and memory to register) for FADD, FSUB, FMUL, FDIV, FSQRT can occur. The exception handler must check for and process an E3 exception first.

ETS, ETE, ETM—Collectively, these fields are referred to as the ETEMP register and normally contain the source operand converted to extended precision. If the instruction specifies a packed decimal real source, bits 63–0 of the operand reside in ETM [63–00], and the ETS and ETE fields are undefined.

FPIARCU—This field contains the instruction address register for the FPU pipeline's conversion unit.

FPTS, FPTE, FPTM—Collectively, these fields are referred to as the FPTEMP register and normally contain the destination operand for dyadic operations converted to extended precision. If the instruction specifies a packed decimal real source, bits 95–64 of the operand reside in FPTM [31–00], and the FPTS, FPTE, and FPTM [63–32] fields are undefined.

OPCLASS—This field refers to bits 15–13 of CMDREG1B. Note that CMDREG1B is identical to the second word of a floating-point arithmetic instruction opcode.

STAG, DTAG—These 3-bit fields specify the data type of the source and destination operands, respectively. STAG is undefined for a packed decimal real source operand. The encodings for STAG and DTAG are as follows:

000 = Normalized
001 = Zero
010 = Infinity
011 = NAN
100 = Extended-Precision Denormalized or Unnormalized Input
101 = Single- or Double-Precision Denormalized Input

T—If set, this bit indicates that a post-instruction exception has occurred. Since only an opclass 3 instruction can indicate a post-instruction exception, this bit indicates that the exception is caused by an FMOVE OUT instruction.

WBTS, WBTE [15,14–00], WBTM [66,65–02,01,00], SBIT—These fields contain the exception operand in internal data format for E3 exceptions. Collectively, these fields are called the WBTEMP and are an image of the intermediate result. WBTM66 is the overflow bit; WBTM1, WBTM0, and SBIT are the guard, round, and sticky bits, respectively.

Table 9-16. State Frame Field Information

FSAVE State Frame Field	Contents
Unimplemented Instruction Exceptions (For Opclass 000 and 010)	
CMDREG1B	Exception Instruction Command Word
ETEMP	Source operand is converted to extended precision. If format is packed, ETM [63–0] contains bits 63–0 of the packed decimal operand.
STAG	Source operand tag (undefined if format is packed).
FPTEMP	Destination operand, if any, is converted to extended precision. If format is packed, FPTM [31–0] contains bits 95–64 of the packed decimal operand.
DTAG	Destination operand tag, if any.
E1	Always 1
T	Always 0
Unsupported Data Type (For Opclass 000 and 010)	
CMDREG1B	Exception Instruction Command Word
ETEMP	Source operand is converted to extended precision. If format is packed, ETM [63–0] contains bits 63–0 of the packed decimal operand.
STAG	Source operand tag (undefined if format is packed).
FPTEMP	Destination operand, if any, is converted to extended precision. If format is packed, FPTM [31–0] contains bits 95–64 of the packed decimal operand.
DTAG	Destination operand tag, if any.
E1	Always 1
T	Always 0
Unsupported Data Type (For Opclass 011)	
CMDREG1B	FMOVE Command Word
ETEMP	Unrounded Source Operand from Floating-Point Data Register
STAG	Source Operand Tag
E1	Always 1
T	Always 1
SNAN (For Opclass 000 and 010)	
CMDREG1B	Exception Instruction Command Word
ETEMP	Source operand is converted to extended precision.
STAG	Source Operand Tag
FPTEMP	Destination operand, if any, is converted to extended precision.
DTAG	Destination operand tag, if any.
E1	Always 1
T	Always 0

Table 9-16. State Frame Field Information (Continued)

FSAVE State Frame Field	Contents
SNAN (For Opclass 011)	
CMDREG1B	FMOVE Instruction Command Word
ETEMP	Unrounded Source Operand from Floating-Point Register, with SNAN bit set.
STAG	Source Operand Tag, indicated NAN.
E1	Always 1
T	Always 1
OPERR (For Opclass 000 and 010)	
CMDREG1B	Exception Instruction Command Word
ETEMP	Source operand is converted to extended precision.
STAG	Source Operand Tag
FPTEMP	Destination operand, if any, is converted to extended precision.
DTAG	Destination operand tag, if any.
E1	Always 1
T	Always 0
OPERR (For Opclass 011)	
CMDREG1B	FMOVE Instruction Command Word
ETEMP	Unrounded Source Operand from Floating-Point Register
STAG	Source Operand Tag
WBTEMP	Contains the rounded integer used to check for erroneous integer overflow.
E1	Always 1
T	Always 1
OVFL (FMOVE to Register, FABS, and FNEG)	
CMDREG1B	Exception Instruction Command Word
FPTEMP	Intermediate result with mantissa rounded to correct precision.
STAG	Source Operand Tag = Normalized
E1	Always 1
T	Always 0
OVFL (FADD, FSUB, FMUL, FDIV, and FSQRT)	
CMDREG3B	Encoded Exception Instruction Command Word
WBTEMP	WBTS, WBTE, and WBTM equal the intermediate result with mantissa rounded to the correct precision.
WBTE15	Bit 15 of the intermediate result's 16-bit exponent = 0 for overflow.
E3	Always 1
T	Either 1 or 0

Table 9-16. State Frame Field Information (Continued)

FSAVE State Frame Field	Contents
OVFL (FMOVE to Memory)	
CMDREG1B	FMOVE instruction command word
FPTEMP	Intermediate result with mantissa rounded to correct precision.
STAG	Source Operand Tag = Normalized
E1	Always 1
T	Always 1
UNFL (FMOVE to Register, FABS, and FNEG)	
CMDREG1B	Exception Instruction Command Word
FPTEMP	Unrounded, Extended-Precision Intermediate Result
STAG	Source Operand Tag = Normalized
E1	Always 1
T	Always 0
UNFL (FADD, FSUB, FMUL, FDIV, and FSQRT)	
CMDREG3B	Encoded Exception Instruction Command Word
WBTEMP	WBTS, WBTE, and WBTM = intermediate result sign, biased 15-bit exponent, and 64-bit mantissa prior to rounding.
WBTE15	Bit 15 of the intermediate result's 16-bit exponent = 1 for underflow.
WBTM1, WBTM0, SBIT	Guard, round, and sticky of intermediate result’s 67-bit mantissa.
E3	Always 1
T	Either 1 or 0
UNFL (FMOVE to Memory)	
CMDREG1B	FMOVE Instruction Command Word
FPTEMP	Intermediate result with mantissa prior to rounding.
STAG	Source Operand Tag = Normalized
E1	Always 1
T	Always 1
DZ	
CMDREG1B	M68040FPSP divide by zero can generate.
ETEMP	Source operand is converted to extended precision.
STAG	Source Operand Tag
FPTEMP	Destination operand is converted to extended precision.
E1	Always 1
T	Always 0

Table 9-16. State Frame Field Information (Concluded)

FSAVE State Frame Field	Contents
INEX (FMOVE to Register, FABS, and FNEG)	
CMDREG1B	Exception Instruction Command Word
FPTEMP	Unrounded, Extended-Precision Intermediate Result
STAG	Source Operand Tag = Normalized
E1	Always 1
T	Always 0
INEX (FADD, FSUB, FMUL, FDIV, and FSQRT)	
CMDREG3B	Encoded Exception Instruction Command Word
WBTEMP	WBTS, WBTE, and WBTM = intermediate result sign, biased 15-bit exponent, and 64-bit mantissa prior to rounding.
WBTE15	Either 1 or 0, generally useless for INEX exceptions.
WBTM1, WBTM0, SBIT	Guard, round, and sticky of intermediate result's 67-bit mantissa.
E3	Always 1
T	Either 1 or 0
INEX (FMOVE to Memory)	
CMDREG1B	FMOVE Instruction Command Word
FPTEMP	Intermediate result with mantissa prior to rounding.
STAG	Source Operand Tag = Normalized
E1	Always 1
T	Always 1

NOTE: If the M68040FPSP unimplemented exception handler is used, the above state frame information applies. The CMDREG1B or CMDREG3B fields of the state frame are modified as appropriate to encode the unimplemented instruction opcode. It is the user exception handler's responsibility to use the E3 and E1 field encodings to recognize which state frame information applies. When E3 = 1 and E1 = 1, E3 takes priority and the state frame information for E3 = 1 must be used.

SECTION 10
INSTRUCTION TIMINGS

This section summarizes instruction timings for the M68040. The timings are divided into two groups: integer unit and floating-point unit instruction timings. Each group is further subdivided to separate more complex instruction timings. Each of these subdivided groups is in alphabetical order with no reference to mode. Table 10-1 alphabetically lists instruction timings and their location in this section.

Table 10.1. Instruction Timing Index

Instruction	Page	Instruction	Page	Instruction	Page
ABCD	10-11	BRA	10-11	EOR	10-13
ADD	10-13	BSET	10-15	EORI	10-13
ADDA	10-13	BSR <offset>	10-11	EORI #<xxx>,CCR	10-11
ADDI	10-13	BTST	10-17	EORI #<xxx>,SR	10-11
ADDQ	10-14	CAS	10-17	EXG	10-11
ADDX	10-11	CAS2	10-11	EXT	10-11
AND	10-13	CHK <ea>, Dn	10-17	EXTB	10-11
ANDI	10-13	CHK2 <ea>, Rn	10-18	FABS	10-30,36
ANDI #<xxx>,CCR	10-11	CLR	10-18	FADD	10-30,35
ANDI #<xxx>,SR	10-11	CINV	10-8	FBcc	10-29
ASL	10-14	CMP	10-18	FCMP	10-30,37
ASR	10-14	CMP2	10-19	FDBcc	10-29
Bcc	10-11	CMPA.L	10-19	FDIV	10-30,35
BCHG	10-15	CMPI	10-19	FMOVE	10-30,36
BCLR	10-15	CMPM	10-11	FMOVE FPn,<ea>	10-31
BFCHG	10-15	CPUSH	10-8	FMOVE/FMOVEM to/from CR	10-32
BFCLR	10-15	DBcc	10-11	FMOVEM	10-37
BFEXTS	10-15	DIVS.L	10-20	FMOVEM <ea>,<list>	10-32
BFEXTU	10-15	DIVS.W	10-20	FMOVEM <list>,<ea>	10-32
BFFFO	10-16	DIVSL.L	10-20	FMUL	10-30,35
BFINS	10-16	DIVU.L	10-20	FNEG	10-30,36
BFSET	10-15	DIVU.W	10-20	FNOP	10-29
BFTST	10-16	DIVUL.L	10-20	FRESTORE <ea>	10-34

Table 10.1. Instruction Timing Index (Continued)

Instruction	Page	Instruction	Page	Instruction	Page
FSAVE <ea>	10-33	MOVEP	10-11	ROL	10-26
FScc	10-32	MOVEQ	10-11	ROR	10-26
FSQRT	10-30,36	MOVES <ea>,An	10-24	ROXL	10-27
FSUB	10-30,35	MOVES <ea>,Dn	10-24	ROXR	10-27
FTRAPcc	10-29	MOVES Rn,<ea>	10-24	RTD	10-11
FTST <ea>, FPn	10-30	MULS.W/L	10-25	RTE	10-11
ILLEGAL	10-11	MULU.W/L	10-25	RTR	10-11
JMP	10-20	NBCD	10-25	RTS	10-11
JSR	10-21	NEG	10-26	SBCD	10-11
LEA	10-21	NEGX	10-26	Scc	10-27
LINK	10-11	NOP	10-11	SUB	10-13
LSL	10-14	NOT	10-26	SUBA	10-27
LSR	10-14	OR	10-13	SUBI	10-13
MOVE	10-9,10	ORI	10-13	SUBQ	10-14
MOVE from CCR	10-21	ORI #<xxx>,CCR	10-11	SUBX	10-11
MOVE from SR	10-22	ORI #<xxx>,SR	10-11	SWAP	10-11
MOVE to CCR	10-22	PACK	10-11	TAS	10-28
MOVE to SR	10-22	PEA	10-26	TRAP#	10-11
MOVE USP	10-11	PFLUSH	10-11	TRAPcc	10-11
MOVE16	10-11	PFLUSHA	10-11	TRAPV	10-11
MOVEA.L	10-23	PFLUSHAN	10-11	TST	10-13
MOVEC	10-11	PFLUSHN (An)	10-11	UNLK	10-11
MOVEM <list>,<ea>	10-23	PTESTR, PTESTW	10-11	UNPK	10-11
MOVEM.L <ea>,<list>	10-23	RESET	10-11		

10.1 OVERVIEW

Refer to **Section 2 Integer Unit** for information on the integer unit pipeline. The <ea> fetch timing is not listed in the following tables because most instructions require one clock in the <ea> fetch stage for each memory access to obtain an operand. An instruction requires one clock to pass through the <ea> fetch stage even if no operand is fetched. Table 10-2 summarizes the number of memory fetches required to access an operand using each addressing mode for long-word aligned accesses. The user must perform his own calculations for <ea> fetch timing for misaligned accesses.

Table 10-2. Number of Memory Accesses

Addressing Mode	Evaluate <ea> And Fetch Operand	Evaluate <ea> And Send To Execution Stage
Dn	0	0
An	0	0
(An)	1	0
(An)+	1	0
–(An)	1	0
(d_{16},An)	1	0
(d_{16},PC)	1	0
(xxx).W, (xxx).L	1	0
#<xxx>	0	0
(d_8,An,Xn)	1	0
(d_8,PC,Xn)	1	0
(BR,Xn)	1	0
(bd,BR,Xn)	1	0
([bd,BR,Xn])	2	1
([bd,BR,Xn],od)	2	1
([bd,BR],Xn)	2	1
([bd,BR],Xn,od)	2	1

In the instruction timing tables, the <ea> calculate column lists the number of clocks required for the instruction to execute in the <ea> calculate stage of the integer unit pipeline. Dual effective address instructions such as ABCD –(Ay),–(Ax) require two calculations in the <ea> calculate stage and two memory fetches. Due to pipelining, the fetch of the first operand occurs in the same clock as the <ea> calculation for the second operand.

The execute column lists the number of clocks required for the instruction to execute in the execute stage of the integer unit pipeline. This number is presented as a lead time and a base time. The lead time is the number of clocks the instruction can stall when entering the execution stage without delaying the instruction execution. If the previous instruction is still executing in the execution stage when the current instruction is ready to move from the <ea> fetch stage, the current instruction stalls until the previous one completes. For

example, if an execution time is listed as 2_L + 1, the lead time is two clocks and the base time is one for a total execution time of three. The instruction can stall for two clocks without delaying the instruction execution time.

The <ea> calculate and execute stages operate in an interlocked manner for all instructions using the brief and full extension word formats. If an instruction using one of these formats is stalled for more than N_L clocks waiting to begin execution in the execute stage, a similar increase in the <ea> calculate time will result. For example, if the execution time listed is 2_L + 1 and the instruction stalls for three clocks, then the <ea> calculate time increases by one clock ($3 - 1 = 2_L$). Write-back times are not listed because they are system dependent and do not affect either <ea> calculate or execute stages of the pipeline.

Not all addressing modes listed in the following tables for an instruction are valid for all variations of the instruction. For example, the table for the integer ADD instruction lists times for both ADD <ea>,Dn and ADD Dn,<ea>. All addressing modes listed are valid for ADD <ea>,Dn. For ADD Dn,<ea> the following invalid modes should be ignored: An, (d_{16},PC), #<xxx>, (d_8,PC,Xn), and modes with BR = PC. Refer to the M68000PM/AD, *M68000 Family Programmer's Reference Manual* for a complete summary of valid instruction and addressing mode combinations. The instruction timings are based on the following suppositions unless otherwise noted:

1. All timings are related to BCLK cycles and are for BR = An or suppressed. For BR = PC, 1 and 1_L clocks to the <ea> calculate and execution times unless otherwise noted. For memory indirect postindexed with suppressed index — ([bd,BR],Xn) or ([bd,BR],Xn,od) with Xn suppressed — times are the same as for memory indirect preindexed with suppressed index — ([bd,BR,Xn]) or ([bd,BR,Xn],od) with Xn suppressed.
2. All memory accesses hit in the caches; no table searches occur as a result of ATC misses except for the operand accesses for the CAS, CAS2, and TAS instructions. These accesses are implicitly noncachable and force external bus accesses. It is assumed that external memory has a zero-wait state in this case and that the bus is granted to the M68040.

   The result increases access time equal to the number of clocks for the memory access (first bus cycle if the operand access results in a line memory access) if aligned accesses miss in the data cache. As an approximation, this time should be added to the execution time for each operand fetch generated by the instruction.
3. All accesses are aligned to a byte boundary that is a multiple of the operand size. For instance, the timing for all long-word accesses assumes that the operands are on long-word boundaries.
4. The integer execution times for floating-point instructions assume that the floating-point unit (FPU) is idle.

10.2 INSTRUCTION TIMING EXAMPLES

The following examples utilize the instruction timing information given in this section. Figure 10-1 illustrates the integer unit pipeline flow for the simple code sequence listed. The three instructions in the code sequence require only a single clock in each pipeline stage. The TRAPF instructions are also single-clock instructions that function as nonsynchronizing NOPs.

LABEL	INSTRUCTION	<ea> CALCULATE	EXECUTE
P1	TRAPF	1	1
A	MOVE.L $1000,D0	1	1
B	ADDQ.L #1,D0	1	1
C	MOVE.L D0,$1000	1	1
N1	TRAPF	1	1
N2	TRAPF	1	1

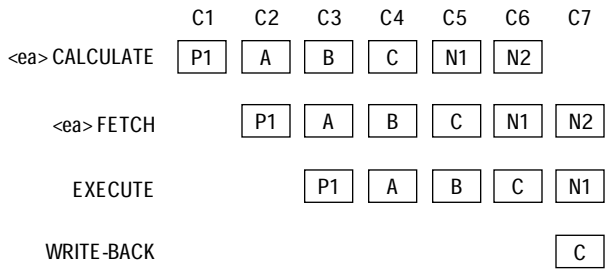


Figure 10-1. Simple Instruction Timing Example

C1 The previous instruction (P1) finishes in the <ea> calculate.

C2 MOVE.L (A) starts in the <ea> calculate and requests an immediate extension word for its effective address.

C3 MOVE.L (A) starts in the <ea> fetch, which fetches the operand at $1000. ADDQ.L (B) starts in the <ea> calculate stage with the operand encoded in the instruction.

C4 MOVE.L (A) executes in the execute stage, storing the fetched operand in register D0. ADDQ.L (B) starts in the <ea> fetch with no operation performed. MOVE.L (C) starts in the <ea> calculate requesting an immediate extension word for its effective address.

C5 ADDQ.L (B) executes in the execution stage, incrementing D0 by 1. MOVE.L (C) passes through the <ea> fetch with no operation performed. The next instruction starts in the <ea> calculate stage.

C6 MOVE.L (C) executes in the execution stage generating a write of D0 to the effective address.

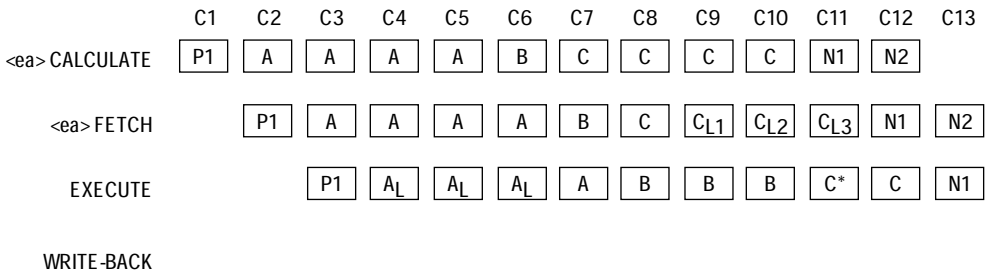
C7 The write to memory by MOVE.L (C) occurs to the data memory unit if it is not busy. If the second TRAPF instruction (N2) in the <ea> fetch stage requires an operand fetch, the write-back for MOVE.L (C) stalls in the write-back stage since it is a lower priority.

The separation of calculation and execution in the <ea> calculate and execute stages allows instruction reordering during compile time to take advantage of potential instruction overlap. Figure 10-2 illustrates this overlap for an instruction requiring multiple clocks in the execute stage and with an instruction with a long lead time. The execution time for LEA (3_L + 1) indicates that the instruction can be stalled three clocks without affecting execution.

When the LEA (A) instruction precedes the ABCD (B) instruction, the execution stalls during C4–C6 (equivalent to the LEA lead time) while the instruction completes in the <ea> calculate and <ea> fetch stages. The resulting execution time for the LEA (A) and ABCD (B) sequence is eight clocks.

However, if the LEA (C) instruction follows the ABCD (B) instruction, the LEA stalls in the <ea> fetch instead, during C9–C11. The LEA then executes in a single clock in the execution stage. The resulting execution time for the LEA (C) and ABCD (B) sequence is five clocks.

LABEL	INSTRUCTION		<ea> CALCULATE	EXECUTE
P1	TRAPF		1	1
A	LEA	$24(PC),A1	4	$3_L + 1$
B	ABCD	D0,D1	1	3
C	LEA	$24(PC),A1	4	$3_L + 1$
N1	TRAPF		1	1
N2	TRAPF		1	1

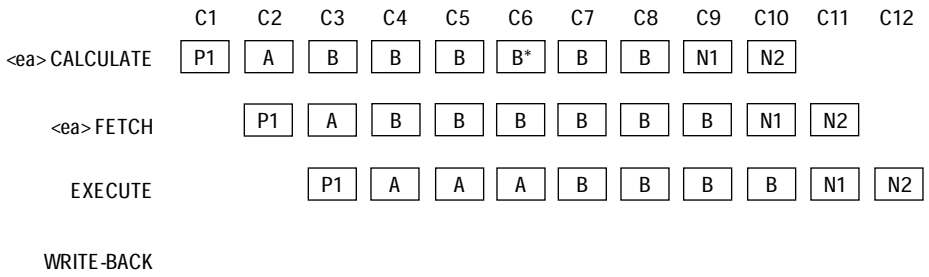


NOTE: *Possible stalls in this stage.

Figure 10-2. Instruction Overlap with Multiple Clocks

Instructions using the brief and full extension word format addressing modes cause the <ea> calculate and execute stages to operate in an interlocked manner. When these instructions wait to begin execution in the execution stage, there is a similar increase in the <ea> calculate time. Figure 10-3 illustrates this effect for an ADD instruction using a brief format extension word. The ADD instruction stalls for two clocks waiting to enter the execution stage. Since this time exceeds by one clock the ADD lead time, the ADD instruction remains in the <ea> calculate stage for one additional clock. If the ADD instruction was in the execution stage for two clocks, the ABCD instruction would not have stalled in the <ea> calculate stage.

LABEL	INSTRUCTION		<ea> CALCULATE	EXECUTE
P1	TRAPF		1	1
A	ABCD	D0,D1	3	3
B	ADD.L	4(A0,D3),D2	5	$1_L + 4$
N1	TRAPF		1	1
N2	TRAPF		1	1



NOTE: *Possible stalls in this stage.

Figure 10-3. Interlocked Stages

10.3 CINV AND CPUSH INSTRUCTION TIMING

The following details the execution time for the CINV and CPUSH instructions used to perform maintenance of the instruction and data caches. These two instructions sample interrupt request ($\overline{IPLx}$) signals on every clock instead of at instruction boundaries. While performing the actual cache invalidate operation, the execution unit stalls to allow previous write-backs and any pending instruction prefetches to complete. The total time required to execute a cache invalidate instruction is dependent on the previous instruction stream. Execution time for this instruction is independent of the selected cache combination. The CINV instructions interlock operation of the <ea> calculate and execution stages to prevent a previous instruction from accessing the caches until the invalidate operation is complete. Idle refers to the number of clocks required for all pending writes and instruction prefetches to complete. Table 10-3 list the CINV timings.

Table 10-3. CINV Timing

Instruction	Execution Time
CINVL	9 + Idle
CINVP	266 + Idle
CINVA	9 + Idle

Execution time for the CPUSH instruction is dependent on several factors, such as the number of dirty cache lines and the size of the resulting push (either long-word or line); the overlapping operations within the data cache and the bus controller; the distribution of dirty cache lines; and the number of wait states in the push access on the bus. The interaction of these factors determines the total time required to execute a CPUSH instruction.

Since the distribution of dirty data within the cache is entirely dependent on the nature of the user's code, it is impossible to provide an equation for execution time that works for all code sequences. Table 10-4 provides baseline information indicating best and worst case execution times for the three CPUSH instruction variants. Best case corresponds to a cache containing no dirty entries, while the worst case corresponds to all lines dirty and requiring line pushes. In Table 10-4, line refers to the number of clocks required in the user's system for a line transfer. Idle refers to the number of clocks required for all pending writes and instruction prefetches to complete.

