Freescale Semiconductor, Inc.

M68020UM/AD
REV. 2

M68020

Datasheet.Company

Freescale Semiconductor, Inc.

MC68020
MC68EC020

MICROPROCESSORS
USER'S MANUAL

For More Information On This Product,
Go to: www.freescale.com

Introduction 1

Processing States 2

Signal Description 3

On-Chip Cache Memory 4

Bus Operation 5

Exception Processing 6

Coprocessor Interface Description 7

Instruction Execution Timing 8

Applications Information 9

Electrical Characteristics 10

Ordering Information and Mechanical Data 11

Appendix A A

Index I

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1 Introduction

2 Processing States

3 Signal Description

4 On-Chip Cache Memory

5 Bus Operation

6 Exception Processing

7 Coprocessor Interface Description

8 Instruction Execution Timing

9 Applications Information

10 Electrical Characteristics

11 Ordering Information and Mechanical Data

A Appendix A

I Index

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

68K FAX-IT

FAX (512) 895-8593

The Motorola High-Performance Microprocessor Technical Publications Department provides a fax number for you to submit any questions and comments about this document. We welcome your suggestions for improving our documentation or any questions concerning our products.

Please provide the document number and revision number (located in upper right-hand corner of the cover) and the title of the document. When referring to items in the manual, please reference the page number, paragraph number, figure number, table number, and line number, if necessary.

When we receive a fax between the hours of 7:30 a.m. and 5:00 p.m. CST, Monday through Friday, we will respond within two hours. If the fax is received after 5:00 p.m. or on a weekend or holiday, we will respond within two hours on the first working day following receipt of the fax.

When sending a fax, please provide your name, company, fax number, and phone number including area code.

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

MOTOROLA

MC68020
MC68EC020

MICROPROCESSORS USER'S MANUAL

First Edition

Freescale Semiconductor, Inc.

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

For More Information On This Product,
Go to: www.freescale.com

PREFACE

The *M68020 User's Manual* describes the capabilities, operation, and programming of the MC68020 32-bit, second-generation, enhanced microprocessor and the MC68EC020 32-bit, second-generation, enhanced embedded microprocessor.

Throughout this manual, "MC68020/EC020" is used when information applies to both the MC68020 and the MC68EC020. "MC68020" and "MC68EC020" are used when information applies only to the MC68020 or MC68EC020, respectively.

For detailed information on the MC68020 and MC68EC020 instruction set, refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual*.

This manual consists of the following sections:

Section 1	Introduction
Section 2	Processing States
Section 3	Signal Description
Section 4	On-Chip Cache Memory
Section 5	Bus Operation
Section 6	Exception Processing
Section 7	Coprocessor Interface Description
Section 8	Instruction Execution Timing
Section 9	Applications Information
Section 10	Electrical Characteristics
Section 11	Ordering Information and Mechanical Data
Appendix A	Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol

NOTE

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

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Documentation Feedback

FAX 512-895-2693: Documentation Comments Only (no technical questions please)
http://www.mot.com/hpesd/docs_survey.html (documentation comments only)

The Technical Communications Department welcomes your suggestions for improving our documentation and encourages you to complete the documentation feedback form at the website listed above. Your help helps us measure how well we are serving your information requirements and how well we are transferring the knowledge you need to be successful with Motorola products.

The Technical Communications Department also provides a fax number for you to submit any questions or comments about this document or how to order other documents. Please provide the part number and revision number (located in the upper right hand corner of the cover) and the title of the document. When referring to items in the manual, please reference the page number, paragraph number, figure or table number. **Please do not fax technical questions to this number.**

When sending a fax, please provide your name, company, fax number, and phone number (including area code).

For Internet Access:

Web Only: http://www.mot.com/aesop

For Hotline Questions:

FAX (US or Canada): 1-800-248-8567

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

MC68020/EC020 ACRONYM LIST

BCD — Binary-Coded Decimal
CAAR — Cache Address Register
CACR — Cache Control Register
CCR — Condition Code Register
CIR — Coprocessor Interface Register
CMOS — Complementary Metal Oxide Semiconductor
CPU — Central Processing Unit
CQFP — Ceramic Quad Flat Pack
DDMA — Dual-Channel Direct Memory Access
DFC — Destination Function Code Register
DMA — Direct Memory Access
DRAM — Dynamic Random Access Memory
FPCP — Floating-Point Coprocessor
HCMOS — High-Density Complementary Metal Oxide Semiconductor
IEEE — Institute of Electrical and Electronic Engineers
ISP — Interrupt Stack Pointer
LMB — Lower Middle Byte
LRAR — Limited Rate Auto Request
LSB — Least Significant Byte
MMU — Memory Management Unit
MPU — Microprocessor Unit
MSB — Most Significant Byte
MSP — Master Stack Pointer
NMOS — n-Type Metal Oxide Semiconductor
PAL — Programmable Array Logic
PC — Program Counter
PGA — Pin Grid Array
PMMU — Paged Memory Management Unit
PPGA — Plastic Pin Grid Array
PQFP — Plastic Quad Flat Pack
RAM — Random Access Memory
SFC — Source Function Code Register
SP — Stack Pointer
SR — Status Register
SSP — Supervisor Stack Pointer
SSW — Special Status Word
UMB — Upper Middle Byte
USP — User Stack Pointer
VBR — Vector Base Register
VLSI — Very Large Scale Integration

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

TABLE OF CONTENTS

Paragraph Number	Title	Page Number
	Section 1 Introduction	
1.1	Features	1-2
1.2	Programming Model	1-4
1.3	Data Types and Addressing Modes Overview	1-8
1.4	Instruction Set Overview	1-10
1.5	Virtual Memory and Virtual Machine Concepts	1-10
1.5.1	Virtual Memory	1-10
1.5.2	Virtual Machine	1-12
1.6	Pipelined Architecture	1-12
1.7	Cache Memory	1-13
	Section 2 Processing States	
2.1	Privilege Levels	2-2
2.1.1	Supervisor Privilege Level	2-2
2.1.2	User Privilege Level	2-3
2.1.3	Changing Privilege Level	2-3
2.2	Address Space Types	2-4
2.3	Exception Processing	2-5
2.3.1	Exception Vectors	2-5
2.3.2	Exception Stack Frame	2-6
	Section 3 Signal Description	
3.1	Signal Index	3-2
3.2	Function Code Signals (FC2–FC0)	3-2
3.3	Address Bus (A31–A0, MC68020)(A23–A0, MC68EC020)	3-2
3.4	Data Bus (D31–D0)	3-2
3.5	Transfer Size Signals (SIZ1, SIZ0)	3-2
3.6	Asynchronous Bus Control Signals	3-4
3.7	Interrupt Control Signals	3-5
3.8	Bus Arbitration Control Signals	3-6
3.9	Bus Exception Control Signals	3-6
3.10	Emulator Support Signal	3-7
3.11	Clock (CLK)	3-7

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
3.12	Power Supply Connections	3-7
3.13	Signal Summary	3-8
	Section 4 On-Chip Cache Memory	
4.1	On-Chip Cache Organization and Operation	4-1
4.2	Cache Reset	4-3
4.3	Cache Control	4-3
4.3.1	Cache Control Register (CACR)	4-3
4.3.2	Cache Address Register (CAAR)	4-4
	Section 5 Bus Operation	
5.1	Bus Transfer Signals	5-1
5.1.1	Bus Control Signals	5-2
5.1.2	Address Bus	5-3
5.1.3	Address Strobe	5-3
5.1.4	Data Bus	5-3
5.1.5	Data Strobe	5-4
5.1.6	Data Buffer Enable	5-4
5.1.7	Bus Cycle Termination Signals	5-4
5.2	Data Transfer Mechanism	5-5
5.2.1	Dynamic Bus Sizing	5-5
5.2.2	Misaligned Operands	5-14
5.2.3	Effects of Dynamic Bus Sizing and Operand Misalignment	5-20
5.2.4	Address, Size, and Data Bus Relationships	5-21
5.2.5	Cache Interactions	5-22
5.2.6	Bus Operation	5-24
5.2.7	Synchronous Operation with $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$	5-24
5.3	Data Transfer Cycles	5-25
5.3.1	Read Cycle	5-26
5.3.2	Write Cycle	5-33
5.3.3	Read-Modify-Write Cycle	5-39
5.4	CPU Space Cycles	5-44
5.4.1	Interrupt Acknowledge Bus Cycles	5-45
5.4.1.1	Interrupt Acknowledge Cycle—Terminated Normally	5-45
5.4.1.2	Autovector Interrupt Acknowledge Cycle	5-48
5.4.1.3	Spurious Interrupt Cycle	5-48
5.4.2	Breakpoint Acknowledge Cycle	5-50
5.4.3	Coprocessor Communication Cycles	5-53
5.5	Bus Exception Control Cycles	5-53
5.5.1	Bus Errors	5-55

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
5.5.2	Retry Operation	5-56
5.5.3	Halt Operation	5-60
5.5.4	Double Bus Fault	5-60
5.6	Bus Synchronization	5-60
5.7	Bus Arbitration	5-62
5.7.1	MC68020 Bus Arbitration	5-62
5.7.1.1	Bus Request (MC68020)	5-63
5.7.1.2	Bus Grant (MC68020)	5-66
5.7.1.3	Bus Grant Acknowledge (MC68020)	5-66
5.7.1.4	Bus Arbitration Control (MC68020)	5-66
5.7.2	MC68EC020 Bus Arbitration	5-67
5.7.2.1	Bus Request (MC68EC020)	5-70
5.7.2.2	Bus Grant (MC68EC020)	5-71
5.7.2.3	Bus Arbitration Control (MC68EC020)	5-71
5.8	Reset Operation	5-73
		5-76

Section 6
Exception Processing

6.1	Exception Processing Sequence	6-1
6.1.1	Reset Exception	6-4
6.1.2	Bus Error Exception	6-4
6.1.3	Address Error Exception	6-6
6.1.4	Instruction Trap Exception	6-6
6.1.5	Illegal Instruction and Unimplemented Instruction Exceptions	6-7
6.1.6	Privilege Violation Exception	6-8
6.1.7	Trace Exception	6-9
6.1.8	Format Error Exception	6-10
6.1.9	Interrupt Exceptions	6-11
6.1.10	Breakpoint Instruction Exception	6-17
6.1.11	Multiple Exceptions	6-17
6.1.12	Return from Exception	6-19
6.2	Bus Fault Recovery	6-21
6.2.1	Special Status Word (SSW)	6-21
6.2.2	Using Software to Complete the Bus Cycles	6-23
6.2.3	Completing the Bus Cycles with RTE	6-24
6.3	Coprocessor Considerations	6-25
6.4	Exception Stack Frame Formats	6-25

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number

Section 7
Coprocessor Interface Description

7.1	Introduction	7-1
7.1.1	Interface Features	7-2
7.1.2	Concurrent Operation Support	7-2
7.1.3	Coprocessor Instruction Format	7-3
7.1.4	Coprocessor System Interface	7-4
7.1.4.1	Coprocessor Classification	7-4
7.1.4.2	Processor-Coprocessor Interface	7-5
7.1.4.3	Coprocessor Interface Register Selection	7-6
7.2	Coprocessor Instruction Types	7-7
7.2.1	Coprocessor General Instructions	7-8
7.2.1.1	Format	7-8
7.2.1.2	Protocol	7-9
7.2.2	Coprocessor Conditional Instructions	7-10
7.2.2.1	Branch on Coprocessor Condition Instruction	7-12
7.2.2.1.1	Format	7-12
7.2.2.1.2	Protocol	7-12
7.2.2.2	Set on Coprocessor Condition Instruction	7-13
7.2.2.2.1	Format	7-13
7.2.2.2.2	Protocol	7-14
7.2.2.3	Test Coprocessor Condition, Decrement, and Branch Instruction	7-14
7.2.2.3.1	Format	7-14
7.2.2.3.2	Protocol	7-15
7.2.2.4	Trap on Coprocessor Condition Instruction	7-15
7.2.2.4.1	Format	7-15
7.2.2.4.2	Protocol	7-16
7.2.3	Coprocessor Context Save and Restore Instructions	7-16
7.2.3.1	Coprocessor Internal State Frames	7-17
7.2.3.2	Coprocessor Format Words	7-18
7.2.3.2.1	Empty/Reset Format Word	7-18
7.2.3.2.2	Not-Ready Format Word	7-19
7.2.3.2.3	Invalid Format Word	7-19
7.2.3.2.4	Valid Format Word	7-20
7.2.3.3	Coprocessor Context Save Instruction	7-20
7.2.3.3.1	Format	7-20
7.2.3.3.2	Protocol	7-21
7.2.3.4	Coprocessor Context Restore Instruction	7-22
7.2.3.4.1	Format	7-22
7.2.3.4.2	Protocol	7-23
7.3	Coprocessor Interface Register Set	7-24

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
7.3.1	Response CIR	7-24
7.3.2	Control CIR	7-24
7.3.3	Save CIR	7-25
7.3.4	Restore CIR	7-25
7.3.5	Operation Word CIR	7-25
7.3.6	Command CIR	7-25
7.3.7	Condition CIR	7-26
7.3.8	Operand CIR	7-26
7.3.9	Register Select CIR	7-27
7.3.10	Instruction Address CIR	7-27
7.3.11	Operand Address CIR	7-27
7.4	Coprocessor Response Primitives	7-27
7.4.1	ScanPC	7-28
7.4.2	Coprocessor Response Primitive General Format	7-28
7.4.3	Busy Primitive	7-30
7.4.4	Null Primitive	7-31
7.4.5	Supervisor Check Primitive	7-33
7.4.6	Transfer Operation Word Primitive	7-33
7.4.7	Transfer from Instruction Stream Primitive	7-34
7.4.8	Evaluate and Transfer Effective Address Primitive	7-35
7.4.9	Evaluate Effective Address and Transfer Data Primitive	7-35
7.4.10	Write to Previously Evaluated Effective Address Primitive	7-37
7.4.11	Take Address and Transfer Data Primitive	7-39
7.4.12	Transfer to/from Top of Stack Primitive	7-40
7.4.13	Transfer Single Main Processor Register Primitive	7-40
7.4.14	Transfer Main Processor Control Register Primitive	7-41
7.4.15	Transfer Multiple Main Processor Registers Primitive	7-42
7.4.16	Transfer Multiple Coprocessor Registers Primitive	7-42
7.4.17	Transfer Status Register and ScanPC Primitive	7-44
7.4.18	Take Preinstruction Exception Primitive	7-45
7.4.19	Take Midinstruction Exception Primitive	7-47
7.4.20	Take Postinstruction Exception Primitive	7-48
7.5	Exceptions	7-49
7.5.1	Coprocessor-Detected Exceptions	7-49
7.5.1.1	Coprocessor-Detected Protocol Violations	7-50
7.5.1.2	Coprocessor-Detected Illegal Command or Condition Words	7-51
7.5.1.3	Coprocessor Data-Processing-Related Exceptions	7-51
7.5.1.4	Coprocessor System-Related Exceptions	7-51
7.5.1.5	Format Errors	7-52
7.5.2	Main-Processor-Detected Exceptions	7-52
7.5.2.1	Protocol Violations	7-52
7.5.2.2	F-Line Emulator Exceptions	7-54

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number
7.5.2.3	Privilege Violations	7-55
7.5.2.4	cpTRAPcc Instruction Traps	7-55
7.5.2.5	Trace Exceptions	7-55
7.5.2.6	Interrupts	7-56
7.5.2.7	Format Errors	7-57
7.5.2.8	Address and Bus Errors	7-57
7.5.3	Coprocessor Reset	7-58
7.6	Coprocessor Summary	7-58
	Section 8 Instruction Execution Timing	
8.1	Timing Estimation Factors	8-1
8.1.1	Instruction Cache and Prefetch	8-1
8.1.2	Operand Misalignment	8-2
8.1.3	Bus/Sequencer Concurrency	8-2
8.1.4	Instruction Execution Overlap	8-3
8.1.5	Instruction Stream Timing Examples	8-4
8.2	Instruction Timing Tables	8-9
8.2.1	Fetch Effective Address	8-13
8.2.2	Fetch Immediate Effective Address	8-14
8.2.3	Calculate Effective Address	8-16
8.2.4	Calculate Immediate Effective Address	8-17
8.2.5	Jump Effective Address	8-19
8.2.6	MOVE Instruction	8-20
8.2.7	Special-Purpose MOVE Instruction	8-29
8.2.8	Arithmetic/Logical Instructions	8-30
8.2.9	Immediate Arithmetic/Logical Instructions	8-31
8.2.10	Binary-Coded Decimal Operations	8-32
8.2.11	Single-Operand Instructions	8-33
8.2.12	Shift/Rotate Instructions	8-34
8.2.13	Bit Manipulation Instructions	8-35
8.2.14	Bit Field Manipulation Instructions	8-36
8.2.15	Conditional Branch Instructions	8-37
8.2.16	Control Instructions	8-38
8.2.17	Exception-Related Instructions	8-39
8.2.18	Save and Restore Operations	8-40
	Section 9 Applications Information	
9.1	Floating-Point Units	9-1
9.2	Byte Select Logic for the MC68020/EC020	9-5
9.3	Power and Ground Considerations	9-9

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

TABLE OF CONTENTS (Concluded)

Paragraph Number	Title	Page Number
9.4	Clock Driver	9-10
9.5	Memory Interface	9-11
9.6	Access Time Calculations	9-12
9.7	Module Support	9-14
9.7.1	Module Descriptor	9-14
9.7.2	Module Stack Frame	9-16
9.8	Access Levels	9-17
9.8.1	Module Call	9-18
9.8.2	Module Return	9-19

Section 10
Electrical Characteristics

10.1	Maximum Ratings	10-1
10.2	Thermal Considerations	10-1
10.2.1	MC68020 Thermal Characteristics and DC Electrical Characteristics	10-2
10.2.2	MC68EC020 Thermal Characteristics and DC Electrical Characteristics	10-4
10.3	AC Electrical Characteristics	10-5

Section 11
Ordering Information and Mechanical Data

11.1	Standard Ordering Information	11-1
11.1.1	Standard MC68020 Ordering Information	11-1
11.1.2	Standard MC68EC020 Ordering Information	11-1
11.2	Pin Assignments and Package Dimensions	11-2
11.2.1	MC68020 RC and RP Suffix—Pin Assignment	11-2
11.2.2	MC68020 RC Suffix—Package Dimensions	11-3
11.2.3	MC68020 RP Suffix—Package Dimensions	11-4
11.2.4	MC68020 FC and FE Suffix—Pin Assignment	11-5
11.2.5	MC68020 FC Suffix—Package Dimensions	11-6
11.2.6	MC68020 FE Suffix—Package Dimensions	11-7
11.2.7	MC68EC020 RP Suffix—Pin Assignment	11-8
11.2.8	MC68EC020 RP Suffix—Package Dimensions	11-9
11.2.9	MC68EC020 FG Suffix—Pin Assignment	11-10
11.2.10	MC68EC020 FG Suffix—Package Dimensions	11-11

Appendix A
Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

LIST OF ILLUSTRATIONS

Figure Number	Title	Page Number
1-1	MC68020/EC020 Block Diagram	1-3
1-2	User Programming Model	1-5
1-3	Supervisor Programming Model Supplement	1-6
1-4	Status Register (SR)	1-7
1-5	Instruction Pipe	1-13
2-1	General Exception Stack Frame	2-6
3-1	Functional Signal Groups	3-1
4-1	MC68020/EC020 On-Chip Cache Organization	4-2
4-2	Cache Control Register	4-3
4-3	Cache Address Register	4-4
5-1	Relationship between External and Internal Signals	5-2
5-2	Input Sample Window	5-2
5-3	Internal Operand Representation	5-6
5-4	MC68020/EC020 Interface to Various Port Sizes	5-6
5-5	Long-Word Operand Write to Word Port Example	5-10
5-6	Long-Word Operand Write to Word Port Timing	5-11
5-7	Word Operand Write to Byte Port Example	5-12
5-8	Word Operand Write to Byte Port Timing	5-13
5-9	Misaligned Long-Word Operand Write to Word Port Example	5-14
5-10	Misaligned Long-Word Operand Write to Word Port Timing	5-15
5-11	Misaligned Long-Word Operand Read from Word Port Example	5-16
5-12	Misaligned Word Operand Write to Word Port Example	5-16
5-13	Misaligned Word Operand Write to Word Port Timing	5-17
5-14	Misaligned Word Operand Read from Word Bus Example	5-18
5-15	Misaligned Long-Word Operand Write to Long-Word Port Example	5-18
5-16	Misaligned Long-Word Operand Write to Long-Word Port Timing	5-19
5-17	Misaligned Long-Word Operand Read from Long-Word Port Example	5-20
5-18	Byte Enable Signal Generation for 16- and 32-Bit Ports	5-23
5-19	Long-Word Read Cycle Flowchart	5-26
5-20	Byte Read Cycle Flowchart	5-27
5-21	Byte and Word Read Cycles—32-Bit Port	5-28
5-22	Long-Word Read—8-Bit Port	5-29
5-23	Long-Word Read—16- and 32-Bit Ports	5-30

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

LIST OF ILLUSTRATIONS (Continued)

Figure Number	Title	Page Number
5-24	Write Cycle Flowchart	5-33
5-25	Read-Write-Read Cycles—32-Bit Port	5-34
5-26	Byte and Word Write Cycles—32-Bit Port	5-35
5-27	Long-Word Operand Write—8-Bit Port	5-36
5-28	Long-Word Operand Write—16-Bit Port	5-37
5-29	Read-Modify-Write Cycle Flowchart	5-40
5-30	Byte Read-Modify-Write Cycle—32-Bit Port (TAS Instruction)	5-41
5-31	MC68020/EC020 CPU Space Address Encoding	5-45
5-32	Interrupt Acknowledge Cycle Flowchart	5-46
5-33	Interrupt Acknowledge Cycle Timing	5-47
5-34	Autovector Operation Timing	5-49
5-35	Breakpoint Acknowledge Cycle Flowchart	5-50
5-36	Breakpoint Acknowledge Cycle Timing	5-51
5-37	Breakpoint Acknowledge Cycle Timing (Exception Signaled)	5-52
5-38	Bus Error without $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$	5-57
5-39	Late Bus Error with $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$	5-58
5-40	Late Retry	5-59
5-41	Halt Operation Timing	5-61
5-42	MC68020 Bus Arbitration Flowchart for Single Request	5-64
5-43	MC68020 Bus Arbitration Operation Timing for Single Request	5-65
5-44	MC68020 Bus Arbitration State Diagram	5-67
5-45	MC68020 Bus Arbitration Operation Timing—Bus Inactive	5-69
5-46	MC68EC020 Bus Arbitration Flowchart for Single Request	5-71
5-47	MC68EC020 Bus Arbitration Operation Timing for Single Request	5-72
5-48	MC68EC020 Bus Arbitration State Diagram	5-73
5-49	MC68EC020 Bus Arbitration Operation Timing—Bus Inactive	5-75
5-50	Interface for Three-Wire to Two-Wire Bus Arbitration	5-76
5-51	Initial Reset Operation Timing	5-77
5-52	RESET Instruction Timing	5-78
6-1	Reset Operation Flowchart	6-5
6-2	Interrupt Pending Procedure	6-12
6-3	Interrupt Recognition Examples	6-13
6-4	Assertion of $\overline{\text{IPEND}}$ (MC68020 Only)	6-14
6-5	Interrupt Exception Processing Flowchart	6-15
6-6	Breakpoint Instruction Flowchart	6-18
6-7	RTE Instruction for Throwaway Four-Word Frame	6-20
6-8	Special Status Word Format	6-22
7-1	F-Line Coprocessor Instruction Operation Word	7-3
7-2	Asynchronous Non-DMA M68000 Coprocessor Interface Signal Usage	7-5
7-3	MC68020/EC020 CPU Space Address Encodings	7-6

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

LIST OF ILLUSTRATIONS (Continued)

Figure Number	Title	Page Number
7-4	Coprocessor Address Map in MC68020/EC020 CPU Space	7-7
7-5	Coprocessor Interface Register Set Map	7-7
7-6	Coprocessor General Instruction Format (cpGEN)	7-8
7-7	Coprocessor Interface Protocol for General Category Instructions	7-10
7-8	Coprocessor Interface Protocol for Conditional Category Instructions	7-11
7-9	Branch on Coprocessor Condition Instruction Format (cpBcc.W)	7-12
7-10	Branch on Coprocessor Condition Instruction Format (cpBcc.L)	7-12
7-11	Set on Coprocessor Condition Instruction Format (cpScc)	7-13
7-12	Test Coprocessor Condition, Decrement, and Branch Instruction Format (cpDBcc)	7-14
7-13	Trap on Coprocessor Condition Instruction Format (cpTRAPcc)	7-15
7-14	Coprocessor State Frame Format in Memory	7-17
7-15	Coprocessor Context Save Instruction Format (cpSAVE)	7-20
7-16	Coprocessor Context Save Instruction Protocol	7-21
7-17	Coprocessor Context Restore Instruction Format (cpRESTORE)	7-22
7-18	Coprocessor Context Restore Instruction Protocol	7-23
7-19	Control CIR Format	7-25
7-20	Condition CIR Format	7-26
7-21	Operand Alignment for Operand CIR Accesses	7-26
7-22	Coprocessor Response Primitive Format	7-28
7-23	Busy Primitive Format	7-30
7-24	Null Primitive Format	7-31
7-25	Supervisor Check Primitive Format	7-33
7-26	Transfer Operation Word Primitive Format	7-33
7-27	Transfer from Instruction Stream Primitive Format	7-34
7-28	Evaluate and Transfer Effective Address Primitive Format	7-35
7-29	Evaluate Effective Address and Transfer Data Primitive Format	7-35
7-30	Write to Previously Evaluated Effective Address Primitive Format	7-37
7-31	Take Address and Transfer Data Primitive Format	7-39
7-32	Transfer to/from Top of Stack Primitive Format	7-40
7-33	Transfer Single Main Processor Register Primitive Format	7-40
7-34	Transfer Main Processor Control Register Primitive Format	7-41
7-35	Transfer Multiple Main Processor Registers Primitive Format	7-42
7-36	Register Select Mask Format	7-42
7-37	Transfer Multiple Coprocessor Registers Primitive Format	7-43
7-38	Operand Format in Memory for Transfer to –(An)	7-44
7-39	Transfer Status Register and ScanPC Primitive Format	7-44
7-40	Take Preinstruction Exception Primitive Format	7-45
7-41	MC68020/EC020 Preinstruction Stack Frame	7-46
7-42	Take Midinstruction Exception Primitive Format	7-47
7-43	MC68020/EC020 Midinstruction Stack Frame	7-47
7-44	Take Postinstruction Exception Primitive Format	7-48

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

LIST OF ILLUSTRATIONS (Concluded)

Figure Number	Title	Page Number
7-45	MC68020/EC020 Postinstruction Stack Frame	7-48
8-1	Concurrent Instruction Execution	8-3
8-2	Instruction Execution for Instruction Timing Purposes	8-3
8-3	Processor Activity for Example 1	8-5
8-4	Processor Activity for Example 2	8-6
8-5	Processor Activity for Example 3	8-7
8-6	Processor Activity for Example 4	8-8
9-1	32-Bit Data Bus Coprocessor Connection	9-2
9-2	Chip Select Generation PAL	9-3
9-3	Chip Select PAL Equations	9-4
9-4	Bus Cycle Timing Diagram	9-4
9-5	Example MC68020/EC020 Byte Select PAL System Configuration	9-7
9-6	MC68020/EC020 Byte Select PAL Equations	9-8
9-7	High-Resolution Clock Controller	9-11
9-8	Alternate Clock Solution	9-11
9-9	Access Time Computation Diagram	9-12
9-10	Module Descriptor Format	9-15
9-11	Module Entry Word	9-15
9-12	Module Call Stack Frame	9-16
9-13	Access Level Control Bus Registers	9-17
10-1	Drive Levels and Test Points for AC Specifications	10-6
10-2	Clock Input Timing Diagram	10-7
10-3	Read Cycle Timing Diagram	10-11
10-4	Write Cycle Timing Diagram	10-12
10-5	Bus Arbitration Timing Diagram	10-13
A-1	Bus Arbitration Circuit—MC68EC020 (Two-Wire) to DMA (Three-Wire)	A-1

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

LIST OF TABLES

Table Number	Title	Page Number
1-1	Addressing Modes	1-9
1-2	Instruction Set	1-11
2-1	Address Space Encodings	2-4
3-1	Signal Index	3-3
3-2	Signal Summary	3-8
5-1	$\overline{DSACK1}/\overline{DSACK0}$ Encodings and Results	5-5
5-2	SIZ1, SIZ0 Signal Encoding	5-7
5-3	Address Offset Encodings	5-7
5-4	Data Bus Requirements for Read Cycles	5-8
5-5	MC68020/EC020 Internal to External Data Bus Multiplexer— Write Cycles	5-9
5-6	Memory Alignment and Port Size Influence on Read/Write Bus Cycles	5-20
5-7	Data Bus Byte Enable Signals for Byte, Word, and Long-Word Ports	5-22
5-8	$\overline{DSACK1}/\overline{DSACK0}$, $\overline{BERR}$, $\overline{HALT}$ Assertion Results	5-54
6-1	Exception Vector Assignments	6-3
6-2	Tracing Control	6-9
6-3	Interrupt Levels and Mask Values	6-12
6-4	Exception Priority Groups	6-18
6-5	Exception Stack Frames	6-26
7-1	cpTRAPcc Opmode Encodings	7-16
7-2	Coprocessor Format Word Encodings	7-18
7-3	Null Coprocessor Response Primitive Encodings	7-32
7-4	Valid Effective Address Field Codes	7-36
7-5	Main Processor Control Register Select Codes	7-41
7-6	Exceptions Related to Primitive Processing	7-53
8-1	Examples 1–4 Instruction Stream Execution Comparison	8-8
8-2	Instruction Timings from Timing Tables	8-11
8-3	Observed Instruction Timings	8-11

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

LIST OF TABLES (Continued)

Table Number	Title	Page Number
9-1	Data Bus Activity for Byte, Word, and Long-Word Ports	9-6
9-2	V_{CC} and GND Pin Assignments—MC68EC020 PPGA (RP Suffix)	9-10
9-3	V_{CC} and GND Pin Assignments—MC68EC020 PQFP (FG Sufffix)	9-10
9-4	Memory Access Time Equations at 16.67 and 25 MHz	9-13
9-5	Calculated t_{AVDV} Values for Operation at Frequencies Less Than or Equal to the CPU Maximum Frequency Rating	9-14
9-6	Access Status Register Codes	9-18
10-1	θ_{JA} vs. Airflow—MC68020 CQFP Package	10-3
10-2	Power vs. Rated Frequency (at T_J Maximum = 110°C)	10-3
10-3	Temperature Rise of Board vs. P_D—MC68020 CQFP Package	10-3
10-4	θ_{JA} vs. Airflow—MC68EC020 PQFP Package	10-4

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 1
INTRODUCTION

The MC68020 is the first full 32-bit implementation of the M68000 family of microprocessors from Motorola. Using VLSI technology, the MC68020 is implemented with 32-bit registers and data paths, 32-bit addresses, a rich instruction set, and versatile addressing modes.

The MC68020 is object-code compatible with earlier members of the M68000 family and has the added features of new addressing modes in support of high-level languages, an on-chip instruction cache, and a flexible coprocessor interface with full IEEE floating-point support (the MC68881 and MC68882). The internal operations of this microprocessor operate in parallel, allowing multiple instructions to be executed concurrently.

The asynchronous bus structure of the MC68020 uses a nonmultiplexed bus with 32 bits of address and 32 bits of data. The processor supports a dynamic bus sizing mechanism that allows the processor to transfer operands to or from external devices while automatically determining device port size on a cycle-by-cycle basis. The dynamic bus interface allows access to devices of differing data bus widths, in addition to eliminating all data alignment restrictions.

The MC68EC020 is an economical high-performance embedded microprocessor based on the MC68020 and has been designed specifically to suit the needs of the embedded microprocessor market. The major differences in the MC68EC020 and the MC68020 are that the MC68EC020 has a 24-bit address bus and does not implement the following signals: $\overline{\text{ECS}}$, $\overline{\text{OCS}}$, $\overline{\text{DBEN}}$, $\overline{\text{IPEND}}$, and $\overline{\text{BGACK}}$. Also, the available packages and frequencies differ for the MC68020 and MC68EC020 (see **Section 11 Ordering Information and Mechanical Data**.) Unless otherwise stated, information in this manual applies to both the MC68020 and the MC68EC020.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.1 FEATURES

The main features of the MC68020/EC020 are as follows:

- Object-Code Compatible with Earlier M68000 Microprocessors
- Addressing Mode Extensions for Enhanced Support of High-Level Languages
- New Bit Field Data Type Accelerates Bit-Oriented Applications—e.g., Video Graphics
- An On-Chip Instruction Cache for Faster Instruction Execution
- Coprocessor Interface to Companion 32-Bit Peripherals—the MC68881 and MC68882 Floating-Point Coprocessors and the MC68851 Paged Memory Management Unit
- Pipelined Architecture with High Degree of Internal Parallelism Allowing Multiple Instructions To Be Executed Concurrently
- High-Performance Asynchronous Bus Is Nonmultiplexed and Full 32 Bits
- Dynamic Bus Sizing Efficiently Supports 8-/16-/32-Bit Memories and Peripherals
- Full Support of Virtual Memory and Virtual Machine
- Sixteen 32-Bit General-Purpose Data and Address Registers
- Two 32-Bit Supervisor Stack Pointers and Five Special-Purpose Control Registers
- Eighteen Addressing Modes and Seven Data Types
- 4-Gbyte Direct Addressing Range for the MC68020
- 16-Mbyte Direct Addressing Range for the MC68EC020
- Selection of Processor Speeds for the MC68020: 16.67, 20, 25, and 33.33 MHz
- Selection of Processor Speeds for the MCEC68020: 16.67 and 25 MHz

A block diagram of the MC68020/EC020 is shown in Figure 1-1.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

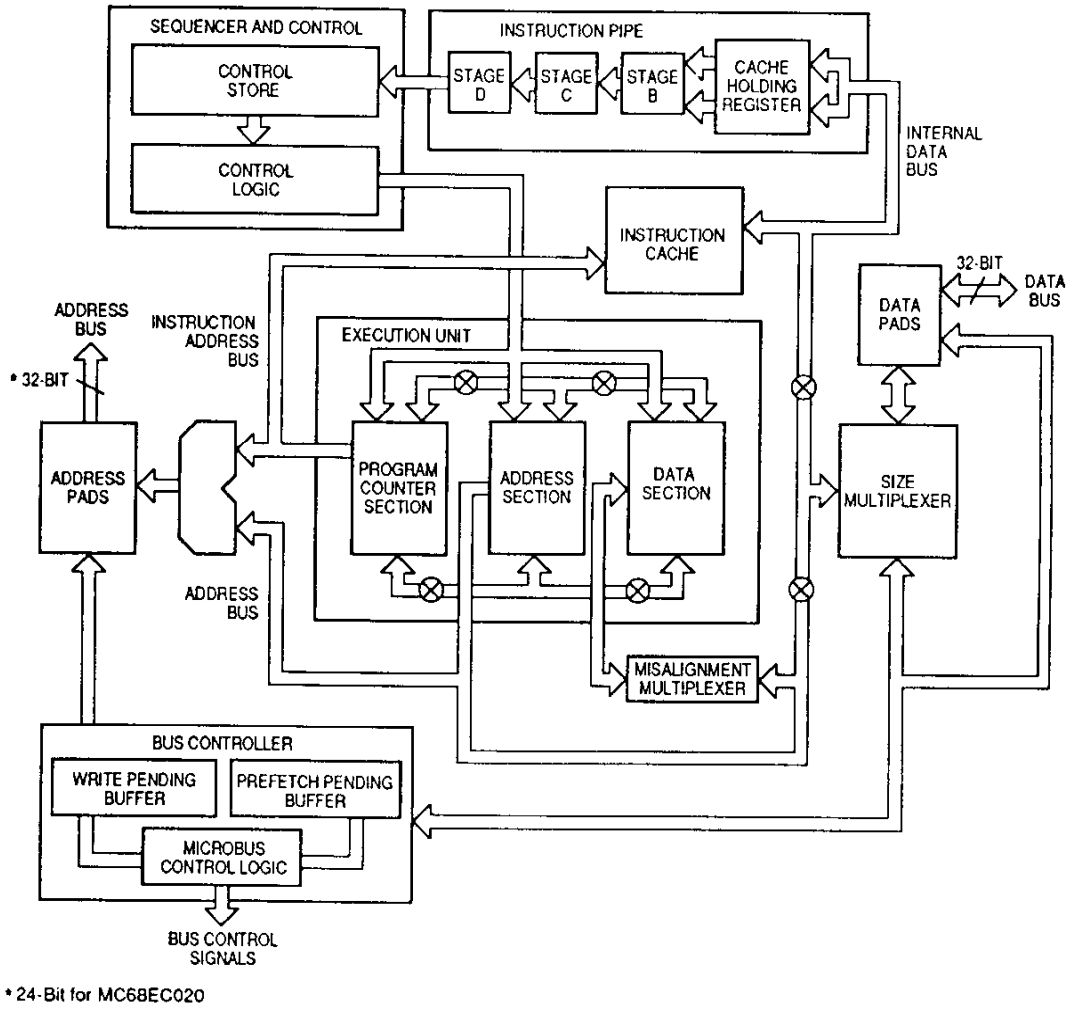


Figure 1-1. MC68020/EC020 Block Diagram

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.2 PROGRAMMING MODEL

The programming model of the MC68020/EC020 consists of two groups of registers, the user model and the supervisor model, that correspond to the user and supervisor privilege levels, respectively. User programs executing at the user privilege level use the registers of the user model. System software executing at the supervisor level uses the control registers of the supervisor level to perform supervisor functions.

As shown in the programming models (see Figures 1-2 and 1-3), the MC68020/EC020 has 16 32-bit general-purpose registers, a 32-bit PC two 32-bit SSPs, a 16-bit SR, a 32-bit VBR, two 3-bit alternate function code registers, and two 32-bit cache handling (address and control) registers.

The user programming model remains unchanged from earlier M68000 family microprocessors. The supervisor programming model supplements the user programming model and is used exclusively by MC68020/EC020 system programmers who utilize the supervisor privilege level to implement sensitive operating system functions. The supervisor programming model contains all the controls to access and enable the special features of the MC68020/EC020. All application software, written to run at the nonprivileged user level, migrates to the MC68020/EC020 from any M68000 platform without modification.

Registers D7–D0 are data registers used 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. Registers A6–A0 and the USP, ISP, and MSP are address registers that may be used as software stack pointers or base address registers. Register A7 (shown as A7 in Figure 1-2 and as A7′ and A7″ in Figure 1-3) is a register designation that applies to the USP in the user privilege level and to either the ISP or MSP in the supervisor privilege level. In the supervisor privilege level, the active stack pointer (interrupt or master) is called the SSP. In addition, the address registers may be used for word and long-word operations. All of the 16 general-purpose registers (D7–D0, A7–A0) may be used as index registers.

The PC contains the address of the next instruction to be executed by the MC68020/EC020. 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.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

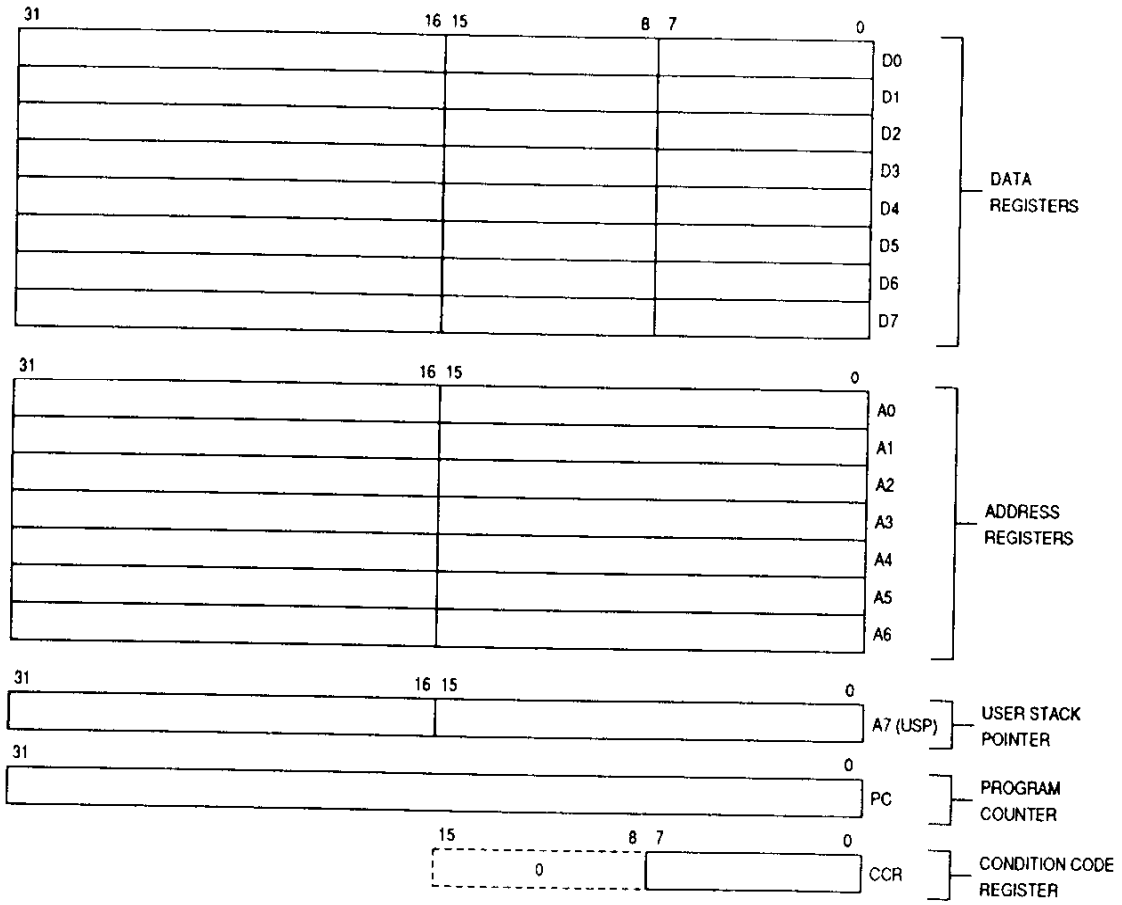


Figure 1-2. User Programming Model

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

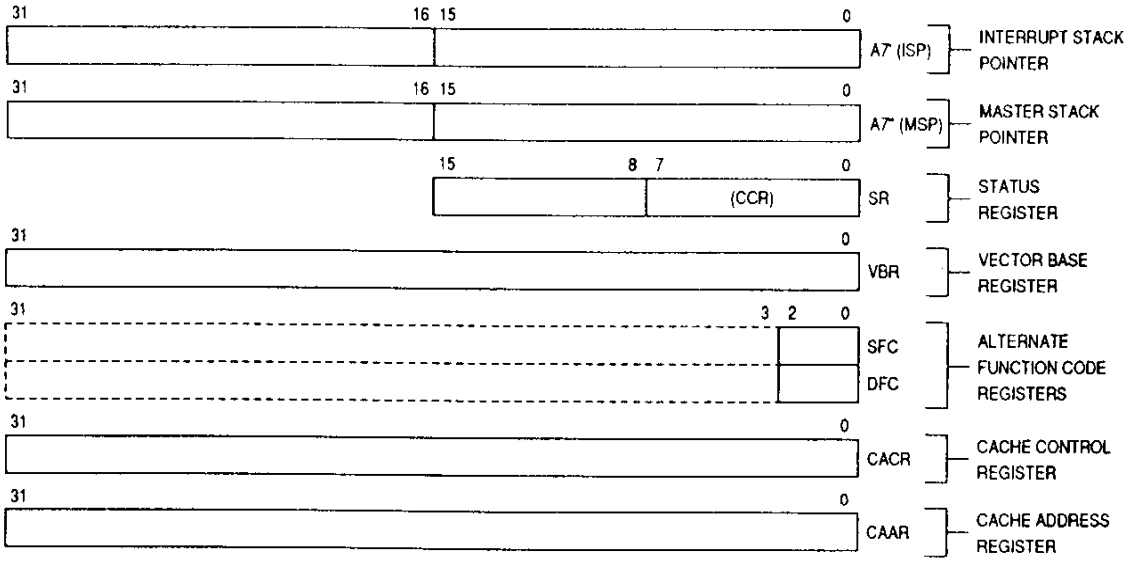


Figure 1-3. Supervisor Programming Model Supplement

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The SR (see Figure 1-4) stores the processor status. It contains the condition codes that reflect the results of a previous operation and can be used for conditional instruction execution in a program. The condition codes are extend (X), negative (N), zero (Z), overflow (V), and carry (C). The user byte, which contains the condition codes, is the only portion of the SR information available in the user privilege level, and it is referenced as the CCR in user programs. In the supervisor privilege level, software can access the entire SR, including the interrupt priority mask (three bits) and control bits that indicate whether the processor is in:

1. One of two trace modes (T1, T0)
2. Supervisor or user privilege level (S)
3. Master or interrupt mode (M)

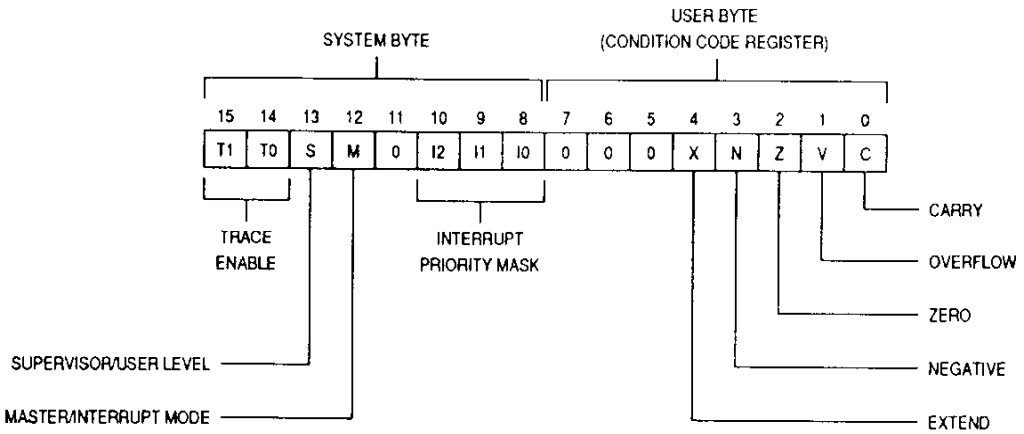


Figure 1-4. Status Register (SR)

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.

The alternate function code registers, SFC and DFC, contain 3-bit function codes. For the MC68020, function codes can be considered extensions of the 32-bit linear address that optionally provide as many as eight 4-Gbyte address spaces; for the MC68EC020, function codes can be considered extensions of the 24-bit linear address that optionally provide as many as eight 16-Mbyte address spaces. Function codes are automatically generated by the processor to select address spaces for data and program at the user and supervisor privilege levels and to select a CPU address space for processor functions (e.g., coprocessor communications). Registers SFC and DFC are used by certain instructions to explicitly specify the function codes for operations.

The CACR controls the on-chip instruction cache of the MC68020/EC020. The CAAR stores an address for cache control functions.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.3 DATA TYPES AND ADDRESSING MODES OVERVIEW

For detailed information on the data types and addressing modes supported by the MC68020/EC020, refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual*.

The MC68020/EC020 supports seven basic data types:

1. Bits
2. Bit Fields (Fields of consecutive bits, 1–32 bits long)
3. BCD Digits (Packed: 2 digits/byte, Unpacked: 1 digit/byte)
4. Byte Integers (8 bits)
5. Word Integers (16 bits)
6. Long-Word Integers (32 bits)
7. Quad-Word Integers (64 bits)

In addition, the MC68020/EC020 instruction set supports operations on other data types such as memory addresses. The coprocessor mechanism allows direct support of floating-point operations with the MC68881 and MC68882 floating-point coprocessors as well as specialized user-defined data types and functions.

The 18 addressing modes listed in Table 1-1 include nine basic types:

1. Register Direct
2. Register Indirect
3. Register Indirect with Index
4. Memory Indirect
5. PC Indirect with Displacement
6. PC Indirect with Index
7. PC Memory Indirect
8. Absolute
9. Immediate

The register indirect addressing modes have postincrement, predecrement, displacement, and index capabilities. The PC modes have index and offset capabilities. Both modes are extended to provide indirect reference through memory. In addition to these addressing modes, many instructions implicitly specify the use of the CCR, stack pointer, and/or PC.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 1-1. 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	(d_{16}, 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)
PC Indirect with Displacement	(d_{16}, PC)
PC Indirect with Index	
8-Bit Displacement	(d_8, PC, Xn)
Base Displacement	(bd, PC, Xn)
PC Indirect	
Postindexed	([bd, PC], Xn, od)
Preindexed	([bd, PC, Xn], od)
Absolute Data Addressing	
Short	(xxx).W
Long	(xxx).L
Immediate	#<data>

NOTE

Dn = Data Register, D7–D0

An = Address Register, A7–A0

d_8, d_{16} = A twos complement or sign-extended displacement added as part of the effective address calculation; size is 8 (d_8) or 16 (d_{16}) bits; when omitted, assemblers use a value of zero.

Xn = Address or data register used as an index register; form is Xn.SIZE*SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.

bd = A twos-complement base displacement; when present, size can be 16 or 32 bits.

od = Outer displacement added as part of effective address calculation after any memory indirection; use is optional with a size of 16 or 32 bits.

PC = Program Counter

<data> = Immediate value of 8, 16, or 32 bits

( ) = Effective Address

[ ] = Use as indirect access to long-word address.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.4 INSTRUCTION SET OVERVIEW

For detailed information on the MC68020/EC020 instruction set, refer to M68000PM/AD, *M68000 Family Programmer's Refcrence Manual*.

The instructions in the MC68020/EC020 instruction set are listed in Table 1-2. The instruction set has been tailored to support structured high-level languages and sophisticated operating systems. Many instructions operate on bytes, words, or long words, and most instructions can use any of the 18 addressing modes.