Table 10-4. CPUSH Best and Worst Case Timing

Instruction	Execution Time	
	Best Case	Worst Case
CPUSHL	6	6 + Line + Idle
CPUSHP CPUSHA	267	11 + 256 × Line + Idle

10.4 MOVE INSTRUCTION TIMING

SOURCE	DESTINATION					
	Dn		(An)		(An)+	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1	1	1	1	1
(An)	1	1	1	1	2	$1_L + 1$
(An)+	1	1	2	$1_L + 1$	2	$1_L + 1$
–(An)	1	1	2	$1_L + 1$	2	$1_L + 1$
(d_{16},An)	1	1	2	$1_L + 1$	2	$1_L + 1$
(d_{16},PC)	3	$2_L + 1$	3	$2_L + 1$	3	$2_L + 1$
(xxx).W, (xxx).L	1	1	1	1	2	$1_L + 1$
#<xxx>	1	1	1	1	2	$1_L + 1$
(d_8,An,Xn)	3	3	4	4	5	5
(d_8,PC,Xn)	5	$1_L + 4$	5	$1_L + 4$	6	$1_L + 5$
(b_{16},BR,Xn)	7	$1_L + 6$	7	$1_L + 6$	8	$1_L + 7$
([bd,BR,Xn])	10	$1_L + 9$	10	$1_L + 9$	11	$1_L + 10$
([bd,BR,Xn],od)	11	$1_L + 10$	11	$1_L + 10$	12	$1_L + 11$
([bd,BR],Xn)	11	$3_L + 8$	11	$3_L + 8$	12	$3_L + 9$
([bd,BR],Xn,od)	12	$3_L + 9$	12	$3_L + 9$	13	$3_L + 10$
	–(An)		(d_{16},An)		(xxx).W, (xxx).L	
Dn	1	1	1	1	1	1
(An)	2	$1_L + 1$	2	$1_L + 1$	1	1
(An)+	2	$1_L + 1$	2	$1_L + 1$	2	$1_L + 1$
–(An)	2	$1_L + 1$	2	$1_L + 1$	2	$1_L + 1$
(d_{16},An)	2	$1_L + 1$	2	$1_L + 1$	2	$1_L + 1$
(d_{16},PC)	3	$2_L + 1$	4	$3_L + 1$	4	$3_L + 1$
(xxx).W, (xxx).L	2	$1_L + 1$	2	$1_L + 1$	2	$1_L + 1$
#<xxx>	2	$1_L + 1$	2	$1_L + 1$	2	$1_L + 1$
(d_8,An,Xn)	5	5	5	5	5	5
(d_8,PC,Xn)	6	$1_L + 5$	6	$1_L + 5$	6	$1_L + 5$
(b_{16},BR,Xn)	8	$1_L + 7$	8	$1_L + 7$	8	$1_L + 7$
([bd,BR,Xn])	11	$1_L + 10$	11	$1_L + 10$	11	$1_L + 10$
([bd,BR,Xn],od)	12	$1_L + 11$	12	$1_L + 11$	12	$1_L + 11$
([bd,BR],Xn)	12	$3_L + 9$	12	$3_L + 9$	12	$3_L + 9$
([bd,BR],Xn,od)	13	$3_L + 10$	13	$3_L + 10$	13	$3_L + 10$

10.4 MOVE INSTRUCTION TIMING (Continued)

SOURCE	DESTINATION					
	(d_8,An,Xn)		(b_{16},An,Xn)		([bd,An,Xn])	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	3	3	7	$1_L + 6$	10	$1_L + 9$
(An)	4	4	7	$1_L + 6$	10	$1_L + 9$
(An)+	4	4	7	$1_L + 6$	10	$1_L + 9$
–(An)	4	4	7	$1_L + 6$	10	$1_L + 9$
(d_{16},An)	4	4	7	$1_L + 6$	10	$1_L + 9$
(d_{16},PC)	8	$4_L + 4$	10	$4_L + 6$	13	$4_L + 9$
(xxx).W, (xxx).L	4	4	7	$1_L + 6$	10	$1_L + 9$
#<xxx>	3	3	7	$1_L + 6$	10	$1_L + 9$
(d_8,An,Xn)	8	8	10	10	13	13
(d_8,PC,Xn)	9	$1_L + 8$	11	$1_L + 10$	14	$1_L + 13$
(b_{16},BR,Xn)	11	$1_L + 10$	13	$1_L + 12$	16	$1_L + 15$
([bd,BR,Xn])	14	$1_L + 13$	16	$1_L + 15$	19	$1_L + 18$
([bd,BR,Xn],od)	15	$1_L + 14$	17	$1_L + 16$	20	$1_L + 19$
([bd,BR],Xn)	15	$3_L + 12$	17	$3_L + 14$	20	$3_L + 17$
([bd,BR],Xn,od)	16	$3_L + 13$	18	$3_L + 15$	21	$3_L + 18$
	([bd,An,Xn],od)		([bd,An],Xn)		([bd,An],Xn,od)	
Dn	11	$1_L + 10$	11	$3_L + 8$	12	$3_L + 9$
(An)	11	$1_L + 10$	11	$3_L + 8$	12	$3_L + 9$
(An)+	11	$1_L + 10$	11	$3_L + 8$	12	$3_L + 9$
–(An)	11	$1_L + 10$	11	$3_L + 8$	12	$3_L + 9$
(d_{16},An)	11	$1_L + 10$	11	$3_L + 8$	12	$3_L + 9$
(d_{16},PC)	14	$4_L + 10$	14	$6_L + 8$	15	$6_L + 9$
(xxx).W, (xxx).L	11	$1_L + 10$	11	$3_L + 8$	12	$3_L + 9$
#<xxx>	11	$1_L + 10$	11	$3_L + 8$	12	$3_L + 9$
(d_8,An,Xn)	14	14	14	14	15	15
(d_8,PC,Xn)	15	$1_L + 14$	15	$1_L + 14$	16	$1_L + 15$
(b_{16},BR,Xn)	17	$1_L + 16$	17	$1_L + 16$	18	$1_L + 17$
([bd,BR,Xn])	20	$1_L + 19$	20	$1_L + 19$	21	$1_L + 20$
([bd,BR,Xn],od)	21	$1_L + 20$	21	$1_L + 20$	22	$1_L + 21$
([bd,BR],Xn)	21	$3_L + 18$	21	$3_L + 18$	22	$3_L + 19$
([bd,BR],Xn,od)	22	$3_L + 19$	22	$3_L + 19$	23	$3_L + 20$

10.5 MISCELLANEOUS INTEGER UNIT INSTRUCTION TIMINGS

Instruction	Condition	<ea> Calculate	Execute
ABCD	Dy,Dx –(Ay),–(Ax)	1 3	3 $1_L + 3$
ADDX	Dy,Dx –(Ay),–(Ax)	1 3	1 $1_L + 2$
ANDI #<xxx>,CCR	—	1	4
ANDI #<xxx>,SR[a]	—	9	$1_L + 8$
Bcc	Branch Taken Branch Not Taken	2 3	2 3
BRA	Branch Taken Branch Not Taken	2 3	2 3
BSR <offset>	—	2	$1_L + 1$
CAS2[b]	True False	56 51	$6_L + 49$ $6_L + 44$
CMPM	—	3	$1_L + 2$
DBcc[c]	False, Count > –1 False, Count = –1 True	3 4 4	3 4 4
EORI #<xxx>,CCR	—	1	4
EORI #<xxx>,SR[a]	—	9	$1_L + 8$
EXG	Dy,Dx Ay,Ax Dy,Ax	1 2 1	1 $1_L + 1$ 1
EXT	Word Long Word	1 1	2 1
EXTB	Long Word	1	1
ILLEGAL[a]	A-Line Unimplemented F-Line Unimplemented	16 16	16 16
LINK	—	3	$2_L + 1$
MOVE USP	USP,An An,USP[a]	3 7	$2_L + 1$ $1_L + 6$
MOVE16[c,d]	(Ax)+,(Ay)+ xxx.L,(An) xxx.L,(An)+ (An),xxx.L (An)+,xxx.L	6 4 5 4 4	$1_L + 7$ 7 8 7 7
MOVEC[b]	Rn,Rc Rc,Rn	7 11	$1_L + 6$ $1_L + 10$
MOVEP[c]	MOVEP.W Dn,d16(An) MOVEP.L Dn,d16(An) MOVEP.W d16(An),Dn MOVEP.L d16(An),Dn	11 13 4 8	$2_L + 9$ $2_L + 11$ $2_L + 5$ $2_L + 8$
MOVEQ	—	1	1
NOP[a]	—	8	$1_L + 7$

10.5 MISCELLANEOUS INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Instruction	Condition	<ea> Calculate	Execute
ORI #<xxx>,CCR	—	1	4
ORI #<xxx>,SR[a]	—	9	$1_L + 8$
PACK	Dx,Dy,#<xxx> –(Ay),–(Ax),#<xxx>	1 3	3 $2_L + 3$
PFLUSH[b]	—	11	$1_L + 10$
PFLUSHA[b]	—	11	$1_L + 10$
PFLUSHAN[b]	—	27	$1_L + 26$
PFLUSHN (An)[b]	—	11	$1_L + 10$
PTESTR, PTESTW[e]	—	25	$11_L + 14$
RESET[a]	—	521	521
RTD[c]	—	6	$1_L + 5$
RTE[a]	Stack Format $0 Stack Format $1 Stack Format $2 Stack Format $3 Stack Format $4 Stack Format $7	2 4 2 3 2 4	13 23 14 20 15 23
RTR[c]	—	7	$1_L + 6$
RTS[c]	—	5	5
SBCD	Dy,Dx –(Ay),–(Ax)	1 3	3 $1_L + 3$
SUBX	Dy,Dx –(Ay),–(Ax)	1 3	1 $1_L + 2$
SWAP	—	1	2
TRAP#[a]	—	16	16
TRAPcc[f]	Taken Not Taken	19 5	19 5
TRAPV[f]	Taken Not Taken	19 5	19 5
UNLK	—	2	$1_L + 1$
UNPK	Dx,Dy,# –(Ay),–(Ax),#	1 3	4 $2_L + 4$

NOTES:

a. Times listed are minimum. This instruction interlocks the <ea> calculate and execute stages and synchronizes some portions of the processor before execution.
b. Times listed are typical. This instruction interlocks the <ea> calculate and execute stages and synchronizes some portions of the processor before execution.
c. This instruction interlocks the <ea> calculate and execute stages.
d. Successive in-line MOVE16 instructions each add eight clocks to the <ea> calculate and execute times.
e. Typical measurement for three-level table search with no descriptor writes, no entries cached, and four-clock memory access times.
f. This instruction interlocks the <ea> calculate and execute stages. For the exception taken, this instruction also synchronizes some portions of the processor before execution; times listed are minimum in this case.

10.6 INTEGER UNIT INSTRUCTION TIMINGS

Addressing Mode	ADD, AND, EOR, OR, SUB, TST		ADDA		ADDI, ANDI, EORI, ORI, SUBI	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1	1	2	1	1
An	1	1	1	1	—	—
(An)	1	1	1	2	1	1
(An)+	1	1	2	$1_L + 2$	2	$1_L + 1$
–(An)	1	1	2	$1_L + 2$	2	$1_L + 1$
(d_{16},An)	1	1	2	$1_L + 2$	2	$1_L + 1$
(d_{16},PC)	3	$2_L + 1$	3	$2_L + 2$	—	—
(xxx).W, (xxx).L	1	1	1	2	2	$1_L + 1$
#<xxx>	1	1	1	1	—	—
(d_8,An,Xn)	3	3	4	5	3	3
(d_8,PC,Xn)	5	$1_L + 4$	5	$1_L + 5$	—	—
(BR,Xn)	6	$1_L + 5$	6	$1_L + 6$	7	$1_L + 6$
(bd,BR,Xn)	7	$1_L + 6$	7	$1_L + 7$	8	$1_L + 7$
([bd,BR,Xn])	10	$1_L + 9$	10	$1_L + 10$	10	$1_L + 10$
([bd,BR,Xn],od)	11	$1_L + 11$	11	$1_L + 12$	11	$1_L + 11$
([bd,BR],Xn)	11	$3_L + 8$	11	$3_L + 9$	11	$3_L + 9$
([bd,BR],Xn,od)	12	$3_L + 10$	12	$3_L + 11$	12	$3_L + 10$

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	ADDQ, SUBQ		ASL		ASR, LSL, LSR	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1	1	3/4*	1	2/3*
An	1	1	—	—	—	—
(An)	1	1	1	3	1	2
(An)+	2	1_L + 1	1	3	1	2
–(An)	2	1_L + 1	1	3	1	2
(d_{16},An)	2	1_L + 1	1	3	1	2
(d_{16},PC)	—	—	—	—	—	—
(xxx).W, (xxx).L	1	1	1	3	1	2
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	3	3	3	5	3	4
(d_8,PC,Xn)	—	—	—	—	—	—
(BR,Xn)	7	1_L + 6	7	1_L + 8	7	1_L + 7
(bd,BR,Xn)	8	1_L + 7	8	1_L + 9	8	1_L + 8
([bd,BR,Xn])	10	1_L + 9	10	1_L + 11	10	1_L + 10
([bd,BR,Xn],od)	11	1_L + 11	11	1_L + 12	11	1_L + 11
([bd,BR],Xn)	11	3_L + 8	11	3_L + 10	11	3_L + 9
([bd,BR],Xn,od)	12	3_L + 10	12	3_L + 11	12	3_L + 10

*Immediate count specified for shift count/shift count specified in register, respectively.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	BCHG, BCLR, BSET[a]		BFCHG, BFCLR, BFSET[b,c]		BFEXTS, BFEXTU[b,d]	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	3/4	3/4[e]	6/7[e]	1/2[e]	4/5[e]
An	—	—	—	—	—	—
(An)	1	3/4	9	2_L + 8	9	2_L + 7
(An)+	1	3/4	—	—	—	—
–(An)	1	3/4	—	—	—	—
(d_{16},An)	2/1	1_L + 3/4	9	2_L + 8	9	2_L + 7
(d_{16},PC)	—	—	—	—	10	3_L + 7
(xxx).W, (xxx).L	2/1	1_L + 3/4	9	2_L + 8	9	2_L + 7
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	3	5/6	10	11	10	10
(d_8,PC,Xn)	—	—	—	—	11	1_L + 10
(BR,Xn)	7	1_L + 8/1_L + 9	13	1_L + 13	13	1_L + 12
(bd,BR,Xn)	8	1_L + 9/1_L + 10	14	1_L + 14	14	1_L + 13
([bd,BR,Xn])	10	1_L + 11/1_L + 12	16	1_L + 16	16	1_L + 15
([bd,BR,Xn],od)	11	1_L + 12/1_L + 13	17	1_L + 17	17	1_L + 16
([bd,BR],Xn)	11	3_L + 10/3_L + 11	17	3_L + 15	17	3_L + 14
([bd,BR],Xn,od)	12	3_L + 11/3_L + 12	18	3_L + 16	18	3_L + 15

NOTES:
a. Bit instruction <ea> calculate and execute times T1/T2 apply to #<xxx>/Dn bit numbers.
b. This instruction interlocks the <ea> calculate and execute stages.
c. If the bit field spans a long-word boundary, add ten and nine clocks to the <ea> calculate and execute times, respectively. Two memory addresses are accessed in this case.
d. If the bit field spans a long-word boundary, add two clocks to the execute time. Two memory addresses are accessed in this case.
e. Immediate count specified for both width and offset and width and/or offset specified in register, respectively.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	BFFFO[a,b]		BFINS[a,c]		BFTST[a]	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	3/4[d]	6/7[d]	2/3[d]	5/6[d]	1/2[d]	3/4[d]
An	—	—	—	—	—	—
(An)	9	2_L + 9	9	2_L + 7	9	2_L + 7
(An)+	—	—	—	—	—	—
–(An)	—	—	—	—	—	—
(d_{16},An)	9	2_L + 9	9	2_L + 7	9	2_L + 7
(d_{16},PC)	10	3_L + 9	—	—	10	3_L + 7
(xxx).W, (xxx).L	9	2_L + 9	9	2_L + 7	9	2_L + 7
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	10	12	10	10	10	10
(d_8,PC,Xn)	11	1_L + 12	—	—	11	1_L + 10
(BR,Xn)	13	1_L + 14	13	1_L + 12	13	1_L + 12
(bd,BR,Xn)	14	1_L + 15	14	1_L + 13	14	1_L + 13
([bd,BR,Xn])	16	1_L + 17	16	1_L + 15	16	1_L + 15
([bd,BR,Xn],od)	17	1_L + 18	17	1_L + 16	17	1_L + 16
([bd,BR],Xn)	17	3_L + 16	17	3_L + 14	17	3_L + 14
([bd,BR],Xn,od)	18	3_L + 17	18	3_L + 15	18	3_L + 15

NOTES:

a. This instruction interlocks the <ea> calculate and execute stages.
b. If the bit field spans a long-word boundary, add two clocks to the execute time. Two memory addresses are accessed in this case.
c. If the bit field spans a long-word boundary, add seven clocks to both the <ea> calculate and execute times. Two memory addresses are accessed in this case.
d. If the bit field spans a long-word boundary, add ten and nine clocks to both the <ea> calculate and execute times, respectively. Two memory addresses are accessed in this case.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	BTST		CAS[b]		CHK[c,d] (<ea>, Dn)	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1/2[a]	—	—	8	$1_L + 7$
An	—	—	—	—	—	—
(An)	1	1/2	36	$6_L + 31$	9	$2_L + 7$
(An)+	1	1/2	37	$5_L + 31$	9	$2_L + 7$
–(An)	1	1/2	37	$5_L + 31$	9	$2_L + 7$
(d_{16},An)	2/1	$1_L + 1/2$	37	$5_L + 31$	9	$2_L + 7$
(d_{16},PC)	3	$2_L + 1/2_L + 2$	—	—	10	$3_L + 7$
(xxx).W, (xxx).L	2/1	$1_L + 1/2$	36	$5_L + 31$	9	$2_L + 7$
#<xxx>	—	—	—	—	8	$1_L + 7$
(d_8,An,Xn)	3	3/4	36	36	10	10
(d_8,PC,Xn)	5	$1_L + 4/1_L + 5$	—	—	11	$1_L + 10$
(BR,Xn)	7/6	$1_L + 6/1_L + 7$	36	$1_L + 35$	12	$1_L + 11$
(bd,BR,Xn)	8/7	$1_L + 7/1_L + 8$	37	$1_L + 36$	13	$1_L + 12$
([bd,BR,Xn])	10/9	$1_L + 9/1_L + 10$	42	40	16	$1_L + 15$
([bd,BR,Xn],od)	11/10	$1_L + 10/1_L + 11$	42	$1_L + 41$	17	$1_L + 16$
([bd,BR],Xn)	11/10	$3_L + 8/3_L + 9$	42	$3_L + 38$	17	$3_L + 14$
([bd,BR],Xn,od)	12/11	$3_L + 9/3_L + 10$	42	$3_L + 39$	18	$3_L + 15$

NOTES:
a. Bit instruction <ea> calculate and execute times T1/T2 apply to #<xxx>/Dn bit numbers.
b. Times listed are typical. This instruction interlocks the <ea> calculate and execute stages and synchronizes some portions of the processor before execution.
c. This instruction interlocks the <ea> calculate and execute stages.
d. Times listed are for Dn within bounds. This instruction interlocks the <ea> calculate and execute stages.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	CHK2* (<ea>, Rn)		CLR		CMP	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	—	—	1	1	1	1
An	—	—	—	—	1	1
(An)	11	2_L + 9	1	1	1	1
(An)+	—	—	1	1	1	1
–(An)	—	—	1	1	1	1
(d_{16},An)	11	2_L + 9	1	1	1	1
(d_{16},PC)	12	3_L + 9	—	—	3	2_L + 1
(xxx).W, (xxx).L	11	2_L + 9	1	1	1	1
#<xxx>	—	—	—	—	1	1
(d_8,An,Xn)	13	1_L + 12	3	3	3	3
(d_8,PC,Xn)	14	2_L + 12	—	—	5	1_L + 4
(BR,Xn)	15	2_L + 13	6	1_L + 5	6	1_L + 5
(bd,BR,Xn)	16	2_L + 14	7	1_L + 6	7	1_L + 6
([bd,BR,Xn])	19	2_L + 17	9	1_L + 8	9	1_L + 8
([bd,BR,Xn],od)	20	2_L + 18	10	1_L + 9	10	1_L + 9
([bd,BR],Xn)	20	4_L + 16	10	3_L + 7	10	3_L + 7
([bd,BR],Xn,od)	21	4_L + 17	11	3_L + 8	11	3_L + 8

*This instruction interlocks the <ea> calculate and execute stages. Timing for Dn within bounds, UB > LB. For UB < LB, add three clocks to <ea> calculate and execute times. For Rn = An, add one clock to <ea> calculate and execute times.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	CMPA.L		CMPI		CMP2*	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1	1	1	—	—
An	1	1	—	—	—	—
(An)	1	1	1	1	13	$2_L + 11$
(An)+	2	$1_L + 1$	2	$1_L + 1$	0	0
–(An)	2	$1_L + 1$	2	$1_L + 1$	0	0
(d_{16},An)	2	$1_L + 1$	2	$1_L + 1$	13	$2_L + 11$
(d_{16},PC)	3	$2_L + 1$	3	$2_L + 1$	14	$3_L + 11$
(xxx).W, (xxx).L	1	1	2	$1_L + 1$	13	$2_L + 11$
#<xxx>	1	1	—	—	—	—
(d_8,An,Xn)	3	3	3	3	15	$1_L + 14$
(d_8,PC,Xn)	5	$1_L + 4$	5	$2_L + 4$	16	$2_L + 14$
(BR,Xn)	6	$1_L + 5$	6	$2_L + 5$	17	$2_L + 15$
(bd,BR,Xn)	7	$1_L + 6$	7	$2_L + 6$	18	$2_L + 16$
([bd,BR,Xn])	9	$1_L + 8$	9	$2_L + 8$	21	$2_L + 19$
([bd,BR,Xn],od)	10	$1_L + 9$	10	$2_L + 9$	22	$2_L + 20$
([bd,BR],Xn)	10	$3_L + 7$	10	$4_L + 7$	22	$4_L + 18$
([bd,BR],Xn,od)	11	$3_L + 8$	11	$4_L + 8$	23	$4_L + 19$

*Times listed are typical.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	DIVS.W, DIVU.W*		DIVS.L, DIVU.L, DIVSL.L, DIVUL.L*		JMP	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	8	27	9	44	—	—
An	—	—	—	—	—	—
(An)	8	27	9	44	3	$2_L + 1$
(An)+	8	27	9	44	—	—
–(An)	8	27	9	44	—	—
(d_{16},An)	8	27	11	$2_L + 44$	4	$3_L + 1$
(d_{16},PC)	11	$3_L + 27$	12	$3_L + 44$	6	$5_L + 1$
(xxx).W, (xxx).L	8	27	11	$2_L + 44$	3	$2_L + 1$
#<xxx>	8	27	10	$1_L + 44$	—	—
(d_8,An,Xn)	11	30	12	47	6	6
(d_8,PC,Xn)	12	$1_L + 30$	13	$1_L + 47$	7	$1_L + 6$
(BR,Xn)	13	$1_L + 31$	14	$1_L + 48$	8	$1_L + 7$
(bd,BR,Xn)	14	$1_L + 32$	15	$1_L + 49$	9	$1_L + 8$
([bd,BR,Xn])	17	$1_L + 35$	18	$1_L + 52$	12	$1_L + 11$
([bd,BR,Xn],od)	18	$1_L + 36$	19	$1_L + 53$	12	$1_L + 11$
([bd,BR],Xn)	18	$3_L + 34$	19	$3_L + 51$	13	$3_L + 10$
([bd,BR],Xn,od)	19	$3_L + 35$	20	$3_L + 52$	14	$3_L + 11$

*This instruction interlocks the <ea> calculate and execute stages. Execution time for a DIV/0 exception taken and exception processing is approximately 16 + <ea> calculate clocks. For example, DIV.W #0,Dn takes approximately 24 clocks in both the <ea> calculate and execute times to execute the divide instruction, perform exception stacking, fetch the exception vector, and prefetch the next instruction.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	JSR		LEA		MOVE from CCR	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	—	—	—	—	1	2
An	—	—	—	—	—	—
(An)	3	$2_L + 1$	1	1	1	2
(An)+	—	—	—	—	1	2
–(An)	—	—	—	—	1	2
(d_{16},An)	4	$3_L + 1$	2	$1_L + 1$	1	2
(d_{16},PC)	6	$5_L + 1$	4	$3_L + 1$	—	—
(xxx).W, (xxx).L	3	$2_L + 1$	1	1	1	2
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	6	6	4	4	3	4
(d_8,PC,Xn)	7	$1_L + 6$	5	$1_L + 4$	—	—
(BR,Xn)	8	$1_L + 7$	6	$1_L + 5$	6	$1_L + 6$
(bd,BR,Xn)	9	$1_L + 8$	7	$1_L + 6$	7	$1_L + 7$
([bd,BR,Xn])	12	$1_L + 11$	9	$1_L + 8$	10	$1_L + 10$
([bd,BR,Xn],od)	13	$1_L + 12$	10	$1_L + 9$	11	$1_L + 11$
([bd,BR],Xn)	13	$3_L + 10$	10	$3_L + 7$	11	$3_L + 9$
([bd,BR],Xn,od)	14	$3_L + 11$	11	$3_L + 8$	12	$3_L + 10$

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	MOVE to CCR		MOVE from SR[a]		MOVE to SR[b]	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	2	2	$1_L + 2$	9	$1_L + 8$
An	—	—	—	—	—	—
(An)	1	2	2	$1_L + 2$	10	$2_L + 8$
(An)+	1	2	2	$1_L + 2$	10	$2_L + 8$
–(An)	1	2	2	$1_L + 2$	10	$2_L + 8$
(d_{16},An)	1	2	2	$1_L + 2$	10	$2_L + 8$
(d_{16},PC)	3	$2_L + 2$	—	—	11	$3_L + 8$
(xxx).W, (xxx).L	1	2	2	$1_L + 2$	10	$2_L + 8$
#<xxx>	1	2	—	—	9	$1_L + 8$
(d_8,An,Xn)	3	4	4	5	11	11
(d_8,PC,Xn)	4	$1_L + 4$	—	—	12	$1_L + 11$
(BR,Xn)	6	$1_L + 6$	6	$1_L + 6$	—	—
(bd,BR,Xn)	7	$1_L + 7$	7	$1_L + 7$	14	$1_L + 13$
([bd,BR,Xn])	10	$1_L + 10$	10	$1_L + 10$	17	$1_L + 16$
([bd,BR,Xn],od)	11	$1_L + 11$	11	$1_L + 11$	18	$1_L + 17$
([bd,BR],Xn)	11	$3_L + 9$	11	$3_L + 9$	18	$3_L + 15$
([bd,BR],Xn,od)	12	$3_L + 10$	12	$3_L + 10$	19	$3_L + 16$

NOTES:
a. This instruction interlocks the <ea> calculate and execute stages.
b. Times listed are minimum. This instruction interlocks the <ea> calculate and execute stages and synchronizes some portions of the processor before execution.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	MOVEA.L[a]		MOVEM <list>,<ea>[b,c]		MOVEM.L <ea>,<list>[b,c]	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1	—	—	—	—
An	1	1	—	—	—	—
(An)	1	1	2 + D' + A'	1_L + 1 + D' + A'	3 + D' + A	1_L + 2 + D' + A'
(An)+	1	1	—	—	3 + D' + A	1_L + 2 + D' + A'
–(An)	1	1	2 + D' + A'	1_L + 1 + D' + A'	—	—
(d_{16},An)	1	1	2 + D' + A'	1_L + 1 + D' + A'	3 + D' + A	1_L + 2 + D' + A'
(d_{16},PC)	3	2_L + 1	—	—	4 + D' + A	2_L + 2 + D' + A'
(xxx).W, (xxx).L	1	1	2 + D' + A'	1_L + 1 + D' + A'	3 + D' + A	1_L + 2 + D' + A'
#<xxx>	1	1	—	—	—	—
(d_8,An,Xn)	4	4	9 + D' + A'	2_L + 7 + D' + A'	10 + D' + A	2_L + 8 + D' + A'
(d_8,PC,Xn)	5	1_L + 4	—	—	11 + D' + A	3_L + 8 + D' + A'
(BR,Xn)	6	1_L + 5	11 + D' + A'	3_L + 8 + D' + A'	12 + D' + A	3_L + 9 + D' + A'
(bd,BR,Xn)	7	1_L + 6	12 + D' + A'	3_L + 9 + D' + A'	13 + D' + A	3_L + 10 + D' + A'
([bd,BR,Xn])	10	1_L + 9	15 + D' + A'	3_L + 12 + D' + A'	16 + D' + A	3_L + 13 + D' + A'
([bd,BR,Xn],od)	11	1_L + 10	16 + D' + A'	3_L + 13 + D' + A'	17 + D' + A	3_L + 14 + D' + A'
([bd,BR],Xn)	11	3_L + 8	16 + D' + A'	5_L + 11 + D' + A'	17 + D' + A	5_L + 12 + D' + A'
([bd,BR],Xn,od)	12	3_L + 9	17 + D' + A'	5_L + 12 + D' + A'	18 + D' + A	5_L + 13 + D' + A'

NOTES:

a. Except for Dn and #<xxx> cases, add one clock to execute times for MOEA.W.
b. This instruction interlocks the <ea> calculate and execute stages.
c. D' and A' indicate the number of data and address registers, respectively (if no data registers specified the number one). For MOVEM.W <ea>,<list>, add N – 2 and N clocks to <ea> calculate and execute times, respectively, for N address registers specified.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	MOVES <ea>,An*		MOVES <ea>,Dn*		MOVES Rn,<ea>*	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	—	—	—	—	—	—
An	—	—	—	—	—	—
(An)	28	$4_L + 24$	20	$4_L + 19$	13	$4_L + 9$
(An)+	28	$4_L + 24$	20	$4_L + 19$	13	$4_L + 9$
–(An)	17	$2_L + 15$	11	12	11	$2_L + 9$
(d_{16},An)	29	$4_L + 24$	21	$4_L + 19$	14	$4_L + 9$
(d_{16},PC)	—	—	—	—	—	—
(xxx).W, (xxx).L	17	$2_L + 15$	11	$4_L + 10$	11	$2_L + 9$
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	29	$1_L + 27$	21	$1_L + 22$	14	$1_L + 12$
(d_8,PC,Xn)	—	—	—	—	—	—
(BR,Xn)	21	$2_L + 19$	15	$2_L + 14$	15	$2_L + 13$
(bd,BR,Xn)	22	$2_L + 20$	16	$2_L + 15$	16	$2_L + 14$
([bd,BR,Xn])	35	$2_L + 32$	26	$2_L + 27$	21	$2_L + 17$
([bd,BR,Xn],od)	31	$2_L + 29$	23	$2_L + 24$	20	$2_L + 18$
([bd,BR],Xn)	36	$4_L + 31$	27	$4_L + 26$	21	$4_L + 16$
([bd,BR],Xn,od)	32	$4_L + 28$	24	$4_L + 23$	21	$4_L + 17$

*Times listed are typical. This instruction interlocks the <ea> calculate and execute stages and synchronizes some portions of the processor before execution.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	MULS.W/L*		MULU.W/L*		NBCD	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	16/20	1	14/20	1	3
An	—	—	—	—	—	—
(An)	1	16/20	1	14/20	1	2
(An)+	1	16/20	1	14/20	1	2
–(An)	1	16/20	1	14/20	1	2
(d_{16},An)	1/2	16/20	1/2	14/20	1	2
(d_{16},PC)	3	2_L + 16/2_L + 20	3	14/20	—	—
(xxx).W, (xxx).L	1/2	16/20	1/2	14/20	1	2
#<xxx>	1	16/20	1	14/20	—	—
(d_8,An,Xn)	3	18/22	3	16/22	3	4
(d_8,PC,Xn)	5	1_L + 19/1_L + 23	5	1_L + 17/1_L + 23	—	—
(BR,Xn)	6	1_L + 20/1_L + 24	6	1_L + 18/1_L + 24	6	1_L + 6
(bd,BR,Xn)	7	1_L + 21/1_L + 25	7	1_L + 19/1_L + 25	7	1_L + 7
([bd,BR,Xn])	9	1_L + 23/1_L + 27	9	1_L + 21/1_L + 27	9	1_L + 9
([bd,BR,Xn],od)	10	1_L + 24/1_L + 28	10	1_L + 22/1_L + 28	10	1_L + 10
([bd,BR],Xn)	10	3_L + 22/3_L + 26	10	3_L + 20/3_L + 26	10	3_L + 8
([bd,BR],Xn,od)	11	3_L + 23/3_L + 27	11	3_L + 21/3_L + 27	11	3_L + 9

*Multiply <ea> calculate and execute times; T1/T2 apply to word/long-word operand size.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	NEG, NEGX, NOT		PEA		ROL, ROR	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	1	—	—	1	3/4*
An	—	—	—	—	—	—
(An)	1	1	2	$1_L + 1$	1	3
(An)+	1	1	—	—	1	3
–(An)	1	1	—	—	1	3
(d_{16},An)	1	1	2	$1_L + 1$	1	3
(d_{16},PC)	—	—	4	$3_L + 1$	—	—
(xxx).W, (xxx).L	1	1	2	$1_L + 1$	1	3
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	3	3	4	$1_L + 3$	3	5
(d_8,PC,Xn)	—	—	6	$2_L + 4$	—	—
(BR,Xn)	6	$1_L + 5$	7	$2_L + 5$	6	$1_L + 7$
(bd,BR,Xn)	7	$1_L + 6$	8	$2_L + 6$	7	$1_L + 8$
([bd,BR,Xn])	9	$1_L + 8$	10	$2_L + 8$	9	$1_L + 10$
([bd,BR,Xn],od)	10	$1_L + 9$	11	$2_L + 9$	10	$1_L + 11$
([bd,BR],Xn)	10	$3_L + 7$	11	$4_L + 7$	10	$3_L + 9$
([bd,BR],Xn,od)	11	$3_L + 8$	12	$4_L + 8$	11	$3_L + 10$

*Immediate count specified for shift count/shift count specified in register, respectively.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Continued)

Addressing Mode	ROXL, ROXR		Scc		SUBA	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	5/6*	1	2	1	1
An	—	—	—	—	1	2
(An)	1	2	1	2	1	2
(An)+	1	2	1	2	2	$1_L + 2$
–(An)	1	2	1	2	2	$1_L + 2$
(d_{16},An)	1	2	1	2	2	$1_L + 2$
(d_{16},PC)	—	—	—	—	3	$2_L + 2$
(xxx).W, (xxx).L	1	2	1	2	1	2
#<xxx>	—	—	—	—	1	2
(d_8,An,Xn)	3	4	4	5	4	5
(d_8,PC,Xn)	—	—	—	—	5	$1_L + 5$
(BR,Xn)	6	$1_L + 6$	6	$1_L + 6$	6	$1_L + 6$
(bd,BR,Xn)	7	$1_L + 7$	7	$1_L + 7$	7	$1_L + 7$
([bd,BR,Xn])	9	$1_L + 9$	10	$1_L + 10$	9	$1_L + 9$
([bd,BR,Xn],od)	10	$1_L + 10$	11	$1_L + 11$	10	$1_L + 10$
([bd,BR],Xn)	10	$3_L + 8$	11	$3_L + 9$	10	$3_L + 8$
([bd,BR],Xn,od)	11	$3_L + 9$	12	$3_L + 10$	11	$3_L + 9$

*Immediate count specified for shift count/shift count specified in register, respectively.