1.5 VIRTUAL MEMORY AND VIRTUAL MACHINE CONCEPTS

The full addressing range of the MC68020 is 4 Gbytes (4,294,967,296 bytes) in each of eight address spaces; the full addressing range of the MC68EC020 is 16 Mbytes (16,777,216 bytes) in each of the eight address spaces. Even though most systems implement a smaller physical memory, the system can be made to appear to have a full 4 Gbytes (MC68020) or 16 Mbytes (MC68EC020) of memory available to each user program by using virtual memory techniques.

In a virtual memory system, a user program can be written as if it has a large amount of memory available, although the physical memory actually present is much smaller. Similarly, a system can be designed to allow user programs to access devices that are not physically present in the system, such as tape drives, disk drives, printers, terminals, and so forth. With proper software emulation, a physical system can appear to be any other M68000 computer system to a user program, and the program can be given full access to all of the resources of that emulated system. Such an emulated system is called a virtual machine.

1.5.1 Virtual Memory

A system that supports virtual memory has a limited amount of high-speed physical memory that can be accessed directly by the processor and maintains an image of a much larger virtual memory on a secondary storage device such as a large-capacity disk drive. When the processor attempts to access a location in the virtual memory map that is not resident in physical memory, a page fault occurs. The access to that location is temporarily suspended while the necessary data is fetched from secondary storage and placed in physical memory. The suspended access is then either restarted or continued.

The MC68020/EC020 uses instruction continuation to support virtual memory. When a bus cycle is terminated with a bus error, the microprocessor suspends the current instruction and executes the virtual memory bus error handler. When the bus error handler has completed execution, it returns control to the program that was executing when the error was detected, reruns the faulted bus cycle (when required), and continues the suspended instruction.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 1-2. Instruction Set

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch Always
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CALLM	Call Module
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bound
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive Or Immediate
EXG	Exchange Registers
EXT, EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE CCR	Move Condition Code Register
MOVE SR	Move Status Register

Mnemonic	Description
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	Logical Inclusive OR Immediate
ORI CCR	Logical Inclusive Or Immediate to Condition Codes
ORI SR	Logical Inclusive OR Immediate to Status Register
PACK	Pack BCD
PEA	Push Effective Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL,ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTM	Return from Module
RTR	Return and Restore Codes
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract Address
SUBI	Subtract Immediate
SUBQ	Subtract Quick
SUBX	Subtract with Extend
SWAP	Swap Register Words
TAS	Test and Set an Operand
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

COPROCESSOR INSTRUCTIONS

Mnemonic	Description
cpBcc	Branch Conditionally
cpDBcc	Test Coprocessor Condition, Decrement and Branch
cpGEN	Coprocessor General Instruction
cpRESTORE	Restore Internal State of Coprocessor
cpSAVE	Save Internal State of Coprocessor
cpScc	Set Conditionally
cpTRAPcc	Trap Conditionally

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1.5.2 Virtual Machine

A typical use for a virtual machine system is the development of software, such as an operating system, for a new machine also under development and not yet available for programming use. In a virtual machine system, a governing operating system emulates the hardware of the new machine and allows the new software to be executed and debugged as though it were running on the new hardware. Since the new software is controlled by the governing operating system, it is executed at a lower privilege level than the governing operating system. Thus, any attempts by the new software to use virtual resources that are not physically present (and should be emulated) are trapped to the governing operating system and performed by its software.

In the MC68020/EC020 implementation of a virtual machine, the virtual application runs at the user privilege level. The governing operating system executes at the supervisor privilege level and any attempt by the new operating system to access supervisor resources or execute privileged instructions causes a trap to the governing operating system.

Instruction continuation is used to support virtual I/O devices in memory-mapped input/output systems. Control and data registers for the virtual device are simulated in the memory map. An access to a virtual register causes a fault, and the function of the register is emulated by software.

1.6 PIPELINED ARCHITECTURE

The MC68020/EC020 contains a three-word instruction pipe where instruction opcodes are decoded. As shown in Figure 1-5, instruction words (instruction operation words and all extension words) enter the pipe at stage B and proceed to stages C and D. An instruction word is completely decoded when it reaches stage D of the pipe. Each stage has a status bit that reflects whether the word in the stage was loaded with data from a bus cycle that was terminated abnormally. Stages of the pipe are only filled in response to specific prefetch requests issued by the sequencer.

Words are loaded into the instruction pipe from the cache holding register. Although the individual stages of the pipe are only 16 bits wide, the cache holding register is 32 bits wide and contains the entire long word. This long word is obtained from the instruction cache or the external bus in response to a prefetch request from the sequencer. When the sequencer requests an even-word (long-word-aligned) prefetch, the entire long word is accessed from the instruction cache or the external bus and loaded into the cache holding register, and the high-order word is also loaded into stage B of the pipe. The instruction word for the next sequential prefetch can then be accessed directly from the cache holding register, and no external bus cycle or instruction cache access is required. The cache holding register provides instruction words to the pipe regardless of whether the instruction cache is enabled or disabled.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

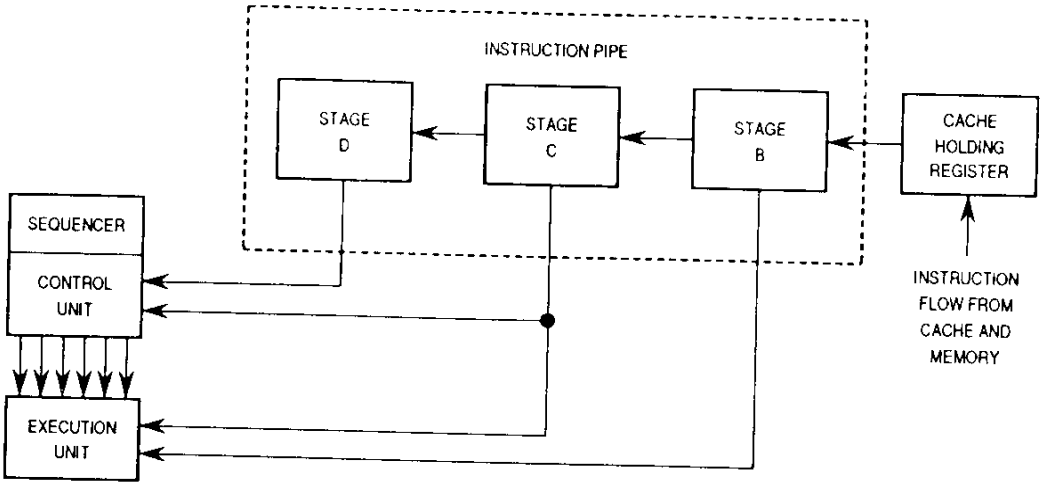


Figure 1-5. Instruction Pipe

The sequencer is either executing microinstructions or awaiting completion of accesses that are necessary to continue executing microcode. The bus controller is responsible for all bus activity. The sequencer controls the bus controller, instruction execution, and internal processor operations such as the calculation of effective addresses and the setting of condition codes. The sequencer initiates instruction word prefetches and controls the validation of instruction words in the instruction pipe.

Prefetch requests are simultaneously submitted to the cache holding register, the instruction cache, and the bus controller. Thus, even if the instruction cache is disabled, an instruction prefetch may hit in the cache holding register and cause an external bus cycle to be aborted.

1.7 CACHE MEMORY

Due to locality of reference, instructions that are used in a program have a high probability of being reused within a short time. Additionally, instructions that reside in proximity to the instructions currently in use also have a high probability of being utilized within a short period. To exploit these locality characteristics, the MC68020/EC020 contains an on-chip instruction cache.

The cache improves the overall performance of the system by reducing the number of bus cycles required by the processor to fetch information from memory and by increasing the bus bandwidth available for other bus masters in the system.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 2
PROCESSING STATES

2

This section describes the processing states of the MC68020/EC020. It describes the functions of the bits in the supervisor portion of the SR and the actions taken by the processor in response to exception conditions.

Unless the processor has halted, it is always in either the normal or the exception processing state. Whenever the processor is executing instructions or fetching instructions or operands, it is in the normal processing state. The processor is also in the normal processing state while it is storing instruction results or communicating with a coprocessor.

NOTE

Exception processing refers specifically to the transition from normal processing of a program to normal processing of system routines, interrupt routines, and other exception handlers. Exception processing includes all stacking operations, the fetch of the exception vector, and the filling of the instruction pipe caused by an exception. Exception processing has completed when execution of the first instruction of the exception handler routine begins.

The processor enters the exception processing state when an interrupt is acknowledged, when an instruction is traced or results in a trap, or when some other exception condition arises. Execution of certain instructions or unusual conditions occurring during the execution of any instruction can cause exceptions. External conditions, such as interrupts, bus errors, and some coprocessor responses, also cause exceptions. Exception processing provides an efficient transfer of control to handlers and routines that process the exceptions.

A catastrophic system failure occurs whenever the processor receives a bus error or generates an address error while in the exception processing state. This type of failure halts the processor. For example, if during the exception processing of one bus error another bus error occurs, the MC68020/EC020 has not completed the transition to normal processing and has not completed saving the internal state of the machine; therefore, the processor assumes that the system is not operational and halts. Only an external reset can restart a halted processor. (When the processor executes a STOP instruction, it is in a special type of normal processing state—one without bus cycles. It is stopped, not halted.)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

2.1 PRIVILEGE LEVELS

The processor operates at one of two privilege levels: the user level or the supervisor level. The supervisor level has higher privileges than the user level. Not all processor or coprocessor instructions are permitted to execute at the lower privileged user level, but all are available at the supervisor level. This arrangement allows a separation of supervisor and user so the supervisor can protect system resources from uncontrolled access. The S-bit in the SR is used to select either the user or supervisor privilege level and either the USP or an SSP for stack operations. The processor identifies a bus access (supervisor or user mode) via the function codes so that differentiation between supervisor level and user level can be maintained.

In many systems, the majority of programs execute at the user level. User programs can access only their own code and data areas and can be restricted from accessing other information. The operating system typically executes at the supervisor privilege level. It has access to all resources, performs the overhead tasks for the user-level programs, and coordinates user-level program activities.

2.1.1 Supervisor Privilege Level

The supervisor level is the higher privilege level. The privilege level is determined by the S-bit of the SR; if the S-bit is set, the supervisor privilege level applies, and all instructions are executable. The bus cycles for instructions executed at the supervisor level are normally classified as supervisor references, and the values of the FC2–FC0 signals refer to supervisor address spaces.

In a multitasking operating system, it is more efficient to have a supervisor stack space associated with each user task and a separate stack space for interrupt-associated tasks. The MC68020/EC020 provides two supervisor stacks, master and interrupt; the M bit of the SR selects which of the two is active. When the M-bit is set, references to the SSP implicitly or to address register seven (A7) explicitly, access the MSP. The operating system sets the MSP for each task to point to a task-related area of supervisor data space. This arrangement separates task-related supervisor activity from asynchronous, I/O-related supervisor tasks that may be only 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 other supervisor stack pointer, the ISP, can be used for interrupt control information and workspace area as interrupt handling routines require.

When the M-bit is clear, the MC68020/EC020 is in the interrupt mode of the supervisor privilege level, and operation is the same as supervisor mode in the MC68000, MC68HC001, MC68008, and MC68010. (The processor is in this mode after a reset operation.) All SSP references access the ISP in this mode.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The value of the M-bit in the SR does not affect execution of privileged instructions; both master and interrupt modes are at the supervisor privilege level. Instructions that affect the M-bit are MOVE to SR, ANDI to SR, EORI to SR, ORI to SR, and RTE. Also, the processor automatically saves the M-bit value and clears it in the SR as part of exception processing for interrupts.

All exception processing is performed at the supervisor privilege level. All bus cycles generated during exception processing are supervisor references, and all stack accesses use the active SSP.

2.1.2 User Privilege Level

The user level is the lower privilege level. The privilege level is determined by the S-bit of the SR; if the S-bit is clear, the processor executes instructions at the user privilege level.

Most instructions execute at either privilege level, but some instructions that have important system effects are privileged and can only be executed at the supervisor level. For instance, user programs are not allowed to execute the STOP instruction or the RESET instruction. To prevent a user program from entering the supervisor privilege level except in a controlled manner, instructions that can alter the S-bit in the SR are privileged. The TRAP #n instruction provides controlled access to operating system services for user programs.

The bus cycles for an instruction executed at the user privilege level are classified as user references, and the values of the FC2–FC0 signals specify user address spaces. While the processor is at the user level, references to the system stack pointer implicitly, or to address register seven (A7) explicitly, refer to the USP.

2.1.3 Changing Privilege Level

To change from the user to the supervisor privilege level, one of the conditions that causes the processor to perform exception processing must occur. This causes a change from the user level to the supervisor level and can cause a change from the master mode to the interrupt mode. Exception processing saves the current values of the S and M bits of the SR (along with the rest of the SR) on the active supervisor stack, and then sets the S-bit, forcing the processor into the supervisor privilege level. When the exception being processed is an interrupt and the M-bit is set, the M-bit is cleared, putting the processor into the interrupt mode. Execution of instructions continues at the supervisor level to process the exception condition.

To return to the user privilege level, a system routine must execute one of the following instructions: MOVE to SR, ANDI to SR, EORI to SR, ORI to SR, or RTE. These instructions execute at the supervisor privilege level and can modify the S-bit of the SR. After these instructions execute, the instruction pipeline is flushed and is refilled from the appropriate address space.

The RTE instruction returns to the program that was executing when the exception occurred. It restores the exception stack frame saved on the supervisor stack. If the frame

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

on top of the stack was generated by an interrupt, trap, or instruction exception, the RTE instruction restores the SR and PC to the values saved on the supervisor stack. The processor then continues execution at the restored PC address and at the privilege level determined by the S-bit of the restored SR. If the frame on top of the stack was generated by a bus fault (bus error or address error exception), the RTE instruction restores the entire saved processor state from the stack.

2.2 ADDRESS SPACE TYPES

The processor specifies a target address space for every bus cycle with the FC2–FC0 signals according to the type of access required. In addition to distinguishing between supervisor/user and program/data, the processor can identify special processor cycles, such as the interrupt acknowledge cycle, and the memory management unit can control accesses and translate addresses appropriately. Table 2-1 lists the types of accesses defined for the MC68020/EC020 and the corresponding values of the FC2–FC0 signals.

Table 2-1. Address Space Encodings

FC2	FC1	FC0	Address Space
0	0	0	(Undefined, Reserved)*
0	0	1	User Data Space
0	1	0	User Program Space
0	1	1	(Undefined, Reserved)*
1	0	0	(Undefined, Reserved)*
1	0	1	Supervisor Data Space
1	1	0	Supervisor Program Space
1	1	1	CPU Space

* Address space 3 is reserved for user definition; 0 and 4 are reserved for future use by Motorola.

The memory locations of user program and data accesses are not predefined; neither are the locations of supervisor data space. During reset, the first two long words beginning at memory location zero in the supervisor program space are used for processor initialization. No other memory locations are explicitly defined by the MC68020/EC020.

A function code of $7 selects the CPU address space. This is a special address space that does not contain instructions or operands but is reserved for special processor functions. The processor uses accesses in this space to communicate with external devices for special purposes. For example, all M68000 processors use the CPU space for interrupt acknowledge cycles. The MC68020/EC020 also generate CPU space accesses for breakpoint acknowledge and coprocessor operations.

Supervisor programs can use the MOVES instruction to access all address spaces, including the user spaces and the CPU address space. Although the MOVES instruction can be used to generate CPU space cycles, this may interfere with proper system operation. Thus, the use of MOVES to access the CPU space should be done with caution.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

2.3 EXCEPTION PROCESSING

An exception is defined as a special condition that preempts normal processing. Both internal and external conditions can cause exceptions. External conditions that cause exceptions are interrupts from external devices, bus errors, coprocessor detected errors, and reset. Instructions, address errors, tracing, and breakpoints are internal conditions that cause exceptions. The TRAP, TRAPcc, TRAPV, cpTRAPcc, CHK, CHK2, RTE, BKPT, CALLM, RTM, cp RESTORE, DIVS and DIVU instructions can generate exceptions as part of their normal execution. In addition, illegal instructions, privilege violations, and coprocessor protocol violations cause exceptions.

Exception processing, which is the transition from the normal processing of a program to the processing required for the exception condition, involves the exception vector table and an exception stack frame. The following paragraphs describe the exception vectors and a generalized exception stack frame. Exception processing is discussed in detail in **Section 6 Exception Processing**. Coprocessor-detected exceptions are discussed in detail in **Section 7 Coprocessor Interface Description**.

2.3.1 Exception Vectors

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

The address of an exception vector is derived from an 8-bit vector number and the VBR. The vector numbers for some exceptions are obtained from an external device; others are supplied automatically by the processor. The processor multiplies the vector number by four to calculate the vector offset, which it adds to the VBR. The sum is the memory address of the vector. All exception vectors are located in supervisor data space, except the reset vector, which is located in supervisor program space. Only the initial reset vector is fixed in the processor's memory map; once initialization is complete, there are no fixed assignments. Since the VBR provides the base address of the vector table, the vector table can be located anywhere in memory; it can even be dynamically relocated for each task that is executed by an operating system. Details of exception processing are provided in **Section 6 Exception Processing**, and Table 6-1 lists the exception vector assignments.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

2.3.2 Exception Stack Frame

Exception processing saves the most volatile portion of the current processor context on the top of the supervisor stack. This context is organized in a format called the exception stack frame. This information always includes a copy of the SR, the PC, the vector offset of the vector, and the frame format field. The frame format field identifies the type of stack frame. The RTE instruction uses the value in the format field to properly restore the information stored in the stack frame and to deallocate the stack space. The general form of the exception stack frame is illustrated in Figure 2-1. Refer to **Section 6 Exception Processing** for a complete list of exception stack frames.

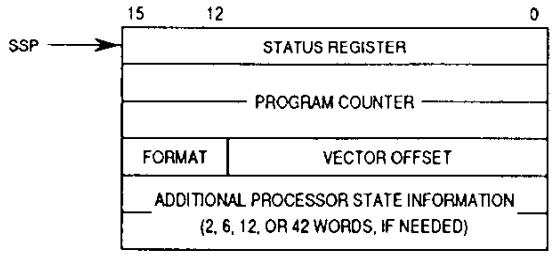


Figure 2-1. General Exception Stack Frame

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

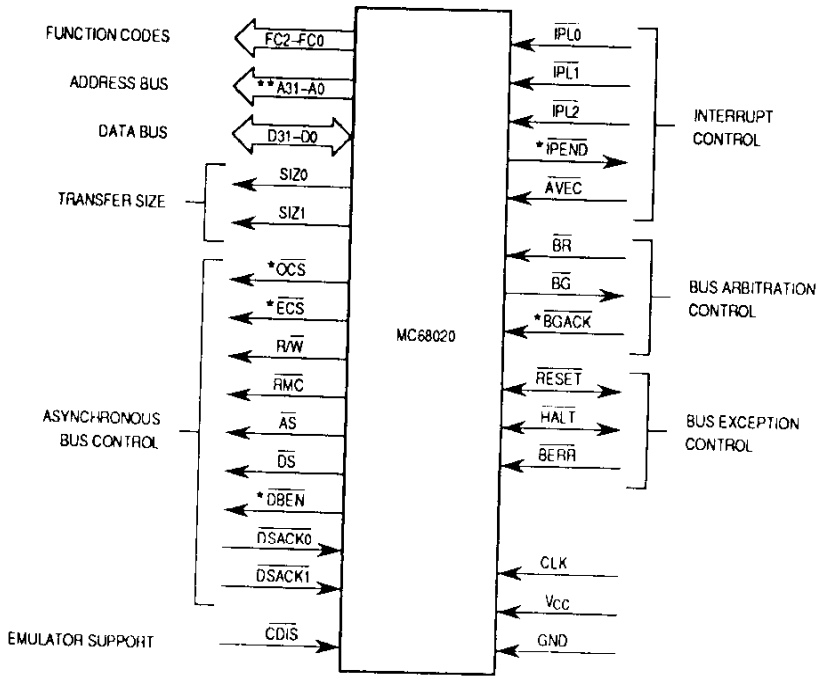
SECTION 3
SIGNAL DESCRIPTION

3

This section contains brief descriptions of the input and output signals in their functional groups, as shown in Figure 3-1. Each signal is explained in a brief paragraph with reference to other sections that contain more detail about the signal and the related operations.

NOTE

In this section and in the remainder of the manual, *assert* and *negate* are used to specify forcing a signal to a particular state. In particular, *assertion* and *assert* refer to a signal that is active or true; *negation* and *negate* indicate a signal that is inactive or false. These terms are used independently of the voltage level (high or low) that they represent.



* This signal is implemented in the MC68020 and not implemented in the MC68EC020.
** A23–A0 for the MC68EC020.

Figure 3-1. Functional Signal Groups

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

3.1 SIGNAL INDEX

The input and output signals for the MC68020/EC020 are listed in Table 3-1. Both the names and mnemonics are shown along with brief signal descriptions. Signals that are implemented in the MC68020, but not in the MC68EC020, have an asterisk (*) preceding the signal name in Table 3-1. Also, note that the address bus is 32 bits wide for the MC68020 and 24 bits wide for the MC68EC020. For more detail on each signal, refer to the paragraph in this section named for the signal and the reference in that paragraph to a description of the related operations.

Timing specifications for the signals listed in Table 3-1 can be found in **Section 10 Electrical Characteristics**.

3.2 FUNCTION CODE SIGNALS (FC2–FC0)

These three-state outputs identify the address space of the current bus cycle. Table 2-1 shows the relationships of the function code signals to the privilege levels and the address spaces. Refer to **Section 2 Processing States** for more information.

3.3 ADDRESS BUS (A31–A0, MC68020)(A23–A0, MC68EC020)

These three-state outputs provide the address for the current bus cycle, except in the CPU address space. Refer to **Section 2 Processing States** for more information on the CPU address space. A31 is the most significant address signal for the MC68020; A23 is the most significant address signal for the MC68EC020. The upper eight bits (A31–A24) are used internally by the MC68EC020 to access the internal instruction cache address tag. Refer to **Section 5 Bus Operation** for information on the address bus and its relationship to bus operation.

3.4 DATA BUS (D31–D0)

These three-state bidirectional signals provide the general-purpose data path between the MC68020/EC020 and all other devices. The data bus can transfer 8, 16, 24, or 32 bits of data per bus cycle. D31 is the most significant bit of the data bus. Refer to **Section 5 Bus Operation** for more information on the data bus and its relationship to bus operation.

3.5 TRANSFER SIZE SIGNALS (SIZ1, SIZ0)

These three-state outputs indicate the number of bytes remaining to be transferred for the current bus cycle. Signals A1, A0, $\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$, SIZ1, and SIZ0 define the number of bits transferred on the data bus. Refer to **Section 5 Bus Operation** for more information on SIZ1 and SIZ0 and their use in dynamic bus sizing.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 3-1. Signal Index

Signal Name	Mnemonic	Function
Function Codes	FC2–FC0	3-bit function code used to identify the address space of each bus cycle.
Address Bus MC68020 MC68EC020	 A31–A0 A23–A0	 32-bit address bus 24-bit address bus
Data Bus	D31–D0	32-bit data bus used to transfer 8, 16, 24, or 32 bits of data per bus cycle.
Size	SIZ1, SIZ0	Indicates the number of bytes remaining to be transferred for this cycle. These signals, together with A1 and A0, define the active sections of the data bus.
*External Cycle Start	$\overline{ECS}$	Provides an indication that a bus cycle is beginning.
*Operand Cycle Start	$\overline{OCS}$	Identical operation to that of $\overline{ECS}$ except that $\overline{OCS}$ is asserted only during the first bus cycle of an operand transfer.
Read/Write	R/$\overline{W}$	Defines the bus transfer as a processor read or write.
Read-Modify-Write Cycle	$\overline{RMC}$	Provides an indicator that the current bus cycle is part of an indivisible read-modify-write operation.
Address Strobe	$\overline{AS}$	Indicates that a valid address is on the bus.
Data Strobe	$\overline{DS}$	Indicates that valid data is to be placed on the data bus by an external device or has been placed on the data bus by the MC68020/EC020.
*Data Buffer Enable	$\overline{DBEN}$	Provides an enable signal for external data buffers.
Data Transfer and Size Acknowledge	$\overline{DSACK1}$, $\overline{DSACK0}$	Bus response signals that indicate the requested data transfer operation has completed. In addition, these two lines indicate the size of the external bus port on a cycle-by-cycle basis and are used for asynchronous transfers.
Interrupt Priority Level	$\overline{IPL2}$–$\overline{IPL0}$	Provides an encoded interrupt level to the processor.
*Interrupt Pending	$\overline{IPEND}$	Indicates that an interrupt is pending.
Autovector	$\overline{AVEC}$	Requests an autovector during an interrupt acknowledge cycle.
Bus Request	$\overline{BR}$	Indicates that an external device requires bus mastership.
Bus Grant	$\overline{BG}$	Indicates that an external device may assume bus mastership.
*Bus Grant Acknowledge	$\overline{BGACK}$	Indicates that an external device has assumed bus mastership.
Reset	$\overline{RESET}$	System reset.
Halt	$\overline{HALT}$	Indicates that the processor should suspend bus activity or that the processor has halted due to a double bus fault.
Bus Error	$\overline{BERR}$	Indicates that an erroneous bus operation is being attempted.
Cache Disable	$\overline{CDIS}$	Statically disables the on-chip cache to assist emulator support.
Clock	CLK	Clock input to the processor.
Power Supply	V_{CC}	Power supply.
Ground	GND	Ground connection.

*This signal is implemented in the MC68020 and not implemented in the MC68EC020.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

3.6 ASYNCHRONOUS BUS CONTROL SIGNALS

The following signals control synchronous bus transfer operations for the MC68020/EC020. Note that $\overline{OCS}$, $\overline{ECS}$, and $\overline{DBEN}$ are implemented in MC68020 and not implemented in the MC68EC020.

Operand Cycle Start ($\overline{OCS}$, MC68020 only)

This output signal indicates the beginning of the first external bus cycle for an instruction prefetch or a data operand transfer. $\overline{OCS}$ is not asserted for subsequent cycles that are performed due to dynamic bus sizing or operand misalignment. Refer to **Section 5 Bus Operation** for information about the relationship of $\overline{OCS}$ to bus operation.

$\overline{OCS}$ is not implemented in the MC68EC020.

External Cycle Start ($\overline{ECS}$, MC68020 only)

This output signal indicates the beginning of a bus cycle of any type. Refer to **Section 5 Bus Operation** for information about the relationship of $\overline{ECS}$ to bus operation.

$\overline{ECS}$ is not implemented in the MC68EC020.

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

This three-state output signal defines the type of bus cycle. A high level indicates a read cycle; a low level indicates a write cycle. Refer to **Section 5 Bus Operation** for information about the relationship of R/$\overline{W}$ to bus operation.

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

This three-state output signal identifies the current bus cycle as part of an indivisible read-modify-write operation; it remains asserted during all bus cycles of the read-modify-write operation. Refer to **Section 5 Bus Operation** for information about the relationship of $\overline{RMC}$ to bus operation.

Address Strobe ($\overline{AS}$)

This three-state output signal indicates that a valid address is on the address bus. The FC2–FC0, SIZ1, SIZ0, and R/$\overline{W}$ signals are also valid when $\overline{AS}$ is asserted. Refer to **Section 5 Bus Operation** for information about the relationship of $\overline{AS}$ to bus operation.

Data Strobe ($\overline{DS}$)

During a read cycle, this three-state output signal indicates that an external device should place valid data on the data bus. During a write cycle, $\overline{DS}$ indicates that the MC68020/EC020 has placed valid data on the bus. During two-clock synchronous write cycles, the MC68020/EC020 does not assert $\overline{DS}$. Refer to **Section 5 Bus Operation** for more information about the relationship of $\overline{DS}$ to bus operation.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Data Buffer Enable ($\overline{DBEN}$, MC68020 only)

This output signal is an enable signal for external data buffers. This signal may not be required in all systems. Refer to **Section 5 Bus Operation** for more information about the relationship of $\overline{DBEN}$ to bus operation.

$\overline{DBEN}$ is not implemented in the MC68EC020.

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

These input signals indicate the completion of a requested data transfer operation. In addition, they indicate the size of the external bus port at the completion of each cycle. These signals apply only to asynchronous bus cycles. Refer to **Section 5 Bus Operation** for more information on these signals and their relationship to dynamic bus sizing.

3.7 INTERRUPT CONTROL SIGNALS

The following signals are the interrupt control signals for the MC68020/EC020. Note that $\overline{IPEND}$ is implemented in the MC68020 and not implemented in the MC68EC020.

Interrupt Priority Level Signals ($\overline{IPL2}$–$\overline{IPL0}$)

These input signals provide an indication of an interrupt condition and the encoding of the interrupt level from a peripheral or external prioritizing circuitry. $\overline{IPL2}$ is the most significant bit of the level number. For example, since the $\overline{IPL2}$–$\overline{IPL0}$ signals are active low, $\overline{IPL2}$–$\overline{IPL0}$ equal to $5 corresponds to an interrupt request at interrupt level 2. Refer to **Section 6 Exception Processing** for information on MC68020/EC020 interrupts.

Interrupt Pending ($\overline{IPEND}$, MC68020 only)

This output signal indicates that an interrupt request exceeding the current interrupt priority mask in the SR has been recognized internally. This output is for use by external devices (coprocessors and other bus masters, for example) to predict processor operation on the following instruction boundaries. Refer to **Section 6 Exception Processing** for interrupt information. Also, refer to **Section 5 Bus Operation** for bus information related to interrupts.

$\overline{IPEND}$ is not implemented in the MC68EC020.

Autovector ($\overline{AVEC}$)

This input signal indicates that the MC68020/EC020 should generate an automatic vector during an interrupt acknowledge cycle. Refer to **Section 5 Bus Operation** for more information about automatic vectors.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

3.8 BUS ARBITRATION CONTROL SIGNALS

The following signals are the bus arbitration control signals used to determine which device in a system is the bus master. Note that $\overline{BGACK}$ is implemented in the MC68020 and not implemented in the MC68EC020.

Bus Request ($\overline{BR}$)

This input signal indicates that an external device needs to become the bus master. $\overline{BR}$ is typically a "wire-ORed" input (but does not need to be constructed from open-collector devices). Refer to **Section 5 Bus Operation** for more information on MC68020 bus arbitration. Refer to **Section 5 Bus Operation** and **Appendix A Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol** for more information on MC68EC020 bus arbitration.

Bus Grant ($\overline{BG}$)

This output signal indicates that the MC68020/EC020 will release ownership of the bus when the current processor bus cycle completes. Refer to **Section 5 Bus Operation** for more information on MC68020 bus arbitration. Refer to **Section 5 Bus Operation** and **Appendix A Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol** for more information on MC68EC020 bus arbitration.

Bus Grant Acknowledge ($\overline{BGACK}$, MC68020 only)

This input signal indicates that an external device has become the bus master. Refer to **Section 5 Bus Operation** for more information on MC68020 bus arbitration. Refer to **Section 5 Bus Operation** and **Appendix A Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol** for more information on MC68EC020 bus arbitration.

$\overline{BGACK}$ is not implemented in the MC68EC020.

3.9 BUS EXCEPTION CONTROL SIGNALS

The following signals are the bus exception control signals for the MC68020/EC020.

Reset ($\overline{RESET}$)

This bidirectional open-drain signal is used to initiate a system reset. An external reset signal resets the MC68020/EC020 as well as all external devices. A reset signal from the processor (asserted as part of the RESET instruction) resets external devices only; the internal state of the processor is not altered. Refer to **Section 5 Bus Operation** for a description of reset bus operation and **Section 6 Exception Processing** for information about the reset exception.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Halt ($\overline{HALT}$)

The assertion of this bidirectional open-drain signal indicates that the processor should suspend bus activity or, when used with $\overline{BERR}$, that the processor should retry the current cycle. Refer to **Section 5 Bus Operation** for a description of the effects of $\overline{HALT}$ on bus operations. When the processor has stopped executing instructions due to a double bus fault condition, the $\overline{HALT}$ line is asserted by the processor to indicate to external devices that the processor has stopped.

Bus Error ($\overline{BERR}$)

This input signal indicates that an invalid bus operation is being attempted or, when used with $\overline{HALT}$, that the processor should retry the current cycle. Refer to **Section 5 Bus Operation** for a description of the effects of $\overline{BERR}$ on bus operations.

3.10 EMULATOR SUPPORT SIGNAL

The following signal supports emulation by providing a means for an emulator to disable the on-chip cache by supplying internal status information to an emulator. Refer to **Section 7 Coprocessor Interface Description** for more detailed information on emulation support.

Cache Disable ($\overline{CDIS}$)

This input signal statically disables the on-chip cache to assist emulator support. Refer to **Section 4 On-Chip Cache Memory** for information about the cache; refer to **Section 9 Applications Information** for a description of the use of this signal by an emulator. $\overline{CDIS}$ does not flush the instruction cache; entries remain unaltered and become available again when $\overline{CDIS}$ is negated.

3.11 CLOCK (CLK)

The CLK signal is the clock input to the MC68020/EC020. This TTL-compatible signal should not be gated off at any time while power is applied to the processor. Refer to **Section 9 Applications Information** for suggestions on clock generation. Refer to **Section 10 Electrical Characteristics** for electrical characteristics.

3.12 POWER SUPPLY CONNECTIONS

The MC68020/EC020 requires connection to a V_{CC} power supply, positive with respect to ground. The V_{CC} connections are grouped to supply adequate current for the various sections of the processor. The ground connections are similarly grouped. **Section 11 Ordering Information and Mechanical Data** describes the groupings of V_{CC} and ground connections, and **Section 9 Applications Information** describes a typical power supply interface.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

3.13 SIGNAL SUMMARY

Table 3-2 provides a summary of the characteristics of the signals discussed in this section. Signal names preceded by an asterisk (*) are implemented in the MC68020 and not implemented in the MC68EC020.

Table 3-2. Signal Summary

Signal Function	Signal Name	Input/Output	Active State	Three-State
Function Codes	FC2–FC0	Output	High	Yes
Address Bus MC68020 MC68EC020	 A31–A0 A23–A0	Output	High	Yes
Data Bus	D31–D0	Input/Output	High	Yes
Transfer Size	SIZ1, SIZ0	Output	High	Yes
*Operand Cycle Start	$\overline{OCS}$	Output	Low	No
*External Cycle Start	$\overline{ECS}$	Output	Low	No
Read/Write	R/$\overline{W}$	Output	High/Low	Yes
Read-Modify-Write Cycle	$\overline{RMC}$	Output	Low	Yes
Address Strobe	$\overline{AS}$	Output	Low	Yes
Data Strobe	$\overline{DS}$	Output	Low	Yes
*Data Buffer Enable	$\overline{DBEN}$	Output	Low	Yes
Data Transfer and Size Acknowledge	$\overline{DSACK1}$, $\overline{DSACK0}$	Input	Low	—
Interrupt Priority Level	$\overline{IPL2}$–$\overline{IPL0}$	Input	Low	—
*Interrupt Pending	$\overline{IPEND}$	Output	Low	No
Autovector	$\overline{AVEC}$	Input	Low	—
Bus Request	$\overline{BR}$	Input	Low	—
Bus Grant	$\overline{BG}$	Output	Low	No
*Bus Grant Acknowledge	$\overline{BGACK}$	Input	Low	—
Reset	$\overline{RESET}$	Input/Output	Low	No**
Halt	$\overline{HALT}$	Input/Output	Low	No**
Bus Error	$\overline{BERR}$	Input	Low	—
Cache Disable	$\overline{CDIS}$	Input	Low	—
Clock	CLK	Input	—	—
Power Supply	V_{CC}	Input	—	—
Ground	GND	Input	—	—

*This signal is implemented in the MC68020 and not implemented in the MC68EC020.

**Open-drain

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 4
ON-CHIP CACHE MEMORY

The MC68020/EC020 incorporates an on-chip cache memory as a means of improving performance. The cache is implemented as a CPU instruction cache and is used to store the instruction stream prefetch accesses from the main memory.

An increase in instruction throughput results when instruction words required by a program are available in the on-chip cache and the time required to access them on the external bus is eliminated. In systems with more than one bus master (e.g., a processor and a DMA device), reduced external bus activity increases overall performance by increasing the availability of the bus for use by external devices without degrading the performance of the MC68020/EC020.

4.1 ON-CHIP CACHE ORGANIZATION AND OPERATION

The MC68020/EC020 on-chip instruction cache is a direct-mapped cache of 64 long-word entries. Each cache entry consists of a tag field (A31–A8 and FC2), one valid bit, and 32 bits (two words) of instruction data. Figure 4-1 shows a block diagram of the on-chip cache organization.

Externally, the MC68EC020 does not use the upper eight bits of the address (A31–A24), and addresses $FF000000 and $00000000 from the MC68EC020 appear the same. However, the MC68EC020 does use A31–A24 internally in the instruction cache address tag, and addresses $FF000000 and $00000000 appear different in the MC68EC020 instruction cache. The MC68020, MC68030/EC030, and MC68040/EC040 use all 32 bits of the address externally. To maintain object-code upgrade compatibility when designing with the MC68EC020, the upper eight bits should be considered part of the address when assigning address spaces in hardware.

When enabled, the MC68020/EC020 instruction cache is used to store instruction prefetches (instruction words and extension words) as they are requested by the CPU. Instruction prefetches are normally requested from sequential memory addresses except when a change of program flow occurs (e.g., a branch taken) or when an instruction is executed that can modify the SR. In these cases, the instruction pipe is automatically flushed and refilled.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

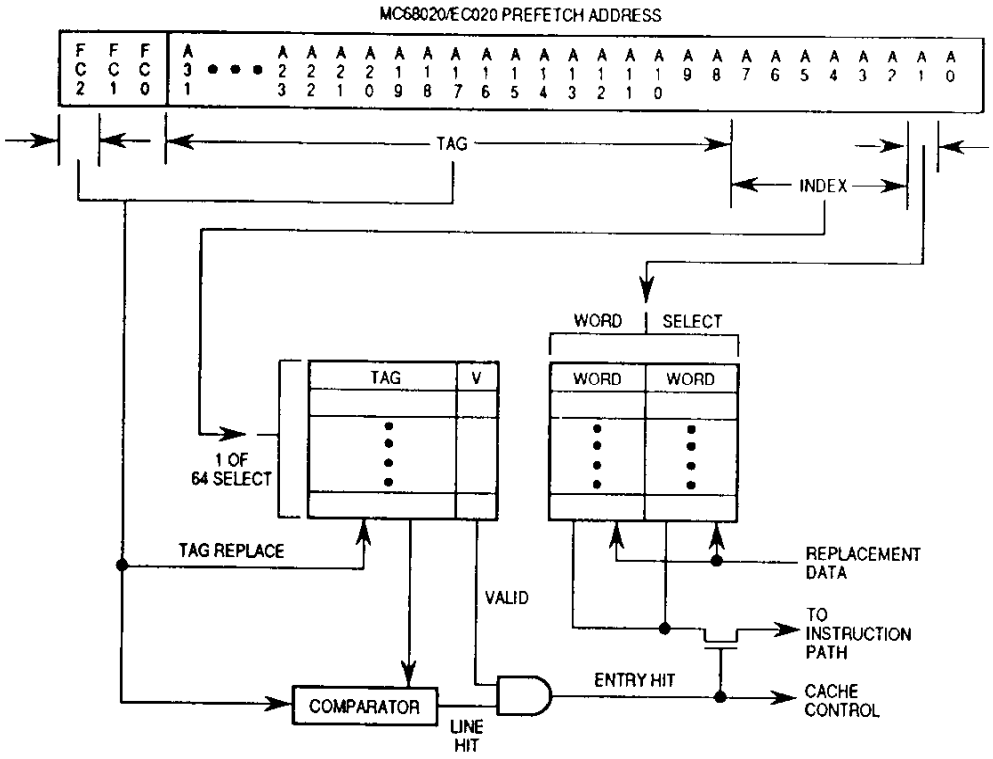


Figure 4-1. MC68020/EC020 On-Chip Cache Organization

When an instruction fetch occurs, the cache (if enabled) is first checked to determine if the word required is in the cache. This check is achieved by first using the index field (A7–A2) of the access address as an index into the on-chip cache. This index selects one of the 64 entries in the cache. Next, A31–A8 and FC2 are compared to the tag of the selected entry. (Note that in the MC68EC020, A31–A24 are used for internal on-chip cache tag comparison.) If there is a match and the valid bit is set, a cache hit occurs. A1 is then used to select the proper word from the cache entry, and the cycle ends. If there is no match or if the valid bit is clear, a cache miss occurs, and the instruction is fetched from external memory. This new instruction is automatically written into the cache entry, and the valid bit is set unless the F-bit in the CACR is set. Since the processor always prefetches instructions externally with long-word-aligned bus cycles, both words of the entry will be updated, regardless of which word caused the miss.

NOTE

Data accesses are not cached, regardless of their associated address space.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.2 CACHE RESET

During processor reset, the cache is cleared by resetting all of the valid bits. The E and F bits in the CACR are also cleared.

4.3 CACHE CONTROL

Only the MC68020/EC020 cache control circuitry can directly access the cache array, but a supervisor program can set bits in the CACR to exercise control over cache operations. The supervisor level also has access to the CAAR, which contains the address for a cache entry to be cleared.

System hardware can assert the $\overline{CDIS}$ signal to disable the cache. The assertion of $\overline{CDIS}$ disables the cache, regardless of the state of the E-bit in the CACR. $\overline{CDIS}$ is primarily intended for use by in-circuit emulators.

4.3.1 Cache Control Register (CACR)

The CACR, shown in Figure 4-2, is a 32-bit register than can be written or read by the MOVEC instruction or indirectly modified by a reset. Four of the bits (3–0) control the instruction cache. Bits 31–4 are reserved for Motorola definition. They are read as zeros and are ignored when written. For future compatibility, writes should not set these bits.

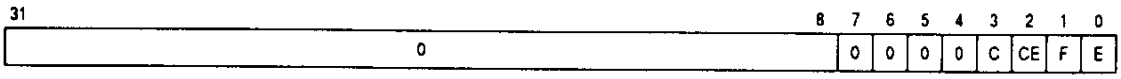


Figure 4-2. Cache Control Register

C—Clear Cache

The C-bit is set to clear all entries in the instruction cache. Operating systems and other software set this bit to clear instructions from the cache prior to a context switch. The processor clears all valid bits in the instruction cache when a MOVEC instruction sets the C-bit. The C-bit is always read as a zero.

CE—Clear Entry In Cache

The CE bit is set to clear an entry in the instruction cache. The index field of the CAAR (see Figure 4-3), corresponding to the index and long-word select portion of an address, specifies the entry to be cleared. The processor clears only the specified long word by clearing the valid bit for the entry when a MOVEC instruction sets the CE bit, regardless of the states of the E and F bits. The CE bit is always read as a zero.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

F—Freeze Cache

The F-bit is set to freeze the instruction cache. When the F-bit is set and a cache miss occurs, the entry (or line) is not replaced. When the F-bit is clear, a cache miss causes the entry (or line) to be filled. A reset operation clears the F-bit.

E—Enable Cache

The E-bit is set to enable the instruction cache. When it is clear, the instruction cache is disabled. A reset operation clears the E-bit. The supervisor normally enables the instruction cache, but it can clear the E-bit for system debugging or emulation, as required. Disabling the instruction cache does not flush the entries. If the cache is reenabled, the previously valid entries remain valid and may be used.

4.3.2 Cache Address Register (CAAR)

The format of the 32-bit CAAR is shown in Figure 4-3.

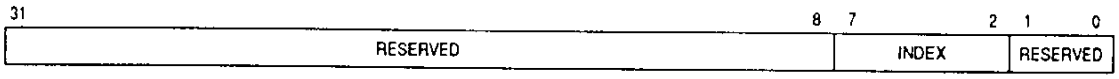


Figure 4-3. Cache Address Register

Bits 31–8, 1, and 0—Reserved

These bits are reserved for use by Motorola.

Index Field

The index field contains the address for the "clear cache entry" operations. The bits of this field, which correspond to A7–A2, specify the index and a long word of a cache line.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 5
BUS OPERATION

This section provides a functional description of the bus, the signals that control it, and the bus cycles provided for data transfer operations. It also describes the error and halt conditions, bus arbitration, and reset operation. Operation of the bus is the same whether the processor or an external device is the bus master; the names and descriptions of bus cycles are from the point of view of the bus master. For exact timing specifications, refer to **Section 10 Electrical Characteristics**.

The MC68020/EC020 architecture supports byte, word, and long-word operands, allowing access to 8-, 16-, and 32-bit data ports through the use of asynchronous cycles controlled by the $\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$ input signals.

The MC68020/EC020 allows byte, word, and long-word operands to be located in memory on any byte boundary. For a misaligned transfer, more than one bus cycle may be required to complete the transfer, regardless of port size. For a port less than 32 bits wide, multiple bus cycles may be required for an operand transfer due to either misalignment or a port width smaller than the operand size. Instruction words and their associated extension words must be aligned on word boundaries. The user should be aware that misalignment of word or long-word operands can cause the MC68020/EC020 to perform multiple bus cycles for the operand transfer; therefore, processor performance is optimized if word and long-word memory operands are aligned on word or long-word boundaries, respectively.

5.1 BUS TRANSFER SIGNALS

The bus transfers information between the MC68020/EC020 and an external memory, coprocessor, or peripheral device. External devices can accept or provide 8 bits, 16 bits, or 32 bits in parallel and must follow the handshake protocol described in this section. The maximum number of bits accepted or provided during a bus transfer is defined as the port width. The MC68020/EC020 contains an address bus that specifies the address for the transfer and a data bus that transfers the data. Control signals indicate the beginning of the cycle, the address space and size of the transfer, and the type of cycle. The selected device then controls the length of the cycle with the signal(s) used to terminate the cycle. Strobe signals, one for the address bus and another for the data bus, indicate the validity of the address and provide timing information for the data.

The bus operates in an asynchronous mode for any port width. The bus and control input signals are internally synchronized to the MC68020/EC020 clock, introducing a delay. This delay is the time period required for the MC68020/EC020 to sample an input signal, synchronize the input to the internal clocks of the processor, and determine whether the

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

input is high or low. Figure 5-1 shows the relationship between the clock signal, a typical input, and its associated internal signal.

Furthermore, for all inputs, the processor latches the level of the input during a sample window around the falling edge of the clock signal. This window is illustrated in Figure 5-2. To ensure that an input signal is recognized on a specific falling edge of the clock, that input must be stable during the sample window. If an input transitions during the window, the level recognized by the processor is not predictable; however, the processor always resolves the latched level to either a logic high or logic low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

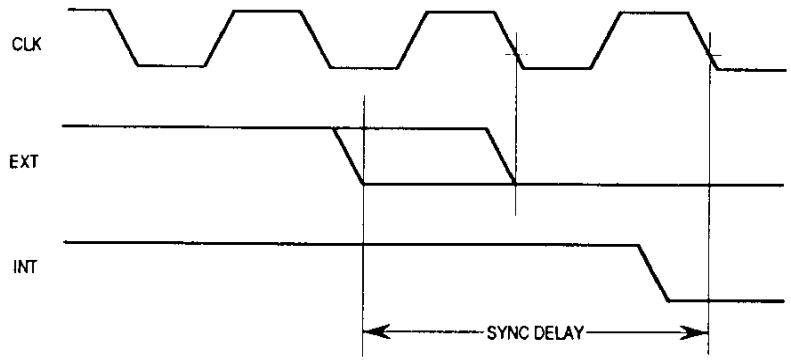


Figure 5-1. Relationship between External and Internal Signals

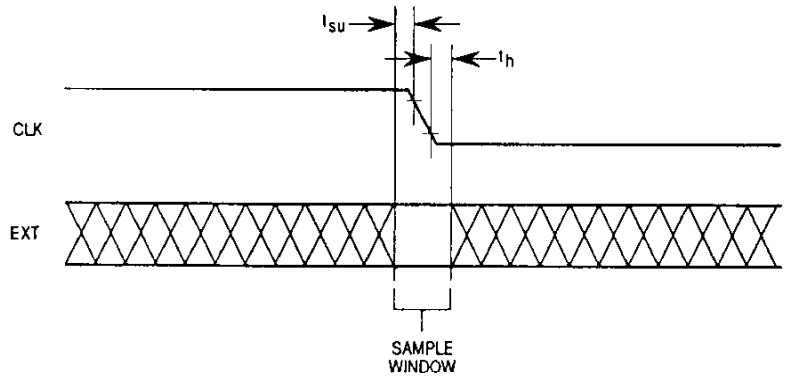


Figure 5-2. Input Sample Window

5.1.1 Bus Control Signals

The MC68020/EC020 initiates a bus cycle by driving the A1–A0, SIZ1, SIZ0, FC2–FC0, and R/$\overline{W}$ outputs. However, if the MC68020/EC020 finds the required instruction in the on-chip cache, the processor aborts the cycle before asserting the $\overline{AS}$.The assertion of $\overline{AS}$ ensures that the cycle has not been aborted by these internal conditions.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

When initiating a bus cycle, the MC68020 asserts $\overline{ECS}$ in addition to A1–A0, SIZ1, SIZ0, FC2–FC0, and R/$\overline{W}$. $\overline{ECS}$ can be used to initiate various timing sequences that are eventually qualified with $\overline{AS}$. Qualification with $\overline{AS}$ may be required since, in the case of an internal cache hit, a bus cycle may be aborted after $\overline{ECS}$ has been asserted. During the first MC68020 external bus cycle of an operand transfer, $\overline{OCS}$ is asserted with $\overline{ECS}$. When several bus cycles are required to transfer the entire operand, $\overline{OCS}$ is asserted only at the beginning of the first external bus cycle. With respect to $\overline{OCS}$, an "operand" is any entity required by the execution unit, whether a program or data item. Note that $\overline{ECS}$ and $\overline{OCS}$ are not implemented in the MC68EC020.

The FC2–FC0 signals select one of eight address spaces (see Table 2-1) to which the address applies. Five address spaces are presently defined. Of the remaining three, one is reserved for user definition, and two are reserved by Motorola for future use. FC2–FC0 are valid while AS is asserted.