10.6 INTEGER UNIT INSTRUCTION TIMINGS (Concluded)

Addressing Mode	TAS*					
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	1	2				
An	—	—				
(An)	26	2_L + 24				
(An)+	26	2_L + 24				
–(An)	26	2_L + 24				
(d_{16},An)	26	2_L + 24				
(d_{16},PC)	—	—				
(xxx).W, (xxx).L	26	2_L + 24				
#<xxx>	—	—				
(d_8,An,Xn)	27	27				
(d_8,PC,Xn)	—	—				
(BR,Xn)	30	1_L + 28				
(bd,BR,Xn)	31	1_L + 29				
([bd,BR,Xn])	33	33				
([bd,BR,Xn],od)	35	34				
([bd,BR],Xn)	34	3_L + 31				
([bd,BR],Xn,od)	36	3_L + 32				

*Times listed are typical. This instruction interlocks the <ea> calculate and execute stages and synchronizes some portions of the processor before execution.

10.7 FLOATING-POINT UNIT INSTRUCTION TIMINGS

For floating-point instructions in the MC68040, the integer pipeline passes the decoded instruction to the floating-point unit for execution, then supports the floating-point unit by calculating effective addresses and transferring operands to and from this unit. For these instructions, the execution times listed in the integer unit timing section show the overhead required by the integer unit to support the floating-point unit, assuming the floating-point unit is not busy with the previous floating-point instructions.

Times in parentheses are the total time that that stage uses to execute an instruction even though the stage can pass data to the next stage early. The order of operands is generally not significant for timing purposes. Different rounding modes (i.e., round to zero, etc.) never incur a time penalty. Instructions with an S or D (e.g., FSADD) have the same effect as setting the rounding precision to S or D. All FMOVEM instructions wait for the pipe to idle before starting. Refer to **Section 9 Floating-Point Unit (MC68040 Only)** for details on the operation of the floating-point unit pipeline.

10.7.1 Miscellaneous Integer Unit Support Timings

Instruction	Condition	<ea> Calculate	Execute
FBcc	Taken Not Taken	7 6	7 6
FDBcc	cc True cc False	9 11	1_L + 7 1_L + 9
FNOP	FPU Idle	6	6
FTRAPcc	Not Taken	6	1_L + 5

10.7.2 Integer Unit Support Timings

Addressing Mode	FABS, FADD, FCMP, FDIV, FMOVE, FMUL, FNEG, FSQRT, FSUB, FTST <ea>,FPn*					
	Byte and Word		Long Word		Single Precision	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
FPn	—	—	—	—	—	—
Dn	2	$1_L + 2$	2	$1_L + 2$	2	$1_L + 2$
(An)	2	2	2	2	2	2
(An)+	2	2	2	2	2	2
–(An)	2	2	2	2	2	2
(d_{16},An)	2	2	2	2	2	2
(d_{16},PC)	4	$2_L + 2$	4	$2_L + 2$	4	$2_L + 2$
(xxx).W, (xxx).L	3	$1_L + 2$	3	$1_L + 2$	3	$1_L + 2$
#<xxx>	5	$3_L + 2$	3	$1_L + 2$	3	$1_L + 2$
(d_8,An,Xn)	5	5	5	5	5	5
(d_8,PC,Xn)	6	$1_L + 5$	6	$1_L + 5$	6	$1_L + 5$
(An,Xn)	7	$1_L + 6$	7	$1_L + 6$	7	$1_L + 6$
(bd,An,Xn)	8	$1_L + 7$	8	$1_L + 7$	8	$1_L + 7$
([bd,An,Xn])	11	$1_L + 10$	11	$1_L + 10$	11	$1_L + 10$
([bd,An,Xn],od)	12	$1_L + 11$	12	$1_L + 11$	12	$1_L + 11$
([bd,An],Xn)	12	$3_L + 9$	12	$3_L + 9$	12	$3_L + 9$
([bd,An],Xn,od)	13	$3_L + 10$	13	$3_L + 10$	13	$3_L + 10$
	Double Precision		Extended Precision			
FPn	—	—	2	$1_L + 2$		
Dn	—	—	—	—		
(An)	2	2	3	3		
(An)+	2	2	3	3		
–(An)	2	2	3	3		
(d_{16},An)	2	2	3	3		
(d_{16},PC)	4	$1_L + 3$	5	$1_L + 4$		
(xxx).W, (xxx).L	3	$1_L + 2$	4	$1_L + 3$		
#<xxx>	4	$2_L + 2$	5	$2_L + 3$		
(d_8,An,Xn)	5	5	6	6		
(d_8,PC,Xn)	6	6	7	7		
(An,Xn)	7	$1_L + 6$	8	$1_L + 7$		
(bd,An,Xn)	8	$1_L + 7$	9	$1_L + 8$		
([bd,An,Xn])	11	$1_L + 10$	12	$1_L + 11$		
([bd,An,Xn],od)	12	$1_L + 11$	13	$1_L + 12$		
([bd,An],Xn)	12	$3_L + 9$	13	$3_L + 10$		
([bd,An],Xn,od)	13	$3_L + 10$	14	$3_L + 11$		

*For BR = PC, add one clock to both <ea> calculate and execute times. Timings are for an idle FPU.

10.7.2 Integer Unit Support Timings (Continued)

Addressing Mode	FMOVE FPn,<ea>*					
	Byte, Word, and Long Word		Single and Double Precision		Extended Precision	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	9	$9_L + 3$	2	$1_L + 3$	—	—
An	—	—	—	—	—	—
(An)	8	$9_L + 2$	2	$1_L + 2$	4	$1_L + 3$
(An)+	8	$9_L + 2$	2	$1_L + 2$	4	$1_L + 3$
–(An)	8	$9_L + 2$	2	$1_L + 2$	4	$1_L + 3$
(xxx).W, (xxx).L	8	$9_L + 2$	3	$1_L + 2$	4	$1_L + 3$
#<xxx>	—	—	—	—	—	—
(d_{16},An)	8	$9_L + 2$	2	$1_L + 2$	4	$1_L + 3$
(d_{16},PC)	—	—	—	—	—	—
(d_8,An,Xn)	8	$6_L + 5$	5	5	6	6
(d_8,PC,Xn)	—	—	—	—	—	—
(An,Xn)	7	$4_L + 6$	7	$1_L + 6$	8	$1_L + 7$
(bd,An,Xn)	8	$4_L + 7$	8	$1_L + 7$	9	$1_L + 8$
([bd,An,Xn])	11	$1_L + 10$	11	$1_L + 10$	12	$1_L + 11$
([bd,An,Xn],od)	12	$1_L + 11$	12	$1_L + 11$	13	$1_L + 12$
([bd,An],Xn)	12	$3_L + 9$	12	$3_L + 9$	13	$3_L + 10$
([bd,An],Xn,od)	13	$3_L + 10$	13	$3_L + 10$	14	$3_L + 11$

*Timings are for an idle floating-point unit.

10.7.2 Integer Unit Support Timings (Continued)

Addressing Mode	FMOVE/FMOVEM to/from 1 Control Register[a]		FMOVEM <list>,<ea> and <ea>,<list>[a,b]		FScc[a]	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
Dn	2	$1_L + 2$	—	—	5	6
An	2	$1_L + 2$	—	—	—	—
(An)	4	$2_L + 3$	17	$2_L + 15$	4	5
(An)+	4	$2_L + 3$	17	$2_L + 15$	6	$2_L + 5$
–(An)	5	$3_L + 3$	16	$1_L + 15$	6	$2_L + 5$
(xxx).W, (xxx).L	4	$2_L + 3$	19	$3_L + 15$	4	5
#<xxx>	4	$2_L + 3$	19	$1_L + 17$	—	—
(d_{16},An)	4	$2_L + 3$	17	$2_L + 15$	4	5
(d_{16},PC)	5	$4_L + 3$	—	—	—	—
(d_8,An,Xn)	5	6	19	18	7	8
(d_8,PC,Xn)	6	$1_L + 6$	20	$1_L + 18$	—	—
(An,Xn)	7	$1_L + 7$	20	$1_L + 19$	9	$1_L + 9$
(bd,An,Xn)	8	$1_L + 8$	21	$1_L + 20$	10	$1_L + 10$
([bd,An,Xn])	11	$1_L + 11$	25	$1_L + 23$	13	$1_L + 13$
([bd,An,Xn],od)	12	$1_L + 13$	25	$1_L + 24$	14	$1_L + 14$
([bd,An],Xn)	12	$3_L + 10$	26	$3_L + 22$	14	$3_L + 12$
([bd,An],Xn,od)	13	$3_L + 12$	26	$3_L + 23$	15	$3_L + 13$

NOTES:

a. Timings are for an idle floating-point unit. Same as FMOVE <ea>,FPCR.

b. Add three clocks to both <ea> calculate and execute times for each additional floating-point register. Add one clock to both <ea> calculate and execute times for dynamic register list.

10.7.2 Integer Unit Support Timings (Continued)

Addressing Mode	FSAVE <ea>*					
	Idle or Null		Short		Long	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
FPn	—	—	—	—	—	—
Dn	—	—	—	—	—	—
An	—	—	—	—	—	—
(An)	12	1_L + 11	33	1_L + 32	50	1_L + 49
(An)+	—	—	—	—	—	—
–(An)	11	11	32	32	49	49
(xxx).W, (xxx).L	13	1_L + 11	34	1_L + 32	51	1_L + 49
#<xxx>	—	—	—	—	—	—
(d_{16},An)	12	1_L + 11	33	1_L + 32	50	1_L + 49
(d_{16},PC)	—	—	—	—	—	—
(d_8,An,Xn)	13	13	34	34	51	51
(d_8,PC,Xn)	—	—	—	—	—	—
(An,Xn)	16	1_L + 14	37	1_L + 35	54	1_L + 52
(bd,An,Xn)	17	1_L + 15	38	1_L + 36	55	1_L + 53
([bd,An,Xn])	19	1_L + 18	40	1_L + 39	57	1_L + 56
([bd,An,Xn],od)	21	1_L + 19	42	1_L + 40	59	1_L + 57
([bd,An],Xn)	20	3_L + 17	41	3_L + 38	58	3_L + 55
([bd,An],Xn,od)	22	3_L + 18	46	3_L + 42	65	3_L + 61

*Timings are for an idle floating-point unit.

10.7.2 Integer Unit Support Timings (Concluded)

Addressing Mode	FRESTORE <ea>*					
	Idle or Null		Short		Long	
	<ea> Calculate	Execute	<ea> Calculate	Execute	<ea> Calculate	Execute
FPn	—	—	—	—	—	—
Dn	—	—	—	—	—	—
An	—	—	—	—	—	—
(An)	13	1_L + 12	26	1_L + 25	40	1_L + 39
(An)+	13	1_L + 12	26	1_L + 25	40	1_L + 39
–(An)	—	—	—	—	—	—
(d_{16},An)	13	1_L + 12	26	1_L + 25	40	1_L + 39
(d_{16},PC)	—	—	—	—	—	—
(xxx).W, (xxx).L	14	1_L + 12	27	1_L + 25	41	1_L + 39
#<xxx>	—	—	—	—	—	—
(d_8,An,Xn)	14	14	27	27	41	41
(d_8,PC,Xn)	—	—	—	—	—	—
(An,Xn)	16	1_L + 14	29	1_L + 27	43	1_L + 41
(bd,An,Xn)	17	1_L + 15	30	1_L + 28	44	1_L + 42
([bd,An,Xn])	20	1_L + 19	33	1_L + 32	47	1_L + 46
([bd,An,Xn],od)	21	1_L + 19	34	1_L + 32	48	1_L + 46
([bd,An],Xn)	21	3_L + 18	34	3_L + 31	48	3_L + 45
([bd,An],Xn,od)	22	3_L + 19	35	3_L + 31	49	3_L + 45

*Timings are for an idle floating-point unit.

10.7.3 Timings in the Floating-Point Unit

Times in parentheses are the total time that the stage uses to execute an instruction even though the stage can pass data to the next stage earlier. So that 2(3) in the conversion stage means that the instruction takes two cycles to execute, but this stage is actually busy for three cycles.

Instruction	Opclass	Size	Precision	Operands	Conversion	Execution	Normalization
FADD,FSUB	0	—	Any	Norm,Norm	2(3)	3	2(3)
	0	—	Any	Norm,Zero	2(3)	3	2(3)
	0	—	Any	Zero,Zero	4	0	0
	0	—	Any	— ,Inf	4	0	0
	0	—	Any	— ,NAN	4	0	0
	0	S,D	Any	Norm,Norm	2(3)	3	2(3)
	2	S,D	Any	Norm,Zero	2(3)	3	2(3)
	2	S,D	Any	Zero,Zero	4	0	0
	2	S,D	Any	— ,Inf	4	0	0
	2	S,D	Any	— ,NAN	4	0	0
	2	X	Any	Norm,Norm	3(4)	3	2(3)
	2	X	Any	Norm,Zero	3(4)	3	2(3)
	2	X	Any	Zero,Zero	5	0	0
	2	X	Any	— ,Inf	5	0	0
	2	X	Any	— ,NAN	5	0	0
FMUL	0	—	Any	Norm,Norm	2(3)	5	2(3)
	0	—	Any	— ,Zero	4	0	0
	0	—	Any	— ,Inf	4	0	0
	0	—	Any	— ,NAN	4	0	0
	2	S,D	Any	Norm,Norm	2(3)	5	2(3)
	2	S,D	Any	— ,Zero	4	0	0
	2	S,D	Any	— ,Inf	4	0	0
	2	S,D	Any	— ,NAN	4	0	0
	2	X	Any	Norm,Norm	3(4)	5	2(3)
	2	X	Any	— ,Zero	5	0	0
	2	X	Any	— ,Inf	5	0	0
	2	X	Any	— ,NAN	5	0	0
FDIV	0		Any	Norm,Norm	2(3)	37.5	2(3)
	0	—	Any	— ,Zero	4	0	0
	0	—	Any	— ,Inf	4	0	0
	0	—	Any	— ,NAN	4	0	0
	2	S,D	Any	Norm,Norm	2(3)	37.5	2(3)
	2	S,D	Any	— ,Zero	4	0	0
	2	S,D	Any	— ,Inf	4	0	0

10.7.3 Timings in the Floating-Point Unit (Continued)

Instruction	Opclass	Size	Precision	Operands	Conversion	Execution	Normalization
FDIV	2	S,D	Any	— ,NAN	4	0	0
	2	X	Any	Norm,Norm	3(4)	37.5	2(3)
	2	X	Any	— ,Zero	5	0	0
	2	—	Any	— ,Inf	5	0	0
	2	X	Any	— ,NAN	5	0	0
FSQRT	0	—	Any	Norm	2(3)	103	2(3)
	0	—	Any	(Zero|Inf|NAN)	4	0	0
	2	S,D	Any	Norm	2(3)	103	2(3)
	2	S,D	Any	(Zero|Inf|NAN)	4	0	0
	2	X	Any	Norm	3(4)	103	2(3)
	2	X	Any	(Zero|Inf|NAN)	5	0	0
FMOVE,	0	—	X	(Norm|Zero|Inf)	2	0	0
FABS,	0	—	X	NAN	3	0	0
FNEG	0	—	S,D	Norm	5	0	0
	0	—	S,D	(Zero|Inf)	3	0	0
	0	—	S,D	NAN	4	0	0
	2	S	Any	(Norm|Zero|Inf)	3	0	0
	2	S	Any	NAN	4	0	0
	2	D	D,X	(Norm|Zero|Inf)	3	0	0
	2	D	D,X	NAN	4	0	0
	2	D	S	Norm	5	0	0
	2	D	S	(Zero|Inf)	4	0	0
	2	D	S	NAN	5	0	0
	2	X	X	(Norm|Zero|Inf)	4	0	0
	2	X	X	NAN	5	0	0
	2	X	S,D	Norm	6	0	0
	2	X	S,D	(Zero|Inf)	5	0	0
	2	X	S,D	NAN	6	0	0
	2	B,W	Any	(+Norm|Zero)	1.5(11)	4.5	2
	2	L	D,X	(+Norm|Zero)	1.5(11)	4.5	2
	2	L	S	(+Norm|Zero)	1.5(12.5)	4.5	2
	2	B,W	Any	—Norm	1.5(11.5)	5	2
	2	L	D,X	—Norm	1.5(11.5)	5	2
	2	L	S	—Norm	1.5(13)	5	2
FMOVE	3	S,D	Any	Any	3	0	0
	3	X	Any	Any	4	0	0
	3	B,W,L	Any	+(Norm|Zero)	3(9)	1.5	3.5
	3	B,W,L	Any	–(Norm|Zero)	3(10)	1.5	4.5

10.7.3 Timings in the Floating-Point Unit (Concluded)

Instruction	Opclass	Size	Precision	Operands	Conversion	Execution	Normalization
FMOVEM	4	—	—	—	2 + (2 per reg)	0	0
	5	—	—	—	2 + (2 per reg)	0	0
	6	—	—	—	2 + (3 per reg)	0	0
	7	—	—	—	2 + (3 per reg)	0	0
FCMP	0	—	Any	Norm,Norm	2(3)	3	1
	0	—	Any	Norm,Zero	2(3)	3	1
	0	—	Any	Zero,Zero	4	0	0
	0	—	Any	— ,Inf	4	0	0
	0	—	Any	— ,NAN	4	0	0
	2	S,D	Any	Norm,Norm	2(3)	3	1
	2	S,D	Any	Norm,Zero	2(3)	3	1
	2	S,D	Any	Zero,Zero	4	0	0
	2	S,D	Any	— ,Inf	4	0	0
	2	S,D	Any	— ,NAN	4	0	0
	2	X	Any	Norm,Norm	3(4)	3	1
	2	X	Any	Norm,Zero	3(4)	3	1
	2	X	Any	Zero,Zero	5	0	0
	2	X	Any	— ,Inf	5	0	0
	2	X	Any	— ,NAN	5	0	0

SECTION 11
MC68040 ELECTRICAL AND THERMAL CHARACTERISTICS

The following paragraphs provide information on the maximum rating and thermal characteristics for the MC68040. This section is subject to change. For the most recent specifications, contact a Motorola sales office or complete the registration card at the end of this manual.

11.1 MAXIMUM RATINGS

Characteristic	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	V
Input Voltage	V_{in}	–0.5 to +7.0	V
Maximum Operating Junction Temperature	T_J	110	°C
Minimum Operating Ambient Temperature	T_A	0	°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. Reliablity of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either GND or V_{CC}).

11.2 THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Rating
Thermal Resistance, Junction to Case—PGA Package	θ_{JC}	3	°C/W

11.3 DC ELECTRICAL SPECIFICATIONS (V_{CC} = 5.0 VDC ±5 %)

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, $\overline{BG}$, $\overline{CDIS}$, $\overline{MDIS}$, $\overline{IPLx}$, PCLK, $\overline{RSTI}$, SCx, $\overline{TBI}$, TLNx , $\overline{TCI}$, TCK, $\overline{TEA}$	I_{in}	20	20	µA
Hi-Z (Off-State) Leakage Current @ 0.5/2.4 V An, $\overline{BB}$, $\overline{CIOUT}$, Dn, $\overline{LOCK}$, $\overline{LOCKE}$, R/$\overline{W}$, SIZx, $\overline{TA}$, TDO, $\overline{TIP}$, TMx, TLNx, $\overline{TS}$, TTx, UPAx	I_{TSI}	20	20	µA
Signal Low Input Current, V_{IL} = 0.8 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 (Small Buffer Mode)	V_{OH}	2.4	—	V
Output Low Voltage, I_{OL} = 5 mA (Small Buffer Mode)	V_{OL}	—	0.5	V
Output High Voltage, I_{OH} = 55 mA (Large Buffer Mode)	V_{OH}	2.4	—	V
Output Low Voltage, I_{OL} = 55 mA (Large Buffer Mode)	V_{OL}	—	0.5	V
Capacitance*, V_{in} = 0 V, f = 1 MHz	C_{in}	—	25	pF

*Capacitance is periodically sampled rather than 100% tested.

11.4 POWER DISSIPATION

Buffer Mode	25 MHz	33 MHz	40 MHz
Worst Case (V_{CC} = 5.25 V , T_A = 0°C)			
Small Unterminated, I_{OL} = I_{OH} = 5 mA	4.9 W	6.2 W	7.2 W
Large Unterminated, I_{OL} = I_{OH} = 5 mA	5.1 W	6.6 W	7.7 W
Large Terminated, 50 Ω, 2.5 V, I_{OL} = I_{OH} = 55 mA	6.5 W	8.0 W	9.1 W
Typical Values (V_{CC} = 5 V, T_J = 90°C)*			
Small	3.0 W	4.1 W	4.5 W
Large Unterminated	3.3 W	4.4 W	4.8 W
Large Terminated, 50 Ω, 2.5 V	4.7 W	5.8 W	6.2 W

*This information is for system reliability purposes.

11.5 CLOCK AC TIMING SPECIFICATIONS (see Figure 11-1)

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
	Frequency of Operation	20	25	20	33	20	40	MHz
1	PCLK Cycle Time	20	25	15	25	12.5	25	ns
2	PCLK Rise Time	—	1.7	—	1.7	—	1.5	ns
3	PCLK Fall Time	—	1.6	—	1.6	—	1.5	ns
4	PCLK Duty Cycle Measured at 1.5 V	47.50	52.50	46.67	53.33	46.00	54.00	%
4a*	PCLK Pulse Width High Measured at 1.5 V	9.50	10.50	7	8	5.75	6.75	ns
4b*	PCLK Pulse Width Low Measured at 1.5 V	9.50	10.50	7	8	5.75	6.75	ns
5	BCLK Cycle Time	40	60	30	60	25	50	ns
6,7	BCLK Rise and Fall Time	—	4	—	3	—	3	ns
8	BCLK Duty Cycle Measured at 1.5 V	40	60	40	60	40	60	%
8a*	BCLK Pulse Width High Measured at 1.5 V	16	24	12	18	10	15	ns
8b*	BCLK Pulse Width Low Measured at 1.5 V	16	24	12	18	10	15	ns
9	PCLK, BCLK Frequency Stability	—	1000	—	1000	—	1000	ppm
10	PCLK to BCLK Skew	—	9	—	n/a	—	n/a	ns

*Specification value at maximum frequency of operation.

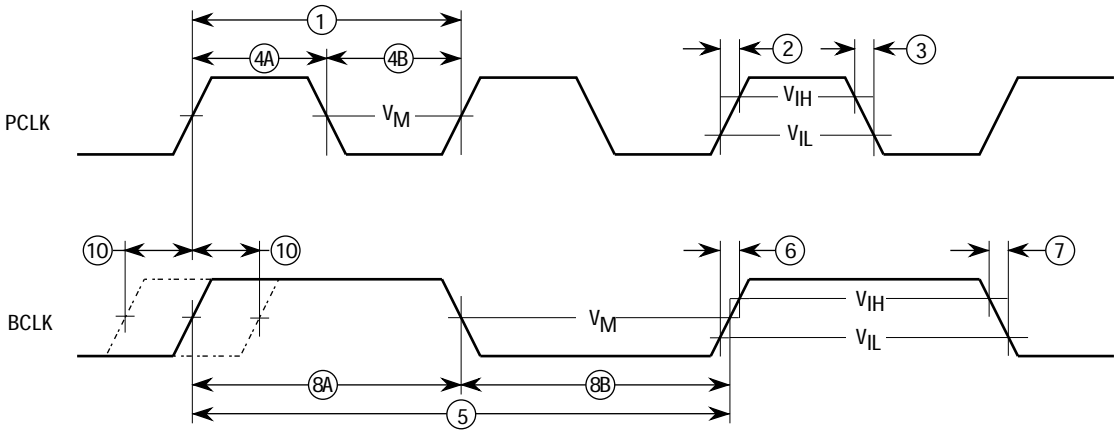


Figure 11-1. Clock Input Timing Diagram

11.6 OUTPUT AC TIMING SPECIFICATIONS (See Figures 11-3 to 11-7)

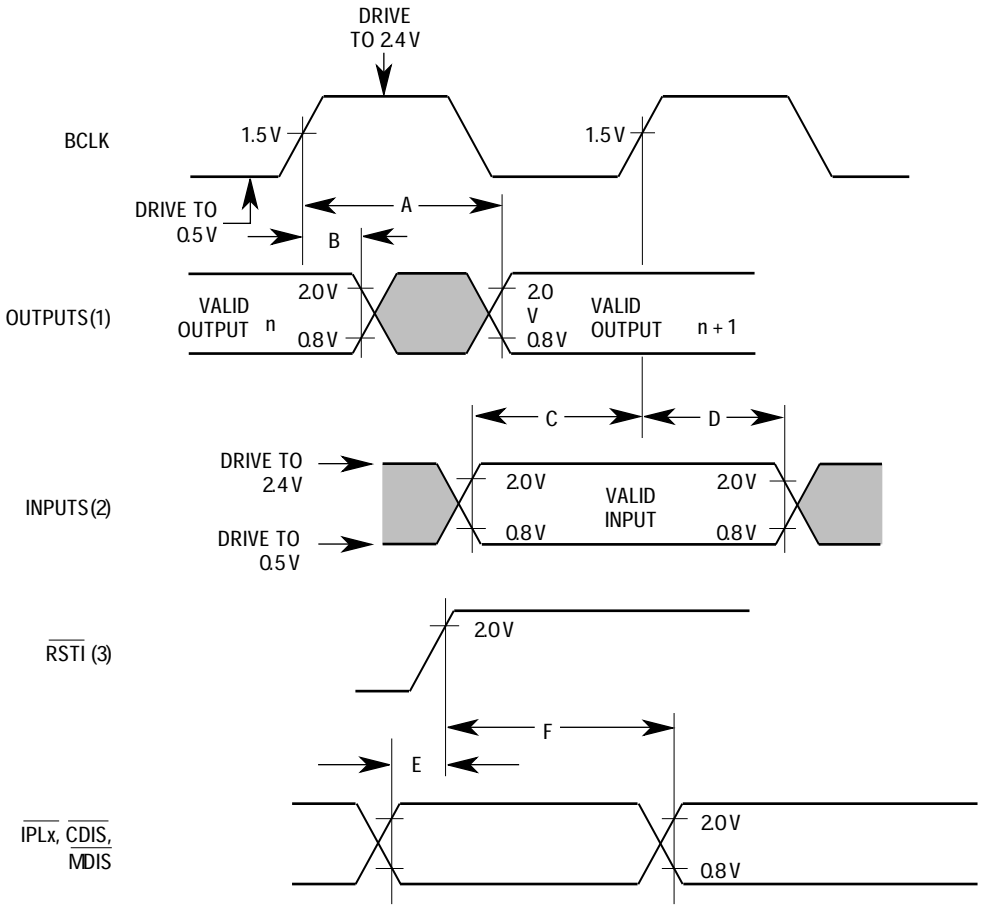
Num	Characteristic	25 MHz				33 MHz				40 Mhz				Unit
		Large[1]		Small[2]		Large[1]		Small[2]		Large[1]		Small[2]		
		Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
11[3]	BCLK to Address $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, R/$\overline{\text{W}}$, SIZx, TLN, TMx, TTx, UPAx Valid	9	21	9	30	6.50	18	6.50	25	5.25	16	5.25	24	ns
12	BCLK to Output Invalid (Output Hold)	9	—	9	—	6.50	—	6.50	—	5.25	—	5.25	—	ns
13	BCLK to $\overline{\text{TS}}$ Valid	9	21	9	30	6.50	18	6.50	25	5.25	16	5.25	24	ns
14	BCLK to $\overline{\text{TIP}}$ Valid	9	21	9	30	6.50	18	6.50	25	5.25	17	5.25	24	ns
18[4]	BCLK to Data Out Valid	9	23	9	32	6.50	20	6.50	27	5.25	18	5.25	26	ns
19[4]	BCLK to Data Out Invalid (Output Hold)	9	—	9	—	6.50	—	6.50	—	5.25	—	5.25	—	ns
20[3,4]	BCLK to Output Low Impedance	9	—	9	—	6.50	—	6.50	—	5.25	—	5.25	—	ns
21[5]	BCLK to Data-Out High Impedance	9	20	9	20	6.50	17	6.50	17	5.25	16	5.25	16	ns
26[3]	BCLK to Multiplexed Address Valid	19	31	19	40	14	26	14	33	13	25	13	32	ns
27[3,5]	BCLK to Multiplexed Address Driven	19	—	19	—	14	—	14	—	13	—	13	—	ns
28[3,4,5]	BCLK to Multiplexed Address High Impedance	9	18	9	18	6.50	15	6.50	15	5.25	14	5.25	14	ns
29[4,5]	BCLK to Multiplexed Data Driven	19	—	19	—	14	20	14	20	13	19	13	19	ns
30[4]	BCLK to Multiplexed Data Valid	19	33	19	42	14	28	14	35	13	27	13	34	ns
38[3]	BCLK to Address, $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, R/$\overline{\text{W}}$, SIZx, $\overline{\text{TS}}$, TLNx, TMx, TTx, UPAx High Impedance	9	18	9	18	6.50	15	6.50	15	5.25	14	5.25	14	ns
39	BCLK to $\overline{\text{BB}}$, $\overline{\text{TA}}$, $\overline{\text{TIP}}$ High Impedance	19	28	19	28	14	23	14	23	11.5	22	11.5	22	ns
40	BCLK to $\overline{\text{BR}}$, $\overline{\text{BB}}$ Valid	9	21	9	30	6.50	18	6.50	25	5.25	16	5.25	24	ns
43	BCLK to $\overline{\text{MI}}$ Valid	9	21	9	30	6.50	18	6.50	25	5.25	17	5.25	24	ns
48	BCLK to $\overline{\text{TA}}$ Valid	9	21	9	30	6.50	18	6.50	25	5.25	17	5.25	24	ns
50	BCLK to $\overline{\text{IPEND}}$, PSTx, $\overline{\text{RSTO}}$ Valid	9	21	9	30	6.50	18	6.50	25	5.25	17	5.25	24	ns

NOTES:

1. Output timing is specified for a valid signal measured at the pin. Large buffer timing is specified driving a 50 Ω transmission line with a length characterized by a 2.5-ns one-way propagation delay, terminated through 50 Ω to 2.5 V. Large buffer output impedance is 4–12 Ω, resulting in incident wave switching for this environment. All large buffer outputs must be terminated to guarantee operation.
2. Small buffer timing is specified driving an unterminated 30 Ω transmission line with a length characterized by a 2.5 ns one-way propagation delay. Small buffer output impedance is typically 30 Ω; the small buffer specifications include approximately 5 ns for the signal to propagate the length of the transmission line and back.
3. Timing specifications 11, 20, and 38 for address bus output timing apply when normal bus operation is selected. Specifications 26, 27, and 28 should be used when the multiplexed bus mode of operation is enabled.
4. Timing specifications 18 and 19 for data bus output timing apply when normal bus operation is selected. Specifications 28 and 29 should be used when the multiplexed bus mode of operation is enabled.
5. Timing specifications 21, 27, 28, and 29 are measured from BCLK edges. By design, the MC68040 cannot drive address and data simultaneously during multiplexed operations.

11.7 INPUT AC TIMING SPECIFICATIONS (see Figures 11-3 to 11-7)

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min.	Max.	
15	Data-In Valid to BCLK (Setup)	5	—	4	—	3	—	ns
16	BCLK to Data-In Invalid (Hold)	4	—	4	—	3	—	ns
17	BCLK to Data-In High Impedance (Read Followed by Write)	—	49	—	36.5	—	30.25	ns
22a	$\overline{TA}$ Valid to BCLK (Setup)	10	—	10	—	8	—	ns
22b	$\overline{TEA}$ Valid to BCLK (Setup)	10	—	10	—	9	—	ns
22c	$\overline{TCI}$ Valid to BCLK (Setup)	10	—	10	—	9	—	ns
22d	$\overline{TBI}$ Valid to BCLK (Setup)	11	—	10	—	9	—	ns
23	BCLK to $\overline{TA}$, $\overline{TEA}$, $\overline{TCI}$, $\overline{TBI}$ Invalid (Hold)	2	—	2	—	2	—	ns
24	$\overline{AVEC}$ Valid to BCLK (Setup)	5	—	5	—	5	—	ns
25	BCLK to $\overline{AVEC}$ Invalid (Hold)	2	—	2	—	2	—	ns
31	DLE Width High	8	—	8	—	8	—	ns
32	Data-In Valid to DLE (Setup)	2	—	2	—	2	—	ns
33	DLE to Data-In Invalid (Hold)	8	—	8	—	8	—	ns
34	BCLK to DLE Hold	3	—	3	—	3	—	ns
35	DLE High to BCLK	16	—	12	—	12	—	ns
36	Data-In Valid to BCLK (DLE Mode Setup)	5	—	5	—	5	—	ns
37	BCLK to Data-In Invalid (DLE Mode Hold)	4	—	4	—	4	—	ns
41a	$\overline{BB}$ Valid to BCLK (Setup)	7	—	7	—	7	—	ns
41b	$\overline{BG}$ Valid to BCLK (Setup)	8	—	7	—	7	—	ns
41c	$\overline{CDIS}$, $\overline{MDIS}$ Valid to BCLK (Setup)	10	—	8	—	8	—	ns
41d	$\overline{IPLx}$ Valid to BCLK (Setup)	4	—	3	—	3	—	ns
42	BCLK to $\overline{BB}$, $\overline{BG}$, $\overline{CDIS}$, $\overline{IPLx}$, $\overline{MDIS}$ Invalid (Hold)	2	—	2	—	2	—	ns
44a	Address Valid to BCLK (Setup)	8	—	7	—	7	—	ns
44b	SIZx Valid to BCLK (Setup)	12	—	8	—	8	—	ns
44c	TTx Valid to BCLK (Setup)	6	—	8.5	—	8.5	—	ns
44d	R/$\overline{W}$ Valid to BCLK (Setup)	6	—	5	—	5	—	ns
44e	SCx Valid to BCLK (Setup)	10	—	11	—	8	—	ns
45	BCLK to Address,SIZx, TTx, R/$\overline{W}$, SCx Invalid (Hold)	2	—	2	—	2	—	ns
46	$\overline{TS}$ Valid to BCLK (Setup)	5	—	9	—	7	—	ns
47	BCLK to $\overline{TS}$ Invalid (Hold)	2	—	2	—	2	—	ns
49	BCLK to $\overline{BB}$ High Impedance (MC68040 Assumes Bus Mastership)	—	9	—	9	—	9	ns
51	$\overline{RSTI}$ Valid to BCLK	5	—	4	—	4	—	ns
52	BCLK to $\overline{RSTI}$ Invalid	2	—	2	—	2	—	ns
53	Mode Select Setup to $\overline{RSTI}$ Negated	20	—	20	—	20	—	ns
54	$\overline{RSTI}$ Negated to Mode Selects Invalid	2	—	2	—	2	—	ns



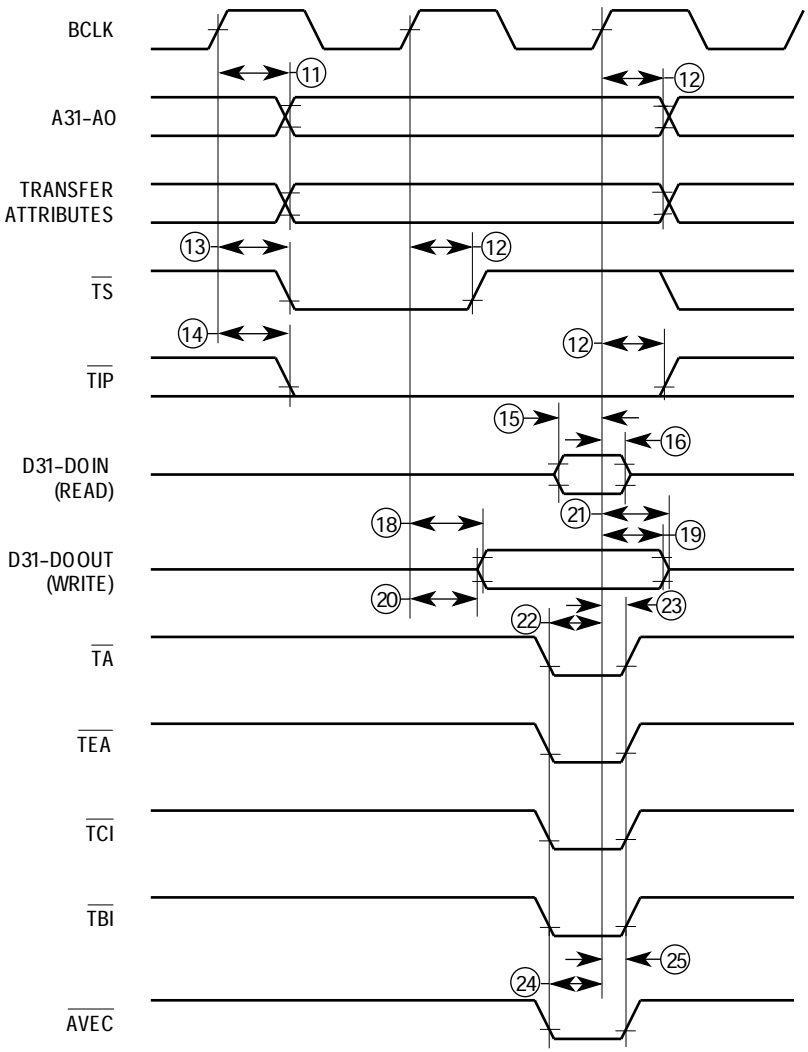
NOTES:

1. This output timing is applicable to all parameters specified relative to the rising edge of the clock.
2. This input timing is applicable to all parameters specified relative to the rising edge of the clock.
3. This timing is applicable to all parameters specified relative to the negation of the RSTI signal.

LEGEND:

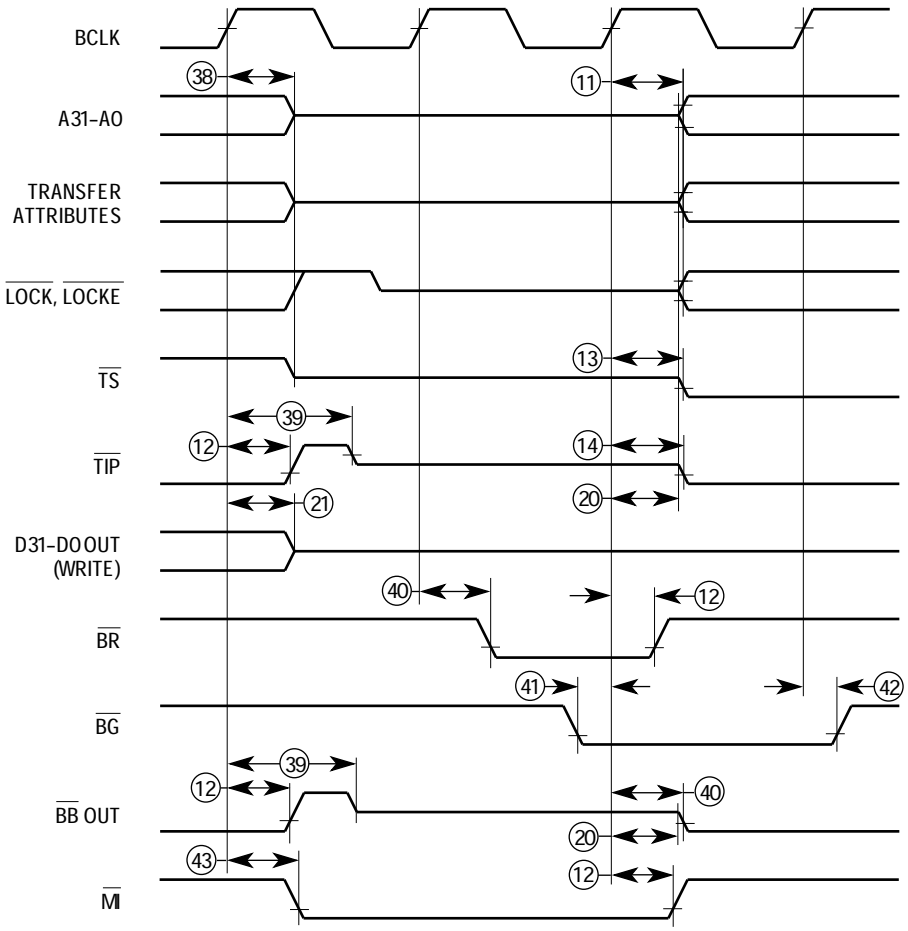
A. Maximum output delay specification.
B. Minimum output hold time.
C. Minimum input setup time specification.
D. Minimum input hold time specification.
E. Mode select setup time to RSTI negated.
F. Mode select hold time from RSTI negated.

Figure 11-2. Drive Levels and Test Points for AC Specifications



NOTE: Transfer Attribute Signals = UPAx, SIZx, TTx, TMx, TLNx, R/$\overline{W}$, $\overline{LOCK}$, $\overline{LOCKE}$, $\overline{CIOUT}$

Figure 11-3. Read/Write Timing



NOTE: Transfer Attribute Signals = UPAx, SIZx, TTx, TMx, TLNx, R/W, CIOUT

Figure 11-4. Bus Arbitration Timing

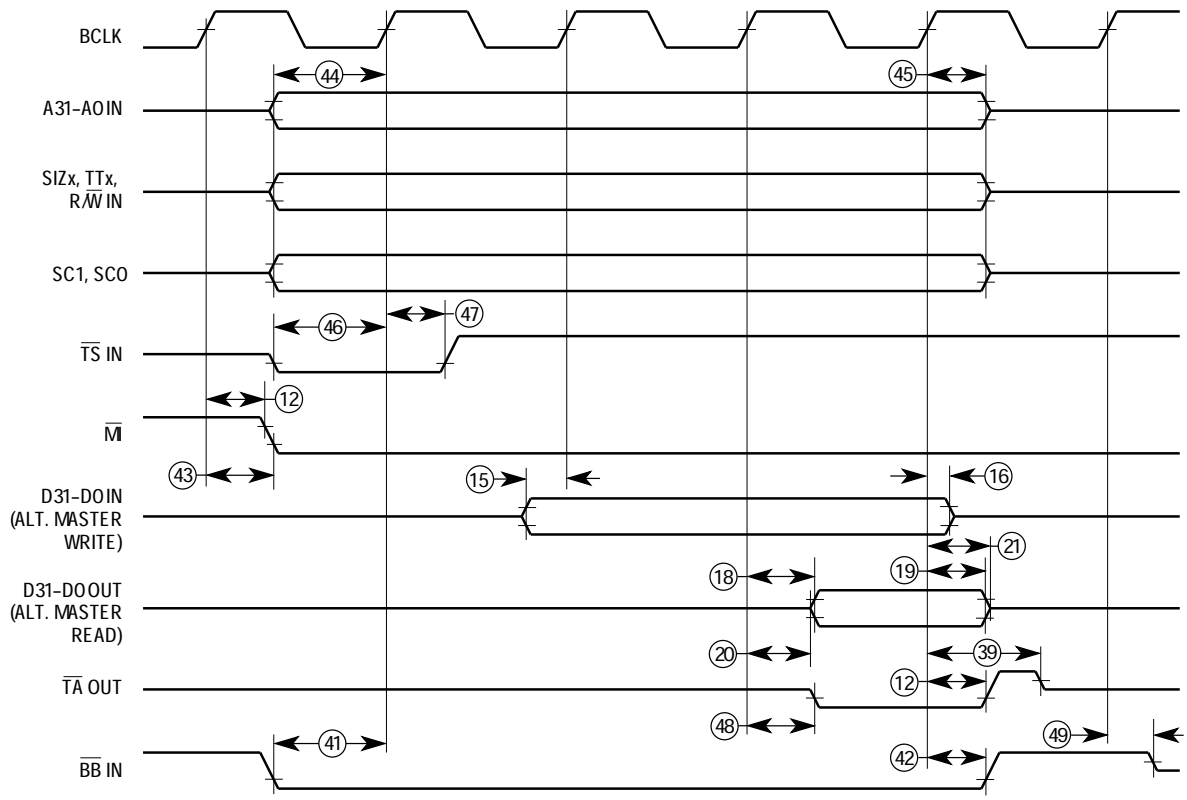


Figure 11-5. Snoop Hit Timing

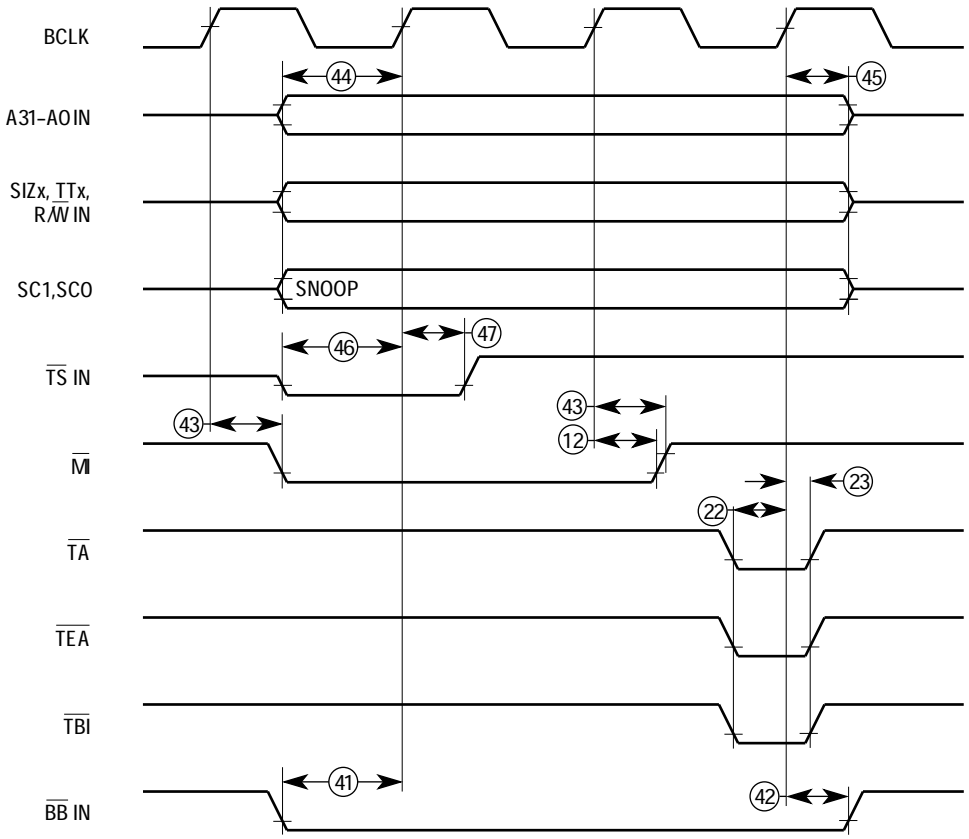


Figure 11-6. Snoop Miss Timing

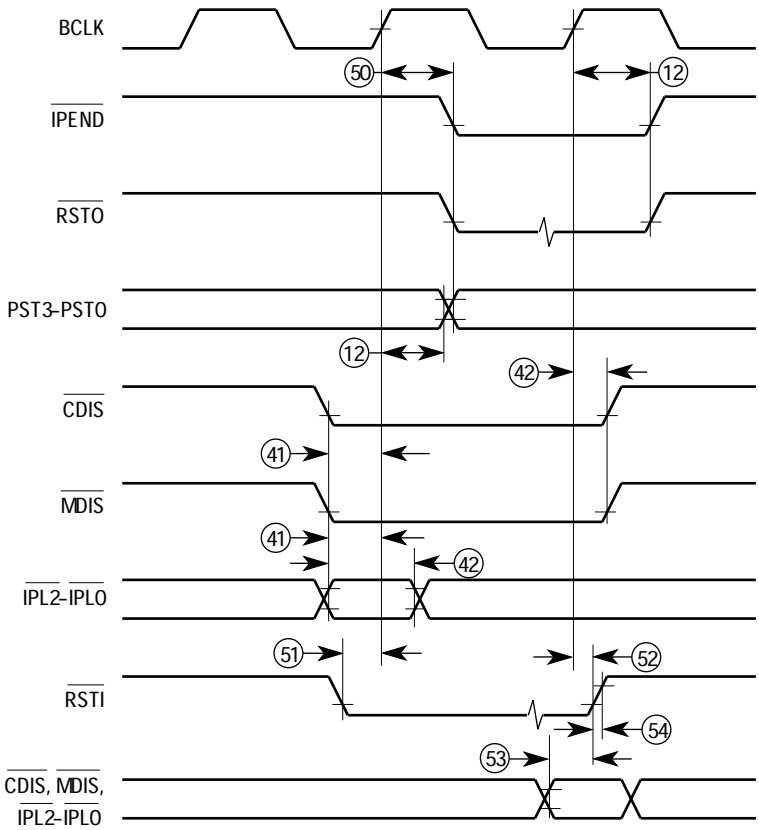


Figure 11-7. Other Signal Timing

11.8 MC68040 THERMAL DEVICE CHARACTERISTICS

The need to efficiently cool microprocessors is becoming more important as they become more complex and require more power. In the past, the M68000 family has been able to provide a 0–70°C ambient temperature part for speeds less than 40 MHz. However, the MC68040, MC68LC040, and MC68EC040 with a 50 MHz processor clock has a maximum power dissipation for a particular operating mode, a maximum junction temperature, and a thermal resistance from the die junction to the case. This section provides a more accurate method of evaluating the environment, taking into consideration both the airflow and ambient temperature. This section also gives the user information to design a cooling method that meets both thermal performance requirements and constraints of the board environment. This section discusses the device characteristics and several methods for thermal management as well as an example of one method for cooling the MC68040, MC68LC040, and MC68EC040. The MC68040, MC68LC040, and MC68EC040 contain inherent characteristics that should be considered when evaluating a method for cooling the device. The following paragraphs discuss these die/package and power considerations for each of these parts.

NOTE

All references to the MC68040 also include the MC68LC040 and MC68EC040. Note that the MC68LC040 and MC68EC040 only implement the small buffer mode.

11.8.1 MC68040 Die and Package

The MC68040 is in a cavity-down alumina-ceramic 179-pin PGA that has a worst case thermal resistance from junction to case of 3°C/W. The package differs from previous M68000 family PGA packages that are cavity-up. The cavity-down design allows the die to be attached to the top surface of the package, increasing the part's ability to dissipate heat through the package surface or an attached heat sink. The system designer needs to determine the specific dimensions and design of the particular heat sink, considering both thermal performance and size requirements.

11.8.2 MC68040 Power Considerations

The MC68040 has a maximum power rating, which varies depending on the combination of output buffer mode and operating frequency. Note that this section assumes large buffers terminated to 2.5 V. The large buffer output mode dissipates more power than the small, and higher frequencies of operation dissipate more power than the lower frequencies.

The MC68040 allows a combination of either large or small buffers on the outputs of the miscellaneous control signals, data bus, and address bus/transfer attribute pins. The large buffers offer quicker output times, allowing for an easier logic design. However, they do so by driving about 10 times as much current as the small buffers. The designer should consider whether the quicker timings present enough advantage to justify the additional consideration to the individual signal terminations, the die power consumption, and the required cooling for the device. Since the MC68040 can be powered up in one of many output buffer modes upon reset, the actual maximum power consumption for an MC68040

rated at a particular maximum operating frequency is dependent upon the power-up mode. Therefore, the MC68040 is rated at a maximum power dissipation for either the large or small buffers at a particular frequency. This allows for control of some of the thermal management upon reset. The following equation provides a rough method to calculate the maximum power consumption for a chosen output buffer mode:

$$P_D = P_{DSB} + (P_{DLB} - P_{DSB}) \times (PINS_{LB} \div PINS_{CLB})$$

where:

P_D = Maximum Power Dissipation for Output Buffer Mode Selected
P_{DSB} = Maximum Power Dissipation for Small Buffer Mode (All Outputs)
P_{DLB} = Maximum Power Dissipation for Large Buffer Mode (All Outputs)
$PINS_{LB}$ = Number of Pins Large Buffer Mode
$PINS_{CLB}$ = Number of Pins Capable of the Large Buffer Mode

Table 11-1 lists the simplified relationship on the maximum power dissipation for eight possible configurations of output buffer modes.

Table 11-1. Maximum Power Dissipation for Output Buffer Mode Configurations

Output Configuration			
Data Bus	**Address Bus and Transfer Attributes**	**Control Signals**	**Maximum Power Dissipation**
Small*	Small	Small	P_{DSB}
Small	Small	Large	$P_{DSB} + (P_{DLB} - P_{DSB}) \times 13\%$
Small	Large	Small	$P_{DSB} + (P_{DLB} - P_{DSB}) \times 52\%$
Small	Large	Large	$P_{DSB} + (P_{DLB} - P_{DSB}) \times 65\%$
Large	Small	Small	$P_{DSB} + (P_{DLB} - P_{DSB}) \times 35\%$
Large	Small	Large	$P_{DSB} + (P_{DLB} - P_{DSB}) \times 48\%$
Large	Large	Small	$P_{DSB} + (P_{DLB} - P_{DSB}) \times 87\%$
Large	Large	Large	$P_{DSB} + (P_{DLB} - P_{DSB}) \times 100\%$

*The MC68LC040 and MC68EC040 only utilize this row of information.

To calculate the specific power dissipation of a design, the termination method of each signal must be considered. For example, a signal output that is not connected would not dissipate any additional power if it were configured in the large rather than the small buffer mode. Since the maximum operating junction temperature is specified as 110°C, the maximum case temperature (T_C) in °C can be obtained from the following equation:

$$T_C = T_J - P_D \times \theta_{JC}$$

where:

T_C = Maximum Case Temperature
T_J = Maximum Junction Temperature
P_D = Maximum Power Dissipation of the Device
θ_{JC} = Thermal Resistance between the Junction of the Die and the Case

In general, the ambient temperature (T_A) in °C is a function of the following equation:

$$T_A = T_J - P_D \times \theta_{JC} - P_D \times \theta_{CA}$$

The thermal resistance from case to outside ambient (θ_{CA}) is the only user-dependent parameter once a buffer output configuration has been determined. Reducing the case to ambient thermal resistance increases the maximum operating ambient temperature. Therefore, by utilizing methods such as heat sinks and ambient air cooling to minimize θ_{CA}, a higher ambient operating temperature and/or a lower junction temperature can be achieved. However, an easier approach to thermal evaluation uses the following equations:

$$T_A = T_J - P_D \times \theta_{JA} \quad \text{or} \quad T_J = T_A + P_D \times \theta_{JA}$$

where:

θ_{JA} = Thermal Resistance from the Junction to the Ambient ($\theta_{JC} + \theta_{CA}$)

The total thermal resistance for a package (θ_{JA}) is a combination of its two components, θ_{JC} and θ_{CA}. These components represent the barrier to heat flow from the semiconductor junction to the package case surface (θ_{JC}) and θ_{CA}. Although θ_{JC} is package related and the user cannot influence it, θ_{CA} is user dependent. Good thermal management by the user, such as heat sink and airflow, can significantly reduce θ_{CA} achieving either a lower semiconductor junction temperature or a higher ambient operating temperature.