The SIZ1 and SIZ0 signals indicate the number of bytes remaining to be transferred during an operand cycle (consisting of one or more bus cycles) or during a cache fill operation from a device with a port size that is less than 32 bits. Table 5-2 lists the encoding of SIZ1 and SIZ0. SIZ1 and SIZ0 are valid while $\overline{AS}$ is asserted.

The R/$\overline{W}$ signal determines the direction of the transfer during a bus cycle. When required, this signal changes state at the beginning of a bus cycle and is valid while $\overline{AS}$ is asserted. R/$\overline{W}$ only transitions when a write cycle is preceded by a read cycle or vice versa. This signal may remain low for two consecutive write cycles.

The $\overline{RMC}$ signal is asserted at the beginning of the first bus cycle of a read-modify-write operation and remains asserted until completion of the final bus cycle of the operation. The $\overline{RMC}$ signal is guaranteed to be negated before the end of state 0 for a bus cycle following a read-modify-write operation.

5.1.2 Address Bus

A31–A0 (for the MC68020) or A23–A0 (for the MC68EC020) define the address of the byte (or the most significant byte) to be transferred during a bus cycle. The processor places the address on the bus at the beginning of a bus cycle. The address is valid while $\overline{AS}$ is asserted. In the MC68EC020, A31–A24 are used internally, but not externally.

5.1.3 Address Strobe

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

5.1.4 Data Bus

D31–D0 comprise a bidirectional, nonmultiplexed parallel bus that contains the data being transferred to or from the processor. A read or write operation may transfer 8, 16, 24, or 32 bits of data (one, two, three, or four bytes) in one bus cycle. During a read cycle, the data is latched by the processor on the last falling edge of the clock for that bus cycle. For

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

a write cycle, all 32 bits of the data bus are driven, regardless of the port width or operand size. The processor places the data on the data bus one-half clock cycle after $\overline{AS}$ is asserted in a write cycle.

5.1.5 Data Strobe

$\overline{DS}$ is a timing signal that applies to the data bus. For a read cycle, the processor asserts $\overline{DS}$ to signal the external device to place data on the bus. $\overline{DS}$ is asserted at the same time as $\overline{AS}$ during a read cycle. For a write cycle, $\overline{DS}$ notifies the external device that the data to be written is valid. The processor asserts $\overline{DS}$ one full clock cycle after the assertion of $\overline{AS}$ during a write cycle.

5.1.6 Data Buffer Enable

The MC68020 $\overline{DBEN}$ signal is used to enable external data buffers while data is present on the data bus. During a read operation, $\overline{DBEN}$ is asserted one clock cycle after the beginning of the bus cycle and is negated as $\overline{DS}$ is negated. In a write operation, $\overline{DBEN}$ is asserted at the time $\overline{AS}$ is asserted and is held active for the duration of the cycle. Note that $\overline{DBEN}$ is implemented in the MC68020 and is not implemented in the MC68EC020.

5.1.7 Bus Cycle Termination Signals

During bus cycles, external devices assert $\overline{DSACK1}/\overline{DSACK0}$ as part of the bus protocol. During a read cycle, $\overline{DSACK1}/\overline{DSACK0}$ assertion signals the processor to terminate the bus cycle and to latch the data. During a write cycle, the assertion of $\overline{DSACK1}/\overline{DSACK0}$ indicates that the external device has successfully stored the data and that the cycle may terminate. $\overline{DSACK1}/\overline{DSACK0}$ also indicate to the processor the size of the port for the bus cycle just completed, as shown in Table 5-1. Refer to **5.3.1 Read Cycle** for timing relationships of $\overline{DSACK1}/\overline{DSACK0}$.

The $\overline{BERR}$ signal is also a bus cycle termination indicator and can be used in the absence of $\overline{DSACK1}/\overline{DSACK0}$ to indicate a bus error condition. It can also be asserted in conjunction with $\overline{DSACK1}/\overline{DSACK0}$ to indicate a bus error condition, provided it meets the appropriate timing described in this section and in **Section 10 Electrical Characteristics**. Additionally, the $\overline{BERR}$ and $\overline{HALT}$ signals can be asserted together to indicate a retry termination. Again, the $\overline{BERR}$ and $\overline{HALT}$ signals can be simultaneously asserted in lieu of, or in conjunction with, the $\overline{DSACK1}/\overline{DSACK0}$ signals.

Finally, the $\overline{AVEC}$ signal can be used to terminate interrupt acknowledge cycles, indicating that the MC68020/EC020 should generate a vector number to locate an interrupt handler routine. $\overline{AVEC}$ is ignored during all other bus cycles.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.2 DATA TRANSFER MECHANISM

The MC68020/EC020 architecture supports byte, word, and long-word operands allowing access to 8-, 16-, and 32-bit data ports through the use of asynchronous cycles controlled by $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$. Byte, word, and long-word operands can be located on any byte boundary, but misaligned transfers may require additional bus cycles, regardless of port size.

5.2.1 Dynamic Bus Sizing

The MC68020/EC020 dynamically interprets the port size of the addressed device during each bus cycle, allowing operand transfers to or from 8-, 16-, and 32-bit ports. During an operand transfer cycle, the slave device signals its port size (byte, word, or long word) and indicates completion of the bus cycle to the processor with the $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ signals. Refer to Table 5-1 for $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ encodings and assertion results.

Table 5-1. $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ Encodings and Results

$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	Result
Negated	Negated	Insert Wait States in Current Bus Cycle
Negated	Asserted	Complete Cycle—Data Bus Port Size is 8 Bits
Asserted	Negated	Complete Cycle—Data Bus Port Size is 16 Bits
Asserted	Asserted	Complete Cycle—Data Bus Port Size is 32 Bits

For example, if the processor is executing an instruction that reads a long-word operand from a long-word-aligned address, it attempts to read 32 bits during the first bus cycle. (Refer to **5.2.2 Misaligned Operands** for the case of a word or byte address.) If the port responds that it is 32 bits wide, the MC68020/EC020 latches all 32 bits of data and continues with the next operation. If the port responds that it is 16 bits wide, the MC68020/EC020 latches the 16 bits of valid data and runs another bus cycle to obtain the other 16 bits. The operation for an 8-bit port is similar, but requires four read cycles. The addressed device uses the $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ signals to indicate the port width. For instance, a 32-bit device always returns $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ for a 32-bit port, regardless of whether the bus cycle is a byte, word, or long-word operation.

Dynamic bus sizing requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 32-bit port must reside on D31–D0, a 16-bit port must reside on D32–D16, and an 8-bit port must reside on D31–D24. This requirement minimizes the number of bus cycles needed to transfer data to 8- and 16-bit ports and ensures that the MC68020/EC020 correctly transfers valid data. The MC68020/EC020 always attempts to transfer the maximum amount of data on all bus cycles; for a long-word operation, it always assumes that the port is 32 bits wide when beginning the bus cycle.

The bytes of operands are designated as shown in Figure 5-3. The most significant byte of a long-word operand is OP0; the least significant byte is OP3. The two bytes of a word-length operand are OP2 (most significant) and OP3. The single byte of a byte-length operand is OP3. These designations are used in the figures and descriptions that follow.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

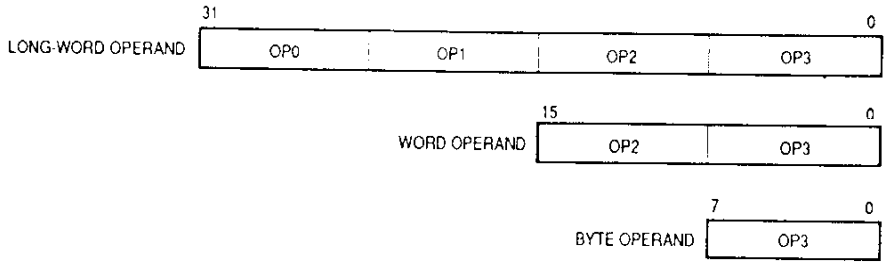


Figure 5-3. Internal Operand Representation

Figure 5-4 shows the required organization of data ports on the MC68020/EC020 bus for 8-, 16-, and 32-bit devices. The four bytes shown in Figure 5-4 are connected through the internal data bus and data multiplexer to the external data bus. This path is the means through which the MC68020/EC020 supports dynamic bus sizing and operand misalignment. Refer to **5.2.2 Misaligned Operands** for the definition of misaligned operand. The data multiplexer establishes the necessary connections for different combinations of address and data sizes.

Freescale Semiconductor, Inc.

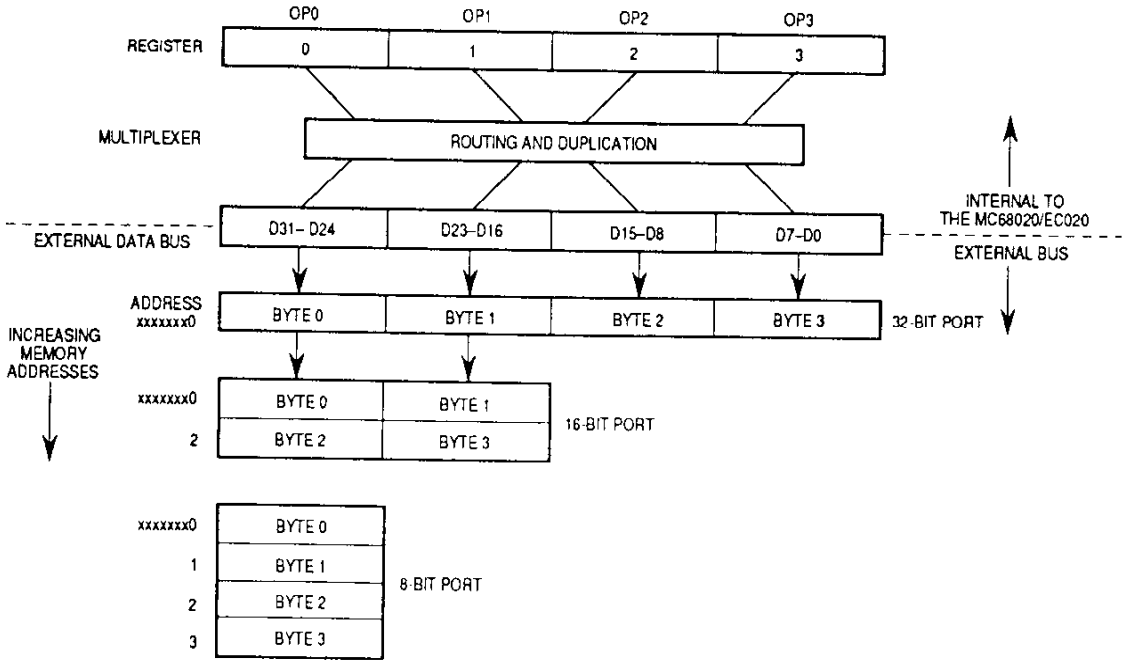


Figure 5-4. MC68020/EC020 Interface to Various Port Sizes

For More Information On This Product,
Go to: www.freescale.com

The multiplexer takes the four bytes of the 32-bit bus and routes them to their required positions. For example, OP0 can be routed to D31–D24, as would be the normal case, or it can be routed to any other byte position to support a misaligned transfer. The same is true for any of the operand bytes. The positioning of bytes is determined by the SIZ1, SIZ0, A1, and A0 outputs.

The SIZ1 and SIZ0 outputs indicate the remaining number of bytes to be transferred during the current bus cycle, as listed in Table 5-2.

Table 5-2. SIZ1, SIZ0 Signal Encoding

SIZ1	SIZ0	Size
Negated	Asserted	Byte
Asserted	Negated	Word
Asserted	Asserted	3 Bytes
Negated	Negated	Long Word

The number of bytes transferred during a write or read bus cycle is equal to or less than the size indicated by the SIZ1 and SIZ0 outputs, depending on port width and operand alignment. For example, during the first bus cycle of a long-word transfer to a word port, the SIZ1 and SIZ0 outputs indicate that four bytes are to be transferred, although only two bytes are moved on that bus cycle.

A1–A0 also affect operation of the data multiplexer. During an operand transfer, A31–A2 (for the MC68020) or A23–A2 (for the MC68EC020) indicate the long-word base address of that portion of the operand to be accessed; A1 and A0 indicate the byte offset from the base. Table 5-3 lists the encodings of A1 and A0 and the corresponding byte offsets from the long-word base.

Table 5-3. Address Offset Encodings

A1	A0	Offset
Negated	Negated	+0 Bytes
Negated	Asserted	+1 Byte
Asserted	Negated	+2 Bytes
Asserted	Asserted	+3 Bytes

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 5-4 lists the bytes required on the data bus for read cycles. The entries shown as OP3, OP2, OP1, and OP0 are portions of the requested operand that are read or written during that bus cycle and are defined by SIZ1, SIZ0, A1, and A0 for the bus cycle.

Table 5-4. Data Bus Requirements for Read Cycles

Transfer Size	Size		Address		Long-Word Port External Data Bytes Required				Word Port External Data Bytes Required		Byte Port External Data Bytes Required
	SIZ1	SIZ0	A1	A0	D31–D24	D23–D16	D15–D8	D7–D0	D31–D24	D23–D16	D31–D24
Byte	0	1	0	0	OP3				OP3		OP3
	0	1	0	1		OP3				OP3	OP3
	0	1	1	0			OP3		OP3		OP3
	0	1	1	1				OP3		OP3	OP3
Word	1	0	0	0	OP2	OP3			OP2	OP3	OP2
	1	0	0	1		OP2	OP3			OP2	OP2
	1	0	1	0			OP2	OP3	OP2	OP3	OP2
	1	0	1	1				OP2		OP2	OP2
3 Bytes	1	1	0	0	OP1	OP2	OP3		OP1	OP2	OP1
	1	1	0	1		OP1	OP2	OP3		OP1	OP1
	1	1	1	0			OP1	OP2	OP1	OP2	OP1
	1	1	1	1				OP1		OP1	OP1
Long Word	0	0	0	0	OP0	OP1	OP2	OP3	OP0	OP1	OP0
	0	0	0	1		OP0	OP1	OP2		OP0	OP0
	0	0	1	0			OP0	OP1	OP0	OP1	OP0
	0	0	1	1				OP0		OP0	OP0

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 5-5 lists the combinations of SIZ1, SIZ0, A1, and A0 and the corresponding pattern of the data transfer for write cycles from the internal multiplexer of the MC68020/EC020 to the external data bus.

Table 5-5. MC68020/EC020 Internal to External Data Bus Multiplexer—Write Cycles

Transfer Size	Size		Address		External Data Bus Connection			
	SIZ1	SIZ0	A1	A0	D31–D24	D23–D16	D15–D8	D7–D0
Byte	0	1	x	x	OP3	OP3	OP3	OP3
Word	1	0	x	0	OP2	OP3	OP2	OP3
	1	0	x	1	OP2	OP2	OP3	OP2
3 Bytes	1	1	0	0	OP1	OP2	OP3	OP0*
	1	1	0	1	OP1	OP1	OP2	OP3
	1	1	1	0	OP1	OP2	OP1	OP2
	1	1	1	1	OP1	OP1	OP2*	OP1
Long Word	0	0	0	0	OP0	OP1	OP2	OP3
	0	0	0	1	OP0	OP0	OP1	OP2
	0	0	1	0	OP0	OP1	OP0	OP1
	0	0	1	1	OP0	OP0	OP1*	OP0

*Due to the current implementation, this byte is output but never used.
x = Don't care
NOTE: The OP tables on the external data bus refer to a particular byte of the operand that is written on that section of the data bus.

5

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Figure 5-5 shows the transfer (write) of a long-word operand to a word port. In the first bus cycle, the MC68020/EC020 places the four operand bytes on the external bus. Since the address is long-word aligned in this example, the multiplexer follows the pattern in the entry of Table 5-5 corresponding to SIZ0, SIZ1, A0, A1 = 0000. The port latches the data on D31–D16, asserts $\overline{\text{DSACK1}}$ ($\overline{\text{DSACK0}}$ remains negated), and the processor terminates the bus cycle. It then starts a new bus cycle with SIZ1, SIZ0, A1, A0 = 1010 to transfer the remaining 16 bits. SIZ1 and SIZ0 indicate that a word remains to be transferred; A1 and A0 indicate that the word corresponds to an offset of two from the base address. The multiplexer follows the pattern corresponding to this configuration of SIZ1, SIZ0, A1, and A0 and places the two least significant bytes of the long word on the word portion of the bus (D31–D16). The bus cycle transfers the remaining bytes to the word-sized port. Figure 5-6 shows the timing of the bus transfer signals for this operation.

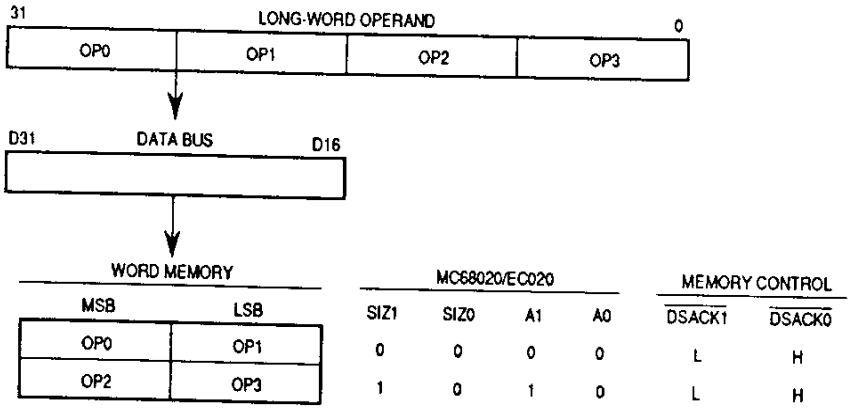
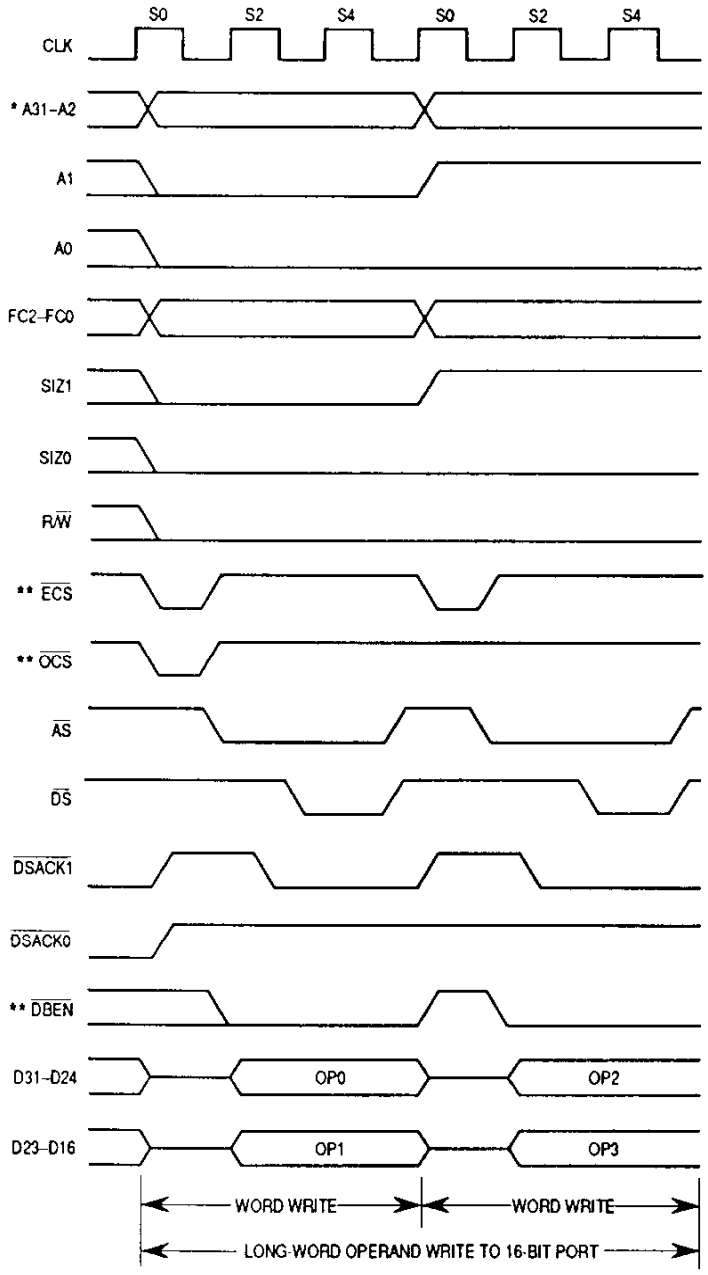


Figure 5-5. Long-Word Operand Write to Word Port Example

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-6. Long-Word Operand Write to Word Port Timing

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Figure 5-7 shows a word write to an 8-bit bus port. Like the preceding example, this example requires two bus cycles. Each bus cycle transfers a single byte. SIZ1 and SIZ0 for the first cycle specify two bytes; for the second cycle, one byte. Figure 5-8 shows the associated bus transfer signal timing.

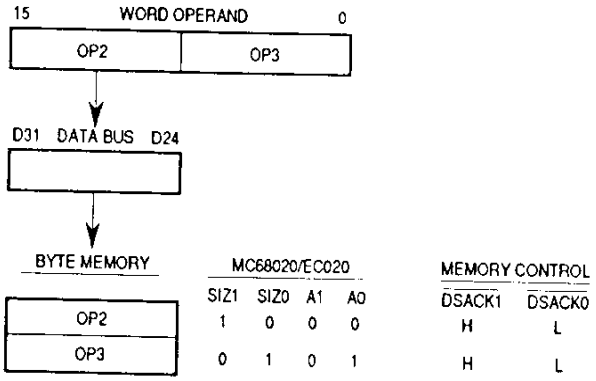
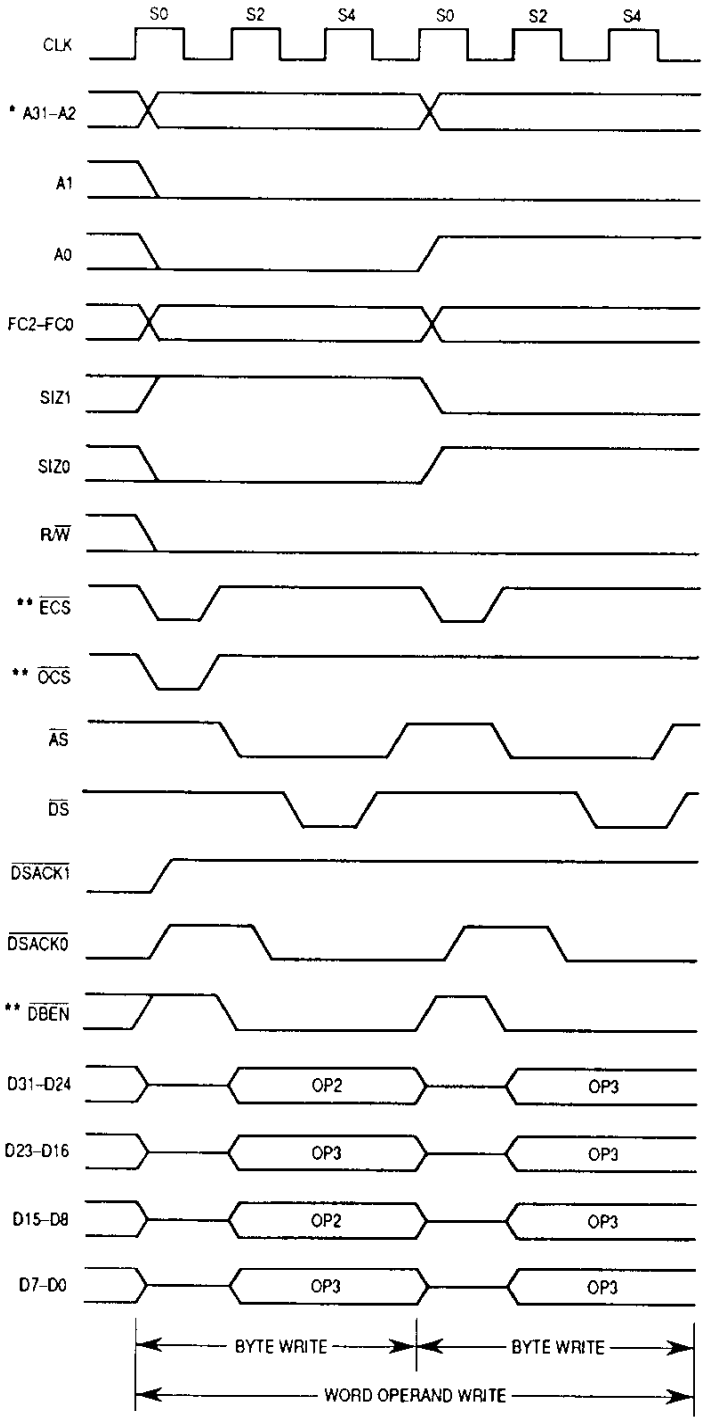


Figure 5-7. Word Operand Write to Byte Port Example

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-8. Word Operand Write to Byte Port Timing

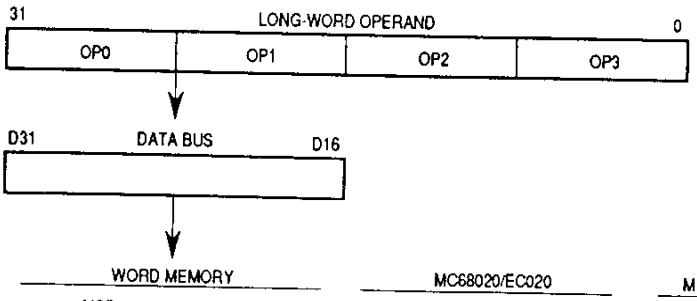
Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.2.2 Misaligned Operands

Since operands may reside at any byte boundary, they may be misaligned. A byte operand is properly aligned at any address; a word operand is misaligned at an odd address; a long word is misaligned at an address that is not evenly divisible by four. The MC68000, MC68008, and MC68010 implementations allow long-word transfers on odd-word boundaries but force exceptions if word or long-word operand transfers are attempted at odd-byte addresses. Although the MC68020/EC020 does not enforce any alignment restrictions for data operands (including PC relative data addresses), 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.

Figure 5-9 shows the transfer (write) of a long-word operand to an odd address in word-organized memory, which requires three bus cycles. For the first cycle, SIZ1 and SIZ0 specify a long-word transfer, and A2–A0 = 001. Since the port width is 16 bits, only the first byte of the long word is transferred. The slave device latches the byte and acknowledges the data transfer, indicating that the port is 16 bits wide. When the processor starts the second cycle, SIZ1 and SIZ0 specify that three bytes remain to be transferred with A2–A0 = 010. The next two bytes are transferred during this cycle. The processor then initiates the third cycle, with SIZ1 and SIZ0 indicating one byte remaining to be transferred with A2–A0 = 100. The port latches the final byte, and the operation is complete. Figure 5-10 shows the associated bus transfer signal timing. Figure 5-11 shows the equivalent operation for a data read cycle.



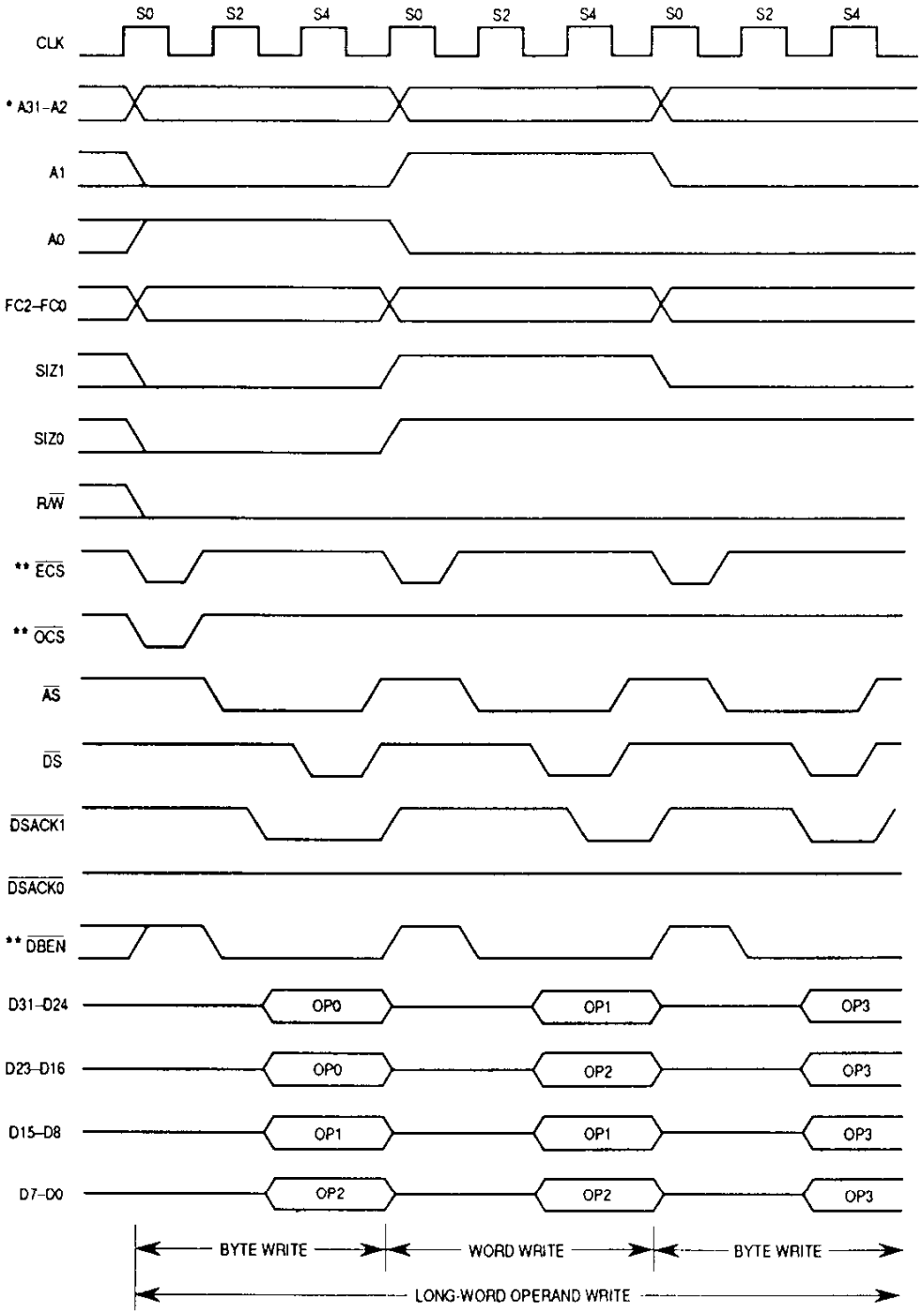
WORD MEMORY	
MSB	LSB
XXX	OP0
OP1	OP2
OP3	XXX

MC68020/EC020					MEMORY CONTROL	
SIZ1	SIZ0	A2	A1	A0	$\overline{DSACK1}$	$\overline{DSACK0}$
0	0	0	0	1	L	H
1	1	0	1	0	L	H
0	1	1	0	0	L	H

Figure 5-9. Misaligned Long-Word Operand Write to Word Port Example

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-10. Misaligned Long-Word Operand Write to Word Port Timing

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

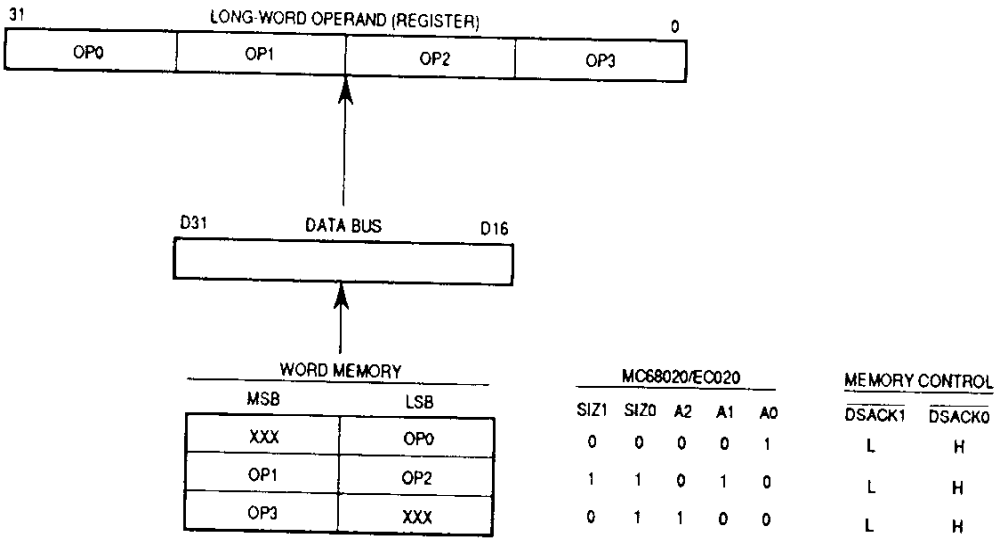


Figure 5-11. Misaligned Long-Word Operand Read from Word Port Example

Figures 5-12 and 5-13 show a word transfer (write) to an odd address in word-organized memory. This example is similar to the one shown in Figures 5-9 and 5-10 except that the operand is word sized and the transfer requires only two bus cycles. Figure 5-14 shows the equivalent operation for a data read cycle.

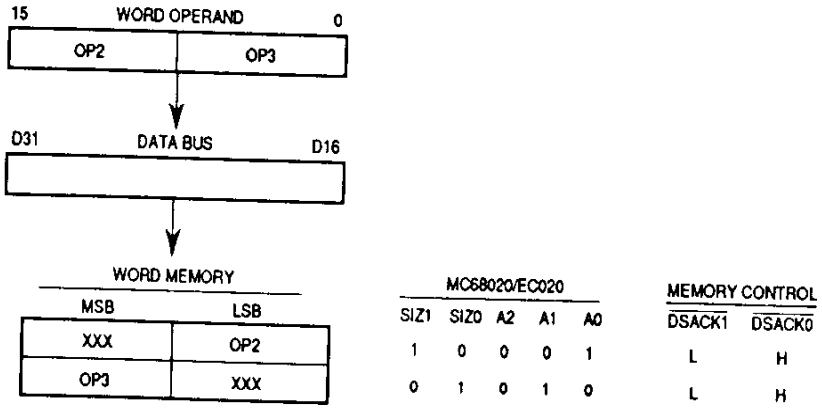
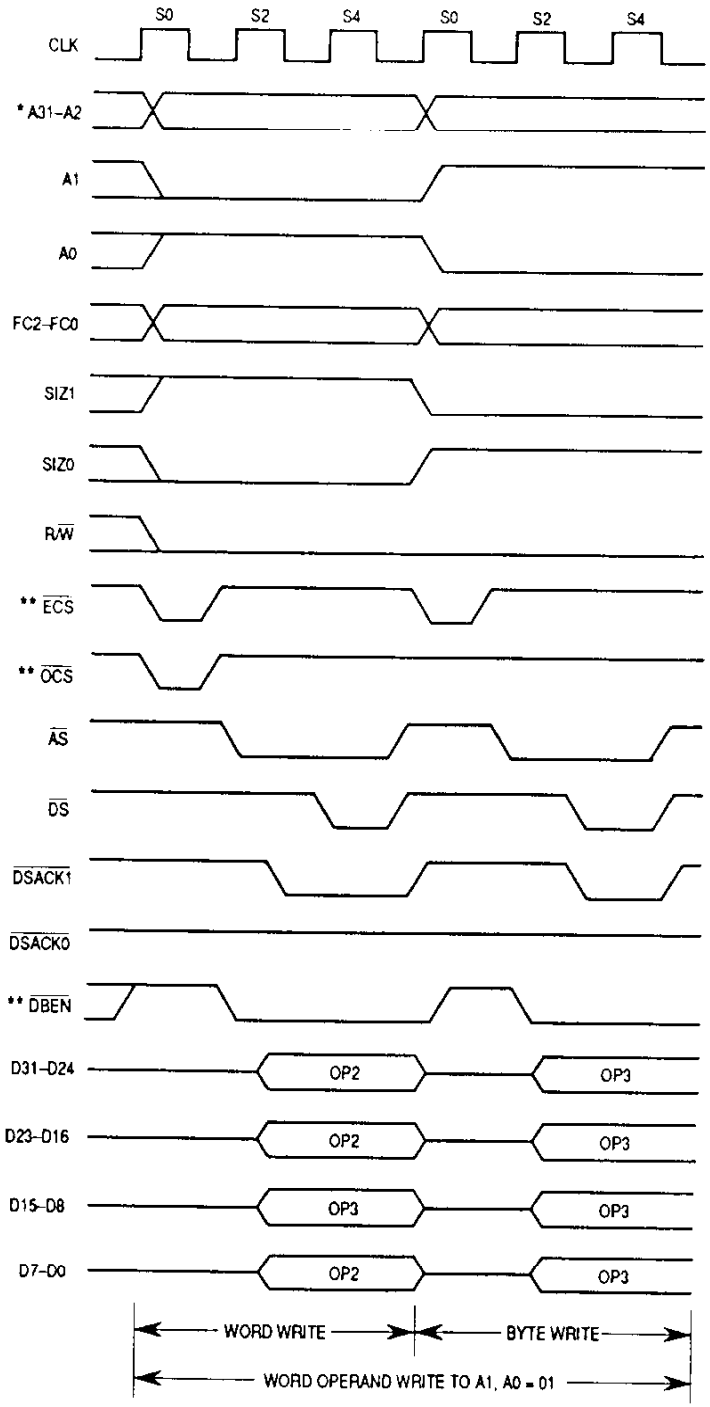


Figure 5-12. Misaligned Word Operand Write to Word Port Example

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020

Figure 5-13. Misaligned Word Operand Write to Word Port Timing

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

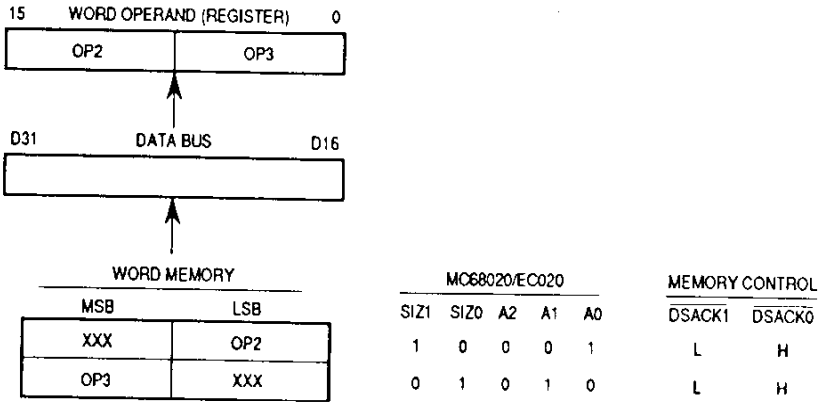


Figure 5-14. Misaligned Word Operand Read from Word Bus Example

Figures 5-15 and 5-16 show an example of a long-word transfer (write) to an odd address in long-word-organized memory. In this example, a long-word access is attempted beginning at the least significant byte of a long-word-organized memory. Only one byte can be transferred in the first bus cycle. The second bus cycle then consists of a three-byte access to a long-word boundary. Since the memory is long word organized, no further bus cycles are necessary. Figure 5-17 shows the equivalent operation for a data read cycle.

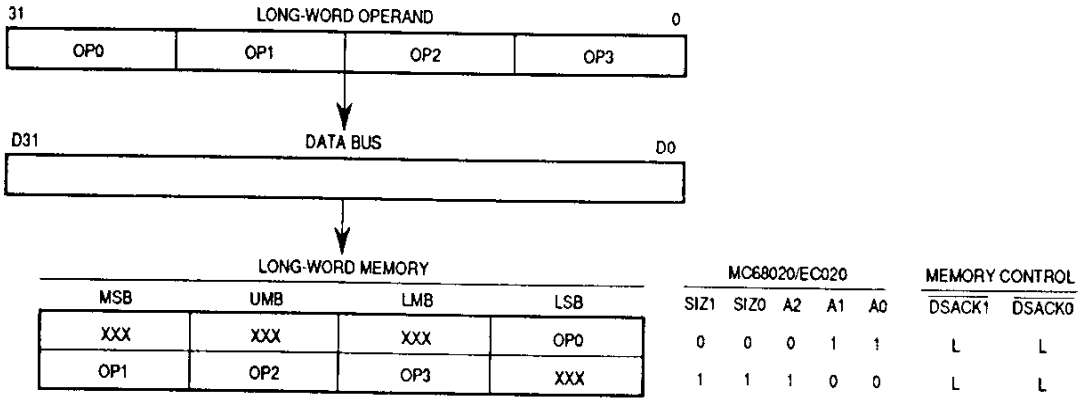
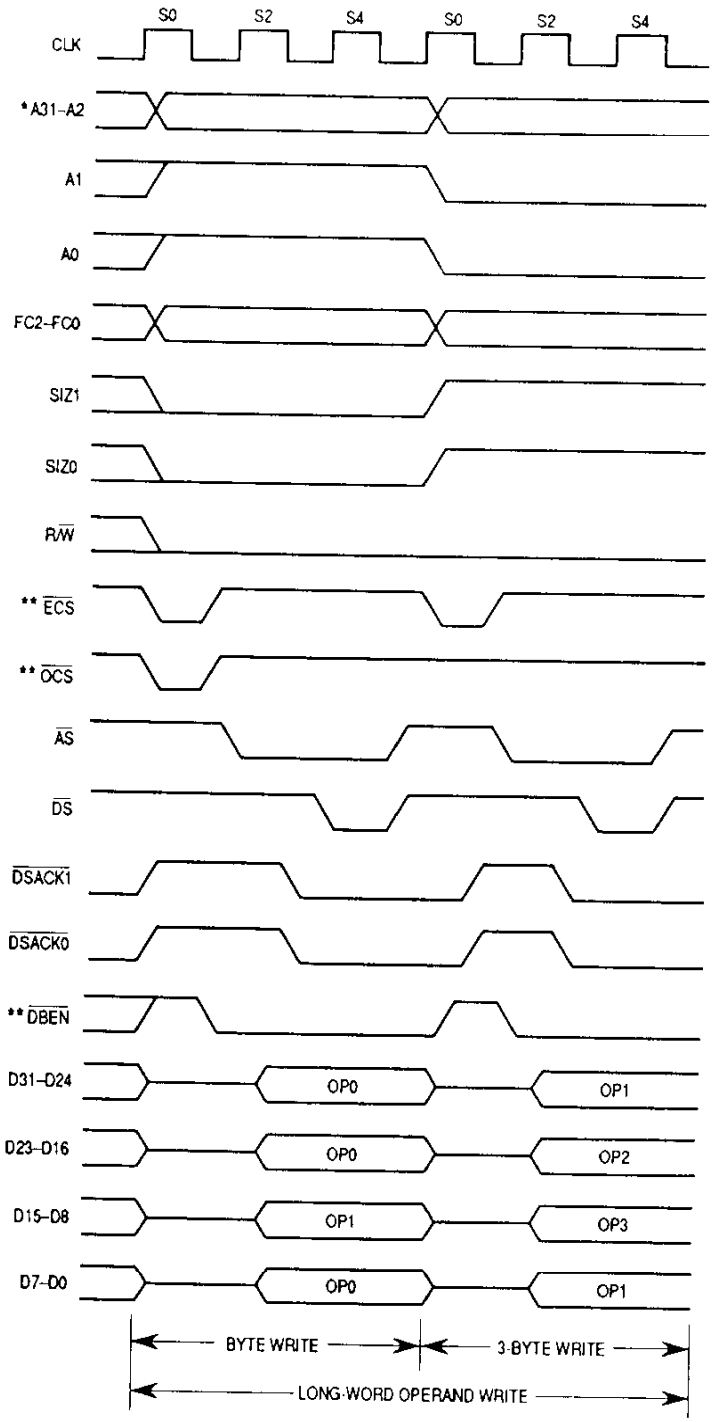


Figure 5-15. Misaligned Long-Word Operand Write to Long-Word Port Example

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020

Figure 5-16. Misaligned Long-Word Operand Write to Long-Word Port Timing

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

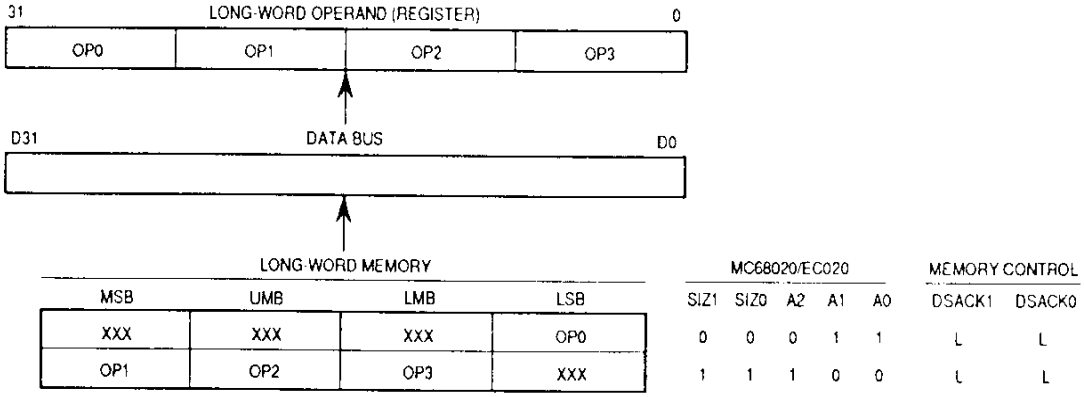


Figure 5-17. Misaligned Long-Word Operand Read from Long-Word Port Example

5.2.3 Effects of Dynamic Bus Sizing and Operand Misalignment

The combination of operand size, operand alignment, and port size determine the number of bus cycles required to perform a particular memory access. Table 5-6 lists the number of bus cycles required for different operand sizes to different port sizes with all possible alignment conditions for read/write cycles.

Table 5-6. Memory Alignment and Port Size Influence on Read/Write Bus Cycles

	Number of Bus Cycles (Data Port Size = 32 Bits:16 Bits:8 Bits)			
	A1, A0			
Operand Size	00	01	10	11
Instruction*	1:2:4	N/A	N/A	N/A
Byte Operand	1:1:1	1:1:1	1:1:1	1:1:1
Word Operand	1:1:2	1:2:2	1:1:2	2:2:2
Long-Word Operand	1:2:4	2:3:4	2:2:4	2:3:4

*Instruction prefetches are always two words from a long-word boundary

Table 5-6 reveals that bus cycle throughput is significantly affected by port size and alignment. The MC68020/EC020 system designer and programmer should be aware of and account for these effects, particularly in time-critical applications.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 5-6 demonstrates that the processor always prefetches instructions by reading a long word from a long-word address (A1, A0 = 00), regardless of port size or alignment. When the required instruction begins at an odd-word boundary, the processor attempts to fetch the entire 32 bits and loads both words into the instruction cache, if possible, although the second one is the required word. Even if the instruction access is not cached, the entire 32 bits are latched into an internal cache holding register from which the two instructions words can subsequently be referenced. Refer to **Section 8 Instruction Execution Timing** for a complete description of the cache holding register and pipeline operation.

5.2.4 Address, Size, and Data Bus Relationships

The data transfer examples show how the MC68020/EC020 drives data onto or receives data from the correct byte sections of the data bus. Table 5-7 shows the combinations of the SIZ1, SIZ0, A1, and A0 signals that can be used to generate byte enable signals for each of the four sections of the data bus for read and write cycles if the addressed device requires them. The port size also affects the generation of these enable signals as shown in the table. The four columns on the right correspond to the four byte enable signals. Letters B, W, and L refer to port sizes: B for 8-bit ports, W for 16-bit ports, and L for 32-bit ports. The letters B, W, and L imply that the byte enable signal should be true for that port size. A dash (—) implies that the byte enable signal does not apply.

The MC68020/EC020 always drives all sections of the data bus because, at the beginning of a write cycle, the bus controller does not know the port size.

Table 5-7 reveals that the MC68020/EC020 transfers the number of bytes specified by SIZ1, SIZ0 to or from the specified address unless the operand is misaligned or unless the number of bytes is greater than the port width. In these cases, the device transfers the greatest number of bytes possible for the port. For example, if the size is four and A1, A0 = 01, a 32-bit slave can only receive three bytes in the current bus cycle. A 16- or 8-bit slave can only receive one byte. The table defines the byte enables for all port sizes. Byte data strobes can be obtained by combining the enable signals with the $\overline{DS}$ signal. Devices residing on 8-bit ports can use the data strobe by itself since there is only one valid byte for every transfer. These enable or strobe signals select only the bytes required for write or read cycles. The other bytes are not selected, which prevents incorrect accesses in sensitive areas such as I/O.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 5-7. Data Bus Byte Enable Signals for Byte, Word, and Long-Word Ports

Transfer Size	SIZ1	SIZ0	A1	A0	Data Bus Active Sections Byte (B), Word (W), Long-Word (L) Ports			
					D31–D24	D23–D16	D15–D8	D7–D0
Byte	0	1	0	0	B W L		—	
	0	1	0	1	B	W L		
	0	1	1	0	B W	—	L	—
	0	1	1	1	B	W	—	L
Word	1	0	0	0	B W L	W L		—
	1	0	0	1	B	W L	L	
	1	0	1	0	B W	W	L	L
	1	0	1	1	B	W	—	L
3 Bytes	1	1	0	0	B W L	W L	L	—
	1	1	0	1	B	W L	L	L
	1	1	1	0	B W	W	L	L
	1	1	1	1	B	W	—	L
Long Word	0	0	0	0	B W L	W L	L	L
	0	0	0	1	B	W L	L	L
	0	0	1	0	B W	W	L	L
	0	0	1	1	B	W	—	L

Figure 5-18 shows a logic diagram of one method for generating byte enable signals for 16- and 32-bit ports from the SIZ1, SIZ0, A1, and A0 encodings and the R/$\overline{W}$ signal.

5.2.5 Cache Interactions

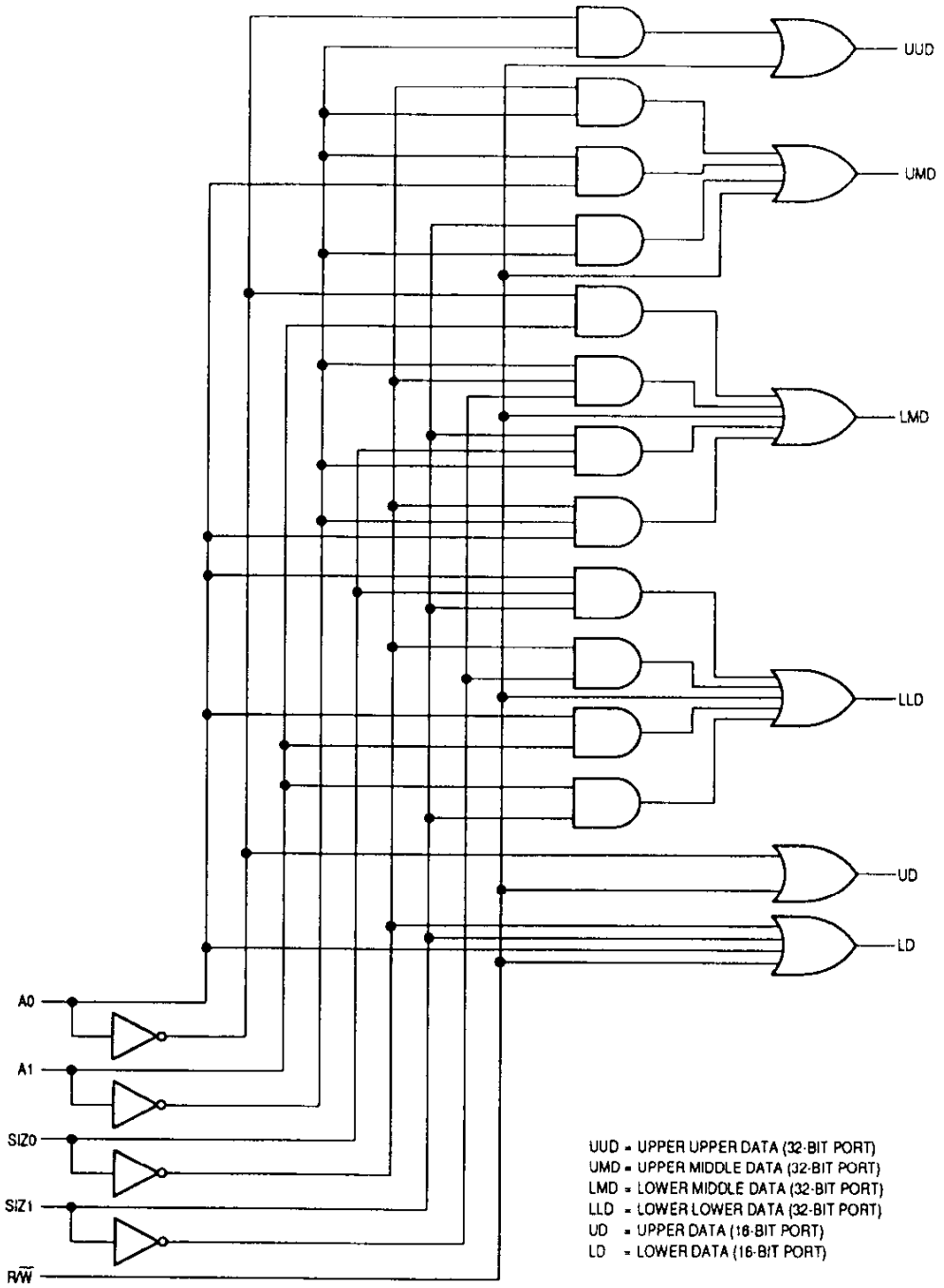
The organization and requirements of the on-chip instruction cache affect the interpretation of $\overline{DSACK1}$ and $\overline{DSACK0}$. Since the MC68020/EC020 attempts to load all instructions into the on-chip cache, the bus may operate differently when caching is enabled. Specifically, on read cycles that terminate normally, the A1, A0, SIZ1, and SIZ0 signals do not apply.

The cache can also affect the assertion of $\overline{AS}$ and the operation of a read cycle. The search of the cache by the processor begins when the sequencer requires an instruction. At this time, the bus controller may also initiate an external bus cycle in case the requested item is not resident in the instruction cache. If an internal cache hit occurs, the external cycle aborts, and $\overline{AS}$ is not asserted.

For the MC68020, if the bus is not occupied with another read or write cycle, the bus controller asserts the $\overline{ECS}$ signal (and the $\overline{OCS}$ signal, if appropriate). It is possible to have $\overline{ECS}$ asserted on multiple consecutive clock cycles. Note that there is a minimum time specified from the negation of $\overline{ECS}$ to the next assertion of $\overline{ECS}$ (refer to **Section 10 Electrical Characteristics**). Instruction prefetches can occur every other clock so that if, after an aborted cycle due to an instruction cache hit, the bus controller asserts $\overline{ECS}$ on the next clock, this second cycle is for a data fetch. Note that, if the bus controller is executing other cycles, these aborted cycles due to cache hits may not be seen externally.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



NOTE: These select lines can be combined with the address decode circuitry or all can be generated within the same programmed array logic unit.

Figure 5-18. Byte Enable Signal Generation for 16- and 32-Bit Ports

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.2.6 Bus Operation

The MC68020/EC020 bus is used in an asynchronous manner allowing external devices to operate at clock frequencies different from the MC68020/EC020 clock. Bus operation uses the handshake lines ($\overline{AS}$, $\overline{DS}$, $\overline{DSACK0}$, $\overline{DSACK1}$, $\overline{BERR}$, and $\overline{HALT}$) to control data transfers. $\overline{AS}$ signals the start of a bus cycle, and $\overline{DS}$ is used as a condition for valid data on a write cycle. Decoding SIZ1, SIZ0, A1, and A0 provides byte enable signals that select the active portion of the data bus. The slave device (memory or peripheral) then responds by placing the requested data on the correct portion of the data bus for a read cycle or latching the data on a write cycle and by asserting the $\overline{DSACK0}/\overline{DSACK1}$ combination that corresponds to the port size to terminate the cycle. If no slave responds or the access is invalid, external control logic asserts $\overline{BERR}$ to abort or $\overline{BERR}$ and $\overline{HALT}$ to retry the bus cycle.

$\overline{DSACK1}/\overline{DSACK0}$ can be asserted before the data from a slave device is valid on a read cycle. The length of time that $\overline{DSACK1}/\overline{DSACK0}$ may precede data is given by parameter #31, and it must be met in any asynchronous system to ensure that valid data is latched into the processor. (Refer to **Section 10 Electrical Characteristics** for timing parameters.) Note that no maximum time is specified from the assertion of $\overline{AS}$ to the assertion of $\overline{DSACK1}/\overline{DSACK0}$. Although the processor can transfer data in a minimum of three clock cycles when the cycle is terminated with $\overline{DSACK1}/\overline{DSACK0}$, the processor inserts wait cycles in clock period increments until $\overline{DSACK1}/\overline{DSACK0}$ is recognized.

The $\overline{BERR}$ and/or $\overline{HALT}$ signals can be asserted after $\overline{DSACK1}/\overline{DSACK0}$ is asserted. $\overline{BERR}$ and/or $\overline{HALT}$ must be asserted within the time given (parameter #48), after $\overline{DSACK1}/\overline{DSACK0}$ is asserted in any asynchronous system. If this maximum delay time is violated, the processor may exhibit erratic behavior.

5.2.7 Synchronous Operation with $\overline{DSACK1}/\overline{DSACK0}$

Although cycles terminated with $\overline{DSACK1}/\overline{DSACK0}$ are classified as asynchronous, cycles terminated with $\overline{DSACK1}/\overline{DSACK0}$ can also operate synchronously in that signals are interpreted relative to clock edges. The devices that use these synchronous cycles must synchronize the responses to the MC68020/EC020 clock. Since these devices terminate bus cycles with $\overline{DSACK1}/\overline{DSACK0}$, the dynamic bus sizing capabilities of the MC68020/EC020 are available. In addition, the minimum cycle time for these synchronous cycles is three clocks.

To support systems that use the system clock to generate $\overline{DSACK1}/\overline{DSACK0}$ and other asynchronous inputs, the asynchronous input setup time (parameter #47A) and the asynchronous input hold time (parameter #47B) are provided in **Section 10 Electrical Characteristics**. (Note: although a misnomer, these "asynchronous" parameters are the setup and hold times for synchronous operation.) If the setup and hold times are met for the assertion or negation of a signal, such as $\overline{DSACK1}/\overline{DSACK0}$, the processor can be guaranteed to recognize that signal level on that specific falling edge of the system clock. If the assertion of $\overline{DSACK1}/\overline{DSACK0}$ is recognized on a particular falling edge of the clock, valid data is latched into the processor (for a read cycle) on the next falling clock edge provided the data meets the data setup time (parameter #27). In this case, parameter #31

Freescale Semiconductor, Inc.
5
For More Information On This Product,
Go to: www.freescale.com

for asynchronous operation can be ignored. All timing parameters referred to are described in **Section 10 Electrical Characteristics**. If a system asserts DSACK1/DSACK0 for the required window around the falling edge of state 2 and obeys the proper bus protocol by maintaining DSACK1/DSACK0 (and/or BERR/HALT) until and throughout the clock edge that negates AS (with the appropriate asynchronous input hold time specified by parameter #47B), no wait states are inserted. The bus cycle runs at its maximum speed of three clocks per cycle for bus cycles terminated with DSACK1/DSACK0.

To ensure proper operation in a synchronous system when BERR or BERR/HALT is asserted after DSACK1/DSACK0, BERR (and HALT) must meet the appropriate setup time (parameter #27A) prior to the falling clock edge one clock cycle after DSACK1/DSACK0 is recognized. This setup time is critical, and the MC68020/EC020 may exhibit erratic behavior if it is violated.

When operating synchronously, the data-in setup (parameter #27) and hold (parameter #30) times for synchronous cycles may be used instead of the timing requirements for data relative to the DS signal.

5.3 DATA TRANSFER CYCLES

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

- Address Bus (A31–A0 for the MC68020) (A23–A0 for the MC68EC020)
- Data Bus (D31–D0)
- Control Signals

The address and data buses are both parallel, nonmultiplexed buses. The bus master moves data on the bus by issuing control signals, and the bus uses a handshake protocol to ensure correct movement of the data. In all bus cycles, the bus master is responsible for de-skewing all signals it issues at both the start and end of the cycle. In addition, the bus master is responsible for de-skewing DSACK1/DSACK0, D31–D0, BERR, HALT, and, for the MC68020, DBEN from the slave devices. The following paragraphs define read, write, and read-modify-write cycle operations.

Each of the bus cycles is defined as a succession of states. These states apply to the bus operation and are different from the processor states described in **Section 2 Processing States**. The clock cycles used in the descriptions and timing diagrams of data transfer cycles are independent of the clock frequency. Bus operations are described in terms of external bus states.

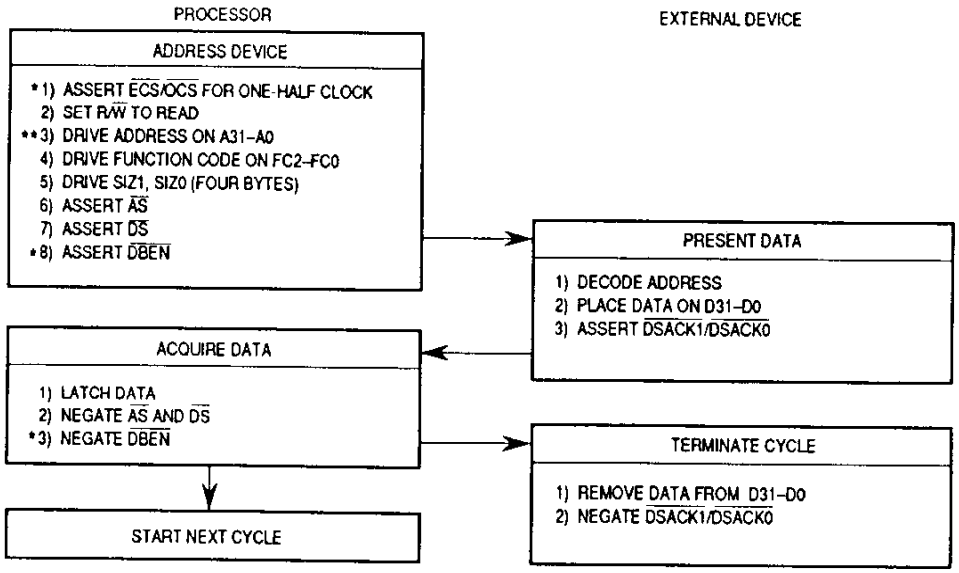
For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.3.1 Read Cycle

During a read cycle, the processor receives data from a memory, coprocessor, or peripheral device. If the instruction specifies a long word operation, the MC68020/EC020 attempts to read four bytes at once. For a word operation, it attempts to read two bytes at once and for a byte operation, one byte. For some operations, the processor requests a three-byte transfer. The processor properly positions each byte internally. The section of the data bus from which each byte is read depends on the operand size, A1–A0, and the port size. Refer to **5.2.1 Dynamic Bus Sizing** and **5.2.2 Misaligned Operands** for more information on dynamic bus sizing and misaligned operands.

Figure 5-19 is a flowchart of a long-word read cycle. Figure 5-20 is a flowchart of a byte read cycle. Figures 5-21–5-23 are read cycle timing diagrams in terms of clock periods. Figure 5-21 corresponds to byte and word read cycles from a 32-bit port. Figure 5-22 corresponds to a long-word read cycle from an 8-bit port. Figure 5-23 also applies to a long-word read cycle, but from 16- and 32-bit ports.

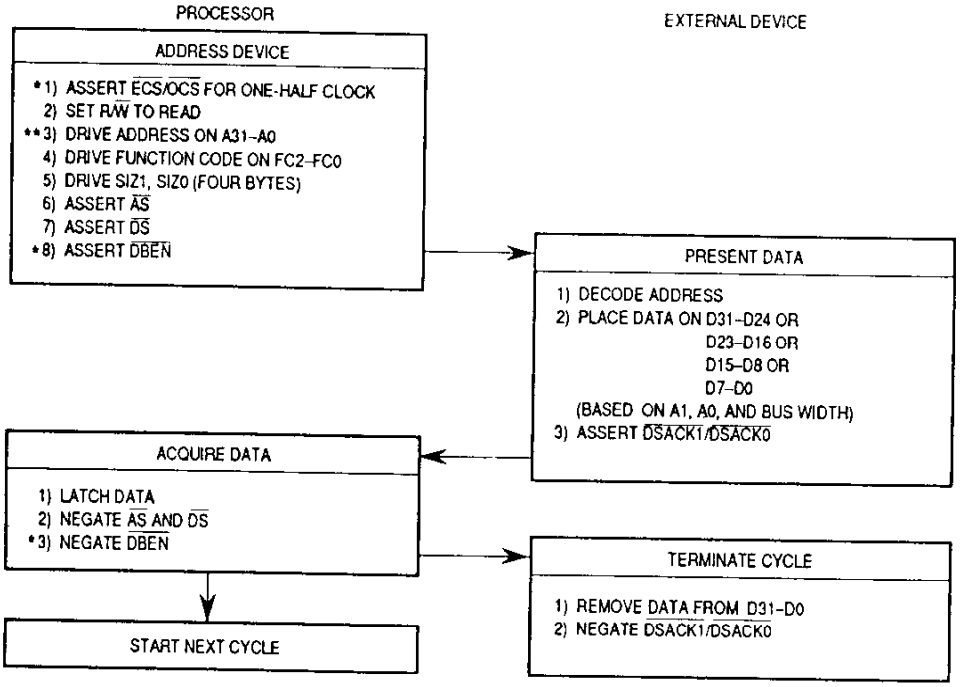


* This step does not apply to the MC68EC020.
** For the MC68EC020, A23–A0.

Figure 5-19. Long-Word Read Cycle Flowchart

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

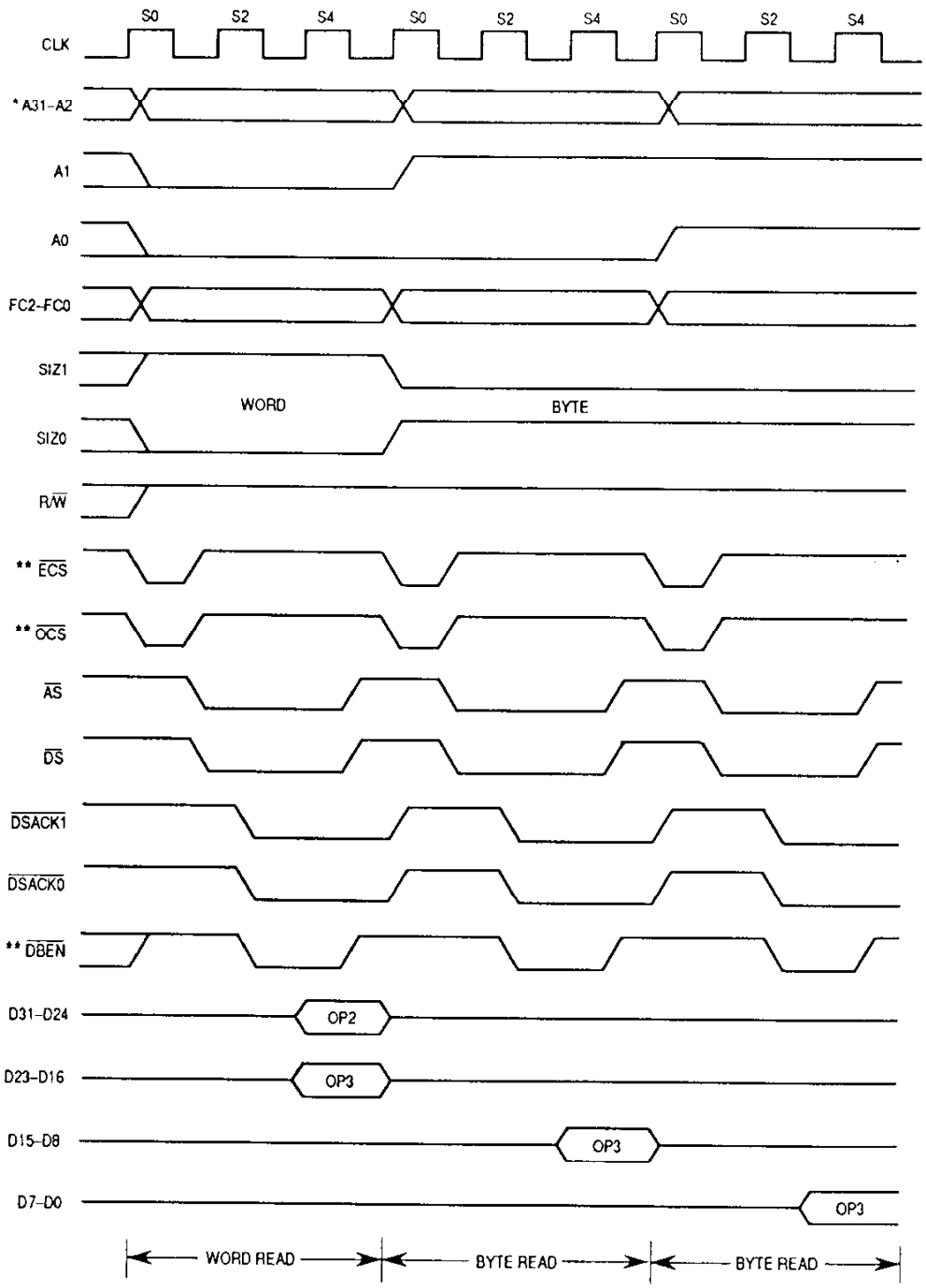


* This step does not apply to the MC68EC020.
** For the MC68EC020, A23–A0.

Figure 5-20. Byte Read Cycle Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

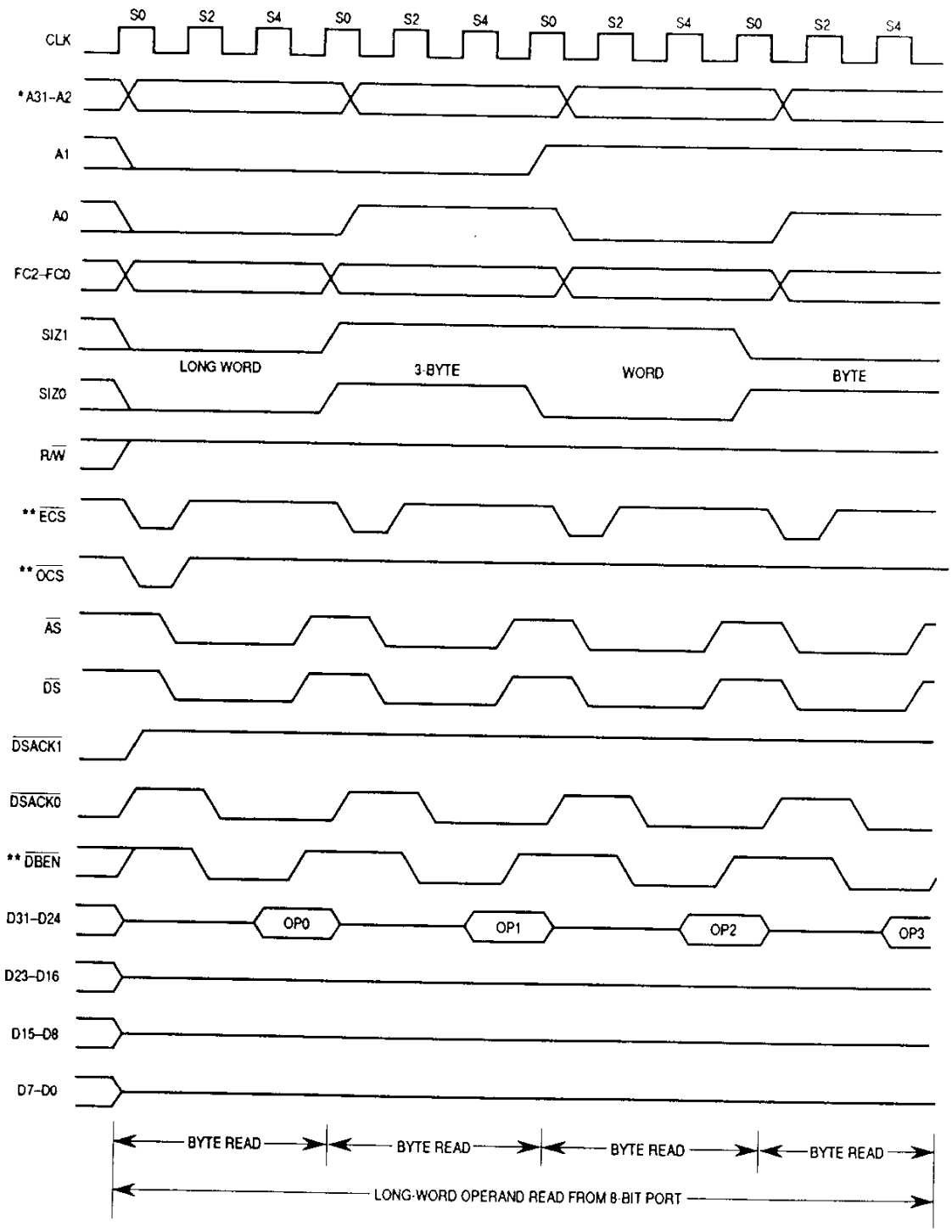


* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020

Figure 5-21. Byte and Word Read Cycles—32-Bit Port

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

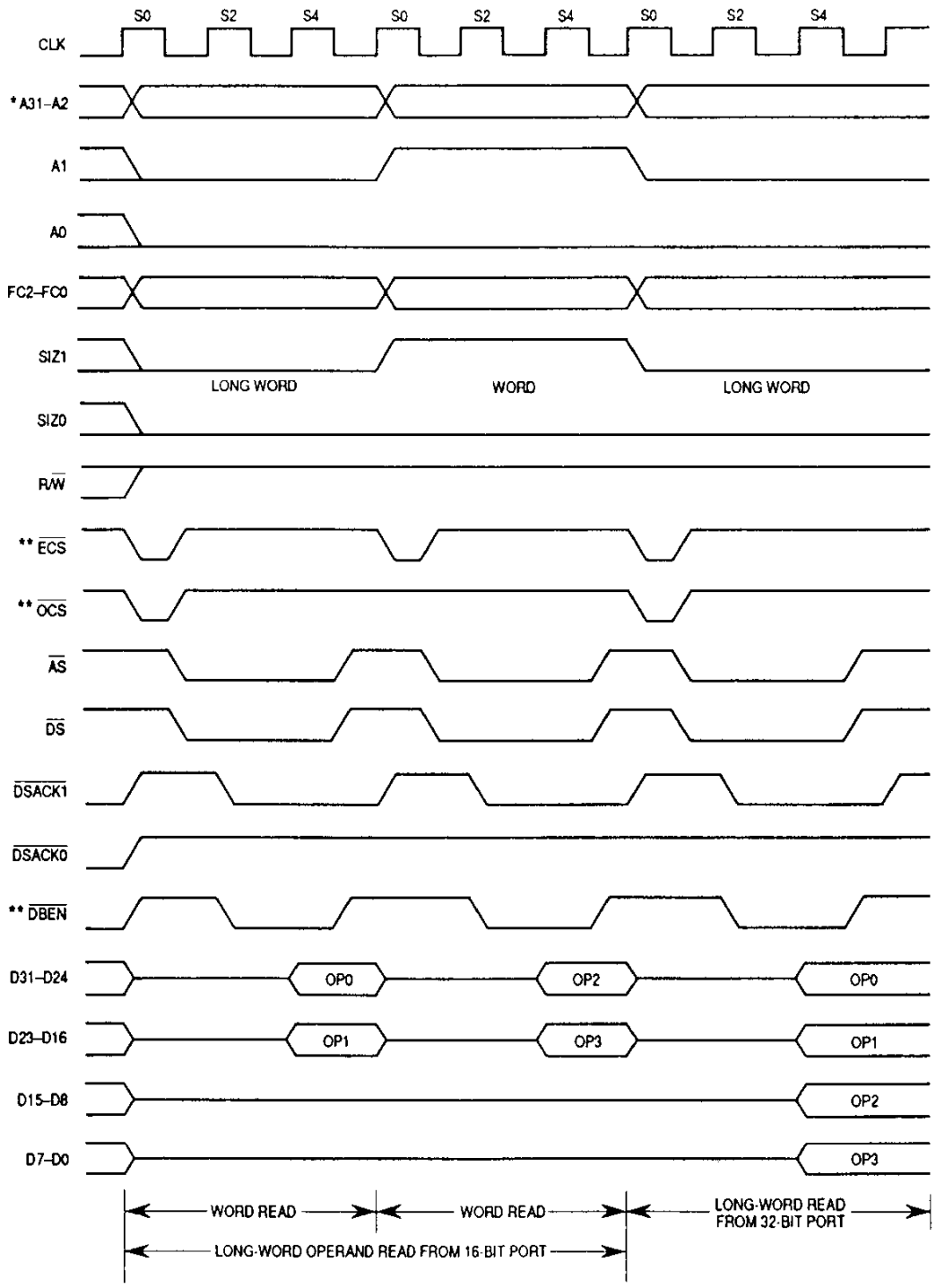


* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-22. Long-Word Read—8-Bit Port

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-23. Long-Word Read—16- and 32-Bit Ports

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

State 0

MC68020—The read cycle starts in state 0 (S0). The processor asserts $\overline{ECS}$, indicating the beginning of an external cycle. If the cycle is the first external cycle of a read operation, $\overline{OCS}$ is asserted simultaneously. During S0, the processor places a valid address on A31–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The processor drives R/$\overline{W}$ high for a read cycle and negates $\overline{DBEN}$ to disable the data buffers. SIZ0 and SIZ1 become valid, indicating the number of bytes requested to be transferred.

MC68EC020—The read cycle starts in S0. During S0, the processor places a valid address on A23–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The processor drives R/$\overline{W}$ high for a read cycle. SIZ0 and SIZ1 become valid, indicating the number of bytes requested to be transferred.

State 1

MC68020—One-half clock later in state 1 (S1), the processor asserts $\overline{AS}$, indicating that the address on the address bus is valid. The processor also asserts $\overline{DS}$ during S1. In addition, the $\overline{ECS}$ (and $\overline{OCS}$, if asserted) signal is negated during S1.

MC68EC020—One-half clock later in S1, the processor asserts $\overline{AS}$, indicating that the address on the address bus is valid. The processor also asserts $\overline{DS}$ during S1.

State 2

MC68020—During state 2 (S2), the processor asserts $\overline{DBEN}$ to enable external data buffers. The selected device uses R/$\overline{W}$, SIZ1–SIZ0, A1–A0, and $\overline{DS}$ to place its 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 and A1–A0. Concurrently, the selected device asserts $\overline{DSACK1}$/$\overline{DSACK0}$.

MC68EC020—During S2, the selected device uses R/$\overline{W}$, SIZ1–SIZ0, A1–A0, and $\overline{DS}$ to place its 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 and A1–A0. Concurrently, the selected device asserts $\overline{DSACK1}$/$\overline{DSACK0}$.

State 3

MC68020/EC020—As long as at least one of the $\overline{DSACK1}$/$\overline{DSACK0}$ signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), data is latched on the next falling edge of the clock, and the cycle terminates. If $\overline{DSACK1}$/$\overline{DSACK0}$ is not recognized by the start of state 3 (S3), the processor inserts wait states instead of proceeding to states 4 and 5. To ensure that wait states are inserted, both $\overline{DSACK1}$ and $\overline{DSACK0}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the processor continues to sample the $\overline{DSACK1}$/$\overline{DSACK0}$ signals on the falling edges of the clock until an assertion is recognized.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

State 4

MC68020/EC020—At the end of state 4 (S4), the processor latches the incoming data.

State 5

MC68020—The processor negates $\overline{AS}$, $\overline{DS}$, and $\overline{DBEN}$ during state 5 (S5). It holds the address valid during S5 to provide address hold time for memory systems. R/$\overline{W}$, SIZ1–SIZ0, and FC2–FC0 also remain valid throughout S5.

The external device keeps its data and $\overline{DSACK1}$/$\overline{DSACK0}$ signals asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove its data and negate $\overline{DSACK1}$/$\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACK1}$/$\overline{DSACK0}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

MC68EC020—The processor negates $\overline{AS}$ and $\overline{DS}$ during state S5. It holds the address valid during S5 to provide address hold time for memory systems. R/$\overline{W}$, SIZ1, SIZ0, and FC2–FC0 also remain valid throughout S5.

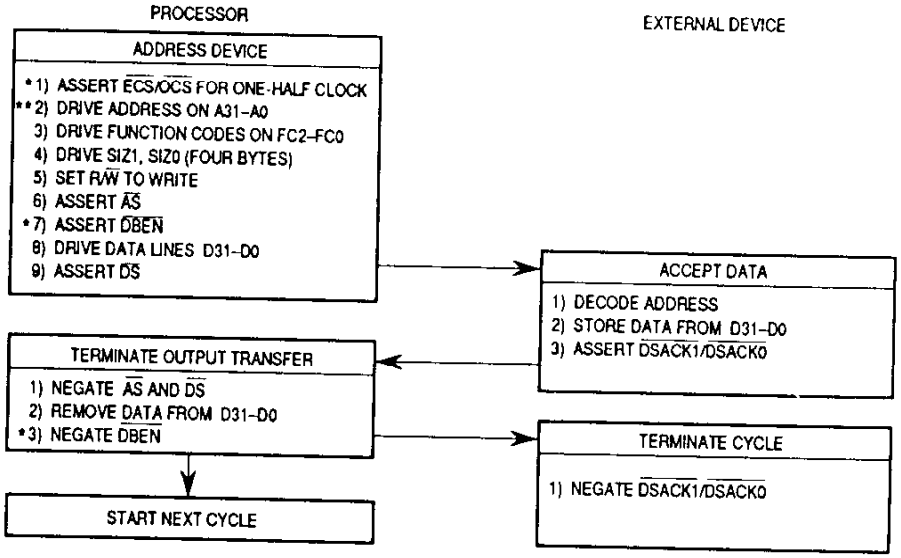
The external device keeps its data and $\overline{DSACK1}$/$\overline{DSACK0}$ signals asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove its data and negate $\overline{DSACK1}$/$\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACK1}$/$\overline{DSACK0}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.3.2 Write Cycle

During a write cycle, the processor transfers data to memory or a peripheral device. Figure 5-24 is a flowchart of a write cycle operation for a long-word transfer. Figures 5-25–5-28 are write cycle timing diagrams in terms of clock periods. Figure 5-25 shows two write cycles (between two read cycles with no idle time in between) for a 32-bit port. Figure 5-26 shows byte and word write cycles to a 32-bit port. Figure 5-27 shows a long-word write cycle to an 8-bit port. Figure 5-28 shows a long-word write cycle to a 16-bit port.



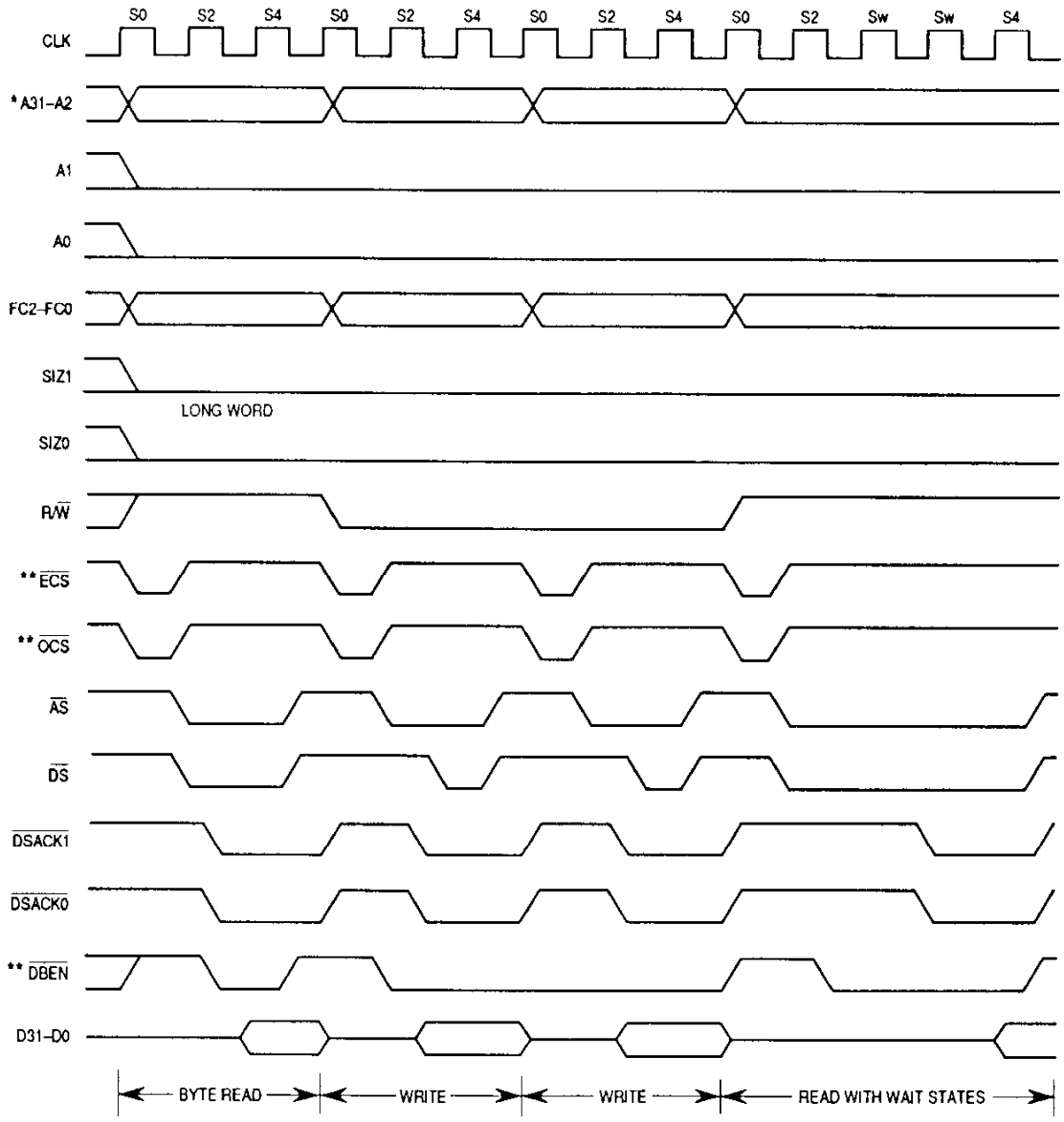
* This step does not apply to the MC68EC020.
** For the MC68EC020, A23–A0.

Figure 5-24. Write Cycle Flowchart

5

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

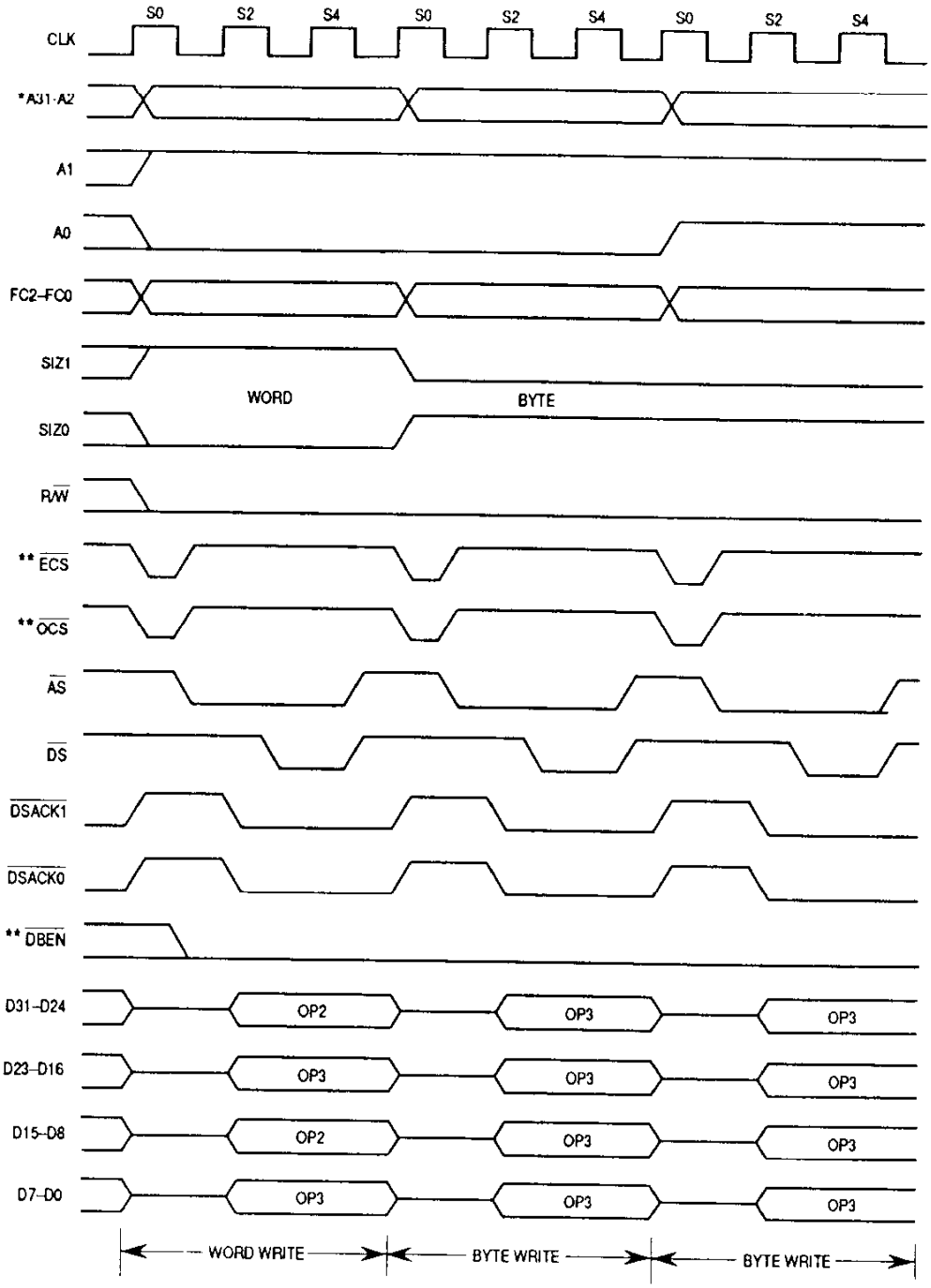


* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-25. Read-Write-Read Cycles—32-Bit Port

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

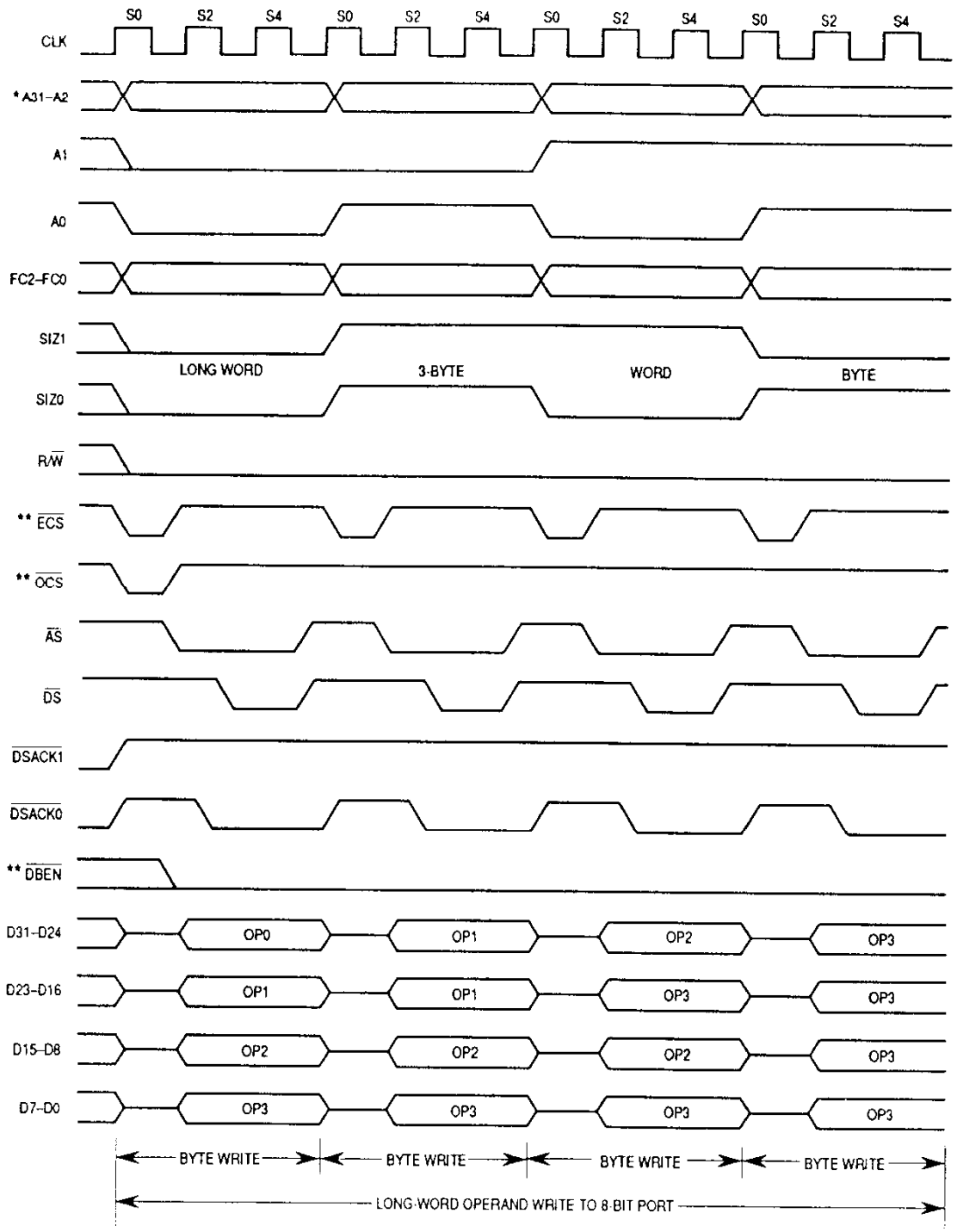


* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-26. Byte and Word Write Cycles—32-Bit Port

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

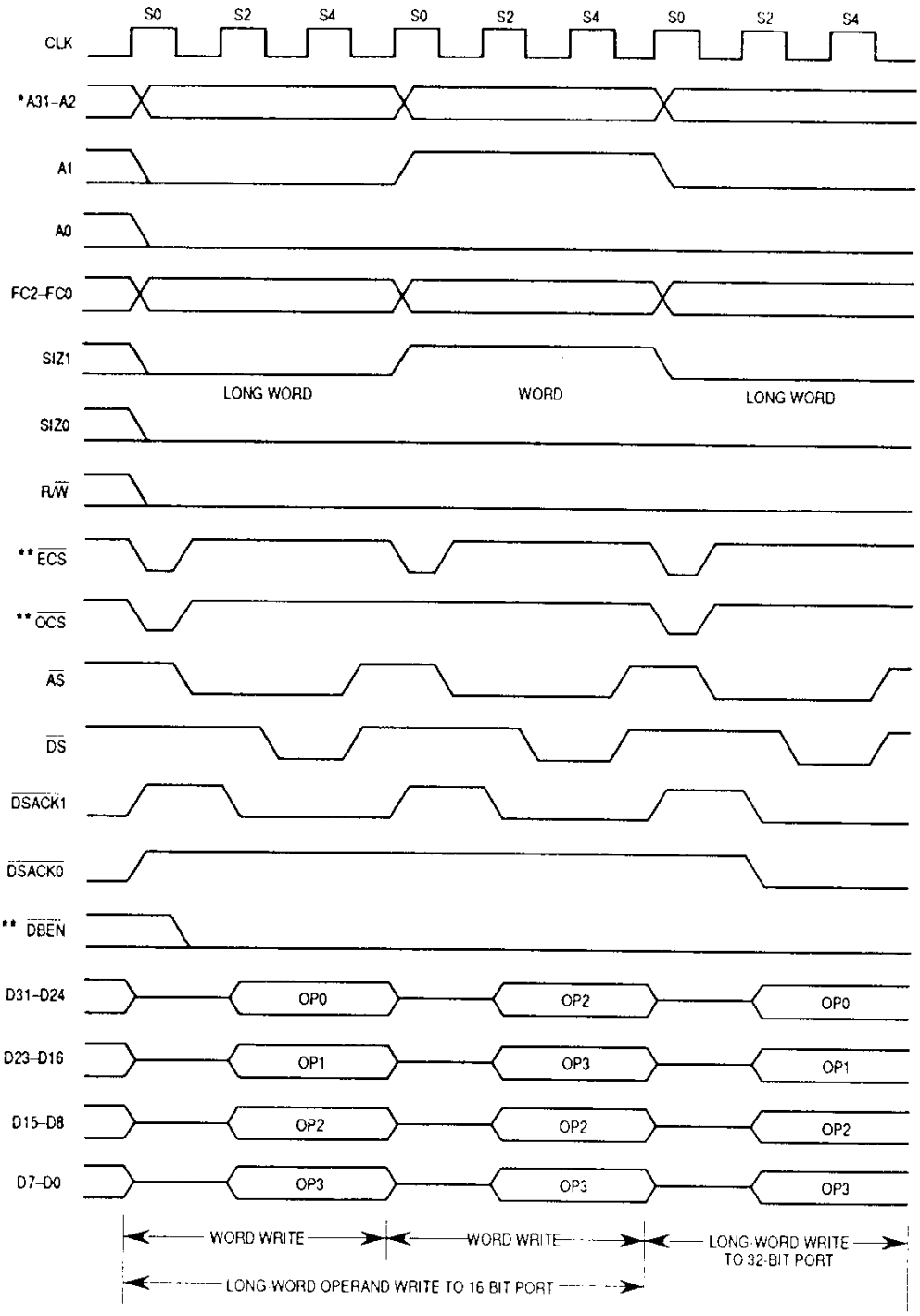


* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-27. Long-Word Operand Write—8-Bit Port

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-28. Long-Word Operand Write—16-Bit Port

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

State 0

MC68020—The write cycle starts in S0. The processor negates $\overline{ECS}$, indicating the beginning of an external cycle. If the cycle is the first external cycle of a write operation, $\overline{OCS}$ is asserted simultaneously. During S0, the processor places a valid address on A31–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The processor drives R/$\overline{W}$ low for a write cycle. SIZ1–SIZ0 become valid, indicating the number of bytes to be transferred.

MC68EC020—The write cycle starts in S0. During S0, the processor places a valid address on A23–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The processor drives R/$\overline{W}$ low for a write cycle. SIZ1, SIZ0 become valid, indicating the number of bytes to be transferred.

State 1

MC68020—One-half clock later in S1, the processor asserts $\overline{AS}$, indicating that the address on the address bus is valid. The processor also asserts $\overline{DBEN}$ during S1, which can enable external data buffers. In addition, the $\overline{ECS}$ (and $\overline{OCS}$, if asserted) signal is negated during S1.

MC68EC020—One-half clock later in S1, the processor asserts $\overline{AS}$, indicating that the address on the address bus is valid.

State 2

MC68020/EC020—During S2, the processor places the data to be written onto D31–D0. At the end of S2, the processor samples $\overline{DSACK1}$/$\overline{DSACK0}$.

State 3

MC68020/EC020—The processor asserts $\overline{DS}$ during S3, indicating that the data on the data bus is stable. As long as at least one of the $\overline{DSACK1}$/$\overline{DSACK0}$ signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), the cycle terminates one clock later. If $\overline{DSACK1}$/$\overline{DSACK0}$ is not recognized by the start of S3, the processor inserts wait states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both $\overline{DSACK1}$ and $\overline{DSACK0}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the processor continues to sample the $\overline{DSACK1}$/$\overline{DSACK0}$ signals on the falling edges of the clock until one is recognized.