11.9 MC68040 THERMAL MANAGEMENT TECHNIQUES

To attain a reasonable maximum ambient operating temperature, the user must reduce the barrier to heat flow from θ_{JA}. The only way to accomplish this is to significantly reduce θ_{CA} by applying thermal management techniques such as heat sinks and forced air cooling. The following paragraphs discuss thermal study results for the MC68040 that did not use thermal management techniques, airflow cooling, heat sink, and heat sink combined with airflow cooling.

The MC68040 power dissipation values given in this section represent the sum of the power dissipated by the internal circuitry and the output buffers of the MC68040. The termination network chosen by the system designer strongly influences this last component of power. Values listed in this section for large buffer terminated entries reflect a termination network as illustrated in Figure 11-8 and are consistent with specifications for the MC68040. For additional termination schemes, refer to AN1051, *Transmission Line Effects in PCB Applications*, or AN1061, *Reflecting on Transmission Line Effects*.

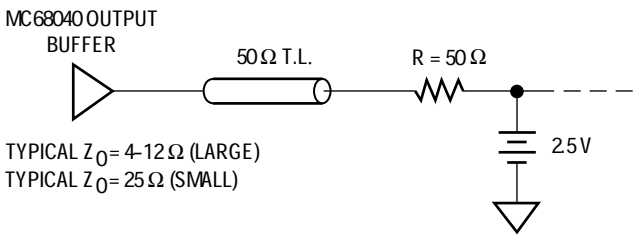


Figure 11-8. MC68040 Termination Network

If a designer uses alternative standard termination methods, such as RC termination network (see Figure 11-9), Thévenin termination network (not illustrated), or no termination method at all, which is not recommended, then the power dissipation of the MC68040 will be significantly less than the large buffer terminated values. For termination networks other than that illustrated in Figure 11-31, the designer must calculate the component of power dissipated in the output buffer and add this value to the small buffer unterminated value.

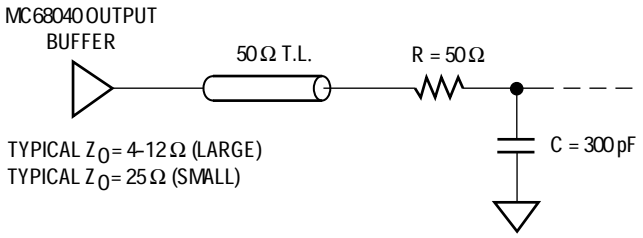


Figure 11-9. Typical Configuration for RC Termination Network

The following paragraphs describe how the large buffer terminated values were calculated. The MC68040 termination network causes current flow through the output buffer of the MC68040, regardless of whether the MC68040 is driving a logic one or a logic zero. The following equation gives the large buffer termination network power dissipation for a given pin:

$$I = (V \div (R + Z_o)) + 5\ mA$$

$$P = I^2 R_{eff}$$

R_{eff} is the effective average output resistance, including typical pullup resistance, typical pulldown resistance, and a duty cycle average of how often the pin is high, low, or three-stated. Typical values for Z_o are 6 Ω for large buffer low output, 12 Ω for large buffer high output, and 25 Ω for small buffer output. Using these values and duty cycle assumptions based on sequential burst write cycles, R_{eff} calculates to 7.7 Ω for the MC68040 large buffer mode and 25 Ω for the small buffer mode.

Maximum termination current in the large buffer mode occurs for output:

Low: $I_{tl} = (2.5\ V \div (50 + 6\ \Omega)) + 5\ mA = 49.6\ mA$

High: $I_{th} = (2.75\ V \div (50 + 12\ \Omega)) + 5\ mA = 50.8\ mA$

Maximum power dissipation in the large buffer mode occurs for output:

Low: $P_{llb} = I^2R = (49.6\ mA)^2 \times 6\ \Omega = 14.8\ mW$

High: $P_{hlb} = I^2R = (50.8\ mA)^2 \times 12\ \Omega = 30.1\ mW$

Similar calculations for unterminated small buffers yield:

$$I = 5\ mA\ (by\ spec)$$

and

$$P = I^2R = (5\ mA)^2 \times 25\ \Omega$$

so

$$P_{hsb} = 0.625\ mW$$
$$P_{lsb} = 0.625\ mW$$

Assuming that the duty cycle of output j is driving a valid logic value instead of being three-stated as given by DC_j, then the following equation approximates total average power dissipation in the output buffers:

$$I_{Total} = \sum_{j=1}^{\text{Number of Outputs Used}} (I_j \times DC_j)^2 \times R_{effj}$$

I_j and Z_{oj} are calculated for every pin as illustrated above. In practice the above summation is carried out by groups of pins instead of individual pins.

Motorola has calculated the values for DC_j for typical situations. On an average clock there will be 37.8 pins high, 41.5 pins low, and 11.7 pins three-stated. The following examples demonstrate how to calculate the power dissipation that is added to small buffer power dissipation numbers, assuming a termination as illustrated in Figure 11-18.

a. For the numbers listed in this section in a large buffer design with no caching.

   $P = (\text{Number of Pins High}) \times (P_{hlb}) + (\text{Number of Pins Low}) \times (P_{llb})$

   $= 37.8\ \text{Pins} \times 30.1\ \text{mW per Pin} + 41.5\ \text{Pins} \times 14.8\ \text{mW per Pin}$

   $= 1.75\ W$

b. For a single bus master system in a large buffer design with no caching or snooping and only standard features (i.e., TLN, UPA, $\overline{BR}$, $\overline{BB}$, $\overline{LOCK}$, $\overline{LOCKE}$, $\overline{CIOUT}$, $\overline{TIP}$, $\overline{MI}$, TDO, $\overline{IPEND}$, PST not used):

   $P = (\text{Number of Pins High}) \times (P_{hlb}) + (\text{Number of Pins Low}) \times (P_{llb})$

   $= 29.8\ \text{Pins} \times 30.1\ \text{mW per Pin} + 34.5\ \text{Pins} \times 14.8\ \text{mW per Pin}$

   $= 1.41\ W$

c. For the example b system with copyback caching, assuming 85% cache hit rate:

   $P = (29.8\ \text{Pins} \times 30.1\ \text{mW per Pin} + 34.5\ \text{Pins} \times 14.8\ \text{mW per Pin}) \times (1 - 0.85)$

   $= 0.21\ W$

d. For the example c system running the data bus in small buffer mode with other outputs in large buffer mode terminated:

P = (Number of Pins Large Buffer High) × (P_{hlb}) + (Number of Pins Large Buffer Low) × (P_{llb}) + (Number of Pins Small Buffer High) × (P_{hsb}) + (Number of Pins Small Buffer Low) × (P_{lsb})

= 19.1 Pins × 30.1 mW per Pin + 23.8 Pins × 14.8 mW per Pin + 10.7 Pins × 0.625 mW per Pin + 10.7 Pins × 0.625 mW per Pin

= 0.94 W × (1 – 0.85)

= 0.14 W

11.9.1 Still Air

In this study, a small sample of MC68040 packages was tested in free-air cooling with no heat sink. Measurements showed that the average θ_{JA} was 22.8°C/W with a standard deviation of 0.44°C/W. The test was performed with approximately 6 W of power being dissipated from within the package. The test determined that θ_{JA} decreases slightly for the increasing power dissipation range possible. Therefore, since the variance in θ_{JA} within the possible power dissipation range is negligible, it can be assumed for calculation purposes that θ_{JA} is valid at all power levels. Using the previous equations, Table 11-2 lists the results of a maximum power dissipation at maximum θ_{JC} with no heat sink or airflow (see Table 11-1 to calculate other power dissipation values). The ambient temperature results illustrate the need to implement some type of thermal management to obtain a reasonable maximum ambient temperature.

Table 11-2. Thermal Parameters with No Heat Sink or Airflow

MHz	Defined Parameters			Measured	Calculated		
	P_D	T_J	θ_{JC}	θ_{JA}	θ_{CA} ($\theta_{JA} - \theta_{JC}$)	T_C ($T_J - P_D \times \theta_{JC}$)	T_A ($T_J - P_D \times \theta_{JA}$)
MC68040							
25	6.3	110 °C	3	22.8	19.8	91.1	–33.64
25	6.6	110 °C	3	22.8	19.8	90.2	–40.48
25	8.6	110 °C	3	22.8	19.8	84.2	–86.08
33	7.7	110 °C	3	22.8	19.8	86.9	–65.56
33	8.0	110 °C	3	22.8	19.8	86.0	–72.40
33	10.0	110 °C	3	22.8	19.8	80.0	–118.00
MC68LC040 and MC68EC040							
20	4	110 °C	3	22.8	19.8	98	18.8
25	5	110 °C	3	22.8	19.8	95	–4
33	6.3	110 °C	3	22.8	19.8	91.1	–33.64

11.9.2 Forced Air

In this study, a small sample of MC68040 packages was tested in forced-air cooling in a wind tunnel with no heat sink. The test was performed with approximately 6 W of power being dissipated from within the package. As previously mentioned, since the variance in θ_{JA} within the possible power range is negligible, it can be assumed for calculation purposes that θ_{JA} is constant at all power levels. Using the previous equations, Table 11-3 lists the results of the maximum power dissipation at maximum θ_{JC} with airflow and no heat sink for the MC68040, and Table 11-4 lists the results for the MC68LC040 and MC68EC040. Refer to Table 11-1 for calculating other power dissipation values.

Table 11-3. Thermal Parameters with Forced Airflow and No Heat Sink for the MC68040

	Thermal Mgmt. Technique	Defined Parameters			Measured	Calculated		
MHz	Airflow Velocity	P_D	T_J	θ_{JC}	θ_{JA}	θ_{CA}	T_C	T_A
25	100 LFM	6.3 W	110 °C	3 °C/W	12.7 °C/W	9.7 °C/W	91.1 °C	29.90 °C
25		6.6 W					90.2 °C	26.18 °C
25		8.6 W					84.9 °C	00.76 °C
33		7.7 W					86.9 °C	12.21 °C
33		8.0 W					86.0 °C	08.40 °C
33		10.0 W					80.0 °C	00.00°C
25	250 LFM	6.3 W	110 °C	3 °C/W	11.0 °C/W	8.0 °C/W	91.1 °C	40.70 °C
25		6.6 W					90.2 °C	37.40 °C
25		8.6 W					84.2 °C	15.40 °C
33		7.7 W					86.9 °C	25.30 °C
33		8.0 W					86.0 °C	22.00 °C
33		10.0 W					80.0 °C	00.00 °C
25	500 LFM	6.3 W	110 °C	3 °C/W	9.9 °C/W	6.9 °C/W	91.1 °C	47.63 °C
25		6.6 W					90.2 °C	44.66 °C
25		8.6 W					84.2 °C	24.86 °C
33		7.7 W					86.9 °C	33.77 °C
33		8.0 W					86.0 °C	30.80 °C
33		10.0 W					80.0 °C	11.00 °C
25	750 LFM	6.3 W	110 °C	3 °C/W	9.5 °C/W	6.5 °C/W	91.1 °C	50.15 °C
25		6.6 W					90.2 °C	47.30 °C
25		8.6 W					84.2 °C	28.30 °C
33		7.7 W					86.9 °C	36.85 °C
33		8.0 W					86.0 °C	34.00 °C
33		10.0 W					80.0 °C	15.00 °C
25	1000 LFM	6.3 W	110 °C	3 °C/W	9.3 °C/W	6.3 °C/W	91.1 °C	51.41 °C
25		6.6 W					90.2 °C	48.62 °C
25		8.6 W					84.2 °C	30.02 °C
33		7.7 W					86.9 °C	38.39 °C
33		8.0 W					80.0 °C	17.00 °C
33		10.0 W					81.8 °C	22.58 °C

Table 11-4. Thermal Parameters with Forced Airflow and No Heat Sink for the MC68LC040 and MC68EC040

	Thermal Mgmt. Technique	Defined Parameters			Measured	Calculated		
MHz	Airflow Velocity	P_D	T_J	θ_{JC}	θ_{JA}	θ_{CA}	T_C	T_A
20 25 33	100 LFM	4 W 5 W 6.3W	110 °C	3 °C/W	12.7 °C/W	9.7 °C/W	98 °C 95 °C 91.1 °C	59.2 °C 46.5 °C 29.9 °C
20 25 33	250 LFM	4 W 5 W 6.3W	110 °C	3 °C/W	11 °C/W	8 °C/W	98 °C 95 °C 91.1 °C	66 °C 55 °C 40.70 °C
20 25 33	500 LFM	4 W 5 W 6.3W	110 °C	3 °C/W	9.9 °C/W	6.9 °C/W	98 °C 95 °C 91.1 °C	70.4 °C 60.5 °C 47.63 °C
20 25 33	750 LFM	4 W 5 W 6.3W	110 °C	3 °C/W	9.5 °C/W	6.5 °C/W	98 °C 95 °C 91.1 °C	72 °C 62.5 °C 50.15 °C
20 25 33	1000 LFM	4 W 5 W 6.3W	110 °C	3 °C/W	9.3 °C/W	6.3 °C/W	98 °C 95 °C 91.1 °C	72.8 °C 63.5 °C 51.41 °C

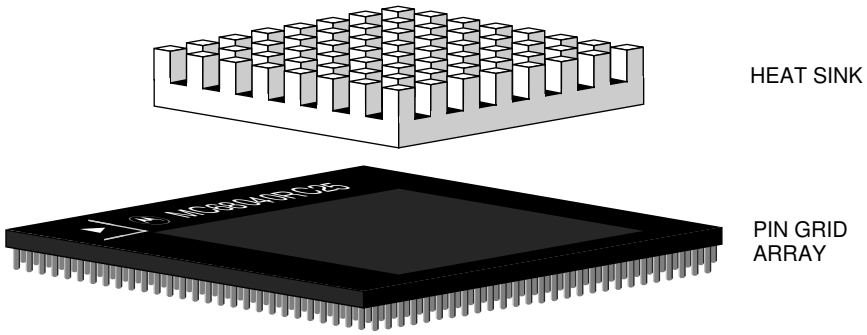
Reviewing the maximum ambient operating temperatures illustrates that using an all small buffer configuration of the MC68040 with a relatively small amount of airflow (100 LFM) achieves a 0–70 °C ambient operating temperature. However, depending on the output buffer configuration and available forced-air cooling, additional thermal management techniques may be required.

11.9.3 With Heat Sink

The designer must consider many factors in choosing a heat sink: heat-sink size and composition, method of attachment, and choice of a dry or wet (i.e., thermal grease) connection. The following paragraphs discuss the relationship of these decisions to the thermal performance of the design noticed during experimentation.

The heat-sink size is one of the most significant parameters to consider in the selection of a heat sink. Obviously a larger heat sink provides better cooling. Under forced-air conditions as low as 100 LFM, the difference between the θ_{CA} is very small (0.4 °C/W or less). This difference continues to decrease as the forced airflow increases.

The area of this example heat-sink base perimeter is 1.8" × 1.8", with a height of 0.65". The heat-sink used a pin-fin (i.e., bed-of-nails) design composed of aluminum alloy. Figure 11-32 illustrates the heat sink, which can be obtained through Thermalloy, Inc.



NOTE: Do not cover up microprocessor markings with an adhesive mounted heat sink.

Figure 11-10. Heat Sink with Adhesive

All pin-fin heat sinks tested were made from extrusion aluminum products. The planar face of the heat-sink matting to the package should have a good degree of planarity; if it has any curvature, the curvature should be convex at the central region of the heat-sink surface to provide intimate physical contact to the PGA surface. This heat sinks meet this criteria. Nonplanar, concave curvature in the central regions of the heat sink results in poor thermal contact to the package.

Although there are several ways to attach a heat sink to the package, it is easiest to use a demountable heat-sink attachment called “E-Z attach for PGA packages” (see Figure 11-33). A steel spring clamps the heat sink and the package to a plastic frame. Besides the height of the heat sink and plastic frame, no additional height is added to the package. The interface between the ceramic package and the aluminum heat sink was evaluated for both dry and wet interfaces in still air. The thermal grease reduced the θ_{CA} quite significantly (about 2.5 °C/W) in still air. An attachment with thermal grease provided about the same thermal performance as if a thermal epoxy had been used.

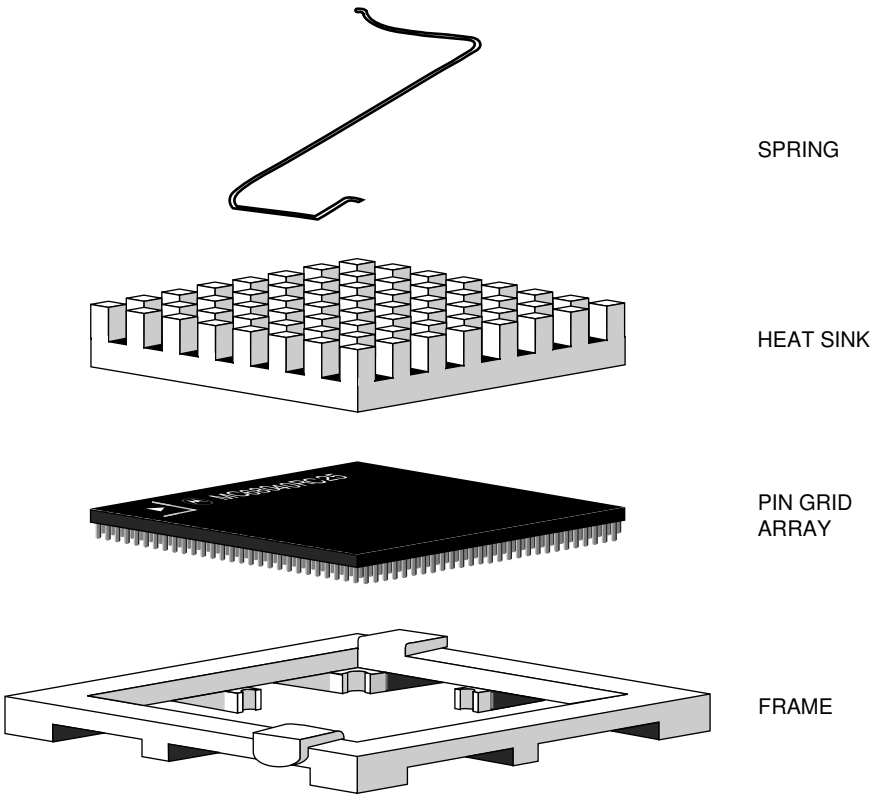


Figure 11-11. Heat Sink with Attachment

In the specification provided by Thermalloy, Inc., a chart illustrates the heat-sink temperature rise above ambient versus heat dissipated. This chart applies if no airflow is used with the heat-sink. Table 11-5 lists the calculations based on this chart.

Table 11-5. Thermal Parameters with Heat Sink and No Airflow

	Thermal Mgmt. Technique	Defined Parameters			Heat-Sink Spec.	Calculated	
MHz	Airflow Velocity	P_D	T_J	θ_{JC}	T_C–T_A	T_C	T_A
MC68040							
25	0	6.3 W	110 °C	3 °C/W	64.4 °C	91.1 °C	26.7 °C
25	0	6.6 W	110 °C	3 °C/W	66.8 °C	90.2 °C	23.4 °C
25	0	8.6 W	110 °C	3 °C/W	82.8 °C	84.2 °C	1.4 °C
33	0	7.7 W	110 °C	3 °C/W	75.6 °C	86.9 °C	11.3 °C
33	0	8.0 W	110 °C	3 °C/W	78.0 °C	86.0 °C	8.0 °C
33	0	10.0 W	110 °C	3 °C/W	94.0 °C	80.0 °C	–14.0 °C
MC68LC040 and MC68EC040							
20	0	4.0 W	110 °C	3 °C/W	45.0 °C	98.0 °C	53.0 °C
25	0	5.0 W	110 °C	3 °C/W	54.0 °C	95.0 °C	41.0 °C
33	0	6.3 W	110 °C	3 °C/W	64.4 °C	91.1 °C	26.7 °C

11.9.4 With Heat Sink and Forced Air

In the specification provided by Thermalloy, Inc., a chart illustrates the air velocity versus thermal resistance. This chart applies if airflow is used with the heat sink. Table 11-6 lists the calculations based on this chart.

Table 11-6. Thermal Parameters with Heat Sink and Airflow

	Thermal Mgmt. Technique	Defined Parameters			Heat-Sink Spec.	Calculated		
MHz	Airflow Velocity	P_D	T_J	θ_{JC} MAX.	θ_{CA}	θ_{JA}	T_C	T_A
MC68040								
25	200 LFM	6.3 W	110 °C	3 °C/W	4.25 °C/W	7.25 °C/W	91.1 °C	64.3 °C
25		6.6 W					90.2 °C	62.2 °C
25		8.6 W					84.2 °C	47.7 °C
33		7.7 W					86.9 °C	54.2 °C
33		8.0 W					86.0 °C	52.0 °C
33		10.0W					80.0 °C	37.5 °C
25	400 LFM	6.3 W	110 °C	3 °C/W	2.30 °C/W	5.25 °C/W	91.1 °C	76.9 °C
25		6.6 W					90.2 °C	75.4 °C
25		8.6 W					84.2 °C	64.9 °C
33		7.7 W					86.9 °C	69.6 °C
33		8.0 W					86.0 °C	68.0 °C
33		10.0W					80.0 °C	57.5 °C
25	600 LFM	6.3 W	110 °C	3 °C/W	1.50 °C/W	4.50 °C/W	91.1 °C	81.7 °C
25		6.6 W					90.2 °C	80.3 °C
25		8.6 W					84.2 °C	71.3 °C
33		7.7 W					86.9 °C	75.4 °C
33		8.0 W					86.0 °C	74.0 °C
33		10.0W					80.0 °C	65.0 °C
25	800 LFM	6.3 W	110 °C	3 °C/W	1.25 °C/W	4.25 °C/W	91.1 °C	83.2 °C
25		6.6 W					90.2 °C	82.0 °C
25		8.6 W					84.2 °C	73.5 °C
33		7.7 W					86.9 °C	77.3 °C
33		8.0 W					86.0 °C	76.0 °C
33		10.0W					80.0 °C	67.5 °C
MC68LC040 and MC68EC040								
20	200 LFM	4.0 W	110 °C	3 °C/W	4.25 °C/W	7.25 °C/W	98.0 °C	81.0 °C
25		5.0 W					95.0 °C	73.8 °C
33		6.3 W					91.1 °C	64.3 °C
20	400 LFM	4.0 W	110 °C	3 °C/W	2.30 °C/W	5.25 °C/W	98.0 °C	89.0 °C
25		5.0 W					95.0 °C	83.8 °C
33		6.3 W					91.1 °C	76.9 °C
20	600 LFM	4.0 W	110 °C	3 °C/W	1.50 °C/W	4.50 °C/W	98.0 °C	92.0°C
25		5.0 W					95.0 °C	87.5 °C
33		6.3 W					91.1 °C	81.7 °C
20	800 LFM	4.0 W	110 °C	3 °C/W	1.25 °C/W	4.25 °C/W	98.0 °C	93.0°C
25		5.0 W					95.0 °C	88.8 °C
33		6.3 W					91.1 °C	83.2 °C

SECTION 12
ORDERING INFORMATION AND MECHANICAL DATA

This section contains the ordering information, pin assignments, and package dimensions of the MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V. The pin assignments depicted in this section for the MC68LC040 also serve as the pin assignments for the MC68040V with a few differences as indicated.

12.1 ORDERING INFORMATION

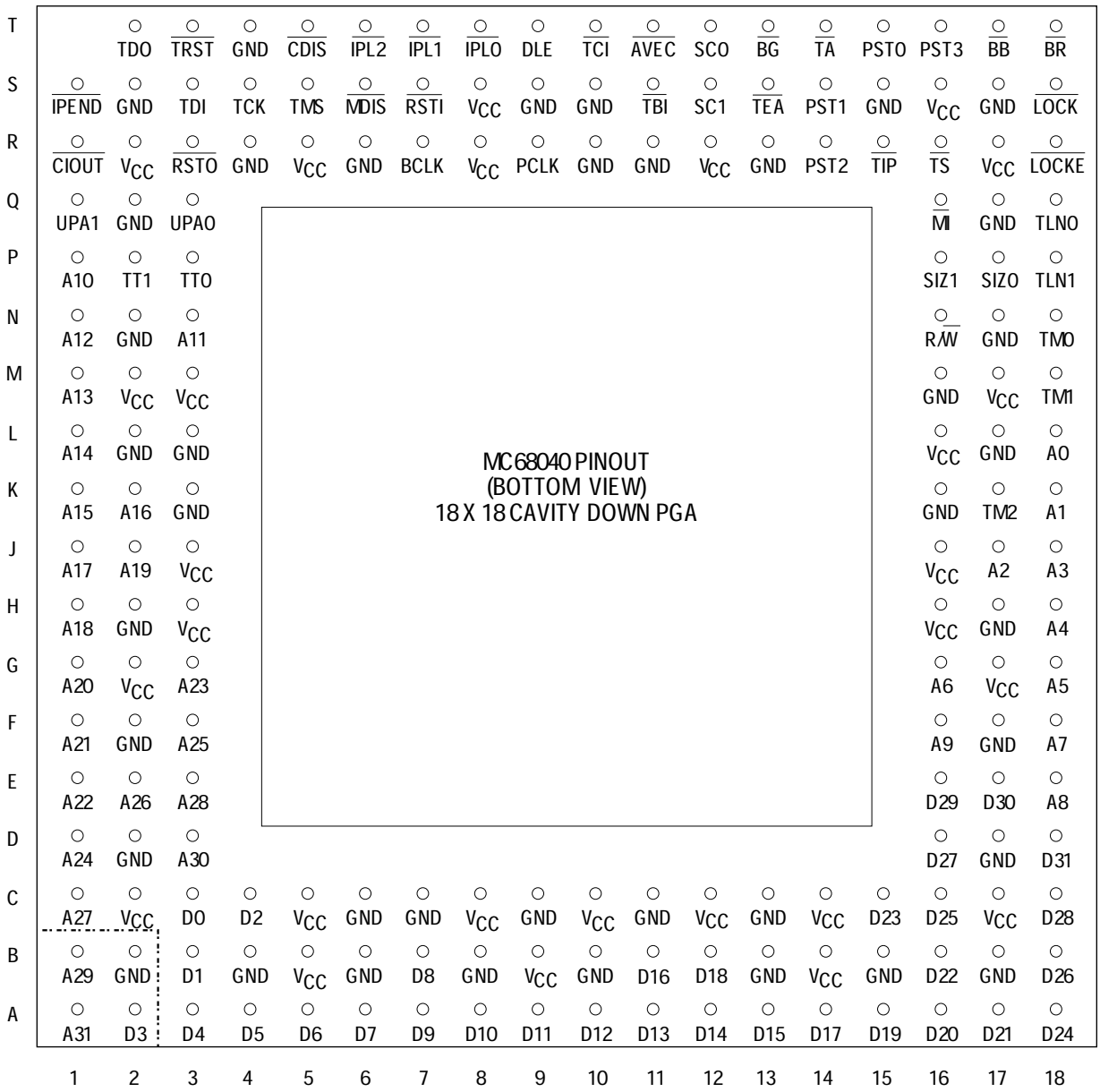
The following table provides ordering information pertaining to the MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V package types, frequencies, temperatures, and Motorola order numbers.