The external device uses R/$\overline{W}$, $\overline{DS}$, SIZ1, SIZ0, A1, and A0 to latch data from the appropriate byte(s) of the data bus (D31–D24, D23–D16, D15–D8, and D7–D0). SIZ1, SIZ0, A1, and A0 select the bytes of the data bus. If it has not already done so, the device asserts $\overline{DSACK1}$/$\overline{DSACK0}$ to signal that it has successfully stored the data.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

State 4

MC68020/EC020—The processor issues no new control signals during S4.

State 5

MC68020—The processor negates $\overline{AS}$ and $\overline{DS}$ during S5. It holds the address and data valid during S5 to provide address hold time for memory systems. R/$\overline{W}$, SIZ1, SIZ0, FC2–FC0, and $\overline{DBEN}$ also remain valid throughout S5.

The external device must keep $\overline{DSACK1}$/$\overline{DSACK0}$ asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must negate $\overline{DSACK1}$/$\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACK1}$/$\overline{DSACK0}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

MC68EC020—The processor negates $\overline{AS}$ and $\overline{DS}$ during S5. It holds the address and data valid during S5 to provide address hold time for memory systems. R/$\overline{W}$, SIZ1, SIZ0, and FC2–FC0 also remain valid throughout S5.

The external device must keep $\overline{DSACK1}$/$\overline{DSACK0}$ asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must negate $\overline{DSACK1}$/$\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACK1}$/$\overline{DSACK0}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

5.3.3 Read-Modify-Write Cycle

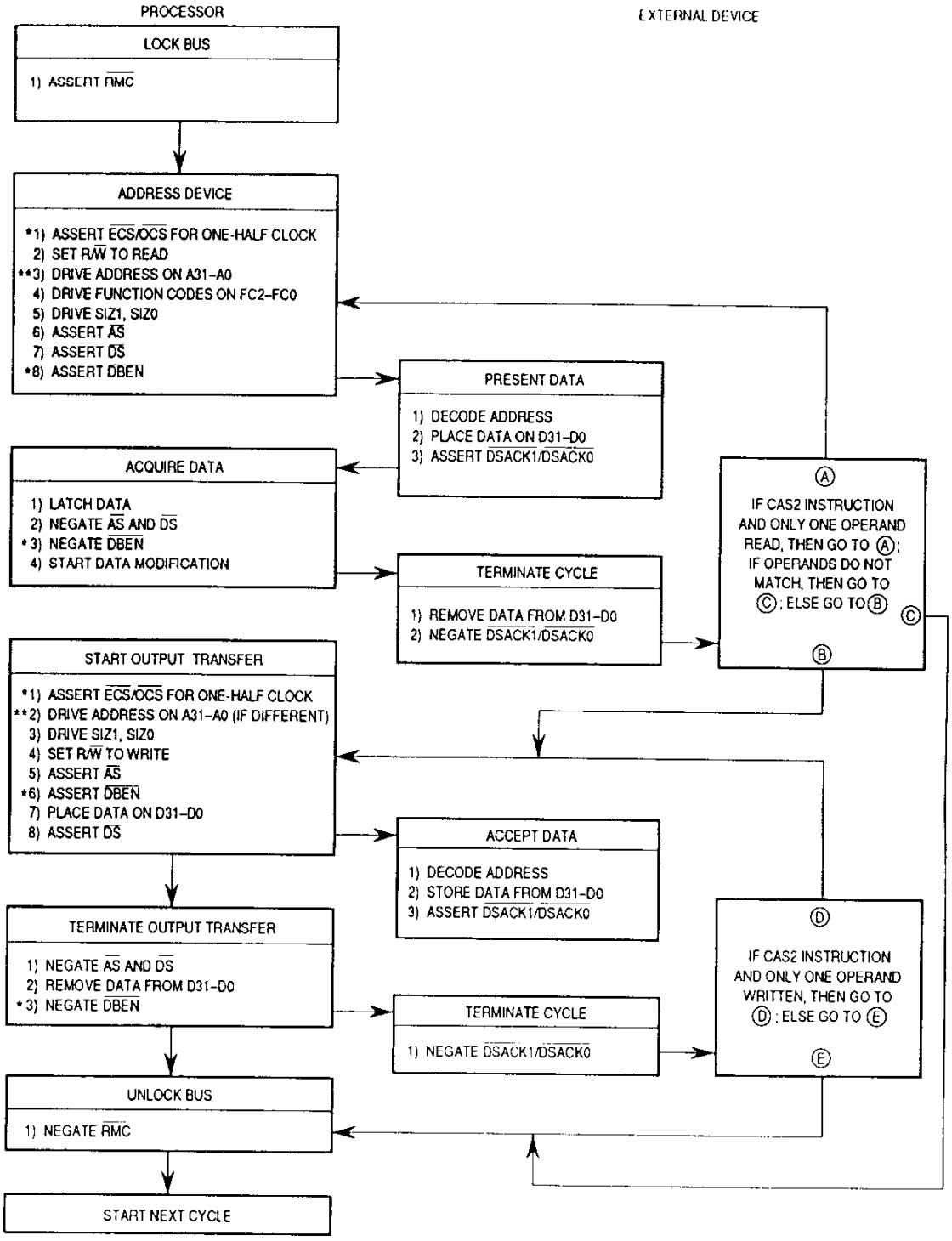
The read-modify-write cycle performs a read, conditionally modifies the data in the arithmetic logic unit, and may write the data out to memory. In the MC68020/EC020, this operation is indivisible, providing semaphore capabilities for multiprocessor systems. During the entire read-modify-write sequence, the MC68020/EC020 asserts $\overline{RMC}$ to indicate that an indivisible operation is occurring. The MC68020/EC020 does not issue a $\overline{BG}$ signal in response to a $\overline{BR}$ signal during this operation.

The TAS, CAS, and CAS2 instructions are the only MC68020/EC020 instructions that utilize read-modify-write operations. Depending on the compare results of the CAS and CAS2 instructions, the write cycle(s) may not occur.

Figure 5-29 is a flowchart of the read-modify-write cycle operation. Figure 5-30 is an example timing diagram of a TAS instruction specified in terms of clock periods.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

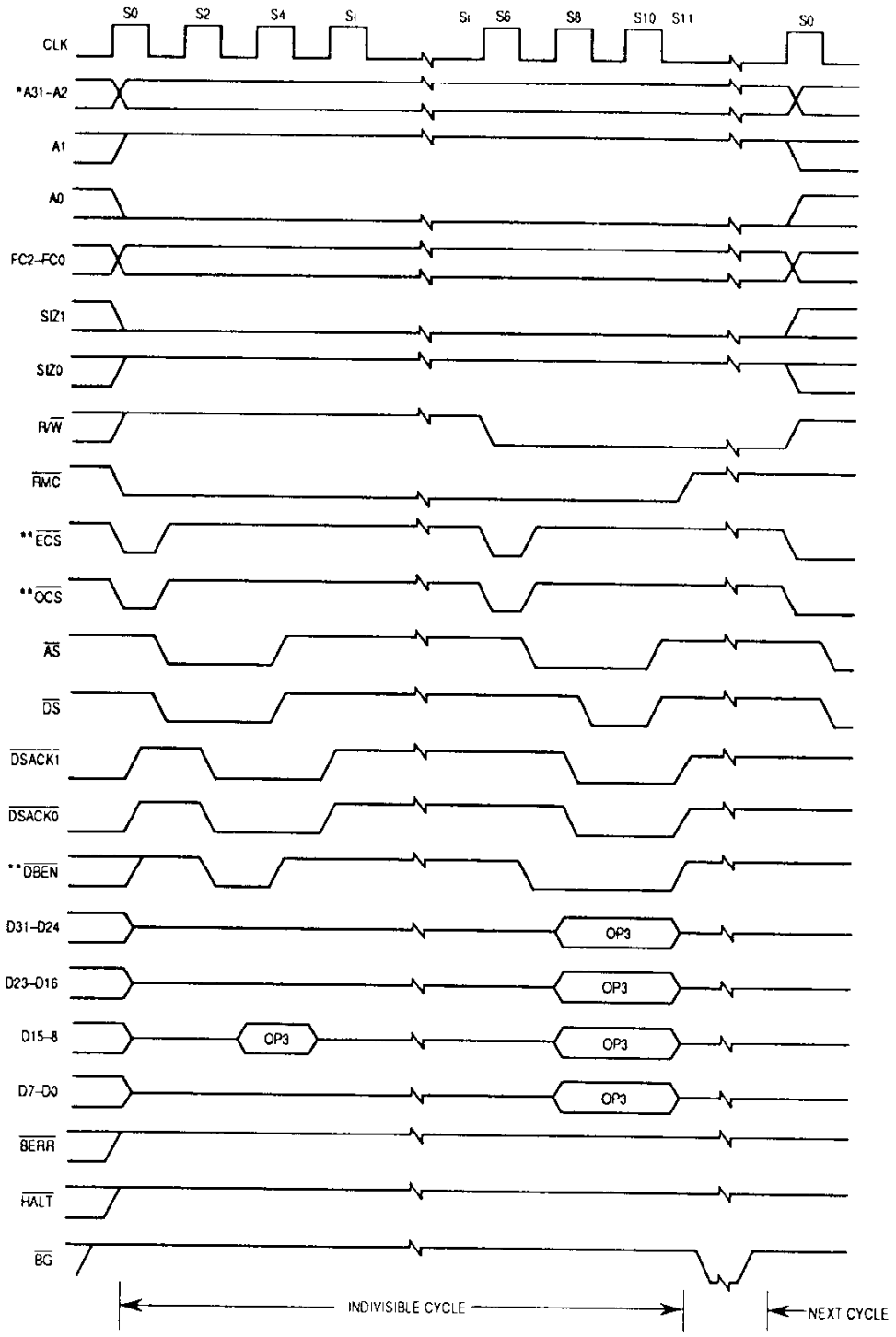


* This step does not apply to the MC68EC020.
** For the MC68EC020, A23–A0.

Figure 5-29. Read-Modify-Write Cycle Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A2.
** This signal does not apply to the MC68EC020.

Figure 5-30. Byte Read-Modify-Write Cycle—32-Bit Port (TAS Instruction)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

State 0

MC68020—The processor asserts $\overline{ECS}$ and $\overline{OCS}$ in S0 to indicate the beginning of an external operand cycle. The processor also asserts $\overline{RMC}$ in S0 to identify a read-modify-write cycle. The processor places a valid address on A31–A0 and valid function codes on FC2–FC0. The function codes select the address space for the operation. SIZ1, SIZ0 become valid in S0 to indicate the operand size. The processor drives R/$\overline{W}$ high for the read cycle.

MC68EC020—The processor asserts $\overline{RMC}$ in S0 to identify a read-modify-write cycle. The processor places a valid address on A23–A0 and valid function codes on FC2–FC0. The function codes select the address space for the operation. SIZ1–SIZ0 become valid in S0 to indicate the operand size. The processor drives R/$\overline{W}$ high for the read cycle.

State 1

MC68020—One-half clock later in S1, the processor asserts $\overline{AS}$ to indicate that the address on the address bus is valid. The processor also asserts $\overline{DS}$ during S1. In addition, the $\overline{ECS}$ (and $\overline{OCS}$, if asserted) signal is negated during S1.

MC68EC020—One-half clock later in S1, the processor asserts $\overline{AS}$ to indicate that the address on the address bus is valid. The processor also asserts $\overline{DS}$ during S1.

State 2

MC68020—During S2, the processor asserts $\overline{DBEN}$ to enable external data buffers. The selected device uses R/$\overline{W}$, SIZ1, SIZ0, A1, A0, and $\overline{DS}$ to place 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 may assert the $\overline{DSACK1}$/$\overline{DSACK0}$ signals.

MC68EC020—During S2, the selected device uses R/$\overline{W}$, SIZ1, SIZ0, A1, A0, and $\overline{DS}$ to place 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 may assert the $\overline{DSACK1}$/$\overline{DSACK0}$ signals.

State 3

MC68020/EC020—As long as at least one of the $\overline{DSACK1}$/$\overline{DSACK0}$ signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), data is latched on the next falling edge of the clock, and the cycle terminates. If $\overline{DSACK1}$/$\overline{DSACK0}$ is not recognized by the start of S3, the processor inserts wait states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both $\overline{DSACK0}$ and $\overline{DSACK1}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the processor continues to sample the $\overline{DSACK1}$/$\overline{DSACK0}$ signals on the falling edges of the clock until one is recognized.

State 4

MC68020/EC020—At the end of S4, the processor latches the incoming data.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

State 5

MC68020—The processor negates $\overline{AS}$, $\overline{DS}$, and $\overline{DBEN}$ during S5. If more than one read cycle is required to read in the operand(s), S0–S5 are repeated for each read cycle. When the read cycle(s) are complete, the processor holds the address, R/$\overline{W}$, and FC2–FC0 valid in preparation for the write portion of the cycle.

The external device keeps its data and $\overline{DSACK1}$/$\overline{DSACK0}$ signals asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove the data and negate $\overline{DSACK1}$/$\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACK1}$/$\overline{DSACK0}$ signals that remain asserted beyond this limit may be prematurely detected for the next portion of the operation.

MC68EC020—The processor negates $\overline{AS}$, $\overline{DS}$, and $\overline{DBEN}$ during S5. If more than one read cycle is required to read in the operand(s), S0–S5 are repeated for each read cycle. When the read cycle(s) is complete, the processor holds the address, R/$\overline{W}$, and FC2–FC0 valid in preparation for the write portion of the cycle.

The external device keeps its data and $\overline{DSACK1}$/$\overline{DSACK0}$ signals asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove the data and negate $\overline{DSACK1}$/$\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACK1}$/$\overline{DSACK0}$ signals that remain asserted beyond this limit may be prematurely detected for the next portion of the operation.

5

Idle States

MC68020/EC020—The processor does not assert any new control signals during the idle states, but it may internally begin the modify portion of the cycle at this time. S6–S11 are omitted if no write cycle is required. If a write cycle is required, the R/$\overline{W}$ signal remains in the read mode until S6 to prevent bus conflicts with the preceding read portion of the cycle; the data bus is not driven until S8.

State 6

MC68020—The processor asserts $\overline{ECS}$ and $\overline{OCS}$ in S6 to indicate that another external cycle is beginning. The processor drives R/$\overline{W}$ low for a write cycle. Depending on the write operation to be performed, the address lines may change during S6.

MC68EC020—During S6, the processor drives R/$\overline{W}$ low for a write cycle. Depending on the write operation to be performed, the address lines may change during S6.

State 7

MC68020—During S7, the processor asserts $\overline{AS}$, indicating that the address on the address bus is valid. The processor also asserts $\overline{DBEN}$, which can be used to enable data buffers. In addition, $\overline{ECS}$ (and $\overline{OCS}$, if asserted) is negated during S7.

MC68EC020—During S7, the processor asserts $\overline{AS}$, indicating that the address on the address bus is valid.

State 8

MC68020/EC020—During S8, the processor places the data to be written onto the data bus.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

State 9

MC68020/EC020 · The processor asserts $\overline{DS}$ during S9, indicating that the data on the data bus is stable. As long as at least one of the $\overline{DSACK1}/\overline{DSACK0}$ signals is recognized by the end of S8 (meeting the asynchronous input setup time requirement), the cycle terminates one clock later. If $\overline{DSACK1}/\overline{DSACK0}$ is not recognized by the start of S9, the processor inserts wait states instead of proceeding to S10 and S11. To ensure that wait states are inserted, both $\overline{DSACK1}$ and $\overline{DSACK0}$ must remain negated throughout the asynchronous input setup and hold times around the end of S8. If wait states are added, the processor continues to sample $\overline{DSACK1}/\overline{DSACK0}$ signals on the falling edges of the clock until one is recognized.

The external device uses R/$\overline{W}$, $\overline{DS}$, SIZ1, SIZ0, A1, and A0 to latch data from the appropriate section(s) of the data bus (D31–D24, D23–D16, D15–D8, and D7–D0). SIZ1, SIZ0, A1, and A0 select the data bus sections. If it has not already done so, the device asserts $\overline{DSACK1}/\overline{DSACK0}$ when it has successfully stored the data.

State 10

MC68020/EC020—The processor issues no new control signals during S10.

State 11

MC68020/EC020—The processor negates $\overline{AS}$ and $\overline{DS}$ during S11. It holds the address and data valid during S11 to provide address hold time for memory systems. R/$\overline{W}$ and FC2–FC0 also remain valid throughout S11.

If more than one write cycle is required, S6–S11 are repeated for each write cycle.

The external device keeps $\overline{DSACK1}/\overline{DSACK0}$ asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove its data and negate $\overline{DSACK1}/\overline{DSACK0}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$.

5.4 CPU SPACE CYCLES

FC2–FC0 select user and supervisor program and data areas as listed in Table 2-1. The area selected by FC2–FC0 = 111 is classified as the CPU space. The interrupt acknowledge, breakpoint acknowledge, module operations, and coprocessor communication cycles described in the following paragraphs utilize CPU space.

The CPU space type is encoded on A19–A16 during a CPU space operation and indicates the function that the processor is performing. On the MC68020/EC020, four of the encodings are implemented as shown in Figure 5-31. All unused values are reserved by Motorola for future use.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

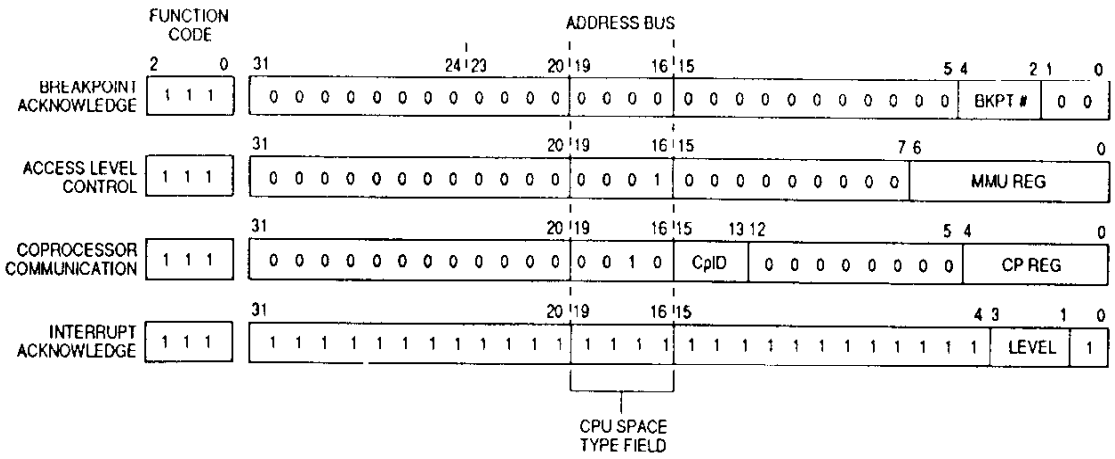


Figure 5-31. MC68020/EC020 CPU Space Address Encoding

5.4.1 Interrupt Acknowledge Bus Cycles

When a peripheral device signals the processor (with the $\overline{IPL2}$–$\overline{IPL0}$ signals) that the device requires service and when the internally synchronized value on these signals indicates a higher priority than the interrupt mask in the status register (or that a transition has occurred in the case of a level 7 interrupt), the processor makes the interrupt a pending interrupt. Refer to **Section 6 Exception Processing** for details on the recognition of interrupts.

The MC68020/EC020 takes an interrupt exception for a pending interrupt within one instruction boundary (after processing any other pending exception with a higher priority). The following paragraphs describe the various kinds of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing.

5.4.1.1 INTERRUPT ACKNOWLEDGE CYCLE—TERMINATED NORMALLY. When the MC68020/EC020 processes an interrupt exception, it performs an interrupt acknowledge cycle to obtain the number of the vector that contains the starting location of the interrupt service routine.

Some interrupting devices have programmable vector registers that contain the interrupt vectors for the routines they use. The following paragraphs describe the interrupt acknowledge cycle for these devices. Other interrupting conditions or devices cannot supply a vector number and use the autovector cycle described in **5.4.1.2 Autovector Interrupt Acknowledge Cycle**.

For More Information On This Product,
Go to: www.freescale.com

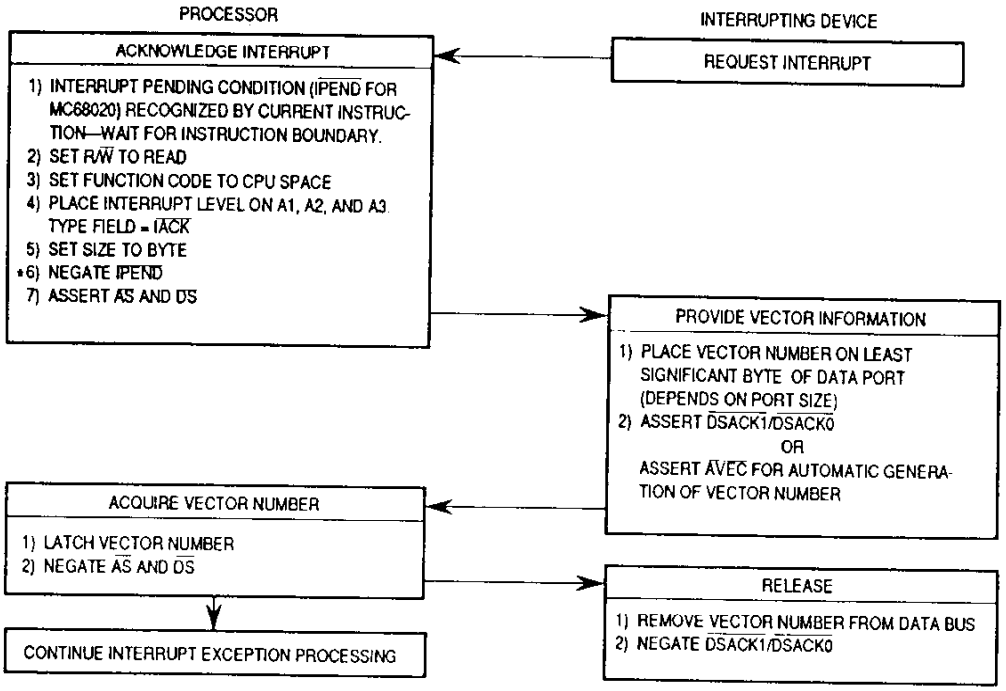
Freescale Semiconductor, Inc.

The interrupt acknowledge cycle is a read cycle. It differs from the read cycle described in **5.3.1 Read Cycle** in that it accesses the CPU address space. Specifically, the differences are:

1. FC2–FC0 are set 111 for CPU address space.
2. A3, A2, and A1 are set to the interrupt request level (the inverted values of $\overline{IPL2}$, $\overline{IPL1}$, and $\overline{IPL0}$, respectively).
3. The CPU space type field (A19–A16) is set to 1111, the interrupt acknowledge code.
4. Other address signals (A31–A20, A15–A4, and A0 for the MC68020; A23–A20, A15–A4, and A0 for the MC68EC020) are set to one.

The responding device places the vector number on the data bus during the interrupt acknowledge cycle. Beyond this, the cycle is terminated normally with $\overline{DSACK1}/\overline{DSACK0}$. Figure 5-32 is the flowchart of the interrupt acknowledge cycle.

Figure 5-33 shows the timing for an interrupt acknowledge cycle terminated with $\overline{DSACK1}/\overline{DSACK0}$.

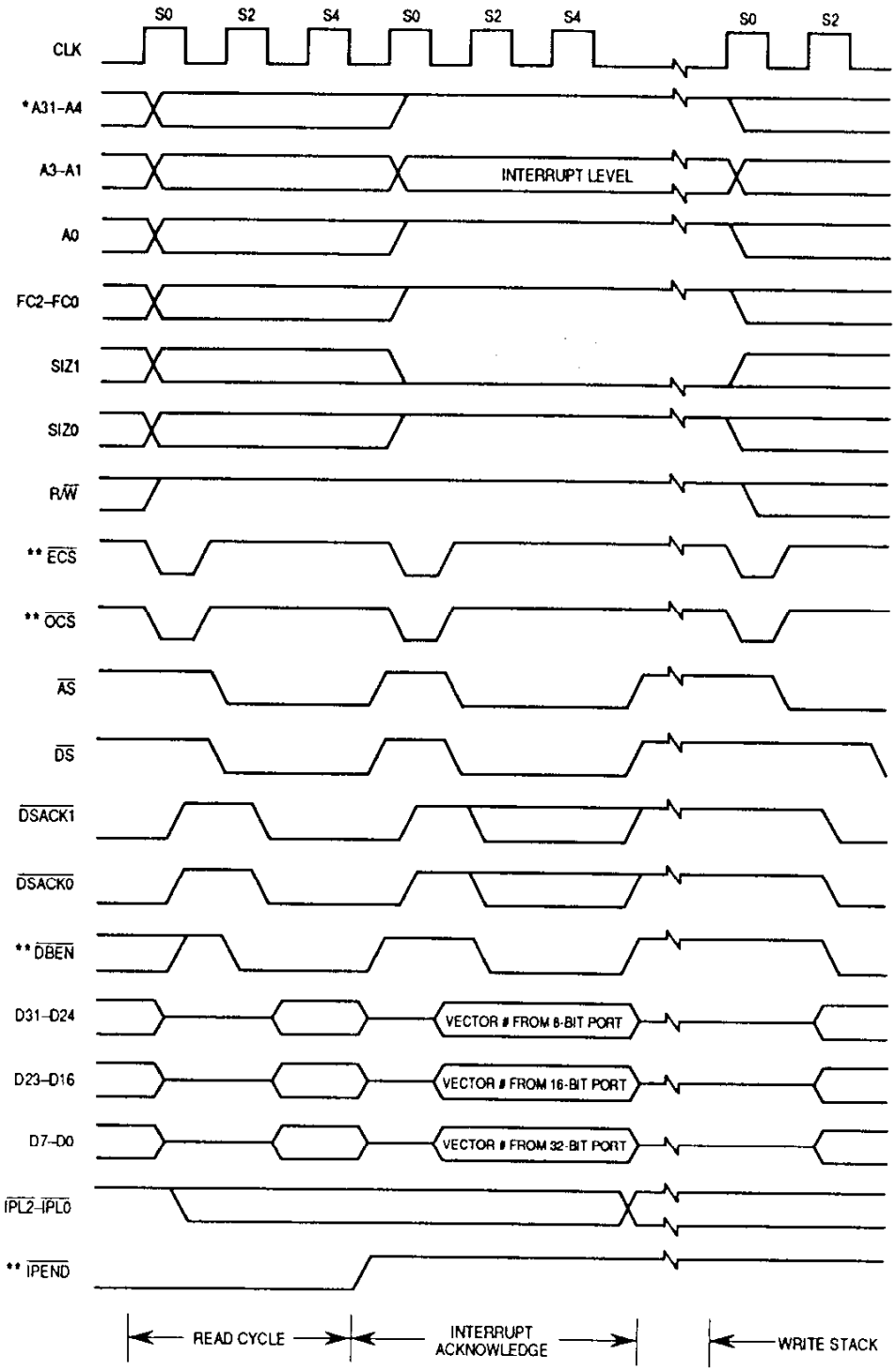


* This step does not apply to the MC68EC020.

Figure 5-32. Interrupt Acknowledge Cycle Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A4.
** This signal does not apply to the MC68EC020.

Figure 5-33. Interrupt Acknowledge Cycle Timing

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

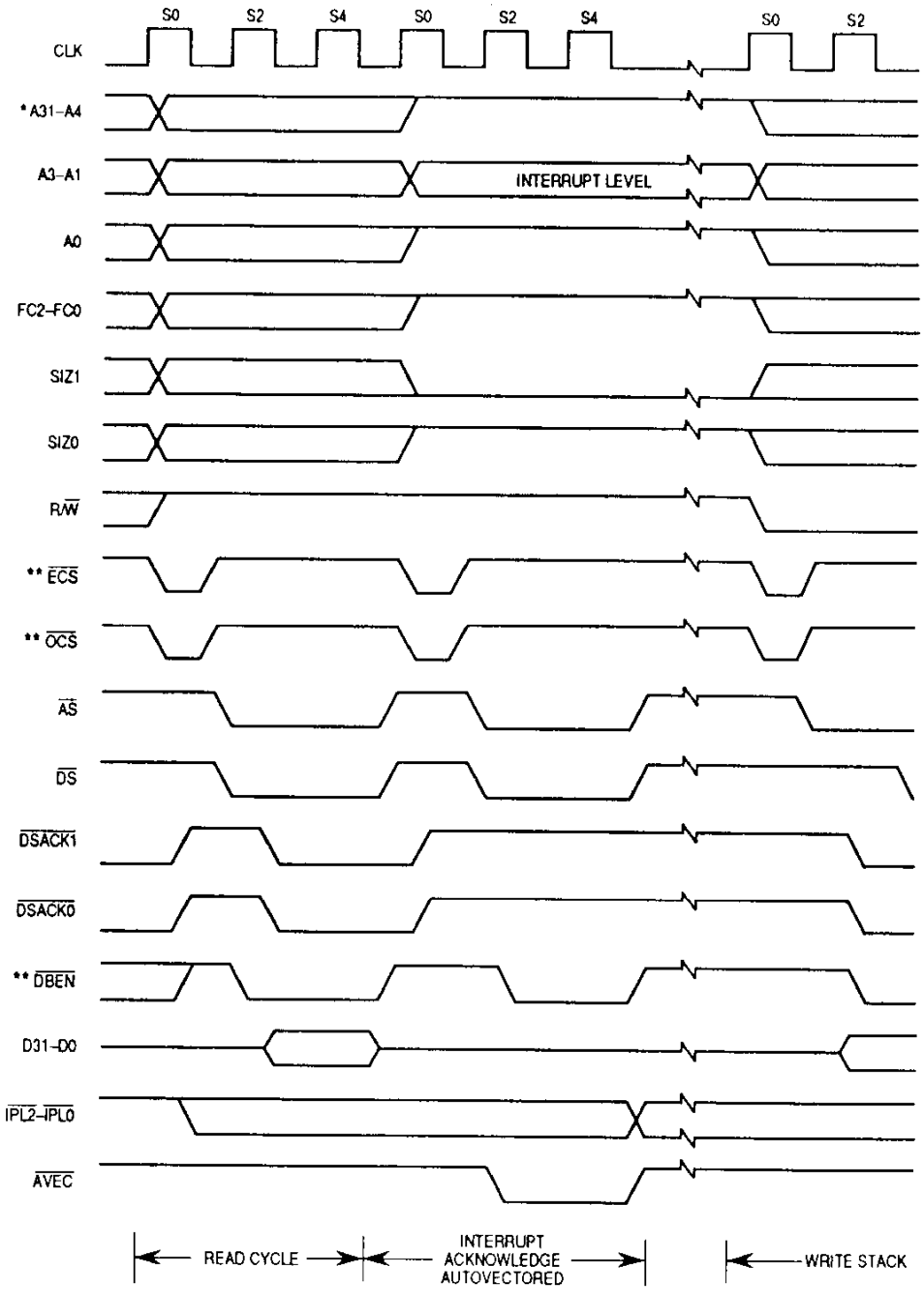
5.4.1.2 AUTOVECTOR INTERRUPT ACKNOWLEDGE CYCLE. When the interrupting device cannot supply a vector number, it requests an automatically generated vector or autovector. Instead of placing a vector number on the data bus and asserting $\overline{DSACK1}/\overline{DSACK0}$, the device asserts $\overline{AVEC}$ to terminate the cycle. The $\overline{DSACK1}/\overline{DSACK0}$ signals may not be asserted during an interrupt acknowledge cycle terminated by $\overline{AVEC}$.

The vector number supplied in an autovector operation is derived from the interrupt level of the current interrupt. When $\overline{AVEC}$ is asserted instead of $\overline{DSACK1}/\overline{DSACK0}$ during an interrupt acknowledge cycle, the MC68020/EC020 ignores the state of the data bus and internally generates the vector number, the sum of the interrupt level plus 24 ($18). Seven distinct autovectors, which correspond to the seven levels of interrupt available with $\overline{IPL2}$–$\overline{IPL0}$, can be used. Figure 5-34 shows the timing for an autovector operation.

5.4.1.3 SPURIOUS INTERRUPT CYCLE. When a device does not respond to an interrupt acknowledge cycle with $\overline{AVEC}$ or $\overline{DSACK1}/\overline{DSACK0}$, the external logic typically returns $\overline{BERR}$. In this case, the MC68020/EC020 automatically generates 24, the spurious interrupt vector number. If $\overline{HALT}$ is also asserted, the processor retries the cycle.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A4.
** This signal does not apply to the MC68EC020.

Figure 5-34. Autovector Operation Timing

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.4.2 Breakpoint Acknowledge Cycle

The breakpoint acknowledge cycle is generated by the execution of a BKPT instruction. The breakpoint acknowledge cycle allows the external hardware to provide an instruction word directly into the instruction pipeline as the program executes. This cycle accesses the CPU space with a type field of zero and provides the breakpoint number specified by the instruction on address lines A4–A2. If the external hardware terminates the cycle with $\overline{DSACK1}/\overline{DSACK0}$, the data on the bus (an instruction word) is inserted into the instruction pipe, replacing the breakpoint opcode, and is executed after the breakpoint acknowledge cycle completes. The BKPT instruction requires a word to be transferred so that if the first bus cycle accesses an 8-bit port, a second cycle is required. If the external logic terminates the breakpoint acknowledge cycle with $\overline{BERR}$ (i.e., no instruction word available), the processor takes an illegal instruction exception. Figure 5-35 is a flowchart of the breakpoint acknowledge cycle. Figure 5-36 shows the timing for a breakpoint acknowledge cycle that returns an instruction word. Figure 5-37 shows the timing for a breakpoint acknowledge cycle that signals an exception.

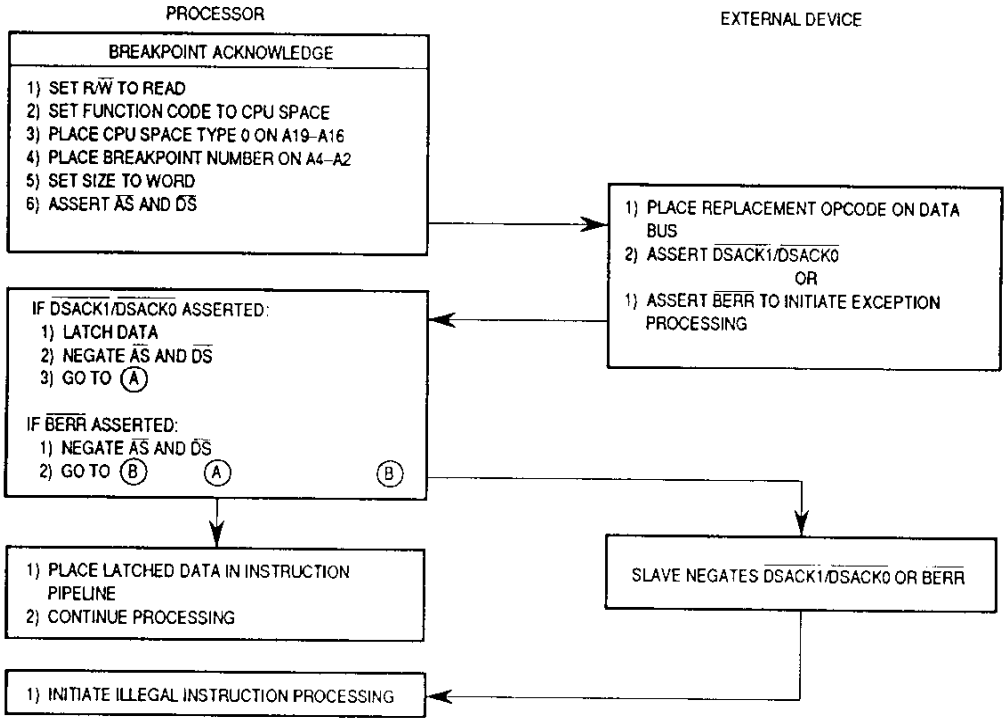
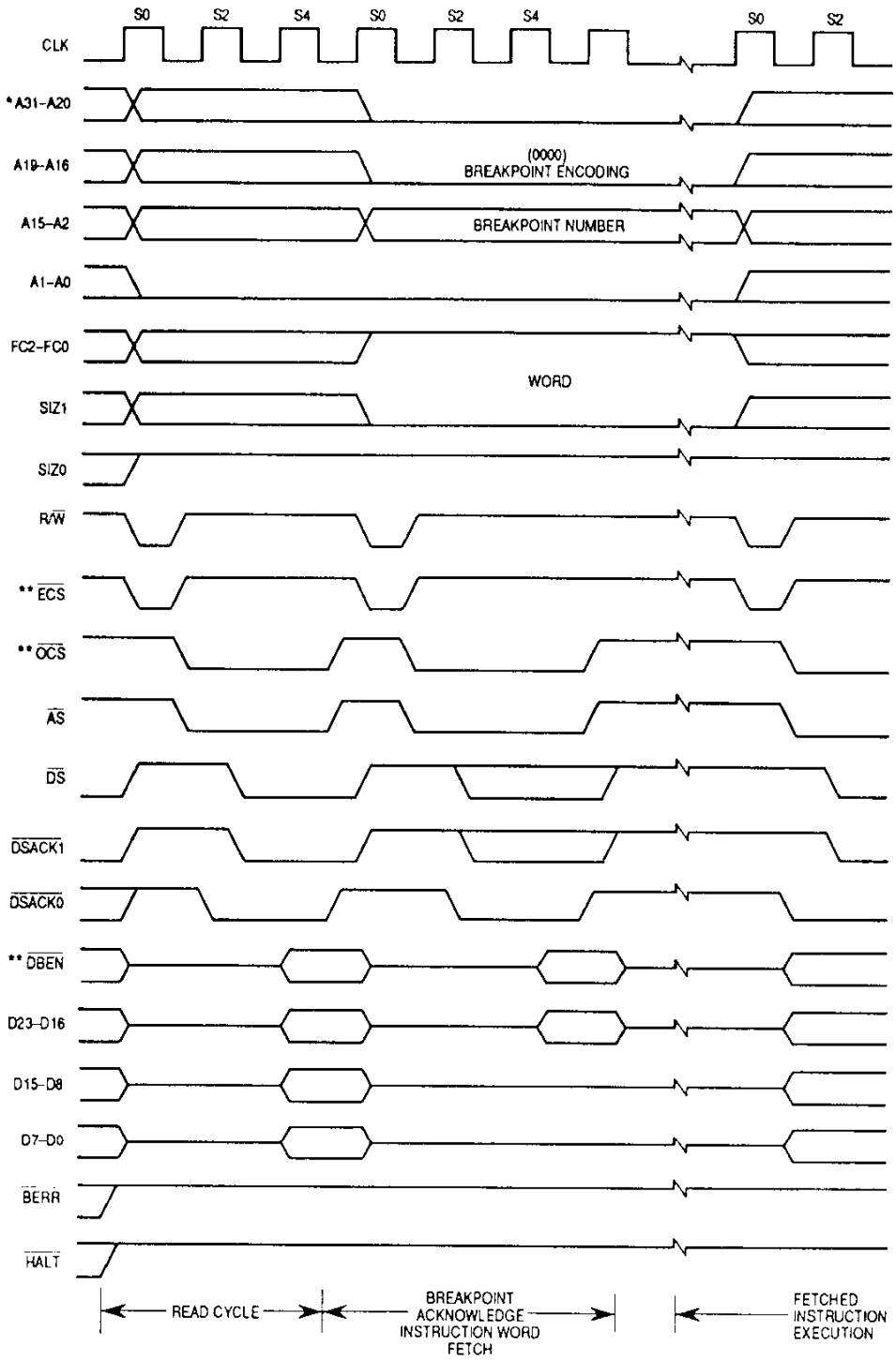


Figure 5-35. Breakpoint Acknowledge Cycle Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

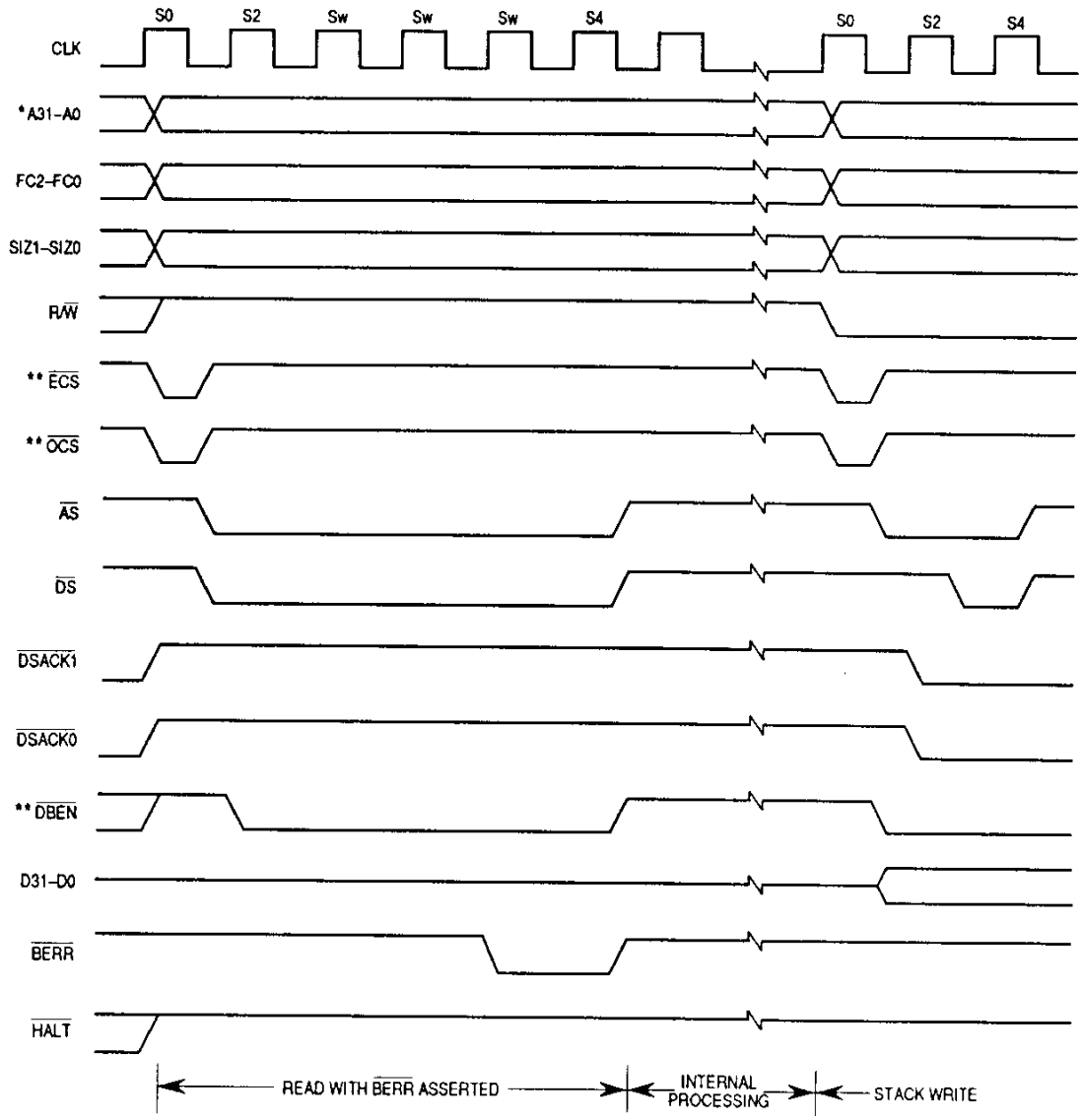


* For the MC68EC020, A23–A20.
** This signal does not apply to the MC68EC020.

Figure 5-36. Breakpoint Acknowledge Cycle Timing

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A0.
** This signal does not apply to the MC68EC020.

Figure 5-37. Breakpoint Acknowledge Cycle Timing (Exception Signaled)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.4.3 Coprocessor Communication Cycles

The MC68020/EC020 coprocessor interface provides instruction-oriented communication between the processor and as many as eight coprocessors. Coprocessor accesses use the MC68020/EC020 bus protocol except that the address bus supplies access information rather than a 32-bit address. The CPU space type field (A19–A16) for a coprocessor operation is 0010. A15–A13 contain the coprocessor identification number (CpID), and A5–A0 specify the coprocessor interface register to be accessed. The memory management unit of an MC68020/EC020 system is always identified by a CpID of zero and has an extended register select field (A7–A0) in CPU space 0001 for use by the CALLM and RTM access level checking mechanism. Refer to **Section 9 Applications Information** for more details.

5.5 BUS EXCEPTION CONTROL CYCLES

The MC68020/EC020 bus architecture requires assertion of $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ from an external device to signal that a bus cycle is complete. $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ or $\overline{\text{AVEC}}$ is not asserted if:

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

External circuitry can assert $\overline{\text{BERR}}$ when no device responds by asserting $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ or $\overline{\text{AVEC}}$ within an appropriate period of time after the processor asserts $\overline{\text{AS}}$. Assertion of $\overline{\text{BERR}}$ allows the cycle to terminate and the processor to enter exception processing for the error condition.

$\overline{\text{HALT}}$ is also used for bus exception control. $\overline{\text{HALT}}$ can be asserted by an external device for debugging purposes to cause single bus cycle operation or can be asserted in combination with $\overline{\text{BERR}}$ to cause a retry of a bus cycle in error.

To properly control termination of a bus cycle for a retry or a bus error condition, $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ can be asserted and negated with the rising edge of the MC68020/EC020 clock. This procedure ensures that when two signals are asserted simultaneously, the required setup time (#47A) and hold time (#47B) for both of them is met for the same falling edge of the processor clock. (Refer to **Section 10 Electrical Characteristics** for timing requirements.) This or some equivalent precaution should be designed into the external circuitry that provides these signals.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The acceptable bus cycle terminations for asynchronous cycles are summarized in relation to $\overline{DSACK1}/\overline{DSACK0}$ assertion as follows (case numbers refer to Table 5-8):

Normal Termination:

$\overline{DSACK1}/\overline{DSACK0}$ is asserted; $\overline{BERR}$ and $\overline{HALT}$ remain negated (case 1).

Halt Termination:

$\overline{HALT}$ is asserted at same time or before $\overline{DSACK1}/\overline{DSACK0}$, and $\overline{BERR}$ remains negated (case 2).

Bus Error Termination:

$\overline{BERR}$ is asserted in lieu of, at the same time, or before $\overline{DSACK1}/\overline{DSACK0}$ (case 3) or after $\overline{DSACK1}/\overline{DSACK0}$ (case 4), and $\overline{HALT}$ remains negated; $\overline{BERR}$ is negated at the same time or after $\overline{DSACK1}/\overline{DSACK0}$.

5

Retry Termination:

$\overline{HALT}$ and $\overline{BERR}$ are asserted in lieu of, at the same time, or before $\overline{DSACK1}/\overline{DSACK0}$ (case 5) or after $\overline{DSACK1}/\overline{DSACK0}$ (case 6); $\overline{BERR}$ is negated at the same time or after $\overline{DSACK1}/\overline{DSACK0}$; $\overline{HALT}$ may be negated at the same time or after $\overline{BERR}$.

Table 5-8. $\overline{DSACK1}/\overline{DSACK0}$, $\overline{BERR}$, $\overline{HALT}$ Assertion Results

Case No.	Control Signal	Asserted on Rising Edge of State		Result
		n	n+2	
1	$\overline{DSACK1}/\overline{DSACK0}$ $\overline{BERR}$ $\overline{HALT}$	A N N	S N X	Normal cycle terminate and continue.
2	$\overline{DSACK1}/\overline{DSACK0}$ $\overline{BERR}$ $\overline{HALT}$	A N A/S	S N S	Normal cycle terminate and halt. Continue when $\overline{HALT}$ negated.
3	$\overline{DSACK1}/\overline{DSACK0}$ $\overline{BERR}$ $\overline{HALT}$	N/A A N	X S N	Terminate and take bus error exception, possibly deferred.
4	$\overline{DSACK1}/\overline{DSACK0}$ $\overline{BERR}$ $\overline{HALT}$	A N N	X A N	Terminate and take bus error exception, possibly deferred.
5	$\overline{DSACK1}/\overline{DSACK0}$ $\overline{BERR}$ $\overline{HALT}$	N/A A A/S	X S S	Terminate and retry when $\overline{HALT}$ negated.
6	$\overline{DSACK1}/\overline{DSACK0}$ $\overline{BERR}$ $\overline{HALT}$	A N N	X A A	Terminate and retry when $\overline{HALT}$ negated.

Legend:

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

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 5-8 lists various combinations of control signal sequences and the resulting bus cycle terminations. To ensure predictable operation, $\overline{BERR}$ and $\overline{HALT}$ should be negated according to parameters #28 and #57 in **Section 10 Electrical Characteristics**. $\overline{DSACK1}/\overline{DSACK0}$, $\overline{BERR}$, and $\overline{HALT}$ may be negated after $\overline{AS}$. If $\overline{DSACK1}/\overline{DSACK0}$ or $\overline{BERR}$ remain asserted into S2 of the next bus cycle, that cycle may be terminated prematurely.

Example A:

A system uses a watchdog timer to terminate accesses to an unpopulated address space. The timer asserts $\overline{BERR}$ after timeout (case 3).

Example B:

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

1. Delay $\overline{DSACK1}/\overline{DSACK0}$ assertion until data is verified and assert $\overline{BERR}$ and $\overline{HALT}$ simultaneously to indicate to the processor to automatically retry the error cycle (case 5) or, if data is valid, assert $\overline{DSACK1}/\overline{DSACK0}$ (case 1).
2. Delay $\overline{DSACK1}/\overline{DSACK0}$ assertion until data is verified and assert $\overline{BERR}$ with or without $\overline{DSACK1}/\overline{DSACK0}$ if data is in error (case 3). This configuration initiates exception processing for software handling of the condition.
3. Assert $\overline{DSACK1}/\overline{DSACK0}$ prior to data verification. If data is invalid, $\overline{BERR}$ is asserted on the next clock cycle (case 4). This configuration initiates exception processing for software handling of the condition.
4. Assert $\overline{DSACK1}/\overline{DSACK0}$ prior to data verification; if data is invalid, assert $\overline{BERR}$ and $\overline{HALT}$ on the next clock cycle (case 6). The memory controller can then correct the RAM prior to or during the automatic retry.