Package Type	Frequency	Maximum Junction Temperature	Minimum Ambient Temperature	Order Number
Pin Grid Array RC Suffix	20 MHz	110 °C	0 °C	MC68LC040RC20B MC68EC040RC20B
Pin Grid Array RC Suffix	25 MHz	110 °C	0 °C	MC68040RC25 MC68LC040RC25B MC68EC040RC25B MC68040RC25V
Pin Grid Array RC Suffix	33 MHz	110 °C	0 °C	MC68040RC33 MC68LC040RC33B MC68EC040RC33B MC68040RC33V
184 Pin QFP FE Suffix	20 MHz	110 °C	0 °C	MC68LC040FE20B MC68EC040FE20B
184 Pin QFP FE Suffix	25 MHz	110 °C	0 °C	MC68LC040FE25B MC68EC040FE25B MC68040FE25V
184 Pin QFP FE Suffix	33 MHz	110 °C	0 °C	MC68LC040FE33B MC68EC040FE33B MC68040FE33V

12.2 PIN ASSIGNMENTS

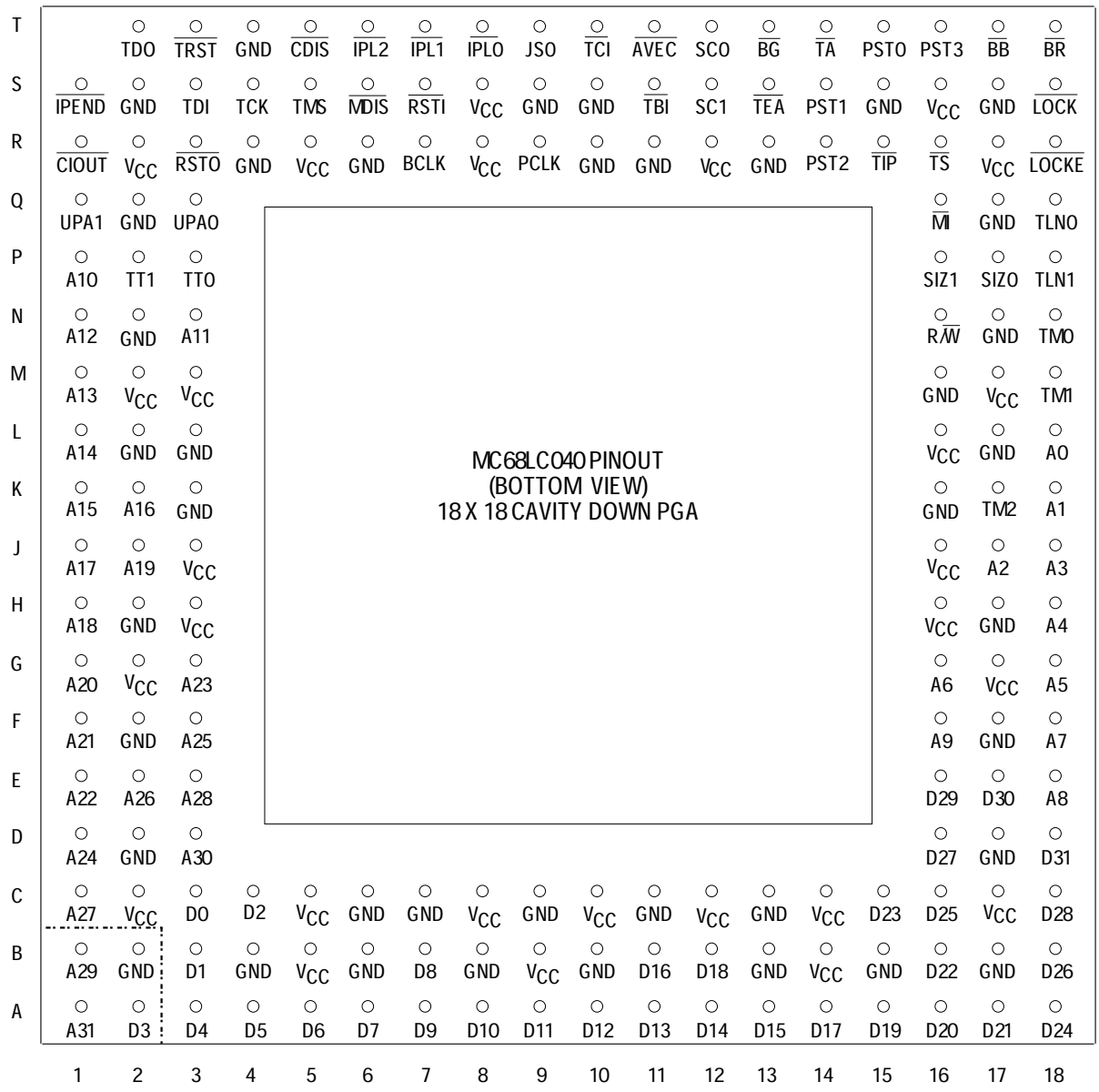
The following are the pin assignments for the MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V package types.

12.2.1 MC68040 Pin Grid Array



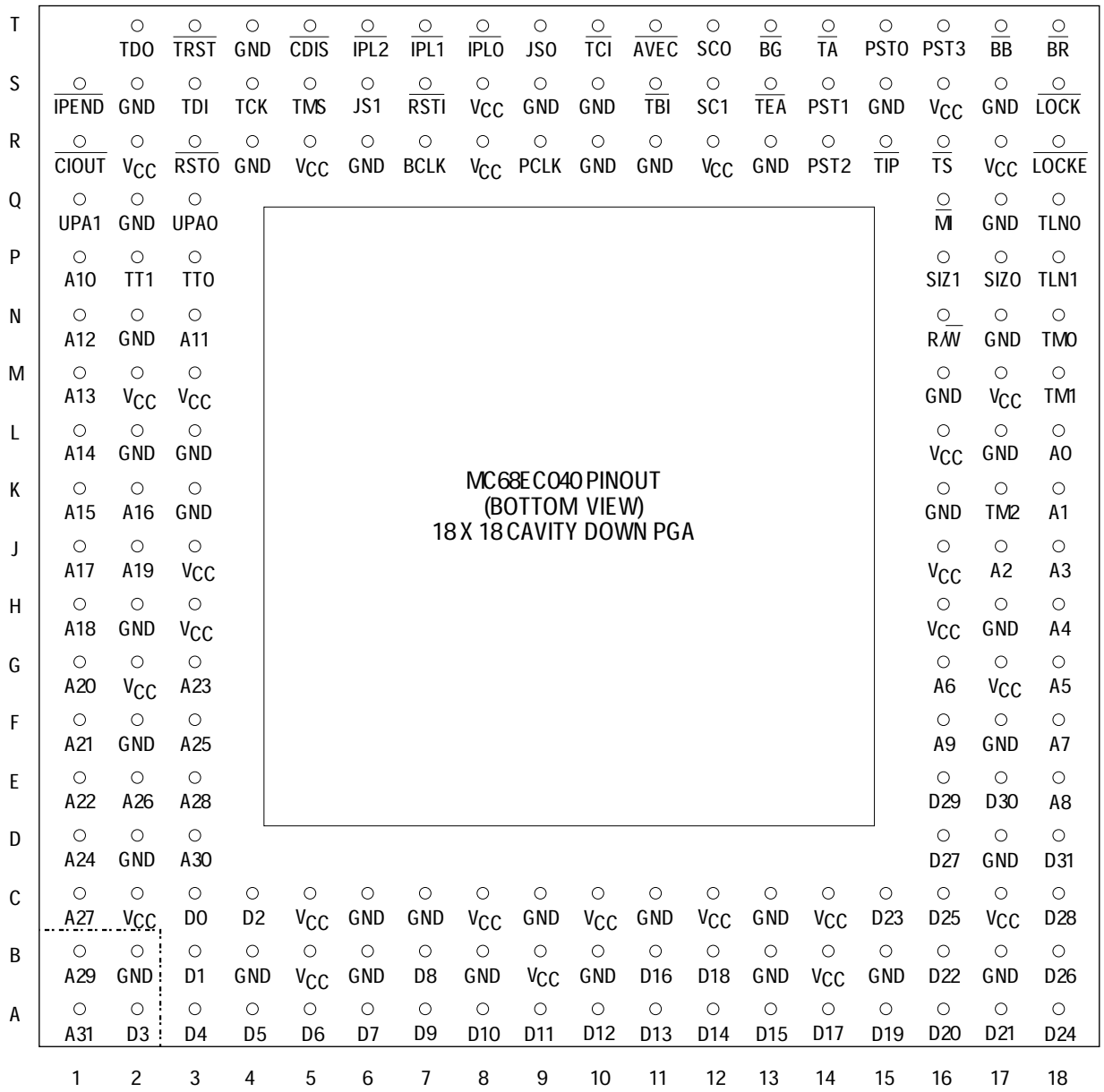
Pin Group	GND	V_{CC}
PLL	S9, R6, R10	R8, S8
Internal Logic	C6, C7, C9, C11, C13, K3, K16, L3, M16, R4, R11, R13, S6, S10, T4	C5, C8, C10, C12, C14, H3, H16, J3, J16, L16, M3, R5, R12
Output Drivers	B2, B4, B6, B8, B10, B13, B15, B17, D2, D17, F2, F17, H2, H17, L2, L17, N2, N17, Q2, Q17, S2, S15, S17	B5, B9, B14, C2, C17, G2, G17, M2, M17, R2, R17, S16

12.2.2 MC68LC040 Pin Grid Array



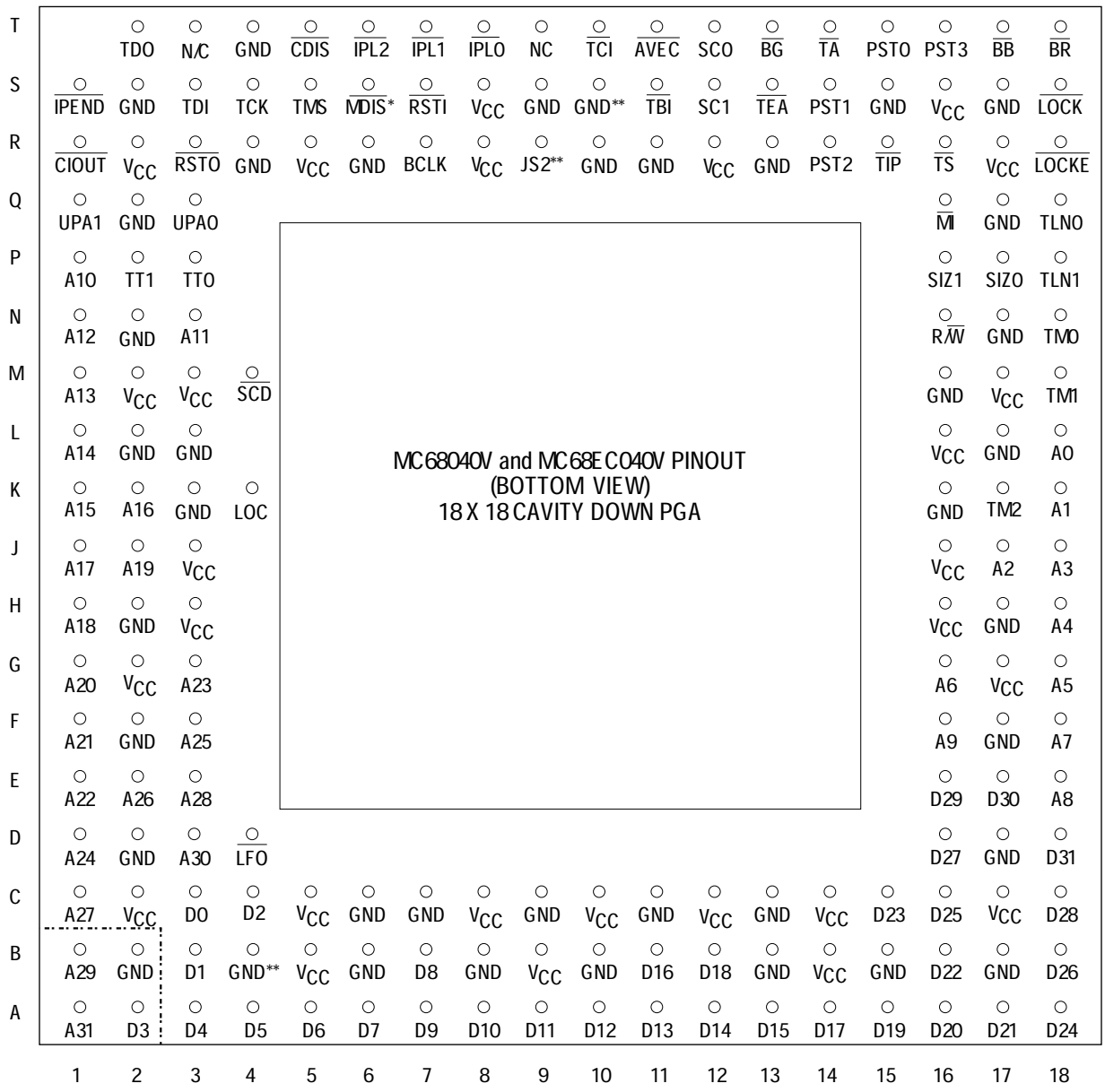
Pin Group	GND	V_{CC}
PLL	S9, R6, R10	R8, S8
Internal Logic	C6, C7, C9, C11, C13, K3, K16, L3, M16, R4, R11, R13, S6, S10, T4	C5, C8, C10, C12, C14, H3, H16, J3, J16, L16, M3, R5, R12
Output Drivers	B2, B4, B6, B8, B10, B13, B15, B17, D2, D17, F2, F17, H2, H17, L2, L17, N2, N17, Q2, Q17, S2, S15, S17	B5, B9, B14, C2, C17, G2, G17, M2, M17, R2, R17, S16

12.2.3 MC68EC040 Pin Grid Array



Pin Group	GND	V_{CC}
PLL	S9, R6, R10	R8, S8
Internal Logic	C6, C7, C9, C11, C13, K3, K16, L3, M16, R4, R11, R13, S6, 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

12.2.4 MC68040V and MC68EC040V Pin Grid Array

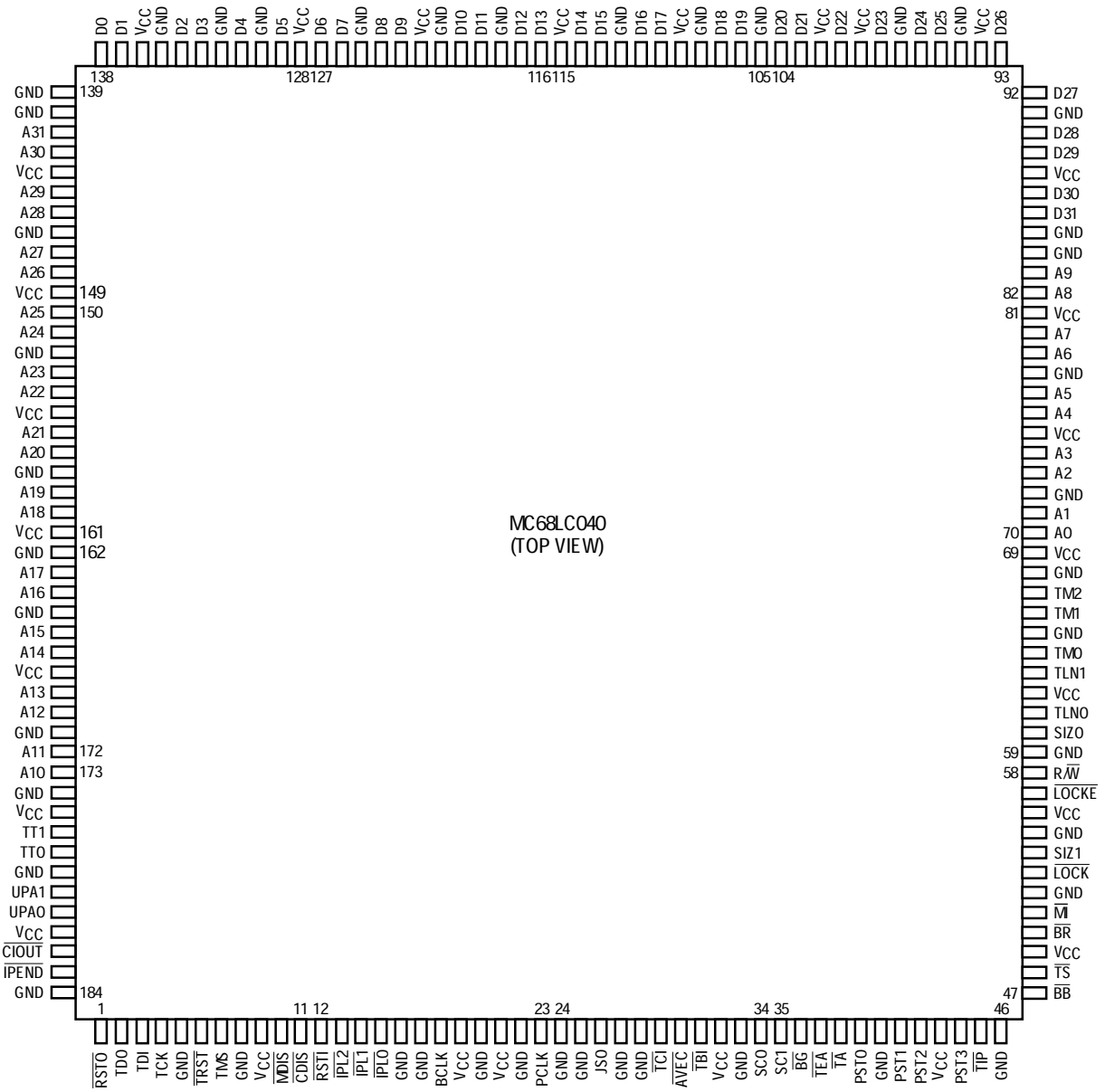


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T		TDO	N/C	GND	CDIS	IPL2	IPL1	IPL0	NC	TCI	AVEC	SCO	BG	TA	PSTO	PST3	BB	BR
S	IPEND	GND	TDI	TCK	TMS	MDIS*	RSTI	V_{CC}	GND	GND**	TBI	SC1	TEA	PST1	GND	V_{CC}	GND	LOCK
R	CIOUT	V_{CC}	RSTO	GND	V_{CC}	GND	BCLK	V_{CC}	JS2**	GND	GND	V_{CC}	GND	PST2	TIP	TS	V_{CC}	LOCKE
Q	UPA1	GND	UPA0													MI	GND	TLN0
P	A10	TT1	TT0													SIZ1	SIZ0	TLN1
N	A12	GND	A11													R/W	GND	TM0
M	A13	V_{CC}	V_{CC}	SCD												GND	V_{CC}	TM1
L	A14	GND	GND													V_{CC}	GND	A0
K	A15	A16	GND	LOC												GND	TM2	A1
J	A17	A19	V_{CC}													V_{CC}	A2	A3
H	A18	GND	V_{CC}													V_{CC}	GND	A4
G	A20	V_{CC}	A23													A6	V_{CC}	A5
F	A21	GND	A25													A9	GND	A7
E	A22	A26	A28													D29	D30	A8
D	A24	GND	A30	LFO												D27	GND	D31
C	A27	V_{CC}	D0	D2	V_{CC}	GND	GND	V_{CC}	GND	V_{CC}	GND	V_{CC}	GND	V_{CC}	D23	D25	V_{CC}	D28
B	A29	GND	D1	GND**	V_{CC}	GND	D8	GND	V_{CC}	GND	D16	D18	GND	V_{CC}	GND	D22	GND	D26
A	A31	D3	D4	D5	D6	D7	D9	D10	D11	D12	D13	D14	D15	D17	D19	D20	D21	D24

NOTES:
* On MC68EC040V this pin is called JS1.
** All these pins are in the JTAG scan chain. On an MC68040 design JS2 = GND; on an MC68060 design JS2 = CLK.

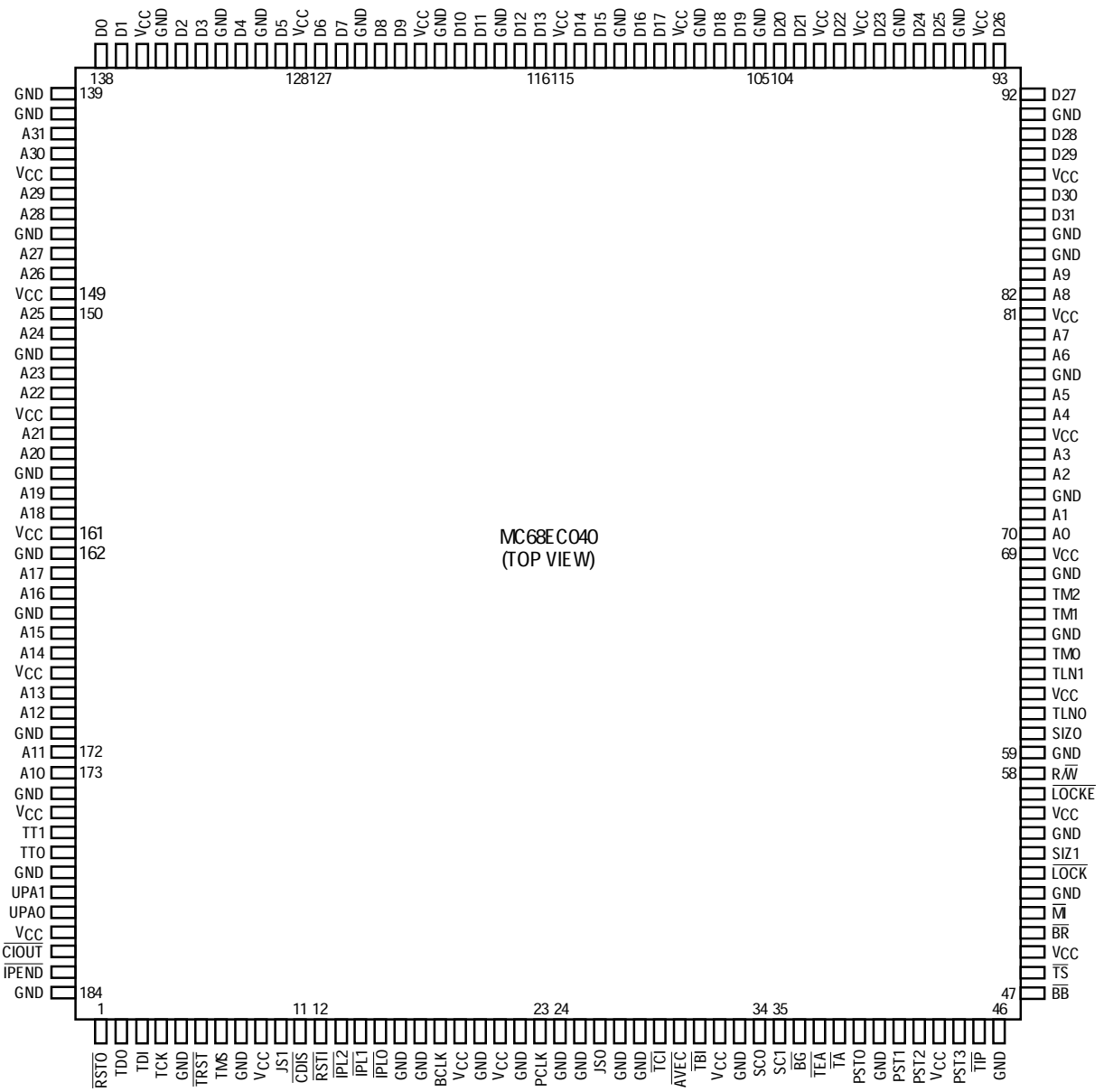
Pin Group	GND	V_{CC}
PLL	S9, R6, R10	R8, S8
Internal Logic	C6, C7, C9, C11, C13, K3, K16, L3, M16, R4, R11, R13, S10, T4	C5, C8, C10, C12, C14, H3, H16, J3, J16, L16, M3, R5, R12
Output Drivers	B2, B4, B6, B8, B10, B13, B15, B17, D2, D17, F2, F17, H2, H17, L2, L17, N2, N17, Q2, Q17, S2, S15, S17	B5,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

12.2.5 MC68LC040 Quad Flat Pack



Pin Group	GND	V_{CC}
PLL	17, 22, 24	19, 21
Internal Logic	5, 8, 10, 27, 28, 33, 55, 68, 95, 108, 121, 162, 130, 135, 174	9, 32, 56, 69, 81, 94, 100, 109, 122, 136, 149, 161, 175
Output Drivers	16, 20, 25, 40, 46, 52, 59, 65, 72, 78, 84, 85, 91, 98, 105, 112, 118, 125, 132, 139, 140, 146, 152, 158, 165, 171, 178, 184	43, 49, 62, 75, 88, 102, 115, 128, 143, 155, 168, 181

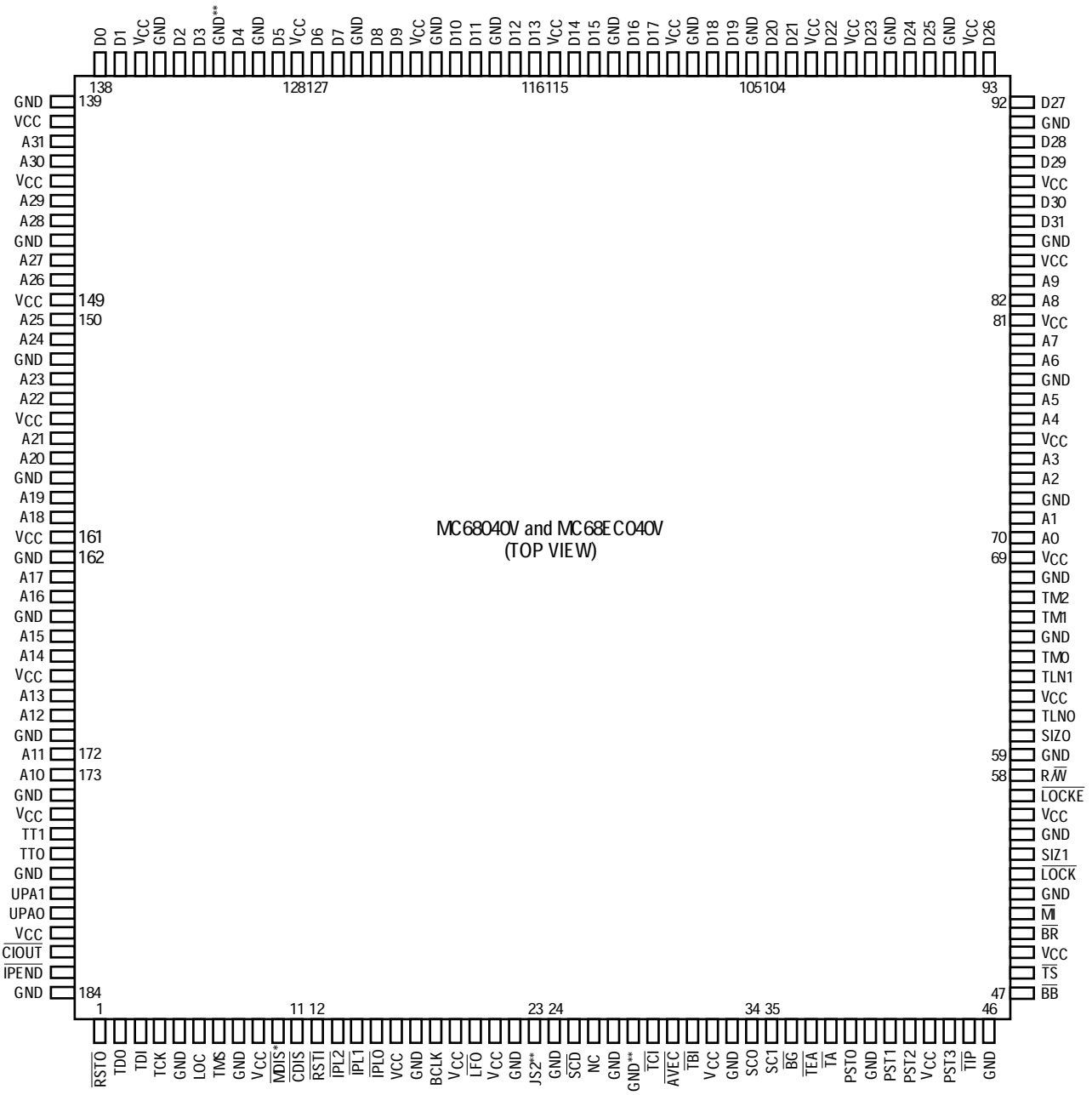
12.2.6 MC68EC040 Quad Flat Pack



MC68EC040 184 Pin QFP Pin Assignment

Pin Group	GND	V_{CC}
PLL	17, 22, 24	19, 21
Internal Logic	5, 8, 10, 27, 28, 33, 55, 68, 95, 108, 121, 162, 130, 135, 174	9, 32, 56, 69, 81, 94, 100, 109, 122, 136, 149, 161, 175
Output Drivers	16, 20, 25, 40, 46, 52, 59, 65, 72, 78, 84, 85, 91, 98, 105, 112, 118, 125, 132, 139, 140, 146, 152, 158, 165, 171, 178, 184	43, 49, 62, 75, 88, 102, 115, 128, 143, 155, 168, 181

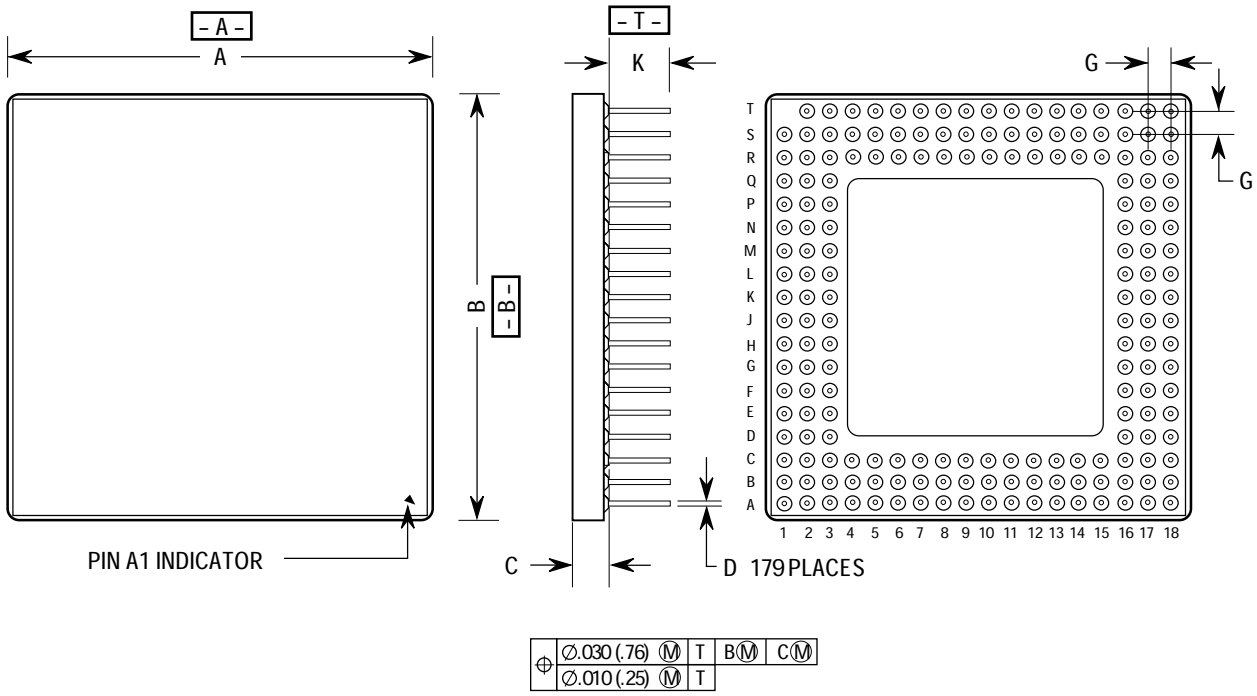
12.2.7 MC68040V and MC68EC040V Quad Flat Pack



NOTES:
* On MC68EC040V this pin is called JS1.
** All these pins are in the JTAG scan chain. On an MC68040 design JS2 = GND; on an MC68060 design JS2 = CLK.