5.5.1 Bus Errors

The $\overline{BERR}$ signal can be used to abort the bus cycle and the instruction being executed. $\overline{BERR}$ takes precedence over $\overline{DSACK1}/\overline{DSACK0}$, provided it meets the timing constraints described in **Section 10 Electrical Characteristics**. If $\overline{BERR}$ does not meet these constraints, it may cause unpredictable operation of the MC68020/EC020. If $\overline{BERR}$ remains asserted into the next bus cycle, it may cause incorrect operation of that cycle.

When $\overline{BERR}$ is issued to terminate a bus cycle, the MC68020/EC020 may enter exception processing immediately following the bus cycle, or it may defer processing the exception. The instruction prefetch mechanism requests instruction words from the bus controller and the instruction cache 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 that instruction word. Should an intervening instruction cause a branch or should a task switch occur, the bus error exception does not occur.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

$\overline{BERR}$ is recognized during a bus cycle in any of the following cases:

1. $\overline{DSACK1}/\overline{DSACK0}$ and $\overline{HALT}$ are negated and $\overline{BERR}$ is asserted.
2. $\overline{HALT}$ and $\overline{BERR}$ are negated and $\overline{DSACK1}/\overline{DSACK0}$ is asserted. $\overline{BERR}$ is then asserted within one clock cycle ($\overline{HALT}$ remains negated).
3. $\overline{BERR}$ and $\overline{HALT}$ are asserted (see **5.5.2 Retry Operation**).

When the processor recognizes a bus error condition, it terminates the current bus cycle in the normal way. Figure 5-38 shows the timing of a bus error for the case in which $\overline{DSACK1}/\overline{DSACK0}$ is not asserted. Figure 5-39 shows the timing for a bus error for the case in which $\overline{BERR}$ is asserted after $\overline{DSACK1}/\overline{DSACK0}$. Exceptions are taken in both cases. (Refer to **Section 6 Exception Processing** for details of bus error exception processing.) When $\overline{BERR}$ is asserted during a read cycle that supplies an instruction to the on-chip cache, the instruction in the cache is marked invalid.

5

When $\overline{BERR}$ is asserted after $\overline{DSACK1}/\overline{DSACK0}$, $\overline{BERR}$ must be asserted within parameter #48 (refer to **Section 10 Electrical Characteristics**) for purely asynchronous operation, or it must be asserted and remain stable during the sample window, defined by parameters #27A and #47B, around the next falling edge of the clock after $\overline{DSACK1}/\overline{DSACK0}$ is recognized. If $\overline{BERR}$ is not stable at this time, the processor may exhibit erratic behavior. In this case, data may be present on the bus, but may not be valid. This sequence may be used by systems that have memory error detection and correction logic and by external cache memories.

5.5.2 Retry Operation

When $\overline{BERR}$ and $\overline{HALT}$ are asserted simultaneously by an external device during a bus cycle, the processor enters the retry sequence. A delayed retry similar to the delayed $\overline{BERR}$ signal described previously can also occur.

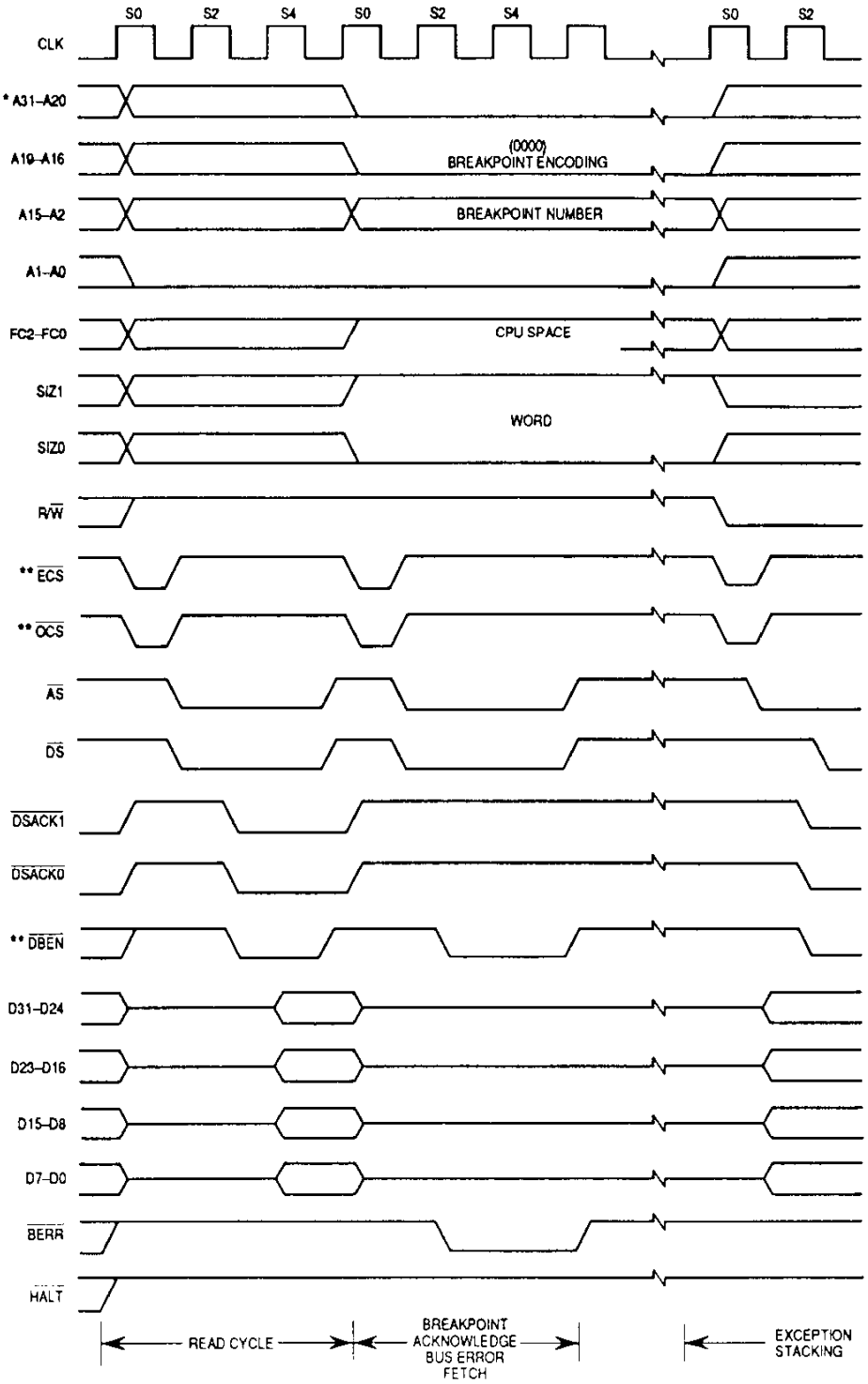
The processor terminates the bus cycle, negates the control signals ($\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, SIZ1, SIZ0, $\overline{RMC}$, and, for the MC68020 only, $\overline{ECS}$ and $\overline{OCS}$), and does not begin another bus cycle until the $\overline{BERR}$ and $\overline{HALT}$ signals have been negated by external logic. After a synchronization delay, the processor retries the previous cycle using the same access information (address, function code, size, etc.) The $\overline{BERR}$ signal should be negated before S2 of the read cycle to ensure correct operation of the retried cycle. Figure 5-40 shows a late retry operation of a cycle.

The processor retries any read or write cycle of a read-modify-write operation separately; $\overline{RMC}$ remains asserted during the entire retry sequence.

Asserting $\overline{BR}$ along with $\overline{BERR}$ and $\overline{HALT}$ provides a relinquish and retry operation. The MC68020/EC020 does not relinquish the bus during a read-modify-write operation. Any device that requires the processor to give up the bus and retry a bus cycle during a read-modify-write cycle must assert $\overline{BERR}$ and $\overline{BR}$ only ($\overline{HALT}$ must not be included). The bus error handler software should examine the read-modify-write bit in the special status word (refer to **Section 6 Exception Processing**) and take the appropriate action to resolve this type of fault when it occurs.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

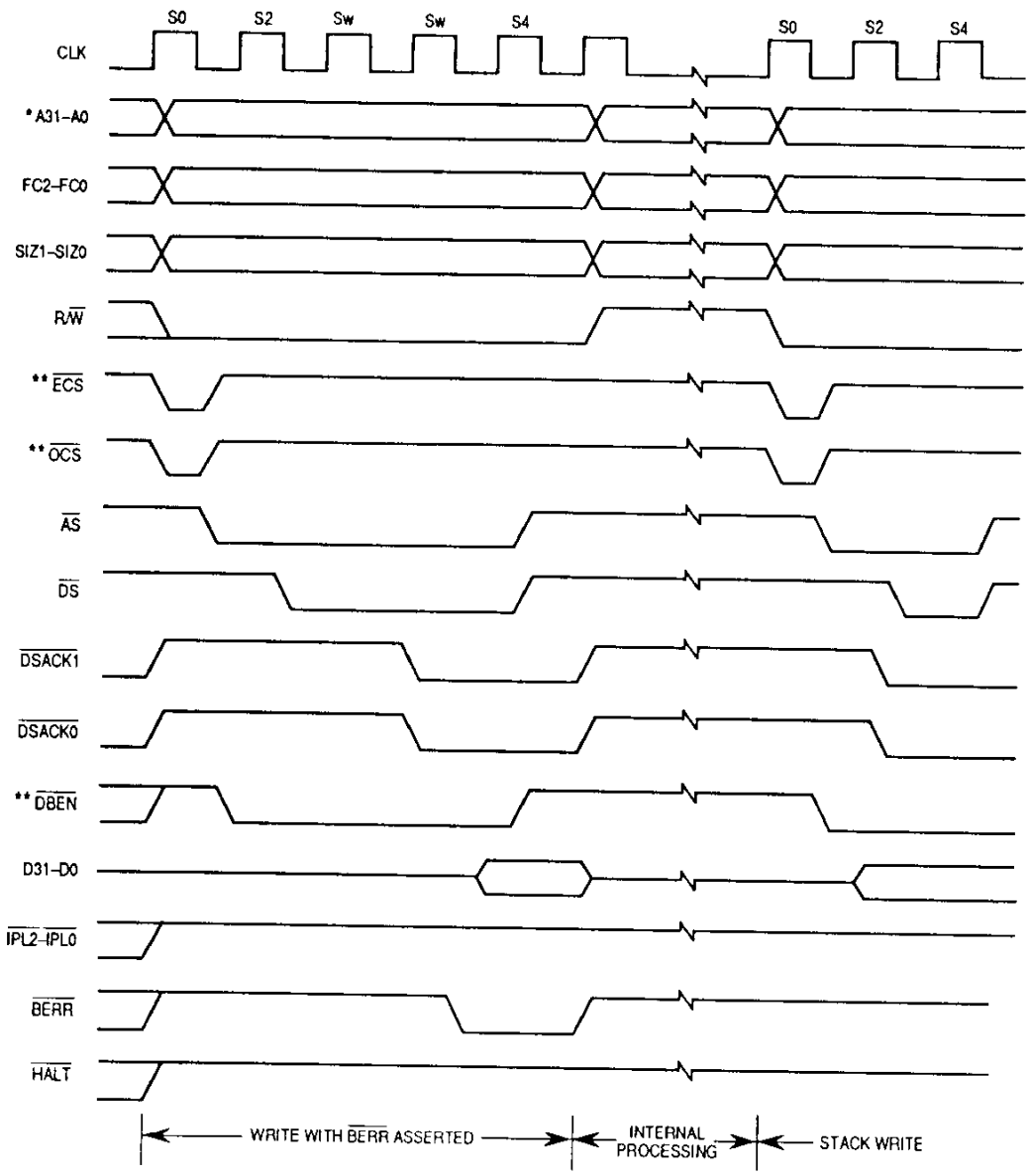


* For the MC68EC020, A23–A20.
** This signal does not apply to the MC68EC020.

Figure 5-38. Bus Error without DSACK1/DSACK0

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

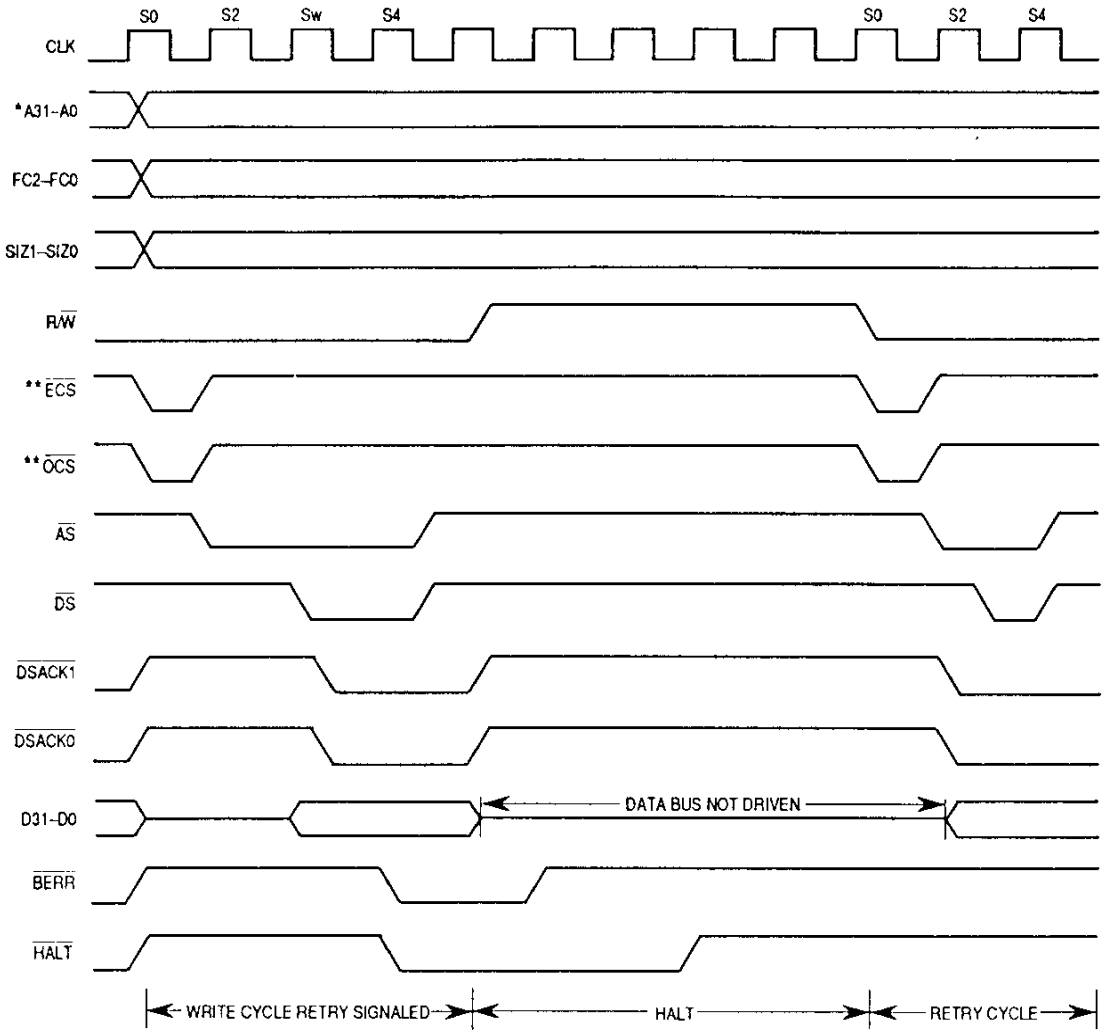


* For the MC68EC020, A23–A0.
** This signal does not apply to the MC68EC020.

Figure 5-39. Late Bus Error with DSACK1/DSACK0

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A0.
** This signal does not apply to the MC68EC020.

Figure 5-40. Late Retry

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.5.3 Halt Operation

When $\overline{HALT}$ is asserted and $\overline{BERR}$ is not asserted, the MC68020/EC020 halts external bus activity at the next bus cycle boundary. $\overline{HALT}$ by itself does not terminate a bus cycle. Negating and reasserting $\overline{HALT}$ in accordance with the correct timing requirements provides a single-step (bus cycle to bus cycle) operation. The $\overline{HALT}$ signal affects external bus cycles only; thus, a program that resides in the instruction cache and does not require use of the external bus may continue executing unaffected by $\overline{HALT}$.

The single-cycle mode allows the user to proceed through (and debug) external processor operations, one bus cycle at a time. Figure 5-41 shows the timing requirements for a single-cycle operation. Since the occurrence of a bus error while $\overline{HALT}$ is asserted causes a retry operation, the user must anticipate retry cycles while debugging in the single-cycle mode. The single-step operation and the software trace capability allow the system debugger to trace single bus cycles, single instructions, or changes in program flow. These processor capabilities, along with a software debugging package, give complete debugging flexibility.

When the processor completes a bus cycle with the $\overline{HALT}$ signal asserted, the data bus is placed in the high-impedance state, and the bus control signals ($\overline{AS}$, $\overline{DS}$, and, for the MC68020 only, $\overline{ECS}$ and $\overline{OCS}$) are negated (not placed in the high-impedance state); A31–A0 for the MC68020 or A23–A0 for the MC68EC020, FC2–FC0, SIZ1, SIZ0, and R/$\overline{W}$ remain in the same state. The halt operation has no effect on bus arbitration (refer to **5.7 Bus Arbitration**). When bus arbitration occurs while the MC68020/EC020 is halted, the address and control signals (A31–A0, FC2–FC0, SIZ1, SIZ0, R/$\overline{W}$, $\overline{AS}$, $\overline{DS}$, and, for the MC68020 only, $\overline{ECS}$ and $\overline{OCS}$) are also placed in the high-impedance state. Once bus mastership is returned to the MC68020/EC020, if $\overline{HALT}$ is still asserted, A31–A0 for the MC68020 or A23–A0 for the MC68EC020, FC2–FC0, SIZ1, SIZ0, and R/$\overline{W}$ are again driven to their previous states. The MC68020/EC020 does not service interrupt requests while it is halted (although the MC68020 may assert the $\overline{IPEND}$ signal as appropriate).

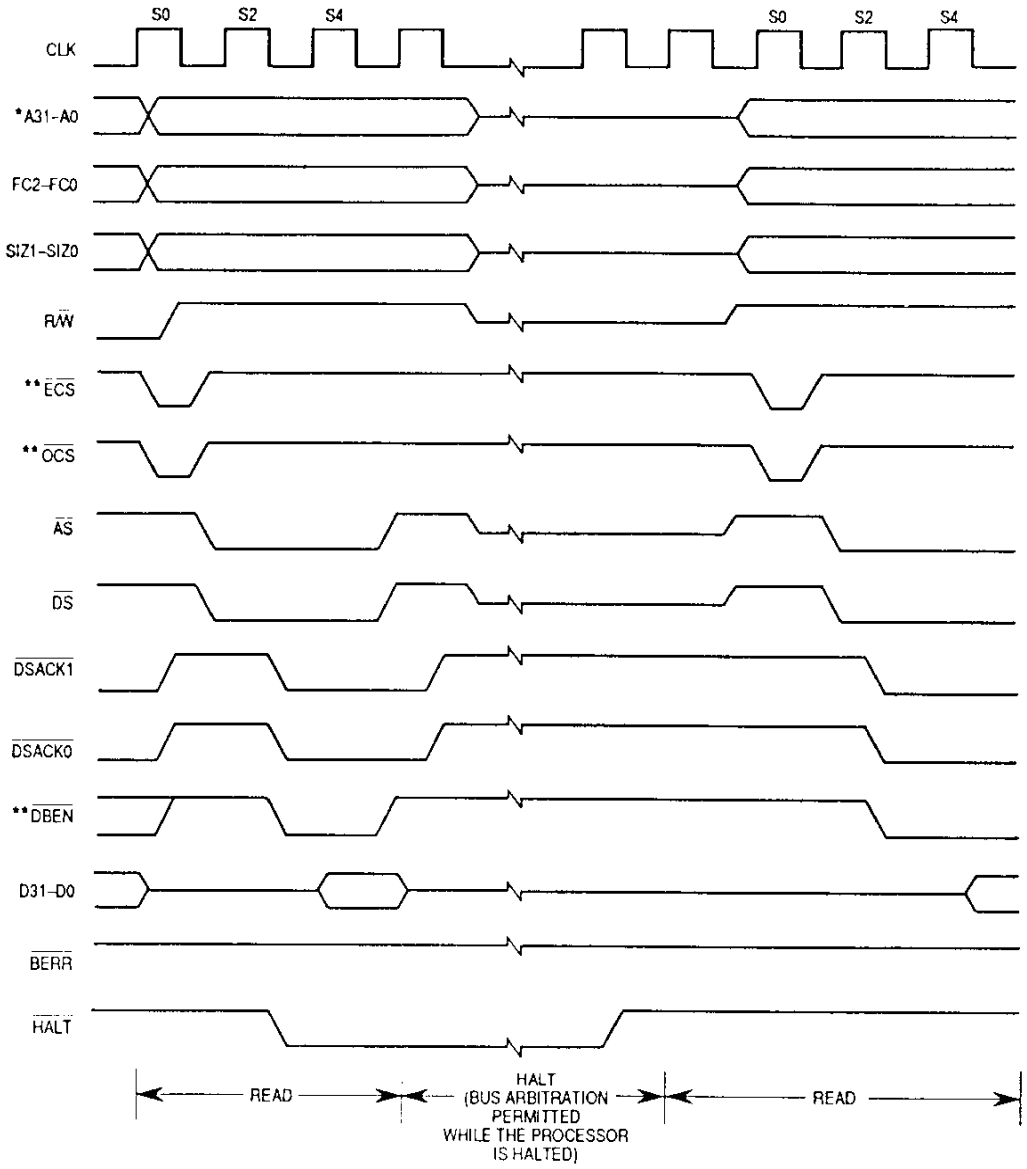
5.5.4 Double Bus Fault

When a bus error or an address error occurs during the exception processing sequence for a previous bus error, a previous address error, or a reset exception, a double bus fault occurs. For example, the processor attempts to stack several words containing information about the state of the machine while processing a bus error exception. If a bus error exception occurs during the stacking operation, the second error is considered a double bus fault. When a double bus fault occurs, the processor halts and asserts $\overline{HALT}$. Only an external reset operation can restart a halted processor. However, bus arbitration can still occur (refer to **5.7 Bus Arbitration**).

A second bus error or address error that occurs after exception processing has completed (during the execution of the exception handler routine or later) does not cause a double bus fault. A bus cycle that is retried does not constitute a bus error or contribute to a double bus fault. The processor continues to retry the same bus cycle as long as the external hardware requests it.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



* For the MC68EC020, A23–A0.
** This signal does not apply to the MC68EC020

Figure 5-41. Halt Operation Timing

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.6 BUS SYNCHRONIZATION

The MC68020/EC020 overlaps instruction execution—that is, during bus activity for one instruction, instructions that do not use the external bus can be executed. Due to the independent operation of the on-chip cache relative to the operation of the bus controller, many subsequent instructions can be executed, resulting in seemingly nonsequential instruction execution. When this 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 by freezing instruction execution until all pending bus cycles have completed.

An example of the use of the NOP instruction for this purpose is the case of 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 $\overline{\text{BERR}}$. Since the MC68020/EC020 cannot process the bus error until the end of the bus cycle, the external hardware has not successfully interrupted 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, bus error exception processing proceeds immediately after the write and 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.

5.7 BUS ARBITRATION

The bus design of the MC68020/EC020 provides for a single bus master at any one time: either the processor or an external device. One or more of the external devices on the bus can have the capability of becoming bus master. Bus arbitration is the protocol by which an external device becomes bus master; the bus controller in the MC68020/EC020 manages the bus arbitration signals so that the processor has the lowest priority.

Bus arbitration differs in the MC68020 and MC68EC020 due to the absence of $\overline{\text{BGACK}}$ in the MC68EC020. Because of this difference, bus arbitration of the MC68020 and MC68EC020 is discussed separately.

External devices that need to obtain the bus must assert the bus arbitration signals in the sequences described in **5.7.1 MC68020 Bus Arbitration** or **5.7.2 MC68EC020 Bus Arbitration**. Systems having several devices that can become bus master require external circuitry to assign priorities to the devices, so that when two or more external devices attempt to become bus master at the same time, the one having the highest priority becomes bus master first.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.7.1 MC68020 Bus Arbitration

The sequence of the MC68020 bus arbitration protocol is as follows:

1. An external device asserts the $\overline{BR}$ signal.
2. The processor asserts the $\overline{BG}$ signal to indicate that the bus will become available at the end of the current bus cycle.
3. The external device asserts the $\overline{BGACK}$ signal to indicate that it has assumed bus mastership.

$\overline{BR}$ may be issued any time during a bus cycle or between cycles. $\overline{BG}$ is asserted in response to $\overline{BR}$; it is usually asserted as soon as $\overline{BR}$ has been synchronized and recognized, except when the MC68020 has made an internal decision to execute a bus cycle. Then, the assertion of $\overline{BG}$ is deferred until the bus cycle has begun. Additionally, $\overline{BG}$ is not asserted until the end of a read-modify-write operation (when $\overline{RMC}$ is negated) in response to a $\overline{BR}$ signal. When the requesting device receives $\overline{BG}$ and more than one external device can be bus master, the requesting device should begin whatever arbitration is required. The external device asserts $\overline{BGACK}$ when it assumes bus mastership, and maintains $\overline{BGACK}$ during the entire bus cycle (or cycles) for which it is bus master. The following conditions must be met for an external device to assume mastership of the bus through the normal bus arbitration procedure:

- The external device must have received $\overline{BG}$ through the arbitration process.
- $\overline{AS}$ must be negated, indicating that no bus cycle is in progress, and the external device must ensure that all appropriate processor signals have been placed in the high-impedance state (by observing specification #7 in **Section 10 Electrical Specifications**).
- The termination signal ($\overline{DSACK1}/\overline{DSACK0}$) for the most recent cycle must have been negated, indicating that external devices are off the bus (optional, refer to **5.7.1.3 Bus Grant Acknowledge (MC68020)**).
- $\overline{BGACK}$ must be inactive, indicating that no other bus master has claimed ownership of the bus.

Figure 5-42 is a flowchart of MC68020 bus arbitration for a single device. Figure 5-43 is a timing diagram for the same operation. This technique allows processing of bus requests during data transfer cycles.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

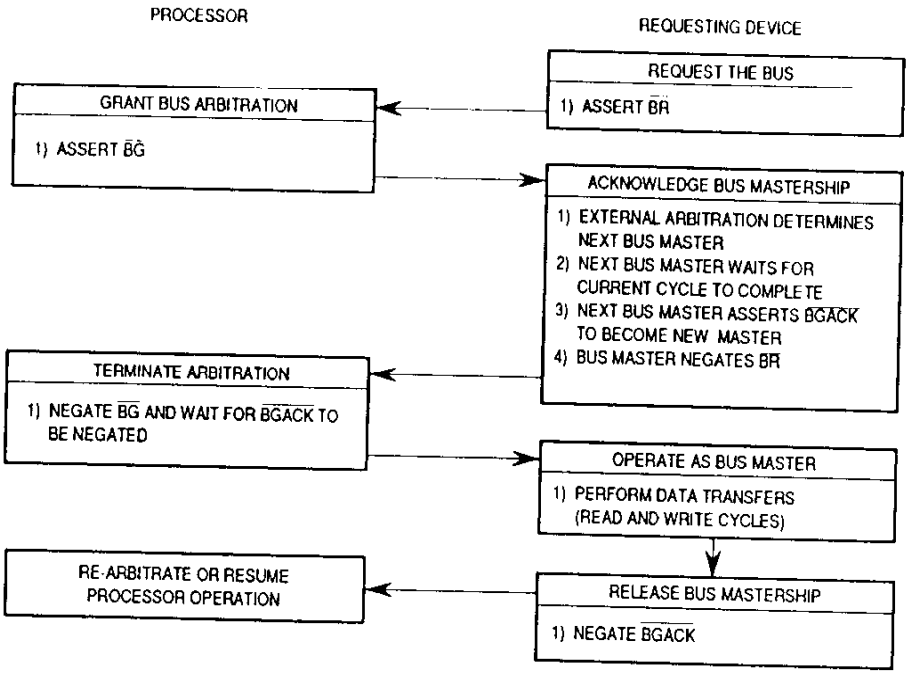


Figure 5-42. MC68020 Bus Arbitration Flowchart for Single Request

The timing diagram (see Figure 5-43) shows that $\overline{BR}$ is negated at the time that $\overline{BGACK}$ is asserted. This type of operation applies to a system consisting of the processor and one device capable of bus mastership. In a system having a number of devices capable of bus mastership, the $\overline{BR}$ line from each device can be wire-ORed to the processor. In such a system, more than one bus request can be asserted simultaneously.

The timing diagram in Figure 5-43 shows that $\overline{BG}$ is negated a few clock cycles after the transition of $\overline{BGACK}$. However, if bus requests are still pending after the negation of $\overline{BG}$, the processor asserts another $\overline{BG}$ within a few clock cycles after it was negated. This additional assertion of $\overline{BG}$ allows external arbitration circuitry to select the next bus master before the current bus master has finished with the bus. The following paragraphs provide additional information about the three steps in the arbitration process.

Bus arbitration requests are recognized during normal processing, $\overline{RESET}$ assertion, $\overline{HALT}$ assertion, and when the processor has halted due to a double bus fault.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

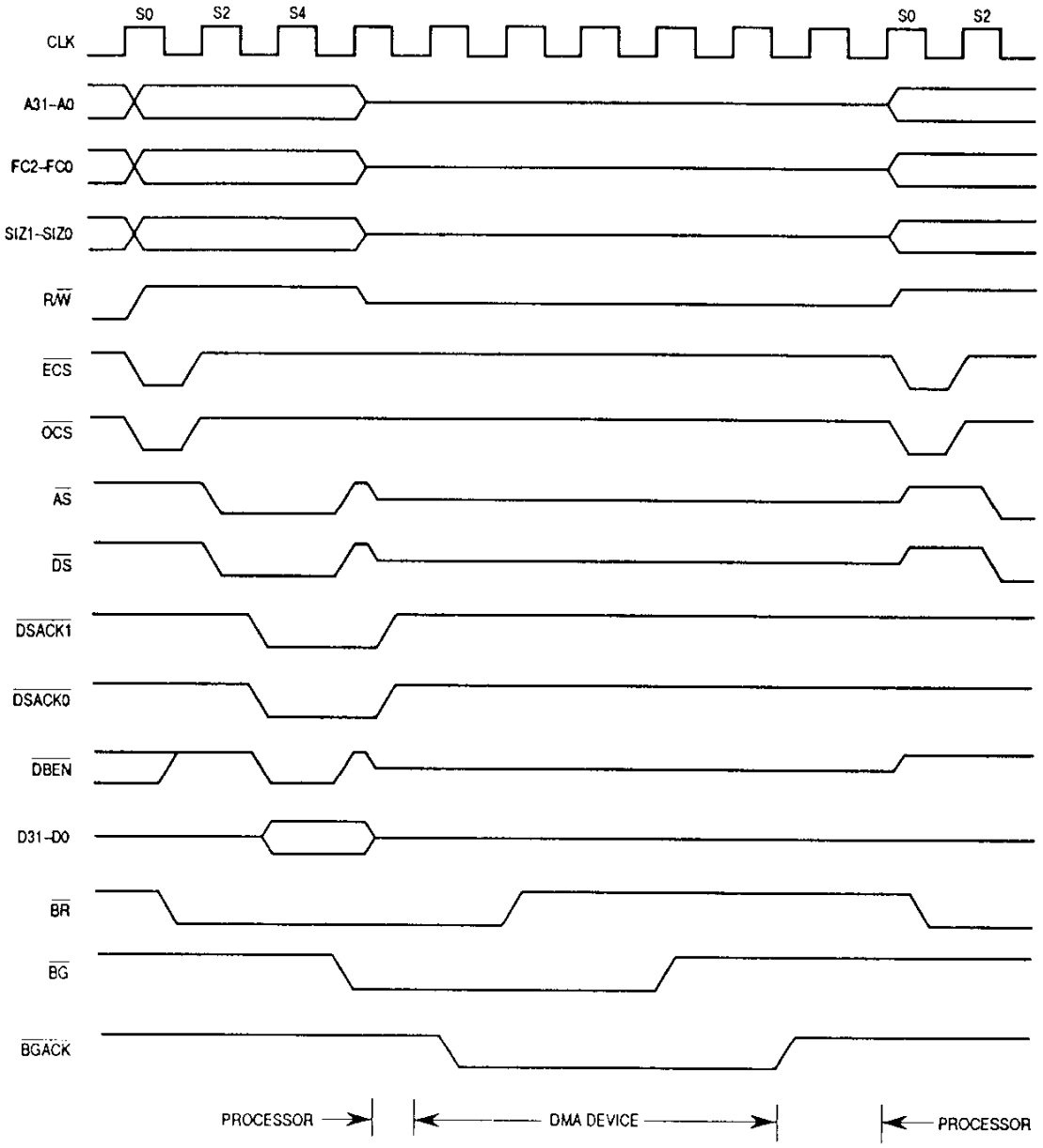


Figure 5-43. MC68020 Bus Arbitration Operation Timing for Single Request

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.7.1.1 BUS REQUEST (MC68020). External devices capable of becoming bus masters request the bus by asserting $\overline{BR}$. $\overline{BR}$ can be a wire-ORed signal (although it need not be constructed from open-collector devices) that indicates to the processor that some external device requires control of the bus. The processor is at a lower bus priority level than the external device and relinquishes the bus after it has completed the current bus cycle (if one has started).

If no $\overline{BGACK}$ is received while $\overline{BR}$ is asserted, the processor remains bus master once $\overline{BR}$ is negated. This prevents unnecessary interference with ordinary processing if the arbitration circuitry inadvertently responds to noise or if an external device determines that it no longer requires use of the bus before it has been granted mastership.

5.7.1.2 BUS GRANT (MC68020). The processor asserts $\overline{BG}$ as soon as possible after receipt of the bus request. $\overline{BG}$ assertion immediately follows internal synchronization except during a read-modify-write cycle or follows an internal decision to execute a bus cycle. During a read-modify-write cycle, the processor does not assert $\overline{BG}$ until the entire operation has completed. $\overline{RMC}$ is asserted to indicate that the bus is locked. In the case of an internal decision to execute another bus cycle, $\overline{BG}$ is deferred until the bus cycle has begun.

$\overline{BG}$ may be routed through a daisy-chained network or through a specific priority-encoded network. The processor allows any type of external arbitration that follows the protocol.

5.7.1.3 BUS GRANT ACKNOWLEDGE (MC68020). Upon receiving $\overline{BG}$, the requesting device waits until $\overline{AS}$, $\overline{DSACK1}/\overline{DSACK0}$, and $\overline{BGACK}$ are negated before asserting its own $\overline{BGACK}$. The negation of $\overline{AS}$ indicates that the previous master releases the bus after specification #7 (refer to **Section 10 Electrical Characteristics**). The negation of $\overline{DSACK1}/\overline{DSACK0}$ indicates that the previous slave has completed its cycle with the previous master. Note that in some applications, $\overline{DSACK1}/\overline{DSACK0}$ might not be used in this way.

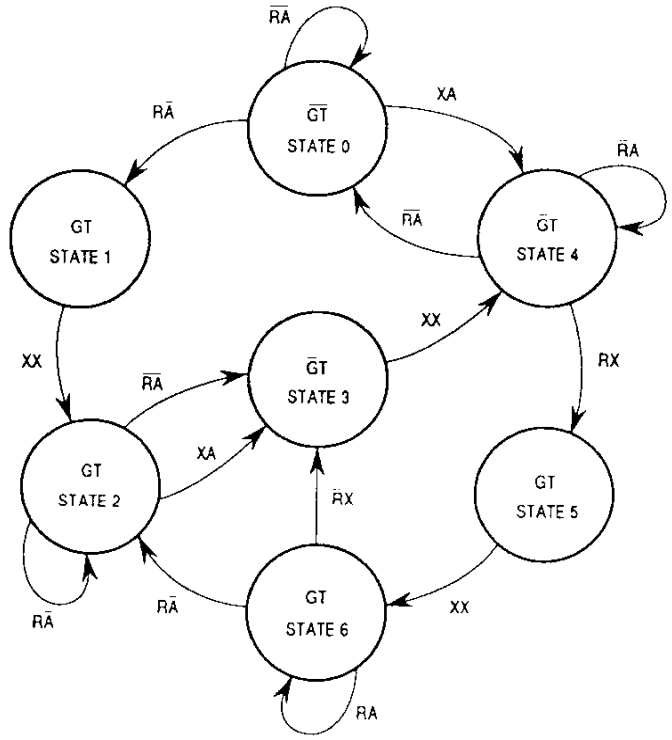
General-purpose devices are connected to be dependent only on $\overline{AS}$. When $\overline{BGACK}$ is asserted, the device is the bus master until it negates $\overline{BGACK}$. $\overline{BGACK}$ should not be negated until all bus cycles required by the alternate bus master have been completed. Bus mastership terminates at the negation of $\overline{BGACK}$. The $\overline{BR}$ from the granted device should be negated after $\overline{BGACK}$ is asserted. If another $\overline{BR}$ is still pending after the assertion of $\overline{BGACK}$, another $\overline{BG}$ is asserted within a few clocks of the negation of the the first $\overline{BG}$, as described in **5.7.1.4 Bus Arbitration Control (MC68020)**. Note that the processor does not perform any external bus cycles before it reasserts $\overline{BG}$ in this case.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.7.1.4 BUS ARBITRATION CONTROL (MC68020). The bus arbitration control unit in the MC68020 is implemented with a finite state machine. As discussed previously, all asynchronous inputs to the MC68020 are internally synchronized in a maximum of two cycles of the processor clock.

As shown in Figure 5-44, input signals labeled R and A are internally synchronized versions of the $\overline{BR}$ and $\overline{BGACK}$ signals, respectively. The $\overline{BG}$ output is labeled G, and the internal high-impedance control signal is labeled T. If T is true, the address, data, and control buses are placed in the high-impedance state after the next rising edge following the negation of $\overline{AS}$ and $\overline{RMC}$. All signals are shown in positive logic (active high), regardless of their true active voltage level.



R—BUS REQUEST
A—BUS GRANT ACKNOWLEDGE
G—BUS GRANT
T—THREE-STATE CONTROL TO BUS CONTROL LOGIC
X—DON'T CARE

NOTE: The $\overline{BG}$ output will not be asserted while $\overline{RMC}$ is asserted

Figure 5-44. MC68020 Bus Arbitration State Diagram

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

State changes occur on the next rising edge of the clock after the internal signal is recognized as valid. The $\overline{BG}$ signal transitions on the falling edge of the clock after a state is reached during which G changes. The bus control signals (controlled by T) are driven by the processor immediately following a state change when bus mastership is returned to the MC68020.

State 0, at the top center of the diagram, in which both G and T are negated, is the state of the bus arbiter while the processor is bus master. Request R and acknowledge A keep the arbiter in state 0 as long as they are both negated. When a request R is received, both grant G and signal T are asserted (in state 1 at the top left). The next clock causes a change to state 2, at the lower left, in which G and T are held. The bus arbiter remains in that state until acknowledge A is asserted or request R is negated. Once either occurs, the arbiter changes to the center state, state 3, and negates grant G. The next clock takes the arbiter to state 4, at the upper right, in which grant G remains negated and signal T remains asserted. With acknowledge A asserted, the arbiter remains in state 4 until A is negated or request R is again asserted. When A is negated, the arbiter returns to the original state, state 0, and negates signal T. This sequence of states follows the normal sequence of signals for relinquishing the bus to an external bus master. Other states apply to other possible sequences of combinations of R and A.

The MC68020 does not allow arbitration of the external bus during the read-modify-write sequence. For the duration of this sequence, the MC68020 ignores the $\overline{BR}$ input. If mastership of the MC68020 bus is required during a read-modify-write operation, $\overline{BERR}$ must be used to abort the read-modify-write sequence. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 5-45.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

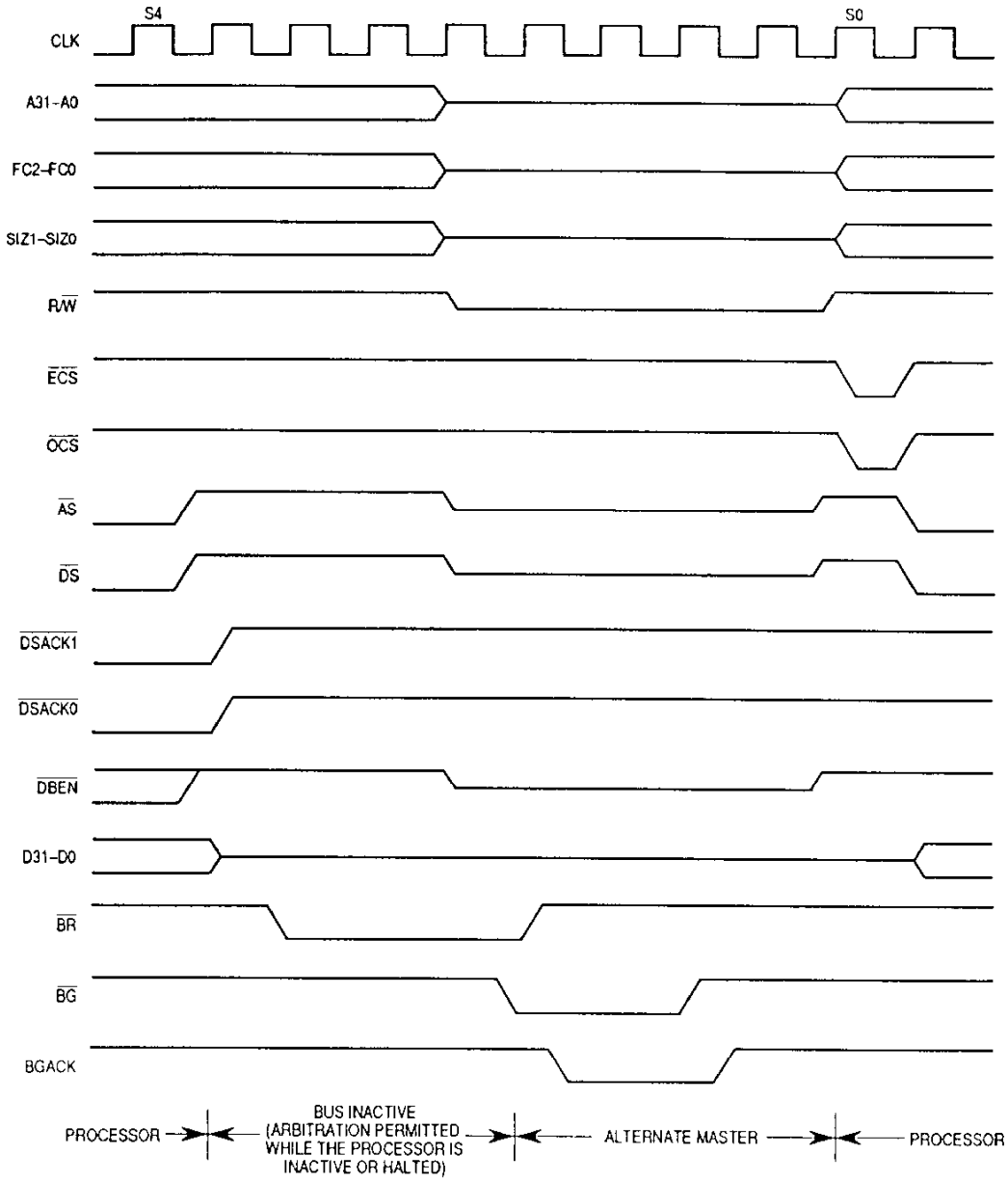


Figure 5-45. MC68020 Bus Arbitration Operation Timing—Bus Inactive

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

5.7.2 MC68EC020 Bus Arbitration

The sequence of the MC68EC020 bus arbitration protocol is as follows.

1. An external device asserts the $\overline{BR}$ signal.
2. The processor asserts the $\overline{BG}$ signal to indicate that the bus will become available at the end of the current bus cycle.
3. The external device asserts the $\overline{BR}$ signal throughout its bus mastership.

$\overline{BR}$ may be issued any time during a bus cycle or between cycles. $\overline{BG}$ is asserted in response to $\overline{BR}$; it is usually asserted as soon as $\overline{BR}$ has been synchronized and recognized, except when the MC68020 has made an internal decision to execute a bus cycle. Then, the assertion of $\overline{BG}$ is deferred until the bus cycle has begun. Additionally, $\overline{BG}$ is not asserted until the end of a read-modify-write operation (when $\overline{RMC}$ is negated) in response to a $\overline{BR}$ signal. When the requesting device receives $\overline{BG}$ and more than one external device can be bus master, the requesting device should begin whatever arbitration is required. The external device continues to assert $\overline{BR}$ when it assumes bus mastership, and maintains $\overline{BR}$ during the entire bus cycle (or cycles) for which it is bus master. The following conditions must be met for an external device to assume mastership of the bus through the normal bus arbitration procedure:

- The external device must have received $\overline{BG}$ through the arbitration process.
- $\overline{AS}$ must be negated, indicating that no bus cycle is in progress, and the external device must ensure that all appropriate processor signals have been placed in the high-impedance state (by observing specification #7 in **Section 10 Electrical Specifications**).
- The termination signal ($\overline{DSACK1}/\overline{DSACK0}$) for the most recent cycle must have been negated, indicating that external devices are off the bus.
- No other bus master has claimed ownership of the bus.

Figure 5-46 is a flowchart of MC68EC020 bus arbitration for a single device. Figure 5-47 is a timing diagram for the same operation. This technique allows processing of bus requests during data transfer cycles.

Bus arbitration requests are recognized during normal processing, $\overline{RESET}$ assertion, $\overline{HALT}$ assertion, and when the processor has halted due to a double bus fault.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

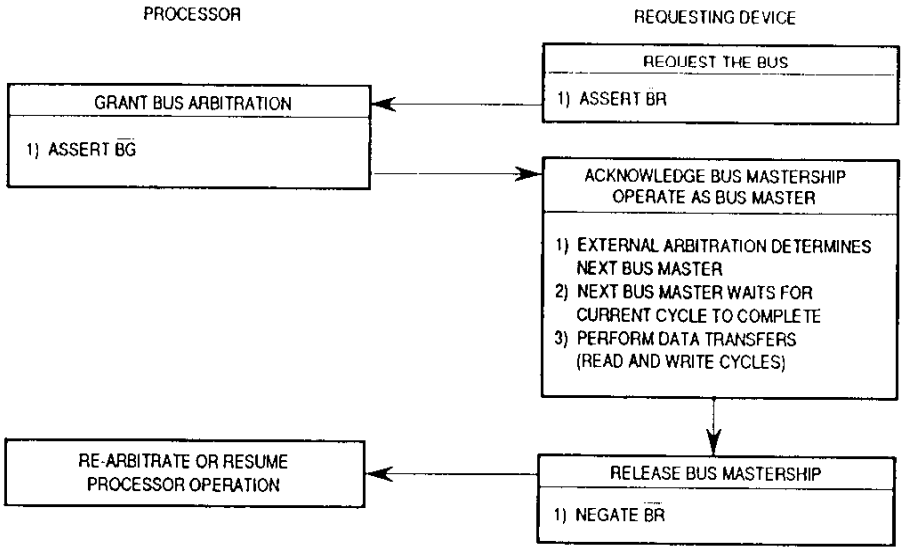


Figure 5-46. MC68EC020 Bus Arbitration Flowchart for Single Request

5.7.2.1 BUS REQUEST (MC68EC020). External devices capable of becoming bus masters request the bus by asserting $\overline{BR}$. $\overline{BR}$ can be a wire-ORed signal (although it need not be constructed from open-collector devices) that indicates to the processor that some external device requires control of the bus. The processor is at a lower bus priority level than the external device and relinquishes the bus after it has completed the current bus cycle (if one has started). $\overline{BR}$ remains asserted throughout the external device's bus mastership.

5.7.2.2 BUS GRANT (MC68EC020). The processor asserts $\overline{BG}$ as soon as possible after receipt of the bus request. $\overline{BG}$ assertion immediately follows internal synchronization except during a read-modify-write cycle or follows an internal decision to execute a bus cycle. During a read-modify-write cycle, the processor does not assert $\overline{BG}$ until the entire operation has completed. $\overline{RMC}$ is asserted to indicate that the bus is locked. In the case of an internal decision to execute another bus cycle, $\overline{BG}$ is deferred until the bus cycle has begun.

$\overline{BG}$ may be routed through a daisy-chained network or through a specific priority-encoded network. The processor allows any type of external arbitration that follows the protocol.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

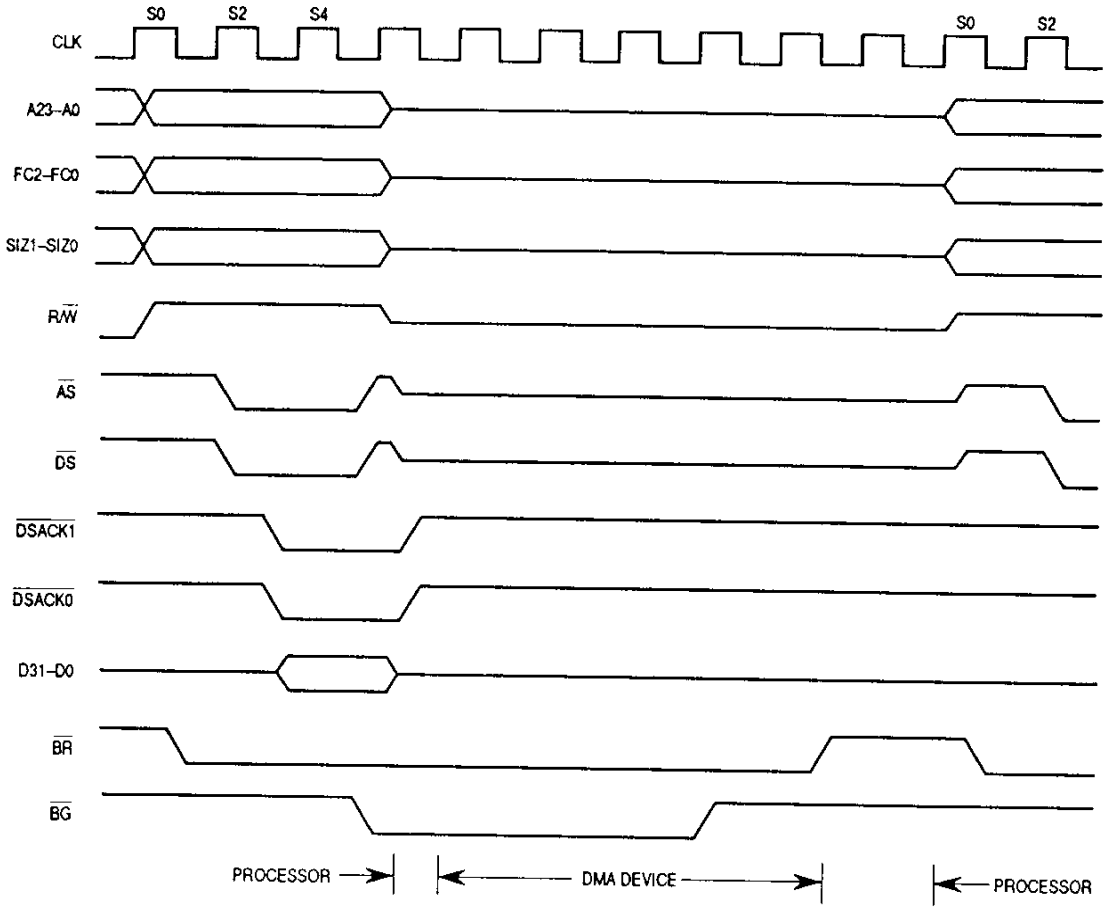


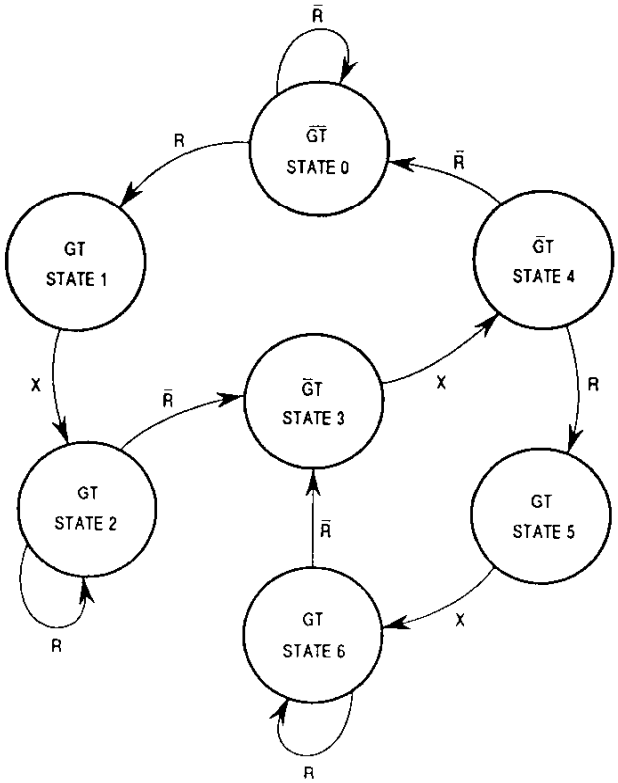
Figure 5-47. MC68EC020 Bus Arbitration Operation Timing for Single Request

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.7.2.3 BUS ARBITRATION CONTROL (MC68EC020). The bus arbitration control unit in the MC68EC020 is implemented with a finite state machine. As discussed previously, all asynchronous inputs to the MC68EC020 are internally synchronized in a maximum of two cycles of the processor clock.

As shown in Figure 5-48, the input signal labeled R is an internally synchronized version of the $\overline{BR}$ signal. The $\overline{BG}$ output is labeled G, and the internal high-impedance control signal is labeled T. If T is true, the address, data, and control buses are placed in the high-impedance state after the next rising edge following the negation of $\overline{AS}$ and $\overline{RMC}$. All signals are shown in positive logic (active high), regardless of their true active voltage level.



R—BUS REQUEST
G—BUS GRANT
T—THREE-STATE CONTROL TO BUS CONTROL LOGIC
X—DON'T CARE

Figure 5-48. MC68EC020 Bus Arbitration State Diagram

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

State changes occur on the next rising edge of the clock after the internal signal is recognized as valid. The $\overline{BG}$ signal transitions on the falling edge of the clock after a state is reached during which G changes. The bus control signals (controlled by T) are driven by the processor immediately following a state change when bus mastership is returned to the MC68EC020.

State 0, at the top center of the diagram, in which both G and T are negated, is the state of the bus arbiter while the processor is bus master. Request R keeps the arbiter in state 0 as long as it is negated. When a request R is received, both grant G and signal T are asserted (in state 1 at the top left). The next clock causes a change to state 2, at the lower left, in which G and T are held. The bus arbiter remains in that state until request R is negated. Then the arbiter changes to the center state, state 3, and negates grant G. The next clock takes the arbiter to state 4, at the upper right, in which grant G remains negated and signal T remains asserted. The arbiter returns to the original state, state 0, and negates signal T. This sequence of states follows the normal sequence of signals for relinquishing the bus to an external bus master. Other states apply to other possible sequences of R.

The MC68EC020 does not allow arbitration of the external bus during the read-modify-write sequence. For the duration of this sequence, the MC68EC020 ignores the $\overline{BR}$ input. If mastership of the MC68EC020 bus is required during a read-modify-write operation, $\overline{BERR}$ must be used to abort the read-modify-write sequence. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 5-49.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

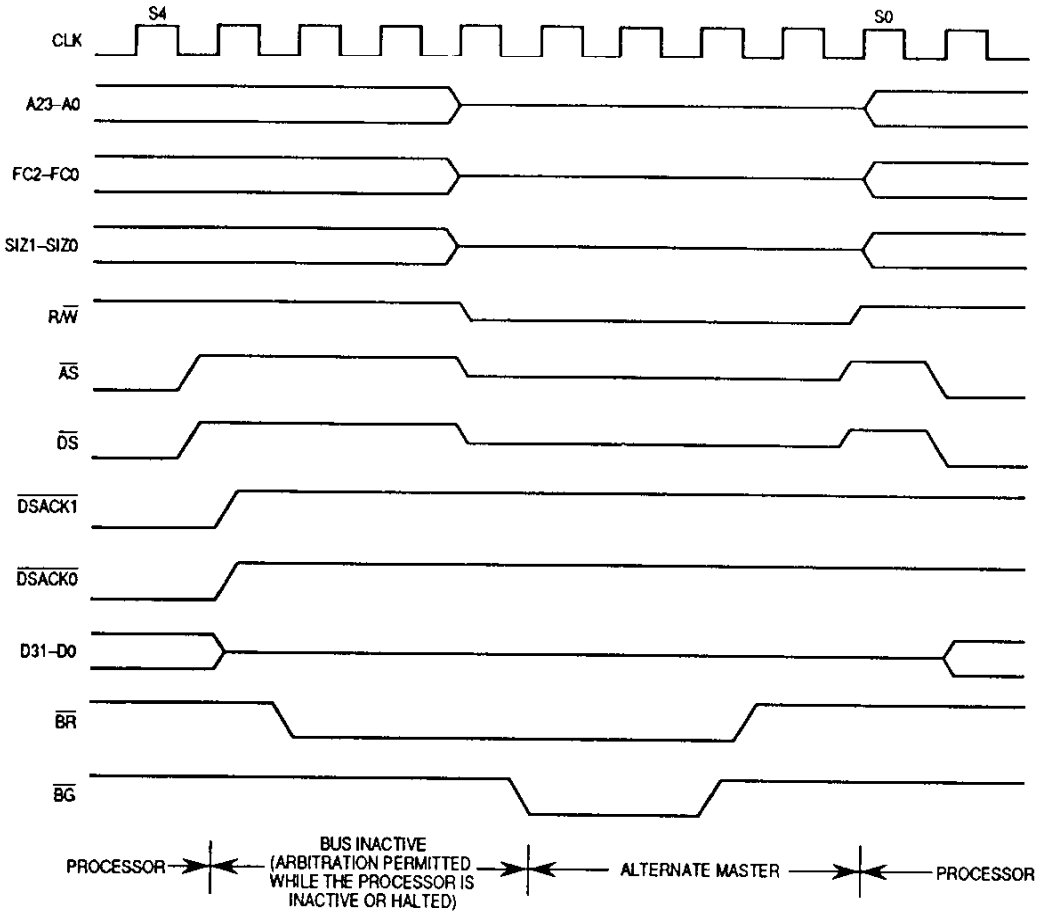


Figure 5-49. MC68EC020 Bus Arbitration Operation Timing—Bus Inactive

The existing three-wire arbitration design ($\overline{BR}$, $\overline{BG}$, and $\overline{BGACK}$) of some peripherals can be converted to the MC68EC020 two-wire arbitration with the addition of an AND gate. Figure 5-50 shows the combination of $\overline{BR}$ and $\overline{BGACK}$ for a three-wire arbitration system to $\overline{BR}$ of the MC68EC020 or $\overline{BR}$ and $\overline{BG}$ from an MC68EC020 to $\overline{BG}$ for a three-wire arbitration system. The speed of the AND gate must be faster than the time between the assertion of $\overline{BGACK}$ and the negation of $\overline{BR}$ by the alternate bus master. Figure 5-50 assumes the alternate bus master does not assume bus mastership until the MC68EC020 $\overline{AS}$ is negated and MC68EC020 $\overline{BG}$ is asserted.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

An example of MC68EC020 bus arbitration to a DMA device that supports three-wire bus arbitration is described in **Appendix A Interfacing an MC68EC020 to a DMA Device That Supports a Three-Wire Bus Arbitration Protocol**.

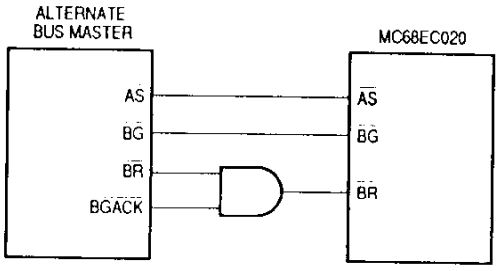


Figure 5-50. Interface for Three-Wire to Two-Wire Bus Arbitration

5.8 RESET OPERATION

$\overline{\text{RESET}}$ is a bidirectional signal with which an external device resets the system or the processor resets external devices. When power is applied to the system, external circuitry should assert $\overline{\text{RESET}}$ for a minimum of 520 clocks after V_{CC} and clock timing have stabilized and are within specification limits. Figure 5-51 is a timing diagram of the power-up reset operation, showing the relationships between $\overline{\text{RESET}}$, V_{CC}, and bus signals. The clock signal is required to be stable by the time V_{CC} reaches the minimum operating specification. During the reset period, the entire bus three-states (except for non-three-statable signals, which are driven to their inactive state). Once $\overline{\text{RESET}}$ negates, all control signals are negated, the data bus is in read mode, and the address bus is driven. After this, the first bus cycle for reset exception processing begins.

The external $\overline{\text{RESET}}$ signal resets the processor and the entire system. Except for the initial reset, $\overline{\text{RESET}}$ should be asserted for at least 520 clock periods to ensure that the processor resets. Asserting $\overline{\text{RESET}}$ for 10 clock periods is sufficient for resetting the processor logic; the additional clock periods prevent a RESET instruction from overlapping the external $\overline{\text{RESET}}$ signal.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

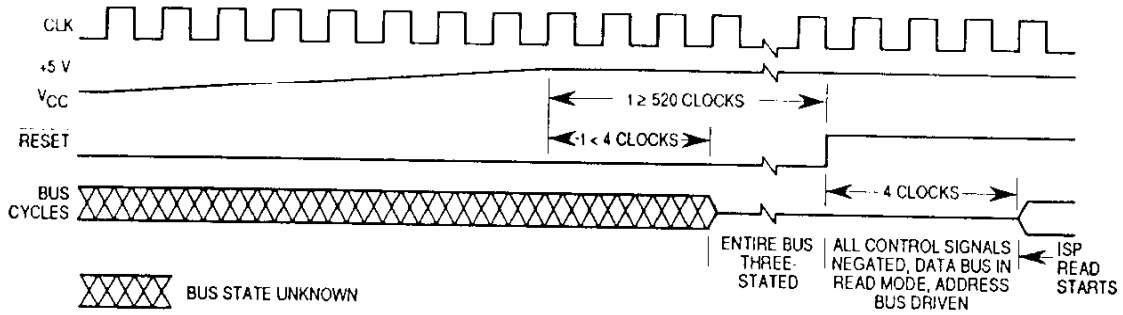


Figure 5-51. Initial Reset Operation Timing

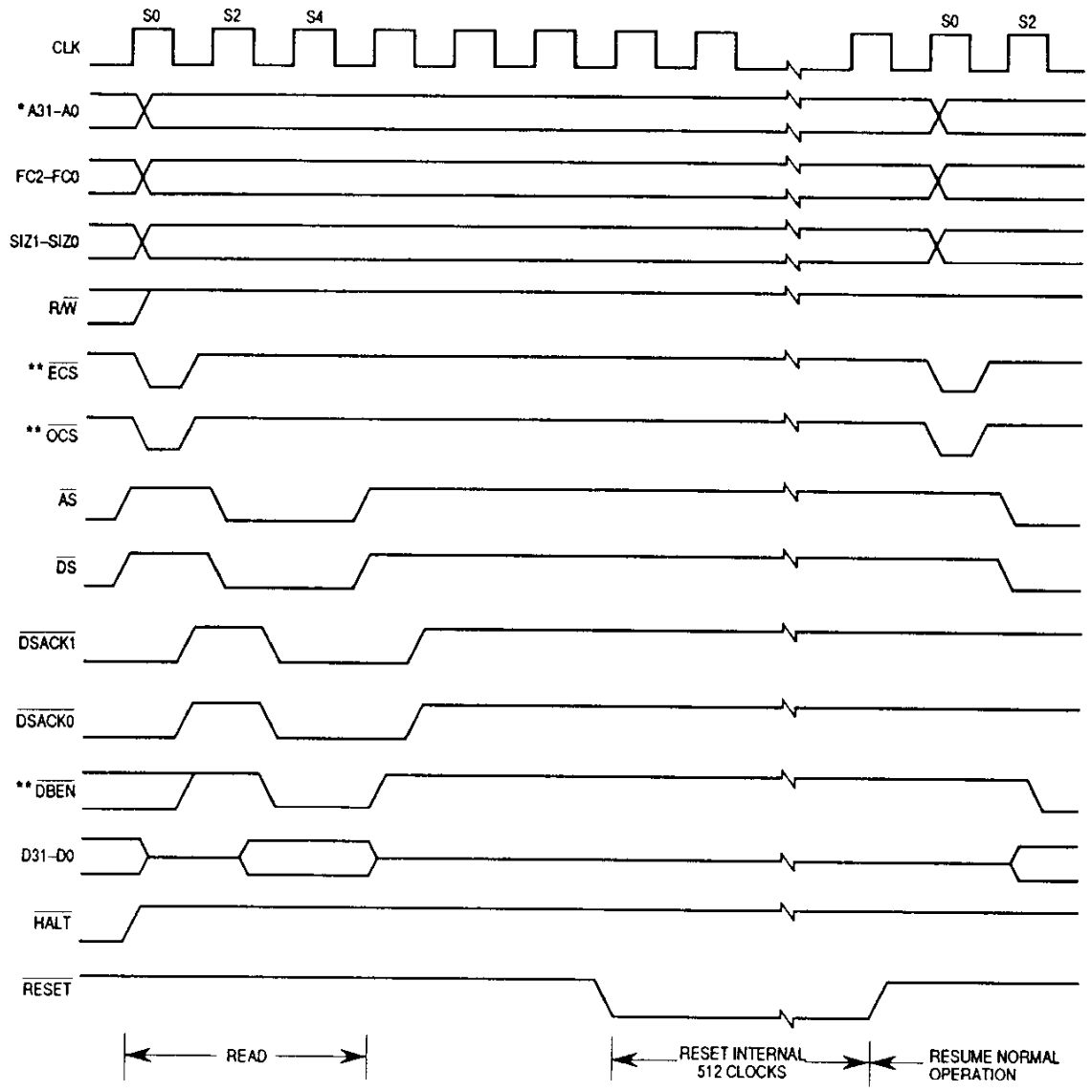
Resetting the processor causes any bus cycle in progress to terminate as if $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ or $\overline{\text{BERR}}$ had been asserted. In addition, the processor initializes registers appropriately for a reset exception. Exception processing for a reset operation is described in **Section 6 Exception Processing**.

When a RESET instruction is executed, the processor drives the $\overline{\text{RESET}}$ signal for 512 clock 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 $\overline{\text{RESET}}$ signal are reset at the completion of the RESET instruction. An external $\overline{\text{RESET}}$ signal that is asserted to the processor during execution of a RESET instruction must extend beyond the reset period of the instruction by at least eight clock cycles to reset the processor. Figure 5-52 shows the timing information for the RESET instruction.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.



* For the MC68EC020, A23–A0.
** This signal does not apply to the MC68EC020.

Figure 5-52. RESET Instruction Timing

For More Information On This Product,
Go to: www.freescale.com

SECTION 6
EXCEPTION PROCESSING

Exception processing is defined as the activities performed by the processor in preparing to execute a handler routine for any condition that causes an exception. In particular, exception processing does not include execution of the handler routine itself. An introduction to exception processing, as one of the processing states of the MC68020/EC020, is given in **Section 2 Processing States**.

This section describes exception processing in detail, describing the processing for each type of exception. It describes the return from an exception and bus fault recovery. This section also describes the formats of the exception stack frames. For more detail on protocol violation and coprocessor-related exceptions, refer to **Section 7 Coprocessor Interface Description**. Also, for more detail on exceptions defined for floating-point coprocessors, refer to MC68881UM/AD, *MC68881/MC68882 Floating-Point Coprocessor User's Manual*.

6.1 EXCEPTION PROCESSING SEQUENCE

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. Nonetheless, all addresses and offsets from the stack pointer are guaranteed to be as described.

The first step of exception processing involves the SR. The processor makes an internal copy of the SR, then sets the S-bit in the SR, changing to the supervisor privilege level. Next, the processor inhibits tracing of the exception handler by clearing the T1 and T0 bits in the SR. For the reset and interrupt exceptions, the processor also updates the interrupt priority mask (bits 10–8 of the SR).

In the second step, the processor determines the vector number of the exception. For interrupts, the processor performs an interrupt acknowledge cycle (a read from the CPU address space type 1111; see Figures 5-32 and 5-33) to obtain the vector number. For coprocessor-detected exceptions, the vector number is included in the coprocessor exception primitive response. (Refer to **Section 7 Coprocessor Interface Description** for a complete discussion of coprocessor exceptions.) For all other exceptions, internal logic provides the vector number. This vector number is used in the last step to calculate the address of the exception vector. Throughout this section, vector numbers are given in decimal notation.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

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

The last step initiates execution of the exception handler. The processor multiplies the vector number by four to determine the exception vector offset. The processor then adds the offset to the value stored in the VBR to obtain the memory address of the exception vector. Next, the processor loads the PC (and the ISP for the reset exception) from the exception vector table in memory. After prefetching the first three words to fill the instruction pipe, the processor resumes normal processing at the address in the PC. Table 6-1 contains a description of all the exception vector offsets defined for the MC68020/EC020.

As shown in Table 6-1, the first 64 vectors are defined by Motorola, and 192 vectors are reserved for interrupt vectors defined by the user. However, external devices may use vectors reserved for internal purposes at the discretion of the system designer.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 6-1. Exception Vector Assignments

Vector Number	Vector Offset		Assignment
	Hex	Space	
0	000	SP	Reset Initial Interrupt Stack Pointer
1	004	SP	Reset Initial Program Counter
2	008	SD	Bus Error
3	00C	SD	Address Error
4	010	SD	Illegal Instruction
5	014	SD	Zero Divide
6	018	SD	CHK, CHK2 Instruction
7	01C	SD	cpTRAPcc, TRAPcc, TRAPV Instructions
8	020	SD	Privilege Violation
9	024	SD	Trace
10	028	SD	Line 1010 Emulator
11	02C	SD	Line 1111 Emulator
12	030	SD	(Unassigned, Reserved)
13	034	SD	Coprocessor Protocol Violation
14	038	SD	Format Error
15	03C	SD	Uninitialized Interrupt
16–23	040	SD	Unassigned, Reserved
	05C	SD	
24	060	SD	Spurious Interrupt
25	064	SD	Level 1 Interrupt Autovector
26	068	SD	Level 2 Interrupt Autovector
27	06C	SD	Level 3 Interrupt Autovector
28	070	SD	Level 4 Interrupt Autovector
29	074	SD	Level 5 Interrupt Autovector
30	078	SD	Level 6 Interrupt Autovector
31	07C	SD	Level 7 Interrupt Autovector
32–47	080	SD	TRAP #0–15 Instruction Vectors
	0BC	SD	
48	0C0	SD	FPCP Branch or Set on Unordered Condition
49	0C4	SD	FPCP Inexact Result
50	0C8	SD	FPCP Divide by Zero
51	0CC	SD	FPCP Underflow
52	0D0	SD	FPCP Operand Error
53	0D4	SD	FPCP Overflow
54	0D8	SD	FPCP Signaling NAN
55	0DC	SD	Unassigned, Reserved
56	0E0	SD	PMMU Configuration
57	0E4	SD	PMMU Illegal Operation
58	0E8	SD	PMMU Access Level Violation
59–63	0EC	SD	Unassigned, Reserved
	0FC	SD	
64–255	100	SD	User-Defined Vectors (192)
	3FC	SD	

SP—Supervisor Program Space
SD—Supervisor Data Space

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.1.1 Reset Exception

Assertion of the $\overline{\text{RESET}}$ signal by external hardware causes a reset exception. For details on the requirements for the assertion of $\overline{\text{RESET}}$, refer to **Section 5 Bus Operation**.

The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. When a reset exception is recognized, it aborts any processing in progress and that processing cannot be recovered. Figure 6-1 is a flowchart of the reset exception, which performs the following operations:

1. Clears the T1 and T0 bits in the SR to disable tracing.
2. Places the processor in the interrupt mode of the supervisor privilege level by setting the S-bit and clearing the M-bit in the SR.
3. Sets the I2–I0 bits in the SR to the highest priority level (level 7).
4. Initializes the VBR to zero ($00000000).
5. Clears the E and F bits in the CACR.
6. Invalidates all entries in the instruction cache.
7. Generates a vector number to reference the reset exception vector (two long words) at offset zero in the supervisor program address space.
8. Loads the first long word of the reset exception vector into the interrupt stack pointer.
9. Loads the second long word of the reset exception vector into the PC.

After the initial instruction prefetches, program execution begins at the address in the PC. The reset exception does not save the value of either the PC or the SR.

As described in **Section 5 Bus Operation**, if a bus error or address error occurs during the exception processing sequence for a reset, a double bus fault occurs. The processor halts and asserts the $\overline{\text{HALT}}$ signal to indicate the halted condition.

Execution of the RESET instruction does not cause a reset exception, nor does it affect any internal registers, but it does cause the MC68020/EC020 to assert the $\overline{\text{RESET}}$ signal, resetting all external devices.

6.1.2 Bus Error Exception

A bus error exception occurs when external logic aborts a bus cycle by asserting the $\overline{\text{BERR}}$ signal. If the aborted bus cycle is a data access, the processor immediately begins exception processing. If the aborted bus cycle is an instruction prefetch, the processor may delay taking the exception until it attempts to use the prefetched information.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

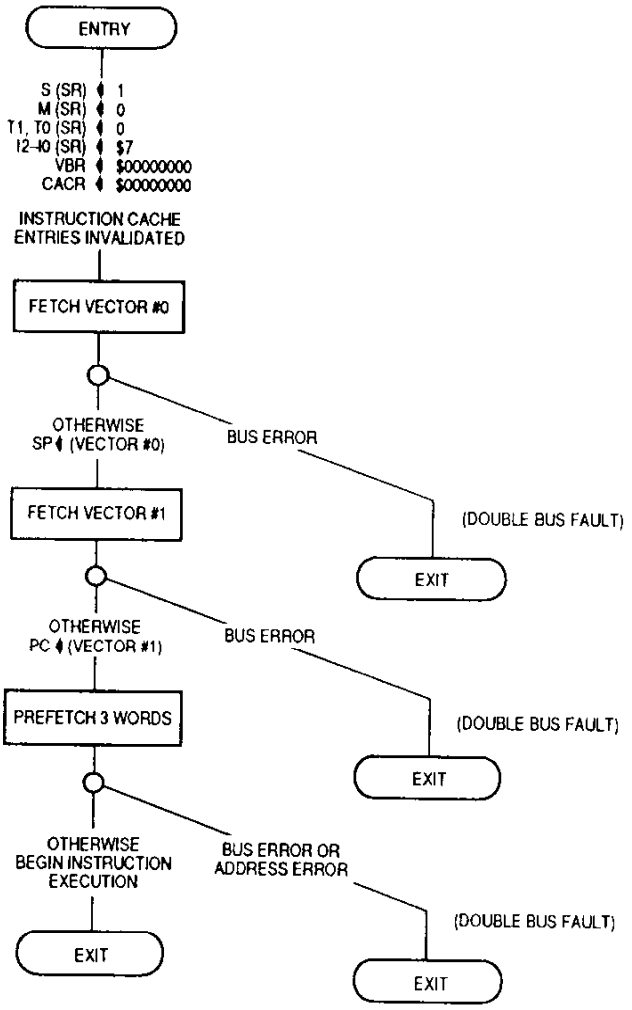


Figure 6-1. Reset Operation Flowchart

The processor begins exception processing for a bus error by making an internal copy of the current SR. The processor then enters the supervisor privilege level (by setting the S-bit in the SR) and clears the T1 and T0 bits in the SR. The processor generates exception vector number 2 for the bus error vector. It saves the vector offset, PC, and the internal copy of the SR on the active supervisor stack. The saved PC value is the logical address of the instruction that was executing at the time the fault was detected. This is not necessarily the instruction that initiated the bus cycle since the processor overlaps

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

execution of instructions. The processor also saves the contents of some of its internal registers. The information saved on the stack is sufficient to identify the cause of the bus fault and recover from the error.

For efficiency, the MC68020/EC020 uses two different bus error stack frame formats. When the bus error exception is taken at an instruction boundary, less information is required to recover from the error, and the processor builds the short bus fault stack frame as shown in Table 6-5. When the exception is taken during the execution of an instruction, the processor must save its entire state for recovery and uses the long bus fault stack frame shown in Table 6-5. The format code in the stack frame distinguishes the two stack frame formats. Stack frame formats are described in detail in **6.4 Exception Stack Frame Formats**.

If a bus error occurs during the exception processing for a bus error, address error, or reset or while the processor is loading internal state information from the stack during the execution of an RTE instruction, a double bus fault occurs and the processor enters the halted state. In this case, the processor does not attempt to alter the current state of memory. Only an external $\overline{\text{RESET}}$ can restart a processor halted by a double bus fault.

6.1.3 Address Error Exception

An address error exception occurs when the processor attempts to prefetch an instruction from an odd address. This exception is similar to a bus error exception but is internally initiated. A bus cycle is not executed, and the processor begins exception processing immediately. After exception processing commences, the sequence is the same as that for bus error exceptions described in the preceding paragraphs, except that the vector number is 3 and the vector offset in the stack frame refers to the address error vector. Either a short or long bus fault stack frame may be generated. If an address error occurs during the exception processing for a bus error, address error, or reset, a double bus fault occurs.

6.1.4 Instruction Trap Exception

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

The DIVS and DIVU instructions force exceptions if a division operation is attempted with a divisor of zero.

When a trap exception occurs, the processor copies the SR internally, enters the supervisor privilege level (by setting the S-bit in the SR), and clears the T1 and T0 bits in the SR. If tracing is enabled for the instruction that caused the trap, a trace exception is taken after the RTE instruction from the trap handler is executed, and the trace corresponds to the trap instruction; the trap handler routine is not traced. The processor generates a vector number according to the instruction being executed; for the TRAP

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

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, a pointer to the instruction that caused the trap is also saved. Instruction execution resumes at the address in the exception vector after the required instruction prefetches.

6.1.5 Illegal Instruction and Unimplemented Instruction Exceptions

An illegal instruction is an instruction that contains any bit pattern in its first word that does not correspond to the bit pattern of the first word of a valid MC68020/EC020 instruction or a MOVEC instruction with an undefined register specification field in the first extension word. An illegal instruction exception corresponds to vector number 4 and occurs when the processor attempts to execute an illegal instruction.

An illegal instruction exception is also taken if a breakpoint acknowledge bus cycle (see **Section 5 Bus Operation**) is terminated with the assertion of the $\overline{\text{BERR}}$ signal. This implies that the external circuitry did not supply an instruction word to replace the BKPT instruction word in the instruction pipe.

Instruction word patterns with bits 15–12 = 1010 are referred to as unimplemented instructions with A-line opcodes. When the processor attempts to execute an unimplemented instruction with an A-line opcode, an exception is generated with vector number 10, permitting efficient emulation of unimplemented instructions.

Instructions that have word patterns with bits 15–12 = 1111, bits 11–9 = 000, and defined word patterns for subsequent words, are legal PMMU instructions. Instructions that have bits 15–12 of the first words = 1111, bits 11–9 = 000, and undefined patterns in the subsequent words, are treated as unimplemented instructions with F-line opcodes when execution is attempted in the supervisor privilege level. When execution of the same instruction is attempted in the user privilege level, a privilege violation exception is taken. The exception vector number for an unimplemented instruction with an F-line opcode is 11.

The word patterns with bits 15–12 = 1111 and bits 11–9 ≠ 000 are used for coprocessor instructions. When the processor identifies a coprocessor instruction, it runs a bus cycle referencing CPU space type $2 (refer to **Section 2 Processing States**) and addressing one of eight coprocessors (0–7, according to bits 11–9). If the addressed coprocessor is not included in the system and the cycle terminates with the assertion of $\overline{\text{BERR}}$, the instruction takes an unimplemented instruction (F-line opcode) exception. The system can emulate the functions of the coprocessor with an F-line exception handler. Refer to **Section 7 Coprocessor Interface Description** for more details.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

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 privilege level (by setting the S bit in the SR), and clears the T1 and T0 bits in the SR, disabling further tracing. The processor generates the vector number, either 4, 10, or 11, according to the exception type. The illegal or unimplemented instruction vector offset, current PC, and copy of the SR are saved on the supervisor stack, with the saved value of the PC being the address of the illegal or unimplemented instruction. Instruction execution resumes at the address contained in the exception vector. It is the responsibility of the handling routine to adjust the stacked PC if the instruction is emulated in software or is to be skipped on return from the handler.

6.1.6 Privilege Violation Exception

To provide system security, the following instructions are privileged:

ANDI to SR
EORI to SR
cpRESTORE
cpSAVE
MOVE from SR
MOVE to SR
MOVE USP
MOVEC
MOVES
ORI to SR
RESET
RTE
STOP

An attempt to execute one of the privileged instructions while at the user privilege level causes a privilege violation exception. Also, a privilege violation exception occurs if a coprocessor requests a privilege check and the processor is at the user level.

Exception processing for privilege violations is similar to that for illegal instructions. When the processor identifies a privilege violation, it begins exception processing before executing the instruction. The processor copies the SR, enters the supervisor privilege level by setting the S-bit in the SR, and clears the T1 and T0 bits in the SR. The processor generates vector number 8, the privilege violation exception vector, and 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.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.1.7 Trace Exception

To aid in program development, the M68000 processors include an instruction-by-instruction tracing capability. The MC68020/EC020 can be programmed to trace all instructions or only instructions that change program flow. In the trace mode, an instruction generates a trace exception after it completes execution, allowing a debugger program to monitor execution of a program.

The T1 and T0 bits in the supervisor portion of the SR control tracing. The state of these bits when an instruction begins execution determines whether the instruction generates a trace exception after the instruction completes. Clearing both the T1 and T0 bits disables tracing, and instruction execution proceeds normally. Clearing the T1 bit and setting the T0 bit causes an instruction that forces a change of flow to take a trace exception. Instructions that increment the PC normally do not take the trace exception. Instructions that are traced in this mode include all branches, jumps, instruction traps, returns, and coprocessor instructions that modify the PC flow. This mode also includes SR manipulations because the processor must re-prefetch instruction words to fill the pipe again any time an instruction that can modify the SR is executed. The execution of the BKPT instruction causes a change of flow if the opcode replacing the BKPT is an instruction that causes a change of flow (i.e., a jump, branch, etc.). Setting the T1 bit and clearing the T0 bit causes the execution of all instructions to force trace exceptions. Table 6-2 shows the trace mode selected by each combination of T1 and T0.

Table 6-2. Tracing Control

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

In general terms, a trace exception is an extension to the function of any traced instruction—i.e., the execution of a traced instruction is not complete until completion of trace exception processing. If an instruction does not complete due to a bus error or address error exception, trace exception processing is deferred until after the execution of the suspended instruction is resumed, and the instruction execution completes normally. If an interrupt is pending at the completion of an instruction, the trace exception processing occurs before the interrupt exception processing starts. If an instruction forces an exception as part of its normal execution, the forced exception processing occurs before the trace exception is processed. See **6.1.11 Multiple Exceptions** for a more complete discussion of exception priorities.

When tracing is enabled and the processor attempts to execute an illegal or unimplemented instruction, that instruction does not cause a trace exception since it is not executed. This is of particular importance to an instruction emulation routine that performs the instruction function, adjusts the stacked PC to skip the unimplemented instruction, and returns. Before returning, the T1 and T0 bits of the SR on the stack should be checked. If

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

tracing is enabled, the trace exception processing should also be emulated for the trace exception handler to account for the emulated instruction.

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

The STOP instruction does not perform its function when it is traced. A STOP instruction that begins execution with T1, T0 = 10 forces a trace exception after it loads the SR. Upon return from the trace handler routine, execution continues with the instruction following the STOP instruction, and the processor never enters the stopped condition.

6.1.8 Format Error Exception

Just as the processor checks that prefetched instructions are valid, the processor (with the aid of a coprocessor, if needed) also performs some checks of data values for control operations, including the type and option fields of the descriptor for CALLM, the coprocessor state frame format word for a cpRESTORE instruction, and the stack frame format for an RTE or an RTM instruction.

The RTE instruction checks the validity of the stack format code. For long bus fault format frames, the RTE instruction also compares the internal version number of the processor to that contained in the frame at memory location SP + 54 (SP + $36). This check ensures that the processor can correctly interpret internal state information from the stack frame.

The CALLM and RTM both check the values in the option and type fields in the module descriptor and module stack frame, respectively. If these fields do not contain proper values or if an illegal access rights change request is detected by an external memory management unit, then an illegal call or return is being requested and is not executed. Refer to **Section 9 Applications Information** for more information on the module call/return mechanism.

The cpRESTORE instruction passes the format word of the coprocessor state frame to the coprocessor for validation. If the coprocessor does not recognize the format value, it signals the MC68020/EC020 to take a format error exception. Refer to **Section 7 Coprocessor Interface Description** for details of coprocessor-related exceptions.

If any of the checks previously described determine that the format of the stacked data is improper, the instruction generates a format error exception. This exception saves a short bus fault 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.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.1.9 Interrupt Exceptions

When a peripheral device requires the services of the MC68020/EC020 or is ready to send information that the processor requires, it may signal the processor to take an interrupt exception. The interrupt exception transfers control to a routine that responds appropriately.

The peripheral device uses the $\overline{IPL2}$–$\overline{IPL0}$ signals to signal an interrupt condition to the processor and to specify the priority of that condition. These three signals encode a value of zero through seven ($\overline{IPL0}$ is the least significant bit). When $\overline{IPL2}$–$\overline{IPL0}$ are all negated, the interrupt request level is zero. $\overline{IPL2}$–$\overline{IPL0}$ values 1–7 specify one of seven levels of prioritized interrupts; level 7 has the highest priority. External circuitry can chain or otherwise merge signals from devices at each level, allowing an unlimited number of devices to interrupt the processor.

The $\overline{IPL2}$–$\overline{IPL0}$ signals must maintain the interrupt request level until the MC68020/EC020 acknowledges the interrupt to guarantee that the interrupt is recognized. The MC68020/EC020 continuously samples the $\overline{IPL2}$–$\overline{IPL0}$ signals on consecutive falling edges of the processor clock to synchronize and debounce these signals. An interrupt request that is the same for two consecutive falling clock edges is considered a valid input. Although the protocol requires that the request remain until the processor runs an interrupt acknowledge cycle for that interrupt value, an interrupt request that is held for as short a period as two clock cycles could be recognized.

The I2–I0 bits in the SR specify the interrupt priority mask. 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. Figure 6-2 is a flowchart of the procedure for making an interrupt pending.

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

Table 6-3 lists the interrupt levels, the states of $\overline{IPL2}$–$\overline{IPL0}$ that define each level, and the mask value that allows an interrupt at each level.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.1.9 Interrupt Excepti

When a peripheral device
send information that the
interrupt exception. The in
appropriately.

The peripheral device uses
processor and to specify th
of zero through seven ($\overline{\text{IPL}}$
the interrupt request level i
prioritized interrupts; level
otherwise merge signals f
devices to interrupt the proc

The $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$ signals mus
acknowledges the interr
MC68020/EC020 continuo
edges of the processor clo
request that is the same f
input. Although the protoco
interrupt acknowledge cycle
short a period as two clock

The I2–I0 bits in the SR spe
is the highest priority level
priority higher than the val
interrupt. Figure 6-2 is a flov

When several devices are c

Freescale Semiconductor, Inc.

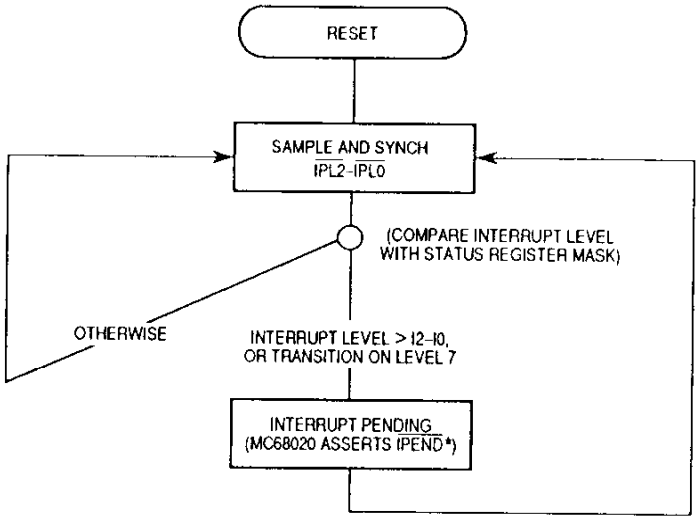
For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



*IPEND is not implemented in the MC68EC020.

Figure 6-2. Interrupt Pending Procedure

Table 6-3. Interrupt Levels and Mask Values

Requested Interrupt Level	Control Line Status			Interrupt Mask Value Required for Recognition
	IPL2	IPL1	IPL0	
0*	N	N	N	N/A*
1	N	N	A	0
2	N	A	N	1 0
3	N	A	A	2-0
4	A	N	N	3-0
5	A	N	A	4-0
6	A	A	N	5-0
7	A	A	A	7 0

*Indicates that no interrupt is requested.

A Asserted

N--- Negated

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 6-3 shows two examples of interrupt recognitions, one for level 6 and one for level 7. When the MC68020/EC020 processes a level 6 interrupt, the interrupt priority mask is automatically updated with a value of 6 before entering the handler routine so that subsequent level 6 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

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

level 6 interrupt is not processed. However, if the MC68020/EC020 is handling a level 7 interrupt (I2–I0 in the SR set to 111) and the external request is lowered to level 3 and then 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 7 interrupt is also generated by a level comparison 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). As shown in Figure 6-3 for level 6 interrupt request level and mask level, this is the case for all interrupt levels.

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.

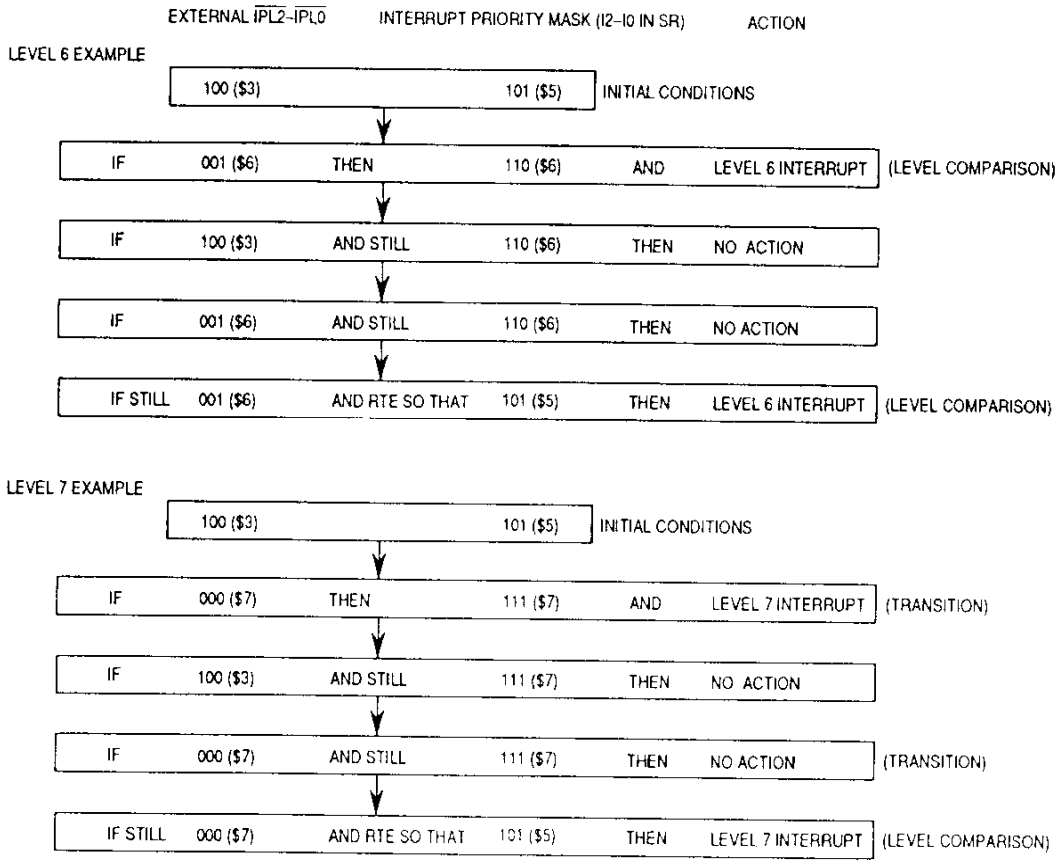


Figure 6-3. Interrupt Recognition Examples

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The MC68020 asserts $\overline{\text{IPEND}}$ (note that $\overline{\text{IPEND}}$ is not implemented in the MC68EC020) when it makes an interrupt request pending. Figure 6-4 shows the assertion of $\overline{\text{IPEND}}$ relative to the assertion of an interrupt level on $\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$. $\overline{\text{IPEND}}$ signals to external devices that an interrupt exception will be taken at an upcoming instruction boundary (following any higher priority exception). The state of the $\overline{\text{IPEND}}$ signal is internally checked by the processor once per instruction, independently of bus operation. In addition, it is checked during the second instruction prefetch associated with exception processing.

Figure 6-5 is a flowchart of the interrupt recognition and associated exception processing sequence.

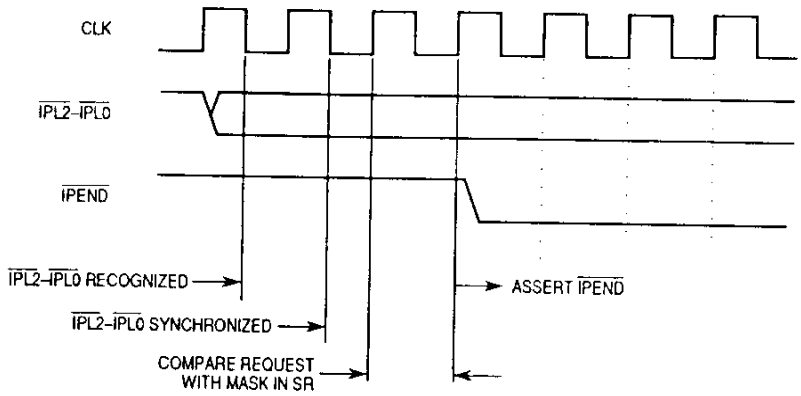
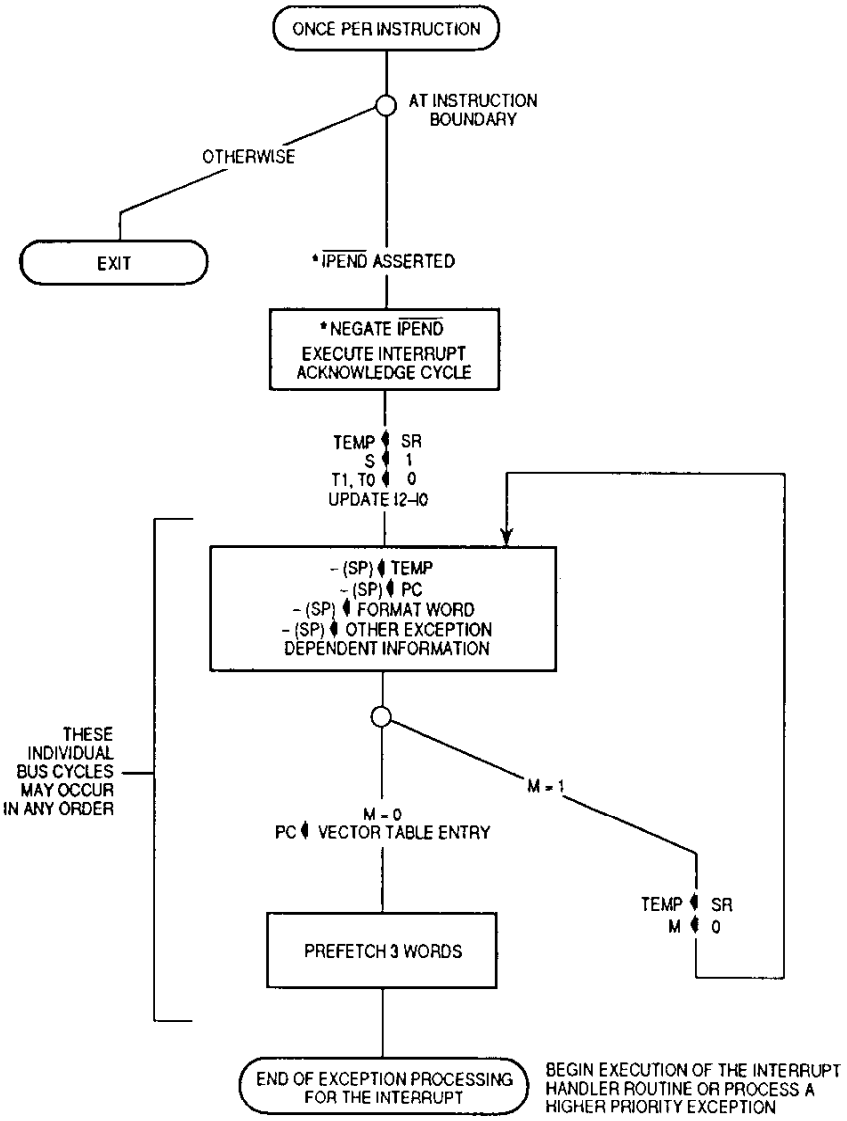


Figure 6-4. Assertion of $\overline{\text{IPEND}}$ (MC68020 Only)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* Does not apply to the MC68EC020.

Figure 6-5. Interrupt Exception Processing Flowchart

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For the MC68020, if no higher priority interrupt has been synchronized, the $\overline{IPEND}$ signal is negated during state 0 (S0) of an interrupt acknowledge cycle, and the $\overline{IPL2}$–$\overline{IPL0}$ signals for the interrupt being acknowledged can be negated at this time. For the MC68EC020, if no higher priority interrupt has been synchronized, the IPL2–IPL0 signals for the interrupt being acknowledged can be negated at this time. Refer to **Section 5 Bus Operation** for more information on interrupt acknowledge cycles.

When processing an interrupt exception, the MC68020/EC020 first makes an internal copy of the SR, sets the privilege level 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 pins A3–A1 of the address bus. For a device that cannot supply an interrupt vector, the $\overline{AVEC}$ signal can be asserted, and the MC68020/EC020 uses an internally generated autovector, which is one of vector numbers 31–25, that corresponds to the interrupt level number. If external logic indicates a bus error during the interrupt acknowledge cycle, the interrupt is considered spurious, and the processor generates the spurious interrupt vector number (24). Refer to **Section 5 Bus Operation** for complete interrupt bus cycle information.

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 interrupt was acknowledged during the execution of a coprocessor instruction, further internal information is saved on the stack so that the MC68020/EC020 can continue executing the coprocessor instruction when the interrupt handler completes execution.

If the M-bit in 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 instead of 0 or 9. The copy of the SR saved on the throwaway frame is exactly the same as that placed on the master stack except that the S-bit is set in the version placed on the interrupt stack. (It may or may not be set in the copy saved on the master stack.) The resulting SR (after exception processing) has the S-bit set and the M-bit cleared.