12.3 MECHANICAL DATA

Figure 12-1 illustrates the MC68040, MC68LC040, and MC68EC040 PGA package dimensions. Figure 12-2 illustrates the MC68040, MC68LC040, and MC68EC040 QFP package dimensions. Due to space limitation, Figure 12-2 is represented by a general (smaller) package outline drawing rather than showing all 184 leads.



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	46.74	47.75	1.840	1.880
B	46.74	47.75	1.840	1.880
C	2.79	3.05	0.110	0.140
D	0.41	0.51	0.016	0.020
G	2.54 BSC		0.100 BSC	
K	3.81	4.32	0.150	0.170

Figure 12-1. PGA Package Dimensions

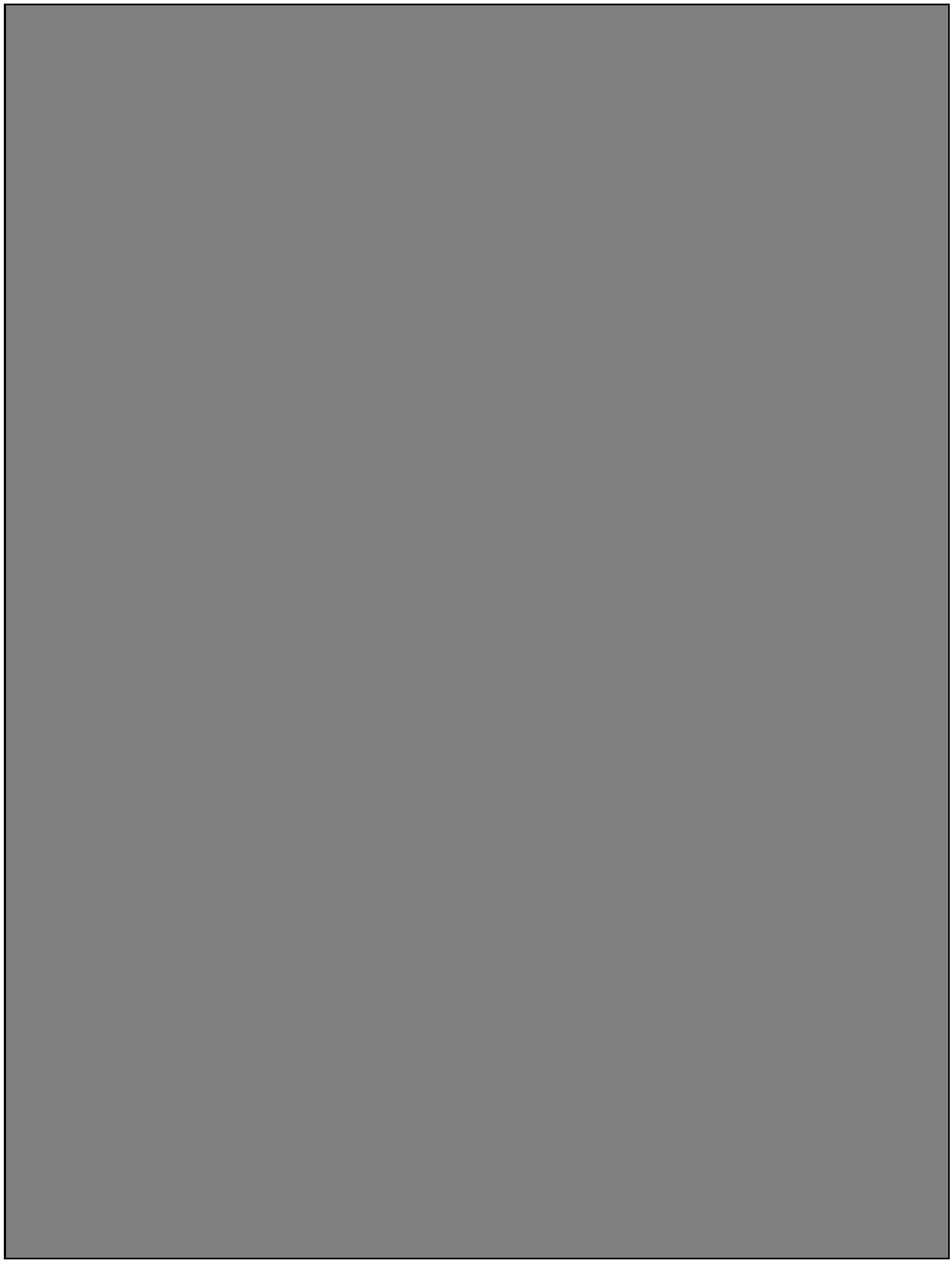


Figure 12-2. QFP Package Dimensions

APPENDIX A
MC68LC040

NOTE

All references to MC68LC040 also apply to the MC68040V. Refer to **Appendix C MC68040V and MC68EC040V** for more information on the MC68040V.

The MC68LC040 is Motorola's integer-only version of the MC68040 third-generation, M68000-compatible, high-performance, 32-bit microprocessor. The MC68LC040 is a virtual memory microprocessor with a highly integrated architecture that provides very high performance in a monolithic HCMOS device. On a single chip, the MC68LC040 integrates an MC68040-compatible integer unit and fully independent instruction and data demand-paged memory management units (MMUs), including independent 4-Kbyte instruction and data caches. A high degree of instruction execution parallelism is achieved through the use of a six-stage instruction pipeline, multiple internal buses, and a full internal Harvard architecture, including separate physical caches for both instruction and data accesses. The MC68LC040 also directly supports cache coherency in multimaster applications with dedicated on-chip bus snooping logic.

The MC68LC040 achieves its high performance through the use of the MC68040 integer unit. The six-stage pipeline operates on up to six instructions concurrent with MMU, cache, and bus controller operations. Multiple internal buses, separate data and instruction caches, and a sophisticated bus controller allow internal units to operate concurrently and decouple the MC68LC040 from the external bus. The internal caches and the decoupling of the external bus allow for an external memory subsystem to be built from slower and less expensive memories with minimal impact to the overall system performance. The potential for a low-cost system design with the price/performance of the MC68LC040 makes it a good choice for embedded microprocessor applications as well as central processor applications.

The MC68LC040 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 high level of performance is ideal for integer-intensive applications. The MC68LC040 is implemented in Motorola's latest HCMOS technology, providing an ideal balance between speed, power, and physical device size. Independent data and instruction MMUs control the main caches and the address translation caches (ATCs). The ATCs speed up logical-to-physical address translations by storing recently used translations. The bus snooper circuit ensures cache coherency in multimaster and multiprocessing applications. The MC68LC040 is pin compatible with the MC68040 and the MC68EC040. Figure A-1 illustrates a simplified block diagram of the MC68LC040.

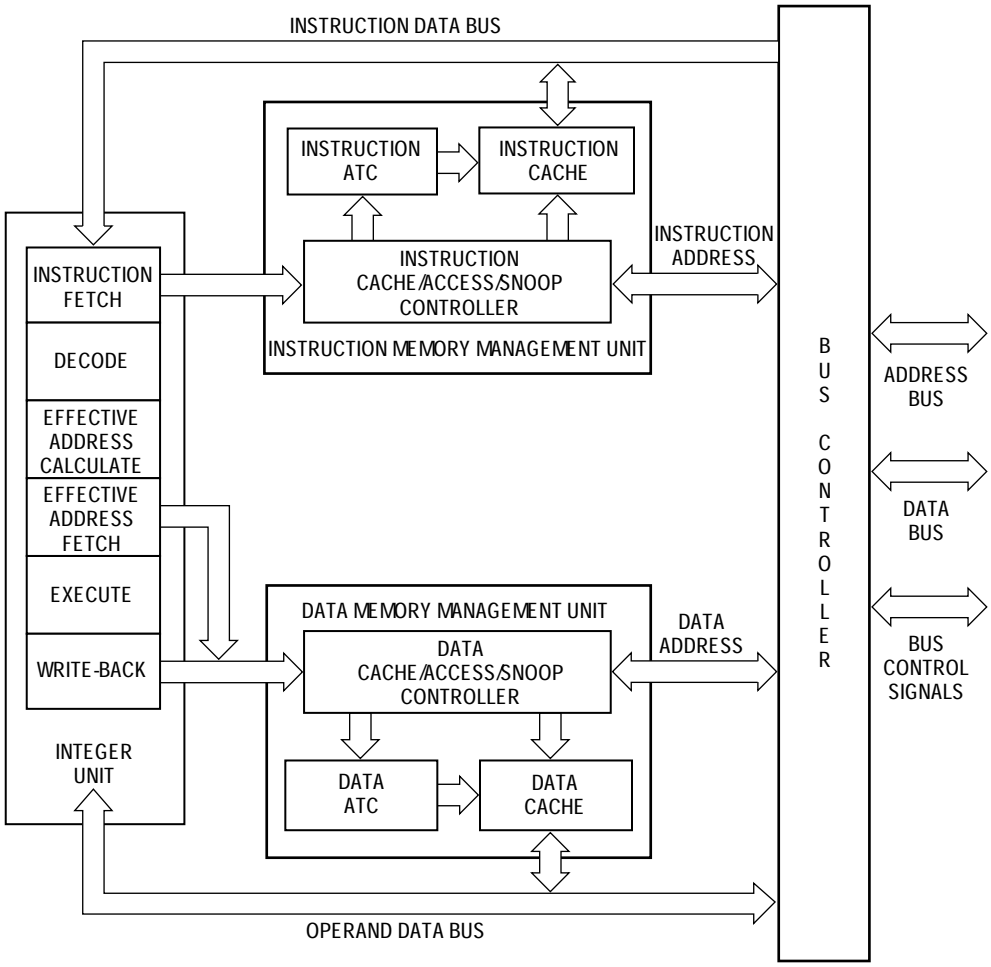


Figure A-1. MC68LC040 Block Diagram

The main features of the MC68LC040 include:

- 22 MIPS Integer Performance at 25 MHz
- Independent Instruction and Data MMUs
- 4-Kbyte Physical Instruction Cache and 4-Kbyte Physical Data Cache Accessible Simultaneously
- 32-Bit, Nonmultiplexed External Address and Data Buses with Synchronous Interface
- User-Object-Code Compatible with All M68000 Microprocessors
- Multimaster/Multiprocessor Support Via Bus Snooping
- Concurrent Integer Unit, MMU, Bus Controller, and Bus Snooper Operation Maximizes Throughput
- 4-Gbyte Direct Addressing Range
- Software Support Including Optimizing C Compiler and UNIX® System V Port

®UNIX is a registered trademark of AT&T Bell Laboratories.

With the exception of the floating-point unit (FPU) and its registers, the MC68LC040 programming model, data formats and types, instruction set (except all instructions beginning with an “F”), caches, and MMUs are the same as those described in **Section 1 Introduction** for the MC68040. Figures A-2 and A-3 illustrate the programming model and functional signal groups for the MC68LC040.

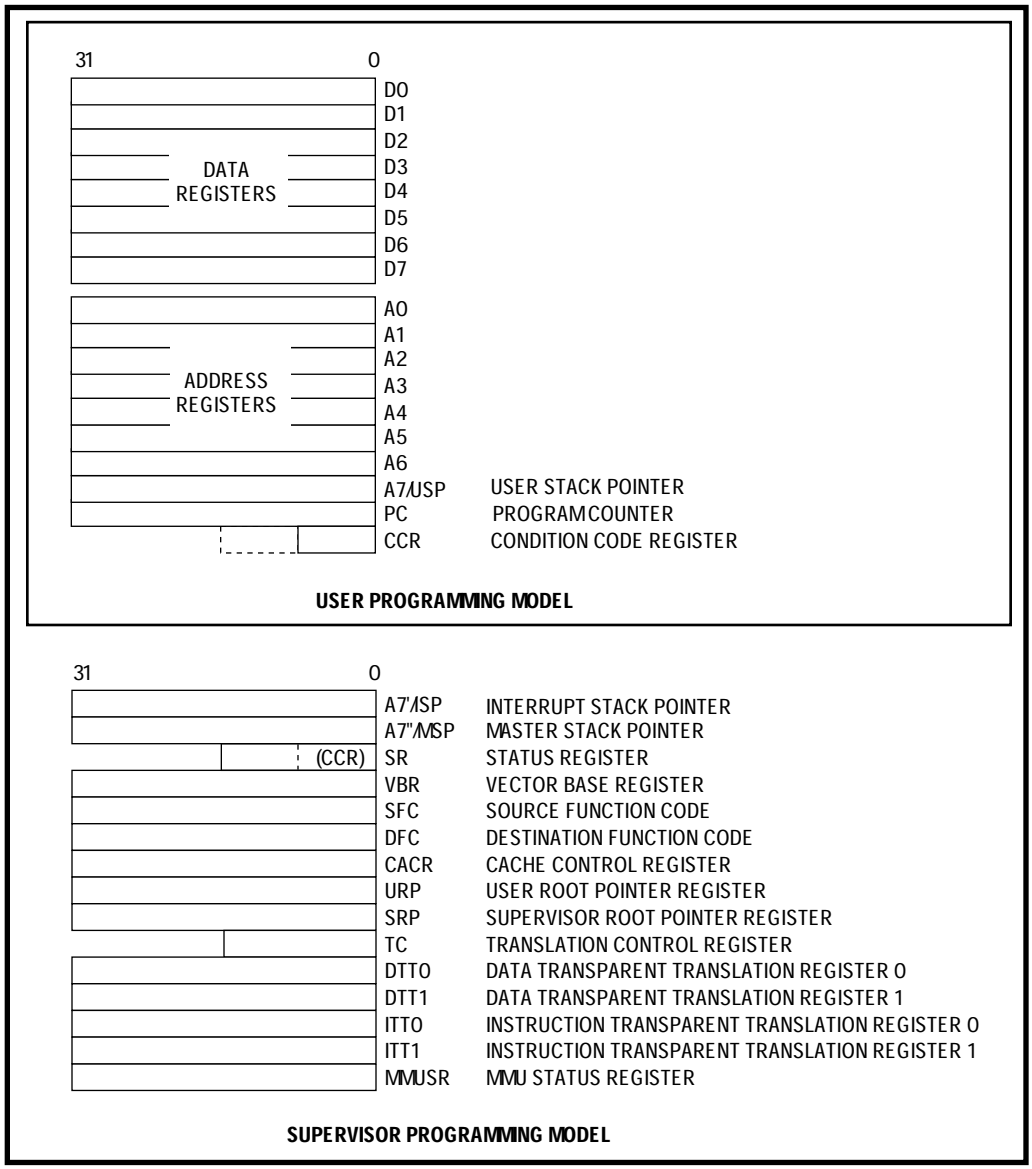


Figure A-2. MC68LC040 Programming Model

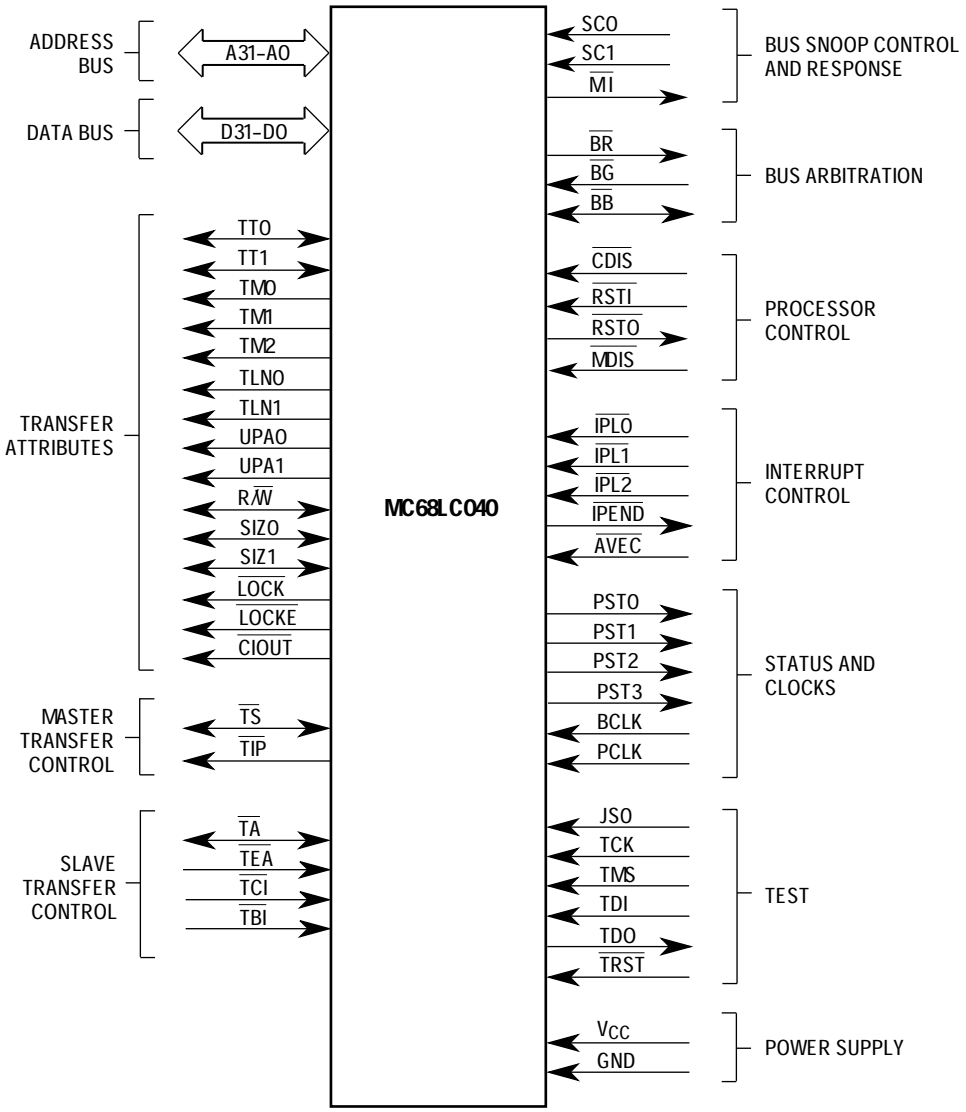


Figure A-3. MC68LC040 Functional Signal Groups

A.1 MC68LC040 DIFFERENCES

The following differences exist between the MC68LC040 and MC68040:

- The MC68LC040 does not implement the small output buffer impedance selection mode.
- The DLE pin name has been changed to JS0.
- The MC68LC040 does not implement the data latch (DLE) or multiplexed bus modes of operation. All timing and drive capabilities of the MC68LC040 are equivalent to those of the MC68040 in small output buffer impedance mode.
- The MC68LC040 does not contain an FPU, which causes unimplemented floating-point exceptions to occur using a new eight-word stack frame format.

A.2 INTERRUPT PRIORITY LEVEL ($\overline{IPL2}$–$\overline{IPL0}$)

The $\overline{IPL2}$–$\overline{IPL0}$ pins do not have any affect on the selection of output buffer impedance.

A.3 JTAG SCAN (JS0)

The MC68040 DLE pin name has been changed to JS0. During normal operation, the JS0 pin cannot float, it must be tied to GND or Vcc directly or through a resistor. During board testing, this pin retains the functionality of the JTAG scan of the MC68040 for compatibility purposes. Refer to **Section 6 IEEE 1149.1 Test Access Port (JTAG)** for details concerning IEEE *1149.1 Standard Test Access Port and Boundary Scan Architecture*.

A.4 DATA LATCH AND MULTIPLEXED BUS MODES

The MC68LC040 does not implement the data latch or multiplexed modes of operation. The $\overline{CDIS}$ pin is ignored at the rising edge of reset. All timing and drive capabilities of the MC68LC040 are equivalent to those of the MC68040 in small output buffer impedance mode.

A.5 FLOATING-POINT UNIT (FPU)

The FPU is not implemented on the MC68LC040. All floating-point instructions cause an unimplemented floating-point exception to be taken with a new eight-word stack frame (format $4). The stack frame contains the status register (SR), program counter (PC), vector offset, effective address of the operand (where applicable), and PC value of the unimplemented floating-point instruction.

A.5.1 Unimplemented Floating-Point Instructions and Exceptions

All legal MC68040 and MC68881/MC68882 floating-point instructions are defined as unimplemented floating-point instructions on the MC68LC040. These instructions generate a format $4 stack frame during exception processing before taking an F-line exception. These instructions trap as an F-line exception, and the F-line exception handler can emulate them in software to maintain user-object-code compatibility.

The MC68LC040 assists the emulation process by distinguishing unimplemented floating-point instructions from other unimplemented F-line instructions. To aid emulation, the effective address is calculated and saved in the format $4 stack frame. This simplifies and speeds up the emulation process by eliminating the need for the emulation routine to determine the effective address and by providing information required to emulate the instruction on the exception stack frame in the supervisor address space. However, the floating-point instruction can reside in user space; therefore, the floating-point unimplemented exception handler may need to access user instruction space. The following processing steps occur for an unimplemented floating-point instruction:

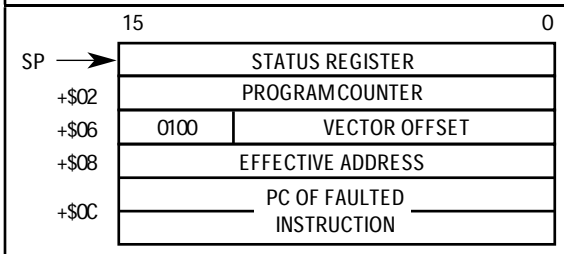
1. When an unimplemented floating-point instruction is encountered, the instruction is partially decoded, and the effective address is calculated, if required.
2. The processor waits for all previous integer instructions, write-backs, and associated exception processing to complete before beginning exception processing for the unimplemented floating-point instruction. Any access error that occurs in completing the write-backs causes an access error exception, and the resulting stack frame indicates a pending unimplemented floating-point instruction exception. The access error exception handler then completes the write-backs in software, 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 and T0). It creates a format $4 stack frame and saves the internal copy of the SR, PC, vector offset, calculated effective address, and PC value of the faulted instruction in the stack frame.

   The effective address field of the format $4 stack frame 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. The saved PC value is the logical address of the instruction that follows the unimplemented floating-point instruction. This value will be restored during RTE execution. The vector offset format number ($4) is used for this eight-word stack frame. 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 instruction is available for software emulation of floating-point instructions. 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. Refer to **Section 8 Exception Processing** for details about exception processing.

A.5.2 MC68LC040 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 MC68LC040 provides five different stack frames for exception processing and allows for an MC68040-specific stack frame. The set of frames includes four- and six-word stack frames, a four-word throwaway stack frame, an access error stack frame, and a new eight-word unimplemented floating-point stack frame. The stack frame that the MC68040 can generate and the MC68LC040 can process is the floating-point post-instruction stack frame. Refer to **Section 8 Exception Processing** for details about exception stack frames.

Eight-Word Stack Frame (Format $4)

Stack Frames	Exception Types	Stacked PC Points To
	• The MC68040 cannot generate or read this stack.	• Effective address field is the address of the faulted instruction operand.

When the MC68LC040 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 format of stack frame for any type of exception. The MC68LC040 does not generate the floating-point post-instruction stack frame. The MC68040 cannot accept the eight-word unimplemented stack frame. The MC68LC040 can handle all MC68040 stack frame formats.

A.6 MC68LC040 ELECTRICAL CHARACTERISTICS

The following paragraphs provide information on the maximum rating and thermal characteristics for the MC68LC040. This section is subject to change. For the most recent specifications, contact a Motorola sales office or complete the registration card at the end of this manual.

A.6.1 Maximum Ratings

Characteristic	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	V
Input Voltage	V_{in}	–0.5 to +7.0	V
Maximum Operating Junction Temperature	T_J	110	°C
Minimum Operating Ambient Temperature	T_A	0	°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. Reliablity of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either GND or V_{CC}).

A.6.2 Thermal Characteristics

Characteristic	Symbol	Value	Rating
Thermal Resistance, Junction to Case—PGA Package	θ_{JC}	3	°C/W

A.6.3 DC Electrical Specifications (V_{CC} = 5.0 Vdc ±5 %)

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, $\overline{BG}$, $\overline{CDIS}$, $\overline{MDIS}$, $\overline{IPLx}$, PCLK, $\overline{RSTI}$, SCx, $\overline{TBI}$, TLNx, $\overline{TCI}$, TCK, $\overline{TEA}$	I_{in}	20	20	µA
Hi-Z (Off-State) Leakage Current @ 0.5–2.4 V An, $\overline{BB}$, $\overline{CIOUT}$, Dn, $\overline{LOCK}$, $\overline{LOCKE}$, R/$\overline{W}$, SIZx, $\overline{TA}$, TDO, $\overline{TIP}$, TMx, TLNx, $\overline{TS}$, TTx, UPAx	I_{TSI}	20	20	µA
Signal Low Input Current, V_{IL} = 0.8 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
Capacitance*, V_{in} = 0 V, f = 1 MHz	C_{in}	—	25	pF

*Capacitance is periodically sampled rather than 100% tested.

A.6.4 Power Dissipation

Frequency	Watts
Maximum Values (V_{CC} = 5.25 V, T_A = 0°C)	
20 MHz	3.2
25 MHz	3.9
33 MHz	4.9
Typical Values (V_{CC} = 5 V, T_A = 25°C)*	
20 MHz	2.0
25 MHz	2.4
33 MHz	3.0

*This information is for system reliability purposes.

A.6.5 Clock AC Timing Specifications (see Figure A-4)

Num	Characteristic	20 MHz		25 MHz		33 MHz		Unit
		Min	Max	Min	Max	Min	Max	
	Frequency of Operation	16.67	20	16.67	25	16.67	33	MHz
1	PCLK Cycle Time	25	30	20	30	15	30	ns
2	PCLK Rise Time	—	1.7	—	1.7	—	1.7	ns
3	PCLK Fall Time	—	1.6	—	1.6	—	1.6	ns
4	PCLK Duty Cycle Measured at 1.5 V	48	52	47.5	52.5	46.67	53.33	%
4a*	PCLK Pulse Width High Measured at 1.5 V	12	13	9.5	10.5	7	8	ns
4b*	PCLK Pulse Width Low Measured at 1.5 V	12	13	9.5	10.5	7	8	ns
5	BCLK Cycle Time	50	60	40	60	30	60	ns
6,7	BCLK Rise and Fall Time	—	4	—	4	—	3	ns
8	BCLK Duty Cycle Measured at 1.5 V	40	60	40	60	40	60	%
8a*	BCLK Pulse Width High Measured at 1.5 V	20	30	16	24	12	18	ns
8b*	BCLK Pulse Width Low Measured at 1.5 V	20	30	16	24	12	18	ns
9	PCLK, BCLK Frequency Stability	—	1000	—	1000	—	1000	ppm
10	PCLK to BCLK Skew	—	9	—	9	—	n/a	ns

*Specification value at maximum frequency of operation.

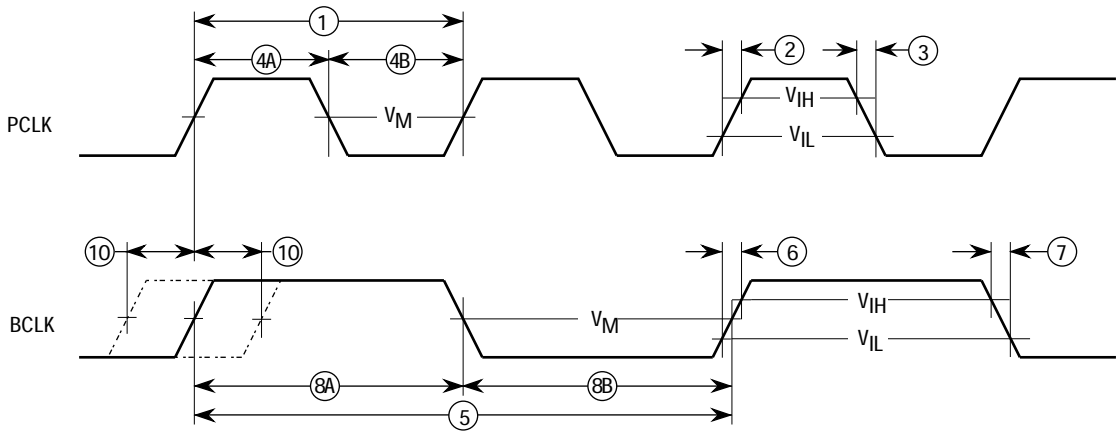


Figure A-4. Clock Input Timing Diagram

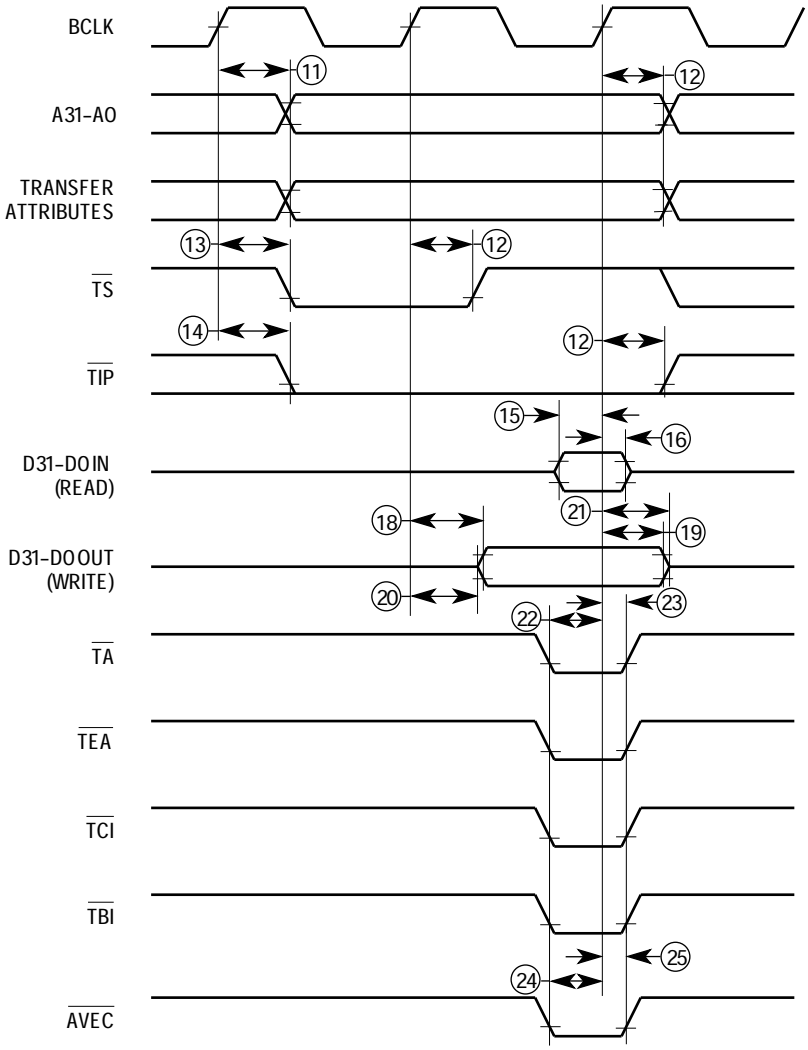
A.6.6 Output AC Timing Specifications (see Figures A-5* to A-9)

Num	Characteristic	20 MHz		25 MHz		33 MHz		Unit
		Min	Max	Min	Max	Min	Max	
11	BCLK to Address, $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, PSTx, R/$\overline{\text{W}}$, SIZx, TLNx ,TMx, TTx, UPAx Valid	11.5	35	9	30	6.5	25	ns
12	BCLK to Output Invalid (Output Hold)	11.5	—	9	—	6.5	—	ns
13	BCLK to $\overline{\text{TS}}$ Valid	11.5	35	9	30	6.5	25	ns
14	BCLK to $\overline{\text{TIP}}$ Valid	11.5	35	9	30	6.5	25	ns
18	BCLK to Data-Out Valid	11.5	37	9	32	6.5	27	ns
19	BCLK to Data-Out Invalid (Output Hold)	11.5	—	9	—	6.5	—	ns
20	BCLK to Output Low Impedance	11.5	—	9	—	6.5	—	ns
21	BCLK to Data-Out High Impedance	11.5	25	9	20	6.5	17	ns
38	BCLK to Address, $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, R/$\overline{\text{W}}$, SIZx, $\overline{\text{TS}}$, TLNx, TMx, TTx, UPAx High Impedance	11.5	23	9	18	6.5	15	ns
39	BCLK to $\overline{\text{BB}}$, $\overline{\text{TA}}$, $\overline{\text{TIP}}$ High Impedance	23	33	19	28	14	25	ns
40	BCLK to $\overline{\text{BR}}$, $\overline{\text{BB}}$ Valid	11.5	35	9	30	6.5	23	ns
43	BCLK to $\overline{\text{MI}}$ Valid	11.5	35	9	30	6.5	25	ns
48	BCLK to $\overline{\text{TA}}$ Valid	11.5	35	9	30	6.5	25	ns
50	BCLK to $\overline{\text{IPEND}}$, PSTx, $\overline{\text{RSTO}}$ Valid	11.5	35	9	30	6.5	25	ns

* Output timing is specified for a valid signal measured at the pin. Timing is specified driving an unterminated 30-Ω transmission line with a length characterized by a 2.5-ns one-way propagation delay. Buffer output impedance is typically 30 Ω; the buffer specifications include approximately 5 ns for the signal to propagate the length of the transmission line and back.

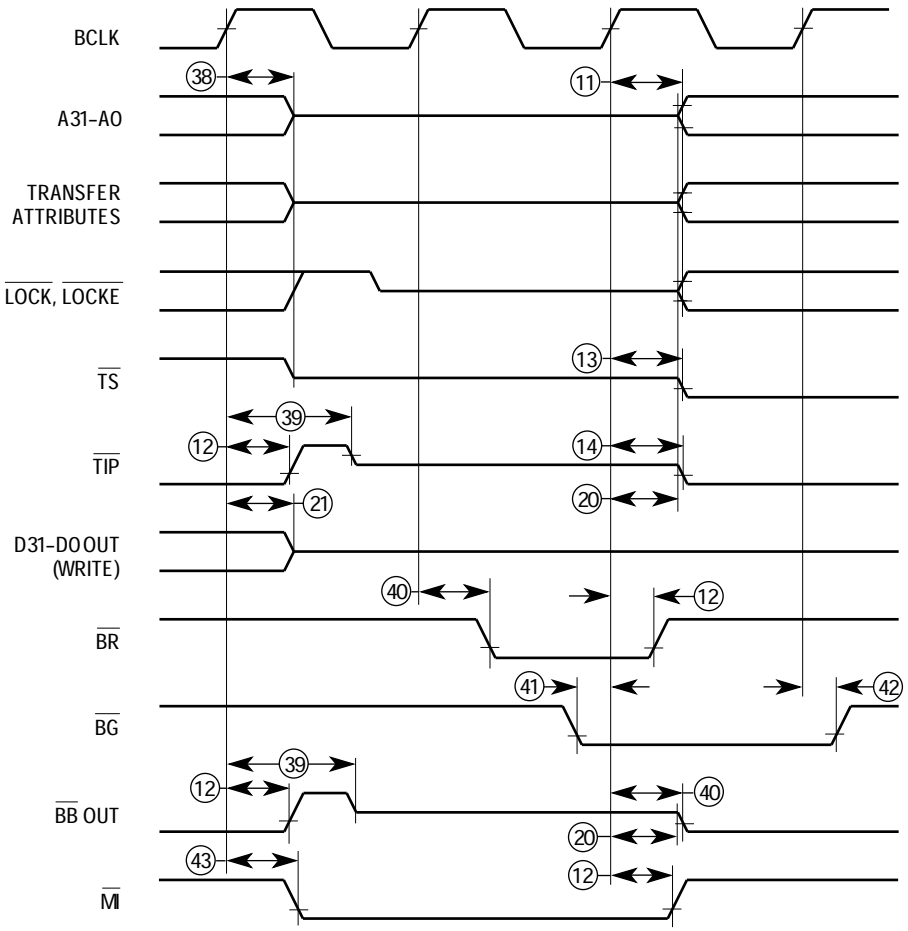
A.6.7 Input AC Timing Specifications (See Figures A-5 to A-9)

Num	Characteristic	20 MHz		25 MHz		33 MHz		Unit
		Min	Max	Min	Max	Min	Max	
15	Data-In Valid to BCLK (Setup)	6	—	5	—	4	—	ns
16	BCLK to Data-In Invalid (Hold)	5	—	4	—	4	—	ns
17	BCLK to Data-In High Impedance (Read Followed by Write)	—	61	—	49	—	36.5	ns
22a	$\overline{TA}$ Valid to BCLK (Setup)	12.5	—	10	—	10	—	ns
22b	$\overline{TEA}$ Valid to BCLK (Setup)	12.5	—	10	—	10	—	ns
22c	$\overline{TCI}$ Valid to BCLK (Setup)	12.5	—	10	—	10	—	ns
22d	$\overline{TBI}$ Valid to BCLK (Setup)	14	—	11	—	10	—	ns
23	BCLK to $\overline{TA}$, $\overline{TEA}$, $\overline{TCI}$, $\overline{TBI}$ Invalid (Hold)	2.5	—	2	—	2	—	ns
24	$\overline{AVEC}$ Valid to BCLK (Setup)	6	—	5	—	5	—	ns
25	BCLK to $\overline{AVEC}$ Invalid (Hold)	2.5	—	2	—	2	—	ns
41a	$\overline{BB}$ Valid to BCLK (Setup)	8	—	7	—	7	—	ns
41b	$\overline{BG}$ Valid to BCLK (Setup)	10	—	8	—	7	—	ns
41c	$\overline{CDIS}$, $\overline{MDIS}$ Valid to BCLK (Setup)	12.5	—	10	—	8	—	ns
41d	$\overline{IPLx}$ Valid to BCLK (Setup)	5	—	4	—	3	—	ns
42	BCLK to $\overline{BB}$, $\overline{BG}$, $\overline{CDIS}$, $\overline{MDIS}$, $\overline{IPLx}$ Invalid (Hold)	2.5	—	2	—	2	—	ns
44a	Address Valid to BCLK (Setup)	10	—	8	—	7	—	ns
44b	SIZx Valid to BCLK (Setup)	15	—	12	—	8	—	ns
44c	TTx Valid to BCLK (Setup)	7.5	—	6	—	8.5	—	ns
44d	R/$\overline{W}$ Valid to BCLK (Setup)	7.7	—	6	—	5	—	ns
44e	SCx Valid to BCLK (Setup)	12.5	—	10	—	11	—	ns
45	BCLK to Address SIZx, TTx, R/$\overline{W}$, SCx Invalid (Hold)	2.5	—	2	—	2	—	ns
46	$\overline{TS}$ Valid to BCLK (Setup)	6	—	5	—	9	—	ns
47	BCLK to $\overline{TS}$ Invalid (Hold)	2.5	—	2	—	2	—	ns
49	BCLK to $\overline{BB}$ High Impedance (MC68LC040 Assumes Bus Mastership)	—	11	—	9	—	9	ns
51	$\overline{RSTI}$ Valid to BCLK	6	—	5	—	4	—	ns
52	BCLK to $\overline{RSTI}$ Invalid	2.5	—	2	—	2	—	ns



NOTE: Transfer Attribute Signals = UPAx, SIZx, TTx, TMx, TLNx, R/$\overline{W}$, $\overline{LOCK}$, $\overline{LOCKE}$, $\overline{CIOUT}$

Figure A-5. Read/Write Timing



NOTE: Transfer Attribute Signals = UPAx, SIZx, TTx, TMx, TLNx, R/W, CIOUT

Figure A-6. Bus Arbitration Timing

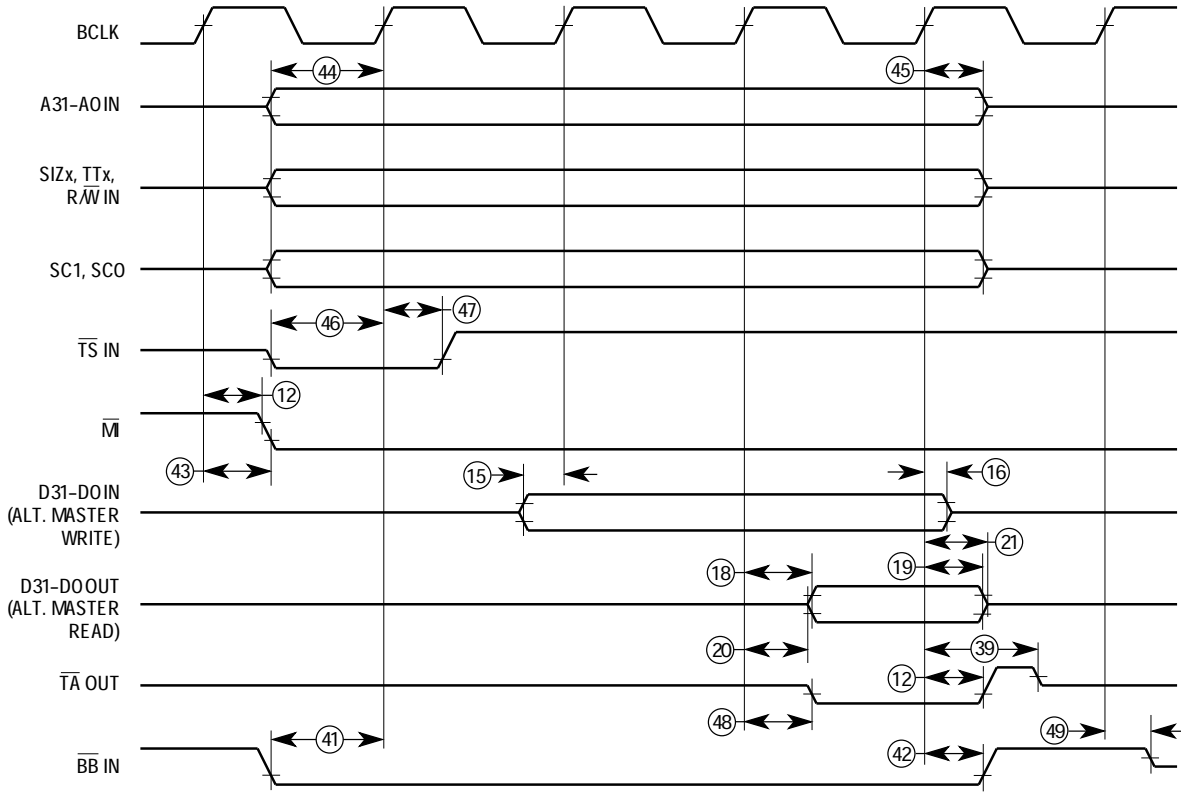


Figure A-7. Snoop Hit Timing

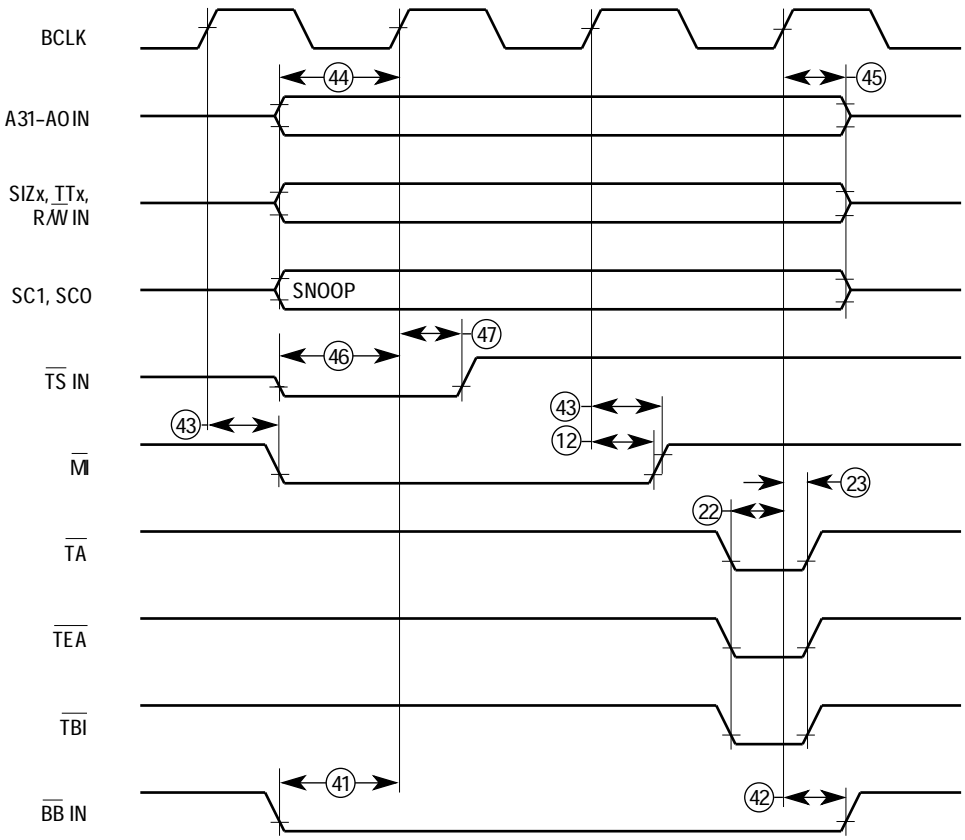


Figure A-8. Snoop Miss Timing

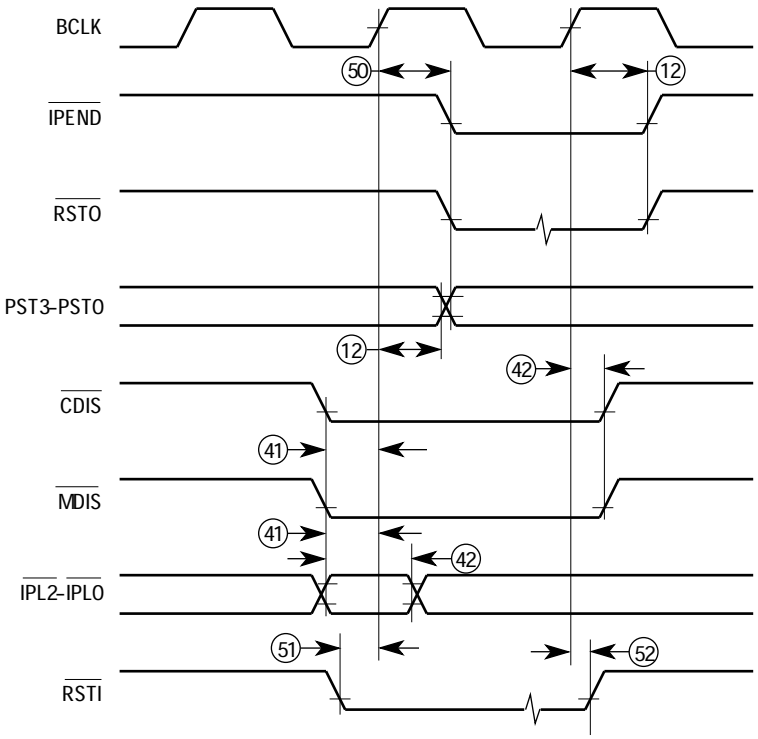


Figure A-9. Other Signal Timing

APPENDIX B
MC68EC040

NOTE

All references to MC68EC040 also apply to the MC68EC040V. Refer to **Appendix C MC68040V and MC68EC040V** for more information on the MC68EC040V.

The MC68EC040 is Motorola's third generation of M68000-compatible, high-performance, 32-bit microprocessors. The MC68EC040 is an 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 4-Kbyte instruction and data caches. A six-stage instruction pipeline, multiple internal buses, and a full internal Harvard architecture, including separate caches for both instruction and data accesses, provides a high degree of instruction execution parallelism. The inclusion of on-chip bus snooping logic, which directly supports cache coherency in multimaster applications, enhances cache functionality.

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 and MC68LC040. The MC68EC040 is implemented in Motorola’s latest HCMOS technology, providing an ideal balance between speed, power, and physical device size. Figure B-1 provides a simplified block diagram of the MC68EC040.

The main features of the MC68EC040 include:

- MC68040-Compatible Integer Execution Unit
- 4-Kbyte Instruction Cache and 4-Kbyte Data Cache Accessible Simultaneously
- 32-Bit, Nonmultiplexed External Address and Data Buses with Synchronous Bursting Interface
- User-Object-Code Compatible with All 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-Gbyte Direct Addressing Range

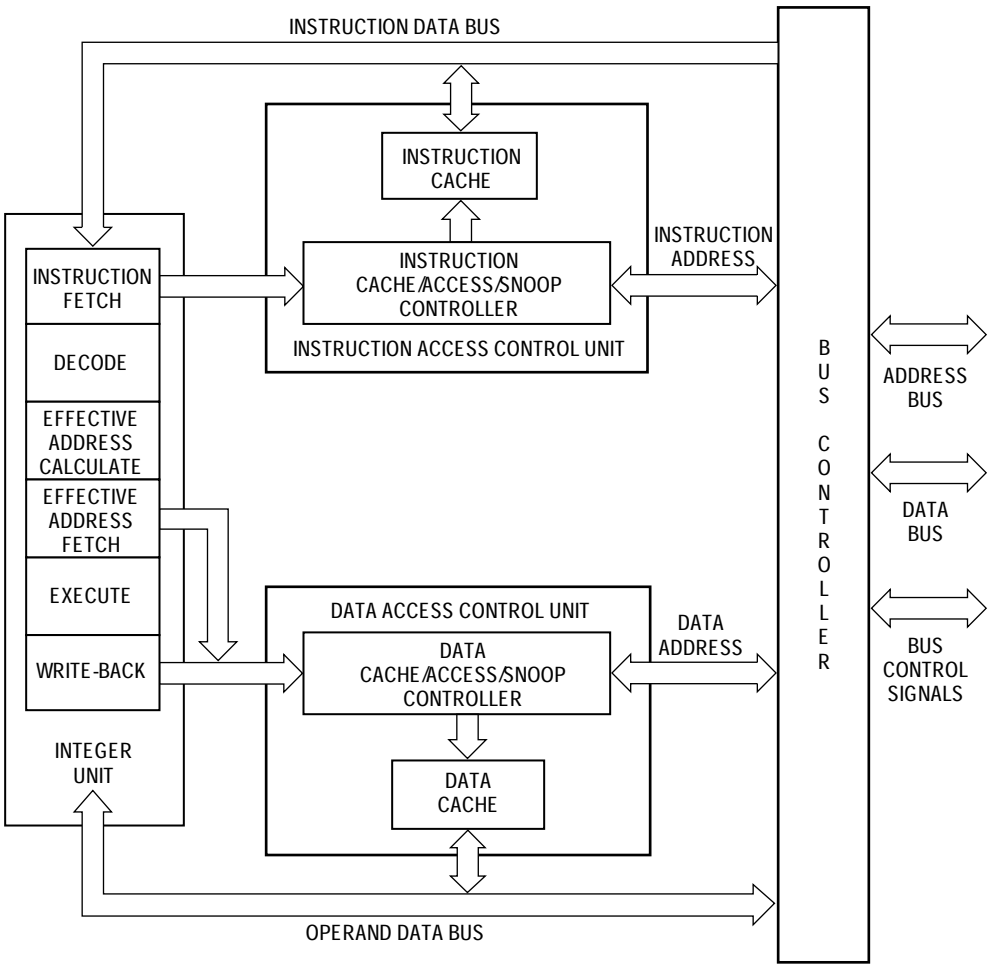


Figure B-1. MC68EC040 Block Diagram

With the exception of the memory management unit (MMU), the floating-point unit (FPU), and their respective registers, the MC68EC040 programming model, data formats and types, instruction set (except all instructions beginning with an “F”, PTEST, and PFLUSH), and caches are the same as described in **Section 1 Introduction** for the MC68040. Figures B-2 and B-3 illustrate the programming model and functional signal groups for the MC68EC040.

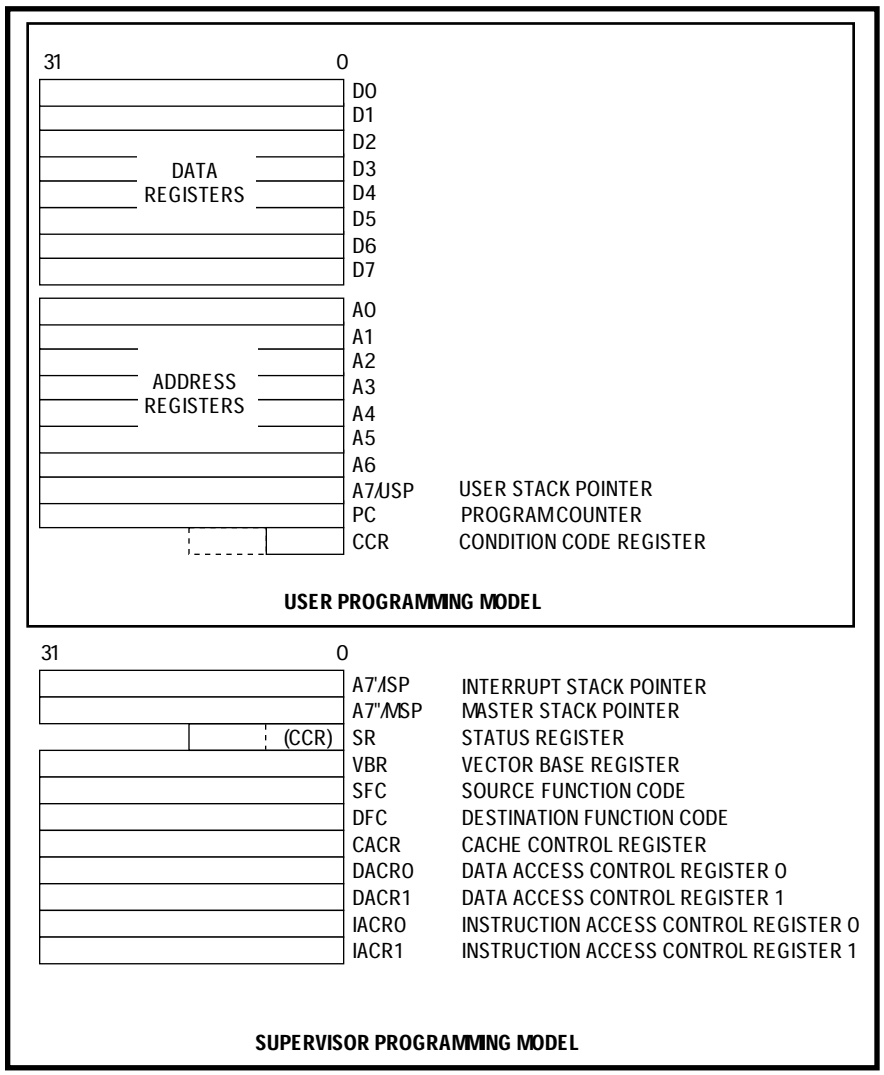


Figure B-2. MC68EC040 Programming Model