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

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

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

6.1.10 Breakpoint Instruction Exception

To use the MC68020/EC020 in a hardware emulator, it must provide a means of inserting breakpoints in the emulator code and of performing appropriate operations at each breakpoint. For the MC68000 and MC68008, this can be done by inserting an illegal instruction at the breakpoint and detecting the illegal instruction exception from its vector location. However, since the VBR on M68000 family processors MC68010 and later allows arbitrary relocation of exception vectors, the exception address cannot reliably identify a breakpoint. The MC68020/EC020 processor provides a breakpoint capability with a set of breakpoint instructions, $4848–$484F, for eight unique breakpoints. The breakpoint facility also allows external hardware to monitor the execution of a program residing in the on-chip instruction cache without severe performance degradation.

When the MC68020/EC020 executes a breakpoint instruction, it performs a breakpoint acknowledge cycle (read cycle) from CPU space type $0 with address lines A4–A2 corresponding to the breakpoint number. Refer to **Section 5 Bus Operation** for a description of the breakpoint acknowledge cycle. The external hardware can return either $\overline{\text{BERR}}$ or $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ with an instruction word on the data bus. If the bus cycle terminates with $\overline{\text{BERR}}$, the processor performs illegal instruction exception processing. If the bus cycle terminates with $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$, the processor uses the data returned to replace the breakpoint instruction in the internal instruction pipe and begins execution of that instruction. The remainder of the pipe remains unaltered. In addition, no stacking or vector fetching is involved with the execution of the instruction. Figure 6-6 is a flowchart of the breakpoint instruction execution.

6.1.11 Multiple Exceptions

When several exceptions occur simultaneously, they are processed according to a fixed priority. Table 6-4 lists the exceptions grouped by characteristics. Each group has a priority from 4–0. Priority 0 has the highest priority.

As soon as the MC68020/EC020 has completed exception processing for a condition when another exception is pending, it begins exception processing for the pending exception instead of executing the exception handler for the original exception condition. Also, whenever a bus error or address error occurs, its exception processing takes precedence over lower priority exceptions and occurs immediately. For example, if a bus error occurs during the exception processing for a trace condition, the system processes the bus error and executes its handler before completing the trace exception processing. However, most exceptions cannot occur during exception processing, and very few combinations of the exceptions shown in Table 6-4 can be pending simultaneously.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

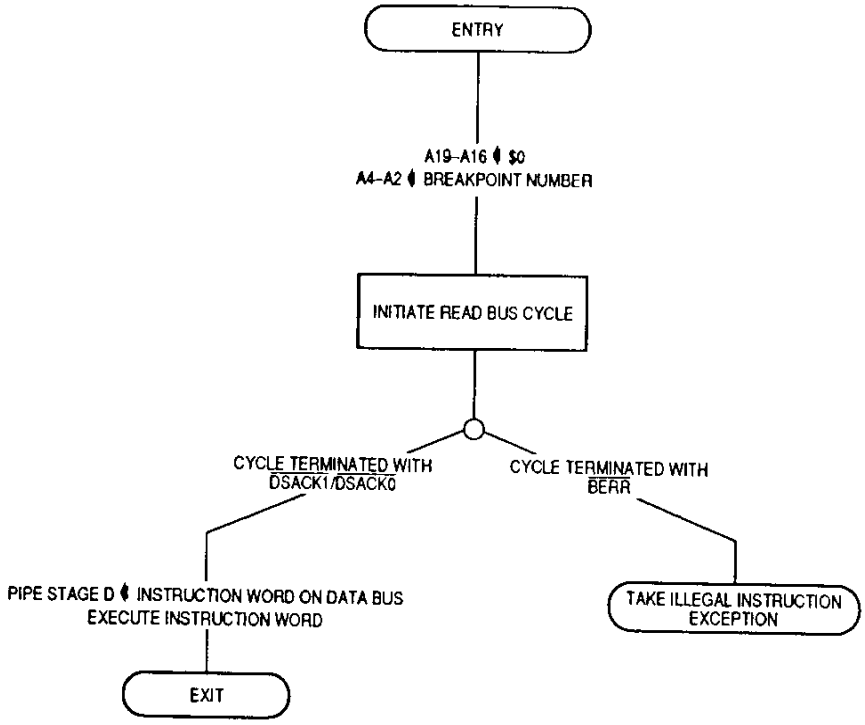


Figure 6-6. Breakpoint Instruction Flowchart

Table 6-4. Exception Priority Groups

Group/ Priority	Exception and Relative Priority	Characteristic
0	0.0—Reset	Aborts all processing (instruction or exception) and does not save old context.
1	1.0—Address Error 1.1—Bus Error	Suspends processing (instruction or exception) and saves internal context.
2	2.0—BKPT, CALLM, CHK, CHK2, cp Midinstruction, cp Protocol Violation, cpTRAPcc, Divide by Zero, RTE, RTM, TRAP, TRAPcc, TRAPV	Exception processing is part of instruction execution.
3	3.0—Illegal Instruction, Line A, Unimplemented Line F, Privilege Violation, cp Preinstruction	Exception processing begins before instruction is executed.
4	4.0—cp Postinstruction 4.1—Trace 4.2—Interrupt	Exception processing begins when current instruction or previous exception processing has completed.

NOTE: 0.0 is the highest priority; 4.2 is the lowest.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The priority scheme is very important in determining the order in which exception handlers execute when several exceptions occur at the same time. As a general rule, the lower the priority of an exception, the sooner the handler routine for that exception executes. For example, if simultaneous trap, trace, and interrupt exceptions are pending, the exception processing for the trap occurs first, followed immediately by exception processing for the trace, and then for the interrupt. When the processor resumes normal instruction execution, it is in the interrupt handler, which returns to the trace handler, which returns to the trap exception handler. This rule does not apply to the reset exception; its handler is executed first even though it has the highest priority because the reset operation clears all other exceptions.

6.1.12 Return from Exception

After the MC68020/EC020 has completed exception processing for all pending exceptions, it resumes normal instruction execution at the address in the vector for the last exception processed. Once the exception handler has completed execution, the processor must return to the system context prior to the exception (if possible). The RTE instruction returns from the handler to the previous system context for any exception.

When the processor executes an RTE instruction, it examines the stack frame on top of the active supervisor stack to determine if it is a valid frame and what type of context restoration it requires. The following paragraphs describe the processing for each of the stack frame types; refer to **6.3 Coprocessor Considerations** for a description of the stack frame type formats.

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

For the throwaway four-word frame, the processor reads the SR value from the frame, increments the active stack pointer by eight, updates the SR with the value read from the stack, and then begins RTE processing again, as shown in Figure 6-7. The processor reads a new format word from the stack frame on top of the active stack (which may or may not be the same stack used for the previous operation) and performs the proper operations corresponding to that format. In most cases, the throwaway frame is on the interrupt stack and when the SR value is read from the stack, the S and M bits are set. In that case, there is a normal four-word frame or a ten-word coprocessor midinstruction frame on the master stack. However, the second frame may be any format (even another throwaway frame) and may reside on any of the three system stacks.

For the six-word stack frame, the processor restores the SR and PC values from the stack, increments the active supervisor stack pointer by 12, and resumes normal instruction execution.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

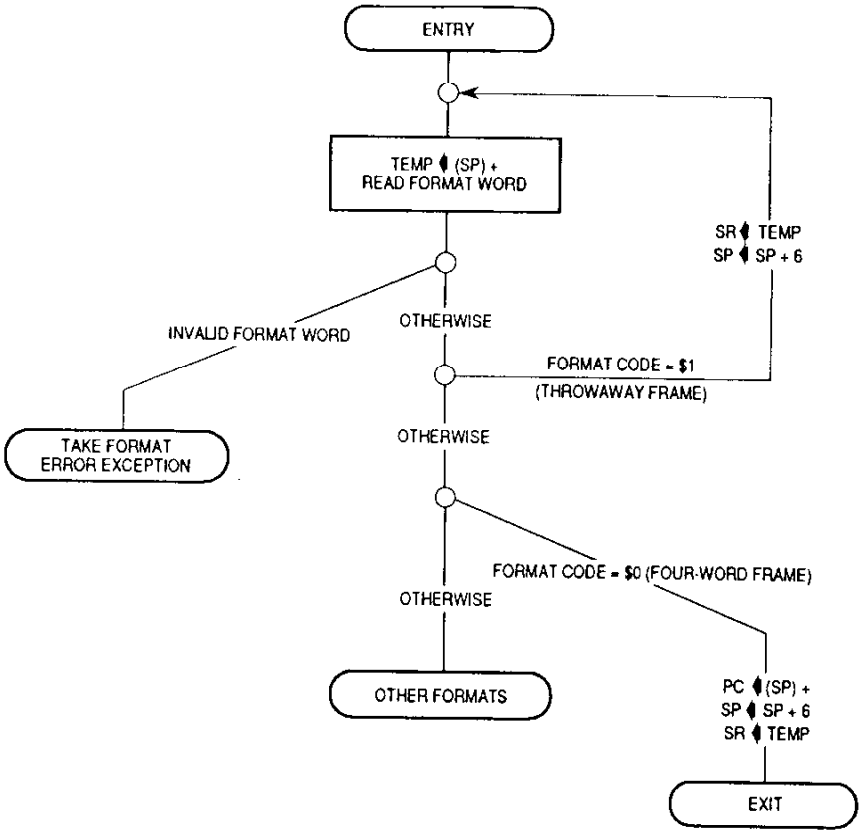


Figure 6-7. RTE Instruction for Throwaway Four-Word Frame

For the coprocessor midinstruction stack frame, the processor reads the SR, PC, instruction address, internal register values, and the evaluated effective address from the stack, restores these values to the corresponding internal registers, and increments the stack pointer by 20. The processor then reads from the response register of the coprocessor that initiated the exception to determine the next operation to be performed. Refer to **Section 7 Coprocessor Interface Description** for details of coprocessor-related exceptions.

For both the short and long bus fault stack frames, the processor first checks the format value on the stack for validity. In addition, for the long stack frame, the processor compares the version number in the stack with its own version number. The version number is located in the most significant nibble (bits 15–12) of the word at location SP + $36 in the long stack frame. This validity check is required in a multiprocessor system to ensure that the data is properly interpreted by the RTE instruction. The RTE instruction also reads from both ends of the stack frame to make sure it is accessible. If the frame is invalid or inaccessible, the processor takes a format error or a bus error exception, respectively. Otherwise, the processor reads the entire frame into the proper internal registers, deallocates the stack, and resumes normal processing. Once the processor begins to load the frame to restore its internal state, the assertion of the $\overline{\text{BERR}}$ signal

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

causes the processor to enter the halted state. Refer to **6.2 Bus Fault Recovery** for a description of the processing that occurs after the frame is read into the internal registers.

If a format error or bus error exception occurs during the frame validation sequence of the RTE instruction, either due to any of the errors previously described or due to an illegal format code, the processor creates a normal four-word or a bus fault stack frame below the frame that it was attempting to use. In this way, the faulty stack frame remains intact. The exception handler can examine or repair the faulty frame. In a multiprocessor system, the faulty frame can be left to be used by another processor of a different type (e.g., an MC68010 or a future M68000 family processor) when appropriate.

6.2 BUS FAULT RECOVERY

An address error exception or a bus error exception indicates a bus fault. The saving of the processor state for a bus error or address error is described in **6.1.2 Bus Error Exception**, and the restoring of the processor state by an RTE instruction is described in **6.1.12 Return from Exception**.

Processor accesses of either data items or the instruction stream can result in bus errors. When a bus error exception occurs while accessing a data item, the exception is taken immediately after the bus cycle terminates. The processor may never access an instruction that is part of the instruction stream. In this case, the bus error would not be processed. For instruction faults, when the short bus fault stack frame applies, the address of the pipe stage B word is the value in the PC plus four, and the address of the stage C word is the value in the PC plus two. For the long format, the long word at SP + $24 contains the address of the stage B word; the address of the stage C word is the address of the stage B word minus two. Address error faults occur only for instruction stream accesses, and the exceptions are taken before the bus cycles are attempted.

6.2.1 Special Status Word (SSW)

The internal SSW (see Figure 6-8) is one of several registers saved as part of the bus fault exception stack frame. Both the short bus fault format and the long bus fault format include this word at offset $A. The bus cycle fault stack frame formats are described in detail in **6.4 Exception Stack Frame Formats**.

The SSW information indicates whether the fault was caused by an access to the instruction stream, data stream, or both. The high-order half of the SSW contains two status bits each for the B and C stages of the instruction pipe. If an address error exception occurs, the fault bits written to the stack frame are not set (they are only set due to a bus error, as previously described), and the rerun bits alone show the cause of the exception. Depending on the state of the pipeline, either RB and RC are set, or only RC is set. To correct the pipeline contents and continue execution of the suspended instruction, software must place the correct instruction stream data in the stage C and/or stage B images requested by the rerun bits and must clear the rerun bits. The least significant half of the SSW applies to data cycles only. Data and instruction stream faults may be pending simultaneously; the fault handler should be able to recognize any combination of the FC, FB, RC, RB, and DF bits.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

15	14	13	12	11	10	9	8	7	6	5 4	3	2 0
FC	FB	RC	RB	0	0	0	DF	RM	RW	SIZE	0	FC2–FC0

Figure 6-8. Special Status Word Format

FC—Fault on Stage C

When the FC bit is set, the processor attempted to use stage C and found it to be marked invalid due to a bus error on the prefetch for that stage. FC can be used by a bus error handler to determine the cause(s) of a bus error exception.

FB—Fault on Stage B

When the FB bit is set, the processor attempted to use stage B and found it to be marked invalid due to a bus error on the prefetch for that stage. FB can be used by a bus error handler to determine the cause(s) of a bus error exception.

RC—Rerun Flag for Stage C

The RC bit is set to indicate that a fault occurred during a prefetch for stage C. The RC bit is always set when the FC bit is set. The RC bit indicates that the word in stage C of the instruction pipe is invalid, and the state of the bit can be used by a handler to repair the values in the pipe after an address error or a bus error, if necessary. If the RC bit is set when the processor executes an RTE instruction, the processor may execute a bus cycle to prefetch the instruction word for stage C of the pipe (if it is required). If the RC and FC bits are set, the RTE instruction automatically reruns the prefetch cycle for stage C. The address space for the bus cycle is the program space for the privilege level indicated in the copy of the SR on the stack. If the RC bit is clear, the words on the stack for stage C of the pipe are accepted as valid; the processor assumes that there is no prefetch pending for stage C and that software has repaired or filled the image of stage C, if necessary.

1 = Rerun faulted bus cycle or run pending prefetch
0 = Do not rerun bus cycle

RB—Rerun Flag for Stage B

The RB bit is set to indicate that a fault occurred during a prefetch for stage B. The RB bit is always set when the FB bit is set. The RB bit indicates that the word in stage B of the instruction pipe is invalid, and the state of the bit can be used by a handler to repair the values in the pipe after an address error or a bus error, if necessary. If the RB bit is set when the processor executes an RTE instruction, the processor may execute a bus cycle to prefetch the instruction word for stage B of the pipe (if it is required). If the RB and FB bits are set, the RTE instruction automatically reruns the prefetch cycle for stage B. The address space for the bus cycle is the program space for the privilege level indicated in the copy of the SR on the stack. If the RB bit is clear, the words on the

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

stack for stage B of the pipe are accepted as valid; the processor assumes that there is no prefetch pending for stage B and that software has repaired or filled the image of stage B, if necessary.

1 = Rerun faulted bus cycle or run pending prefetch
0 = Do not rerun bus cycle

Bits 11–9—Reserved by Motorola

DF—Fault/Rerun Flag

If the DF bit is set, a data fault has occurred and caused the exception. If the DF bit is set when the processor reads the stack frame, it reruns the faulted data access; otherwise, it assumes that the data input buffer value on the stack is valid for a read or that the data has been correctly written to memory for a write (or that no data fault occurred).

1 = Rerun faulted bus cycle or run pending prefetch
0 = Do not rerun bus cycle

RM—Read-Modify-Write

1 = Read-modify-write operation on data cycle
0 = Not a read-modify-write operation

RW—Read/Write

1 = Read on data cycle
0 = Write on data cycle

SIZE—Size Code

The SIZE field indicates the size of the operand access for the data cycle.

Bit 3—Reserved by Motorola

FC2–FC0—Specifies the address space for data cycle

6.2.2 Using Software to Complete the Bus Cycles

One method of completing a faulted bus cycle is to use a software handler to emulate the cycle. This is the only method for correcting address errors. The handler should emulate the faulted bus cycle in a manner that is transparent to the instruction that caused the fault. For instruction stream faults, the handler may need to run bus cycles for both the B and C stages of the instruction pipe. The RB and RC bits of the SSW identify the stages that may require a bus cycle; the FB and FC bits of the SSW indicate that a stage was invalid when an attempt was made to use its contents. Those stages must be repaired. For each faulted stage, the software handler should copy the instruction word from the proper address space as indicated by the S-bit of the copy of the SR saved on the stack to the image of the appropriate stage in the stack frame. In addition, the handler must clear the RB or RC bit associated with the stage that it has corrected. The handler should not change the FB and FC bits.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

To repair data faults (indicated by DF = 1), the software should first examine the RM bit in the SSW to determine if the fault was generated during a read-modify-write operation. If RM = 0, the handler should then check the RW bit of the SSW to determine if the fault was caused by a read or a write cycle. For data write faults, the handler must transfer the properly sized data from the data output buffer on the stack frame to the location indicated by the data fault address in the address space defined by the SSW. (Both the data output buffer and the data fault address are part of the stack frame at SP + $18 and SP + $10, respectively.) Data read faults only generate the long bus fault frame, and the handler must transfer properly sized data from the location indicated by the fault address and address space to the image of the data input buffer at location SP + $2C of the long format stack frame. Byte, word, and 3-byte operands are right justified in the 4-byte data buffers. In addition, the software handler must clear the DF bit of the SSW to indicate that the faulted bus cycle has been corrected.

To emulate a read-modify-write cycle, the exception handler must first read the operation word at the PC address (SP + 2 of the stack frame). This word identifies the CAS, CAS2, or TAS instruction that caused the fault. Then the handler must emulate this entire instruction (which may consist of up to four long-word transfers) and update the CCR portion of the SR appropriately, because the RTE instruction expects the entire operation to have been completed if the RM bit is set and the DF bit is cleared. This is true even if the fault occurred on the first read cycle.

To emulate the entire instruction, the handler must save the data and address registers for the instruction (with a MOVEM instruction, for example). Next, the handler reads and modifies (if necessary) the memory location. It clears the DF bit in the SSW of the stack frame and modifies the condition codes in the SR copy and the copies of any data or address registers required for the CAS and CAS2 instructions. Last, the handler restores the registers that it saved at the beginning of the emulation. Except for the data input buffer, the copy of the SR, and the SSW, the handler should not modify a bus fault stack frame. The only bits in the SSW that may be modified are DF, RB, and RC; all other bits, including those defined for internal use, must remain unchanged.

Address error faults must be repaired in software. Address error faults can be distinguished from bus error faults by the value in the vector offset field of the format word.

6.2.3 Completing the Bus Cycles with RTE

Another method of completing a faulted bus cycle is to allow the processor to rerun the bus cycles during execution of the RTE instruction that terminates the exception handler. This method cannot be used to recover from address errors. The RTE instruction is always executed. Unless the handler routine has corrected the error and cleared the fault (and cleared the RB/RC and DF bits of the SSW), the RTE instruction cannot complete the bus cycle(s). If the DF bit is still set at the time of the RTE execution, the faulted data cycle is rerun by the RTE instruction. If the FB or FC bit is set and the corresponding rerun bit (RB or RC) was not cleared by the software, the RTE reruns the associated instruction prefetch. The fault occurs again unless the cause of the fault, such as a nonresident page in a virtual memory system, has been corrected. If the RB or RC bit is set and the

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

corresponding fault bit (FB or FC) is cleared, the associated prefetch cycle may or may not be run by the RTE instruction (depending on whether the stage is required).

If a fault occurs when the RTE instruction attempts to rerun the bus cycle(s), the processor creates a new stack frame on the supervisor stack after deallocating the previous frame, and address error or bus error exception processing starts in the normal manner.

The read-modify-write operations of the MC68020/EC020 can also be completed by the RTE instruction that terminates the handler routine. The rerun operation, executed by the RTE instruction with the DF bit of the SSW set, reruns the entire instruction. If the cause of the error has been corrected, the handler does not need to emulate the instruction but can leave the DF bit set and execute the RTE instruction.

6.3 COPROCESSOR CONSIDERATIONS

Exception handler programmers should consider carefully whether to save and restore the context of a coprocessor at the beginning and end of handler routines for exceptions that can occur during the execution of a coprocessor instruction (i.e., bus errors, interrupts, and coprocessor-related exceptions). The nature of the coprocessor and the exception handler routine determines whether or not saving the state of one or more coprocessors with the cpSAVE and cpRESTORE instructions is required. If the coprocessor allows multiple coprocessor instructions to be executed concurrently, it may require its state to be saved and restored for all coprocessor-generated exceptions, regardless of whether or not the coprocessor is accessed during the handler routine. The MC68882 floating-point coprocessor is an example of this type of coprocessor. On the other hand, the MC68881 floating-point coprocessor requires FSAVE and FRESTORE instructions within an exception handler routine only if the exception handler itself uses the coprocessor.

6.4 EXCEPTION STACK FRAME FORMATS

The MC68020/EC020 provides six different stack frames for exception processing. The set of frames includes the normal four- and six-word stack frames, the four-word throwaway stack frame, the coprocessor midinstruction stack frame, and the short and long bus fault stack frames.

When the MC68020/EC020 writes or reads a stack frame, it uses long-word operand transfers wherever possible. Using a long-word-aligned stack pointer with memory that is on a 32-bit port greatly enhances exception processing performance. The processor does not necessarily read or write the stack frame data in sequential order.

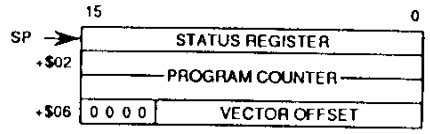
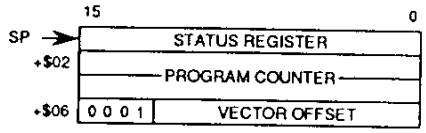
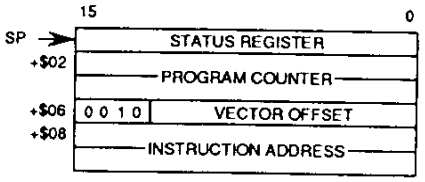
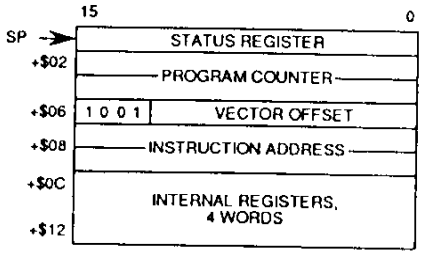
The system software should not depend on a particular exception generating a particular stack frame. For compatibility with future devices, the software should be able to handle any type of stack frame for any type of exception.

Table 6-5 summarizes the stack frames defined for the MC68020/EC020.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

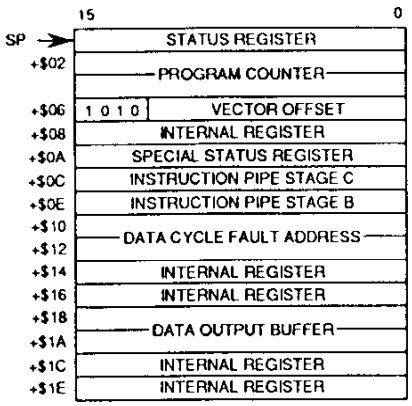
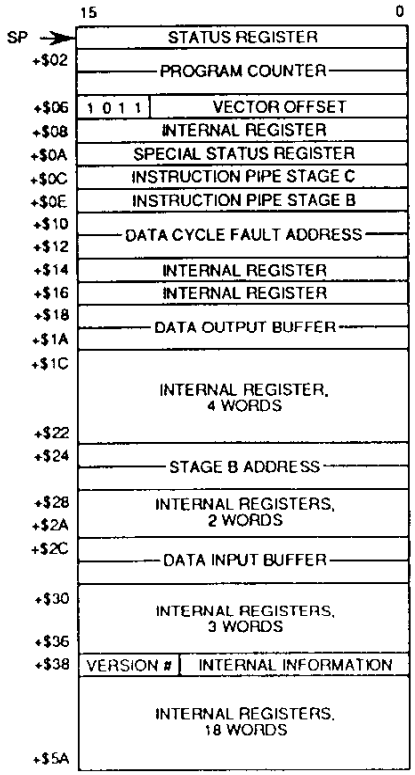
Table 6-5. Exception Stack Frames

Stack Frames	Exception Types (Stacked PC Points to)
 FOUR-WORD STACK FRAME — FORMAT $0	• Interrupt [Next instruction] • Format Error [RTE or cpRESTORE instruction] • TRAP [NEXT instruction] • Illegal Instruction [Illegal instruction] • A-Line Instruction [A-line instruction] • F-Line Instruction [F-line instruction] • Privilege Violation [First word of instruction causing Privilege Violation] • Coprocessor Preinstruction [Opword of instruction that returned the 'take preinstruction' primitive]
 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]
 SIX-WORD STACK FRAME — FORMAT $2	• CHK • CHK2 • cpTRAPcc • TRAPcc • TRAPV • Trace • Zero Divide • MMU Configuration • Coprocessor Postinstruction [Next instruction for all these exceptions] INSTRUCTION ADDRESS is the address of the instruction that caused the exception
 COPROCESSOR MIDINSTRUCTION STACK FRAME (10 WORDS) — FORMAT $9	• Coprocessor Midinstruction • Main-Detected Protocol Violation • Interrupt Detected During Coprocessor Instruction (supported with 'null come again with interrupts allowed' primitive) [Next word to be fetched from instruction stream for all these exceptions] INSTRUCTION ADDRESS is the address of the instruction that caused the exception

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 6-5. Exception Stack Frames (Continued)

Stack Frames	Exception Types (Stacked PC Points to)
 SHORT BUS FAULT STACK FRAME (16 WORDS) — FORMAT $A	• Address Error or Bus Error — Execution Unit at Instruction Boundary [Next instruction]
 LONG BUS FAULT STACK FRAME (46 WORDS) — FORMAT $B	• Address Error or Bus Error — Instruction Execution in Progress [Address of instruction in execution when fault occurred — may not be the instruction that generated the faulted bus cycle]

6

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 7
COPROCESSOR INTERFACE DESCRIPTION

The M68000 family of general-purpose microprocessors provides a level of performance that satisfies a wide range of computer applications. Special-purpose hardware, however, can often provide a higher level of performance for a specific application. The coprocessor concept allows the capabilities and performance of a general-purpose processor to be enhanced for a particular application without encumbering the main processor architecture. A coprocessor can efficiently meet specific capability requirements that must typically be implemented in software by a general-purpose processor. With a general-purpose main processor and the appropriate coprocessor(s), the processing capabilities of a system can be tailored to a specific application.

The MC68020/EC020 supports the M68000 coprocessor interface described in this section. This section is intended for designers who are implementing coprocessors to interface with the MC68020/EC020.

The designer of a system that uses one or more Motorola coprocessors (the MC68881 or MC68882 floating-point coprocessor, for example) does not require a detailed knowledge of the M68000 coprocessor interface. Motorola coprocessors conform to the interface described in this section. Typically, they implement a subset of the interface, and that subset is described in the coprocessor user's manual. These coprocessors execute Motorola-defined instructions that are described in the user's manual for each coprocessor.

7.1 INTRODUCTION

The distinction between standard peripheral hardware and an M68000 coprocessor is important from a programming model perspective. The programming model of the main processor consists of the instruction set, register set, and memory map. An M68000 coprocessor is a device or set of devices that communicates with the main processor through the protocol defined as the M68000 coprocessor interface. The programming model for a coprocessor is different than that for a peripheral device. A coprocessor adds additional instructions and generally additional registers and data types to the programming model that are not directly supported by the main processor architecture. The additional instructions are dedicated coprocessor instructions that utilize the coprocessor capabilities. The necessary interactions between the main processor and the coprocessor that provide a given service are transparent to the programmer. That is, the programmer does not need to know the specific communication protocol between the main processor and the coprocessor because this protocol is implemented in hardware. Thus, the coprocessor can provide capabilities to the user without appearing separate from the main processor.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

In contrast, standard peripheral hardware is generally accessed through interface registers mapped into the memory space of the main processor. To use the services provided by the peripheral, the programmer accesses the peripheral registers with standard processor instructions. While a peripheral could conceivably provide capabilities equivalent to a coprocessor for many applications, the programmer must implement the communication protocol between the main processor and the peripheral necessary to use the peripheral hardware.

The communication protocol defined for the M68000 coprocessor interface is described in **7.2 Coprocessor Instruction Types**. The algorithms that implement the M68000 coprocessor interface are provided in the microcode of the MC68020/EC020 and are completely transparent to the MC68020/EC020 programming model. For example, floating-point operations are not implemented in the MC68020/EC020 hardware. In a system utilizing both the MC68020/EC020 and the MC68881 or MC68882 floating-point coprocessor, a programmer can use any of the instructions defined for the coprocessor without knowing that the actual computation is performed by the MC68881 or MC68882 hardware.

7.1.1 Interface Features

The M68000 coprocessor interface design incorporates a number of flexible capabilities. The physical coprocessor interface uses the main processor external bus, which simplifies the interface since no special-purpose signals are involved. With the MC68020/EC020, a coprocessor uses the asynchronous bus transfer protocol. Since standard bus cycles transfer information between the main processor and the coprocessor, the coprocessor can be implemented in whatever technology is available to the coprocessor designer. A coprocessor can be implemented as a VLSI device, as a separate system board, or even as a separate computer system.

Since the main processor and a M68000 coprocessor can communicate using the asynchronous bus, they can operate at different clock frequencies. The system designer can choose the speeds of a main processor and coprocessor that provide the optimum performance for a given system. Both the MC68881 and MC68882 floating-point coprocessors use the asynchronous bus handshake protocol.

The M68000 coprocessor interface also facilitates the design of coprocessors. The coprocessor designer must only conform to the coprocessor interface and does not need an extensive knowledge of the architecture of the main processor. Also, the main processor can operate with a coprocessor without having explicit provisions made in the main processor for the capabilities of that coprocessor. This type of interface provides a great deal of freedom in the implementation of a given coprocessor.

7.1.2 Concurrent Operation Support

The programming model for the M68000 family of microprocessors is based on sequential, nonconcurrent instruction execution, which implies that the instructions in a given sequence must appear to be executed in the order in which they occur. To maintain a uniform programming model, any coprocessor extensions should also maintain the

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

model of sequential, nonconcurrent instruction execution at the user level. Consequently, the programmer can assume that the images of registers and memory affected by a given instruction have been updated when the next instruction in the sequence accessing these registers or memory locations is executed.

The M68000 coprocessor interface provides full support of all operations necessary for nonconcurrent operation of the main processor and its associated coprocessors. Although the M68000 coprocessor interface allows concurrency in coprocessor execution, the coprocessor designer is responsible for implementing this concurrency while maintaining a programming model based on sequential nonconcurrent instruction execution.

For example, if the coprocessor determines that instruction B does not use or alter resources to be altered or used by instruction A, instruction B can be executed concurrently (if the execution hardware is also available). Thus, the required instruction interdependencies and sequences of the program are always respected. The MC68882 coprocessor offers concurrent instruction execution; whereas, the MC68881 coprocessor does not. However, the MC68020/EC020 can execute instructions concurrently with coprocessor instruction execution in the MC68881.

7.1.3 Coprocessor Instruction Format

The instruction set for a given coprocessor is defined by the design of that coprocessor. When a coprocessor instruction is encountered in the main processor instruction stream, the MC68020/EC020 hardware initiates communication with the coprocessor and coordinates any interaction necessary to execute the instruction with the coprocessor. A programmer needs to know only the instruction set and register set defined by the coprocessor to use the functions provided by the coprocessor hardware.

The instruction set of an M68000 coprocessor uses a subset of the F-line operation words in the M68000 instruction set. The operation word is the first word of any M68000 family instruction. The F-line operation word contains ones in bits 15–12 (refer to Figure 7-1); the remaining bits are coprocessor and instruction dependent. The F-line operation word may be followed by as many extension words as are required to provide additional information necessary for the execution of the coprocessor instruction.

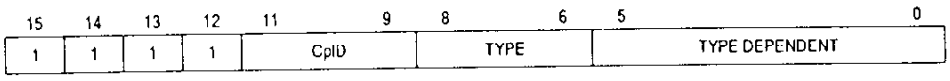


Figure 7-1. F-Line Coprocessor Instruction Operation Word

As shown in Figure 7-1, bits 11–9 of the F-line operation word encode the coprocessor identification (CpID) field. The MC68020/EC020 uses the CpID field to indicate the coprocessor to which the instruction applies. F-line operation words, in which the CpID is zero, are not coprocessor instructions for the MC68020/EC020. Instructions with a CpID of zero and a nonzero type field are unimplemented instructions that cause the

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

MC68020/EC020 to begin exception processing. The MC68020/EC020 never generates coprocessor interface bus cycles with the CpID equal to zero (except via the MOVES instruction).

CpID codes of 000–101 are reserved for current and future Motorola coprocessors, and CpID codes of 110–111 are reserved for user-defined coprocessors. The Motorola CpID code of 001 designates the MC68881 or MC68882 floating-point coprocessor. By default, Motorola assemblers will use a CpID code of 001 when generating the instruction operation codes for the MC68881 or MC68882.

The encoding of bits 8–0 of the coprocessor instruction operation word is dependent on the particular instruction being implemented (refer to **7.2 Coprocessor Instruction Types**).

7.1.4 Coprocessor System Interface

The communication protocol between the main processor and coprocessor necessary to execute a coprocessor instruction uses a group of interface registers, CIRs, resident within the coprocessor. By accessing one of the CIRs, the MC68020/EC020 hardware initiates coprocessor instructions. The coprocessor uses a set of response primitive codes and format codes defined for the M68000 coprocessor interface to communicate status and service requests to the main processor through these registers. The CIRs are also used to pass operands between the main processor and the coprocessor. The CIR set, response primitives, and format codes are discussed in **7.3 Coprocessor Interface Register Set** and **7.4 Coprocessor Response Primitives**.

7.1.4.1 COPROCESSOR CLASSIFICATION. M68000 coprocessors can be classified into two categories depending on their bus interface capabilities. The first category, non-DMA coprocessors, consists of coprocessors that always operate as bus slaves. The second category, DMA coprocessors, consists of coprocessors that operate as bus slaves while communicating with the main processor across the coprocessor interface. These coprocessors also have the ability to operate as bus masters, directly controlling the system bus.

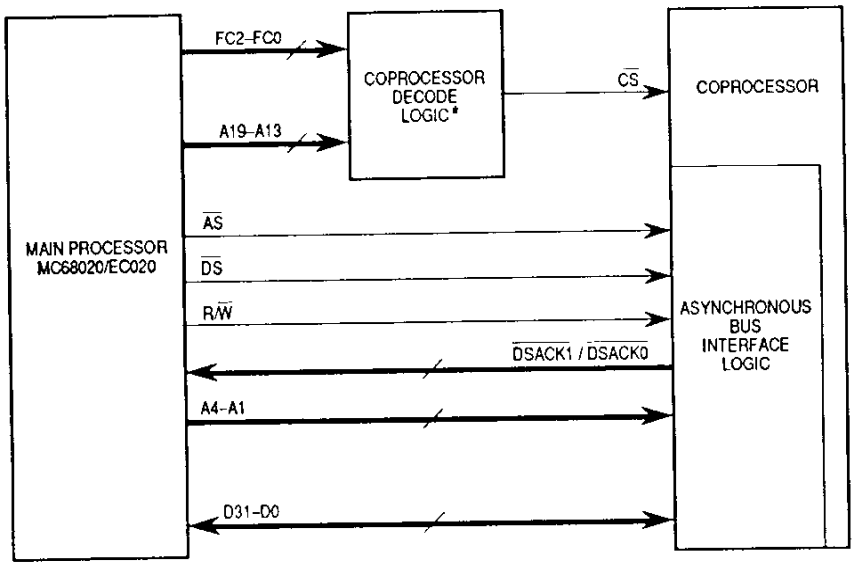
If the operation of a coprocessor does not require a large portion of the available bus bandwidth or has special requirements not directly satisfied by the main processor, that coprocessor can be efficiently implemented as a non-DMA coprocessor. Since non-DMA coprocessors always operate as bus slaves, all external bus-related functions that the coprocessor requires are performed by the main processor. The main processor transfers operands from the coprocessor by reading the operand from the appropriate CIR and then writing the operand to a specified effective address with the appropriate address space specified on the FC2–FC0. Likewise, the main processor transfers operands to the coprocessor by reading the operand from a specified effective address (and address space) and then writing that operand to the appropriate CIR using the coprocessor interface. The bus interface circuitry of a coprocessor operating as a bus slave is not as complex as that of a device operating as a bus master.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

To improve the efficiency of operand transfers between memory and the coprocessor, a coprocessor that requires a relatively high amount of bus bandwidth or has special bus requirements can be implemented as a DMA coprocessor. The DMA coprocessor provides all control, address, and data signals necessary to request and obtain the bus and then performs DMA transfers using the bus. DMA coprocessors, however, must still act as bus slaves when they require information or services of the main processor using the M68000 coprocessor interface protocol.

7.1.4.2 PROCESSOR-COPROCESSOR INTERFACE. Figure 7-2 is a block diagram of the signals involved in an asynchronous non-DMA M68000 coprocessor interface. Since the CpID on signals A15–A13 of the address bus is used with other address signals to select the coprocessor, the system designer can use several coprocessors of the same type and assign a unique CpID to each one.



FC2–FC0 = 111 ⇒ CPU SPACE CYCLE
A19–A16 = 0010 ⇒ COPROCESSOR ACCESS IN CPU SPACE
A15–A13 = xxx ⇒ COPROCESSOR IDENTIFICATION
A4–A1 = rrrr ⇒ COPROCESSOR INFERFACE REGISTER SELECTOR

*Chip select logic may be integrated into the coprocessor.
Address lines not specified above are "0" during coprocessor access.

Figure 7-2. Asynchronous Non-DMA M68000 Coprocessor Interface Signal Usage

The MC68020/EC020 accesses the registers in the CIR set using standard asynchronous bus cycles. Thus, the bus interface implemented by a coprocessor for its interface register set must satisfy the MC68020/EC020 address, data, and control signal timing. The MC68020/EC020 bus operation is described in detail in **Section 5 Bus Operation**.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

During coprocessor instruction execution, the MC68020/EC020 executes CPU space bus cycles to access the CIR set. The MC68020/EC020 asserts FC2–FC0, identifying a CPU space bus cycle. The CIR set is mapped into CPU space in the same manner that a peripheral interface register set is generally mapped into data space. The information encoded on FC2–FC0 and the address bus of the MC68020/EC020 during a coprocessor access is used to generate the chip select signal for the coprocessor being accessed. Other address lines select a register within the interface set. The information encoded on the function code and address lines of the MC68020/EC020 during a coprocessor access is illustrated in Figure 7-3.

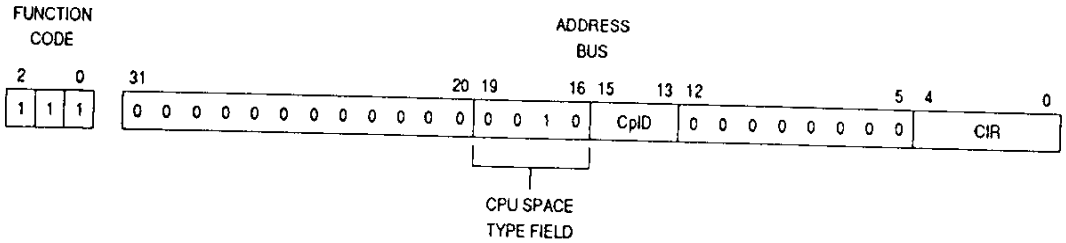


Figure 7-3. MC68020/EC020 CPU Space Address Encodings

Signals A19–A16 of the MC68020/EC020 address bus specify the CPU space cycle type for a CPU space bus cycle. The types of CPU space cycles currently defined for the MC68020/EC020 are interrupt acknowledge, breakpoint acknowledge, module support operations, and coprocessor access cycles. CPU space type $2 (A19–A16 = 0010) specifies a coprocessor access cycle.

A15–A13 specify the CpID code for the coprocessor being accessed. This code is transferred from bits 11–9 of the coprocessor instruction operation word (refer to Figure 7-1) to the address bus during each coprocessor access. Thus, decoding the MC68020/EC020 FC2–FC0 and A19–A13 signals provides a unique chip select signal for a given coprocessor. The FC2–FC0 and A19–A16 signals indicate a coprocessor access; A15–A13 indicate which of the possible eight coprocessors (000–111) is being accessed. Bits A31–A20 and A12–A5 of the MC68020 address bus and bits A23–A20 and A12–A5 of the MC68EC020 address bus are always zero during a coprocessor access.

7.1.4.3 COPROCESSOR INTERFACE REGISTER SELECTION. Figure 7-4 shows that the value on the MC68020/EC020 address bus during a coprocessor access addresses a unique region of the main processor's CPU address space. Signals A4–A0 of the MC68020/EC020 address bus select the CIR being accessed. The register map for the M68000 coprocessor interface is shown in Figure 7-5. The individual registers are described in detail in **7.3 Coprocessor Interface Register Set**.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

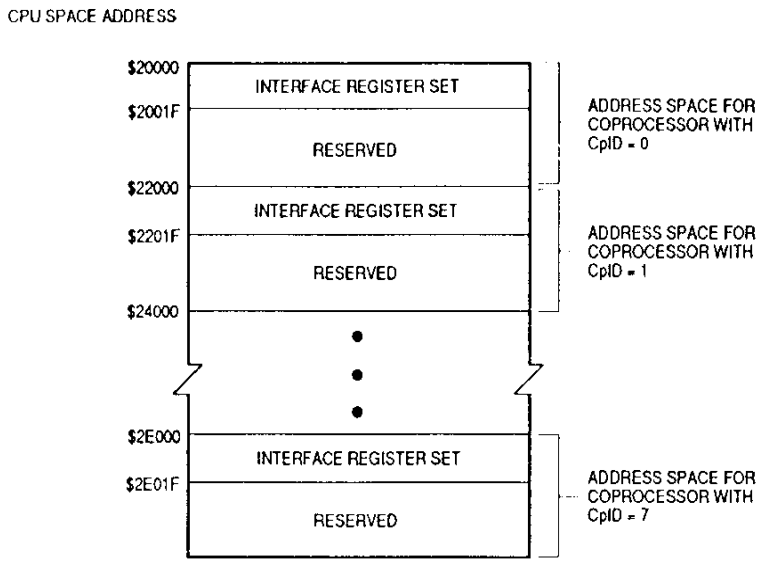


Figure 7-4. Coprocessor Address Map in MC68020/EC020 CPU Space

	31 — 16	15 — 0
$00	RESPONSE	CONTROL
$04	SAVE	RESTORE
$08	OPERATION WORD	COMMAND
$0C	(RESERVED)	CONDITION
$10	OPERAND	
$14	REGISTER SELECT	(RESERVED)
$18	INSTRUCTION ADDRESS	
$1C	OPERAND ADDRESS	

Figure 7-5. Coprocessor Interface Register Set Map

7.2 COPROCESSOR INSTRUCTION TYPES

The M68000 coprocessor interface supports four categories of coprocessor instructions: general, conditional, context save, and context restore. The category name indicates the type of operations provided by the coprocessor instructions in the category. The instruction category also determines the CIR accessed by the MC68020/EC020 to initiate instruction and communication protocols between the main processor and the coprocessor necessary for instruction execution.

During the execution of instructions in the general or conditional categories, the coprocessor uses the set of coprocessor response primitive codes defined for the M68000 coprocessor interface to request services from and indicate status to the main processor. During the execution of the instructions in the context save and context

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

restore categories, the coprocessor uses the set of coprocessor format codes defined for the M68000 coprocessor interface to indicate its status to the main processor.

7.2.1 Coprocessor General Instructions

The coprocessor general instruction category contains data processing instructions and other general-purpose instructions for a given coprocessor.

7.2.1.1 FORMAT. Figure 7-6 shows the format of a coprocessor general instruction.

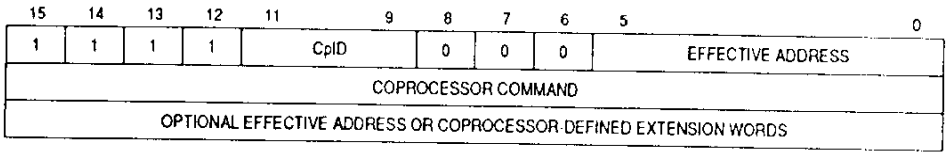


Figure 7-6. Coprocessor General Instruction Format (cpGEN)

The mnemonic cpGEN is a generic mnemonic used in this discussion for all general instructions. The mnemonic of a specific general instruction usually suggests the type of operation it performs and the coprocessor to which it applies. The actual mnemonic and syntax used to represent a coprocessor instruction is determined by the syntax of the assembler or compiler that generates the object code.

A coprocessor general instruction consists of at least two words. The first word of the instruction is an F-line operation code (bits 15–12 = 1111). The CpID field of the F-line operation code is used during the coprocessor access to indicate which coprocessor in the system executes the instruction. During accesses to the CIRs (refer to **7.1.4.2 Processor-Coprocessor Interface**), the processor places the CpID on address lines A15–A13.

Bits 8–6 = 000 of the first word of an instruction indicate that the instruction is in the general instruction category. Bits 5–0 of the F-line operation code sometimes encode a standard M68000 effective address specifier (refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual*). During the execution of a cpGEN instruction, the coprocessor can use a coprocessor response primitive to request that the MC68020/EC020 perform an effective address calculation necessary for that instruction. Using the effective address specifier field of the F-line operation code, the processor then determines the effective addressing mode. If a coprocessor never requests effective address calculation, bits 5–0 can have any value (don't cares).

The second word of the general type instruction is the coprocessor command word. The main processor writes this command word to the command CIR to initiate execution of the instruction by the coprocessor.

An instruction in the coprocessor general instruction category optionally includes a number of extension words following the coprocessor command word. These words can provide additional information required for the coprocessor instruction. For example, if

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

the coprocessor requests that the MC68020/EC020 calculate an effective address during coprocessor instruction execution, information required for the calculation must be included in the instruction format as effective address extension words.

7.2.1.2 PROTOCOL. The execution of a cpGEN instruction follows the protocol shown in Figure 7-7. The main processor initiates communication with the coprocessor by writing the instruction command word to the command CIR. The coprocessor decodes the command word to begin processing the cpGEN instruction. Coprocessor design determines the interpretation of the coprocessor command word; the MC68020/EC020 does not attempt to decode it.

While the coprocessor is executing an instruction, it requests any required services from and communicates status to the main processor by placing coprocessor response primitive codes in the response CIR. After writing to the command CIR, the main processor reads the response CIR and responds appropriately. When the coprocessor has completed the execution of an instruction or no longer needs the services of the main processor to execute the instruction, it provides a response to release the main processor. The main processor can then execute the next instruction in the instruction stream. However, if a trace exception is pending, the MC68020/EC020 does not terminate communication with the coprocessor until the coprocessor indicates that it has completed all processing associated with the cpGEN instruction (refer to **7.5.2.5 Trace Exceptions**).

The coprocessor interface protocol shown in Figure 7-7 allows the coprocessor to define the operation of each coprocessor general type instruction. That is, the main processor initiates the instruction execution by writing the instruction command word to the command CIR and by reading the response CIR to determine its next action. The execution of the coprocessor instruction is then defined by the internal operation of the coprocessor and by its use of response primitives to request services from the main processor. This instruction protocol allows a wide range of operations to be implemented in the general instruction category.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

MAIN PROCESSOR		COPROCESSOR
M1 RECOGNIZE COPROCESSOR INSTRUCTION F-LINE OPERATION WORD		
M2 WRITE COPROCESSOR COMMAND WORD TO COMMAND CIR	→	C1 DECODE COMMAND WORD AND INITIATE COMMAND EXECUTION
		C2 WHILE (MAIN PROCESSOR SERVICE IS REQUIRED) DO STEPS 1) AND 2) BELOW
M3 READ COPROCESSOR RESPONSE PRIMITIVE CODE FROM RESPONSE CIR 1) PERFORM SERVICE REQUESTED BY RESPONSE PRIMITIVE 2) IF (COPROCESSOR RESPONSE PRIMITIVE INDICATES "COME AGAIN") GO TO M3 (SEE NOTE 1)	↔	1) REQUEST SERVICE BY PLACING APPROPRIATE RESPONSE PRIMITIVE CODE IN RESPONSE CIR 2) RECEIVE SERVICE FROM MAIN PROCESSOR
		C3 REFLECT "NO COME AGAIN" IN RESPONSE CIR
		C4 COMPLETE COMMAND EXECUTION
M4 PROCEED WITH EXECUTION OF NEXT INSTRUCTION (SEE NOTE 2)		C5 REFLECT "PROCESSING FINISHED" STATUS IN RESPONSE CIR

NOTES: 1. "Come Again" indicates that further service of the main processor is being requested by the coprocessor.
2. The next instruction should be the operation word pointed to by the ScanPC at this point. The operation of the MC68020/EC020 ScanPC is discussed in **7.4.1 ScanPC**.

Figure 7-7. Coprocessor Interface Protocol for General Category Instructions

7.2.2 Coprocessor Conditional Instructions

The conditional instruction category provides program control based on the operations of the coprocessor. The coprocessor evaluates a condition and returns a true/false indicator to the main processor. The main processor completes the execution of the instruction based on this true/false condition indicator.

The implementation of instructions in the conditional category promotes efficient use of both the main processor and the coprocessor hardware. The condition specified for the instruction is related to the coprocessor operation and is therefore evaluated by the coprocessor. However, the instruction completion following the condition evaluation is directly related to the operation of the main processor. The main processor performs the change of flow, the setting of a byte, or the TRAP operation, since its architecture explicitly implements these operations for its instruction set.

Figure 7-8 shows the protocol for a conditional category coprocessor instruction. The main processor initiates execution of an instruction in this category by writing a condition selector to the condition CIR. The coprocessor decodes the condition selector to determine the condition to evaluate. The coprocessor can use response primitives to request that the main processor provide services required for the condition evaluation.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

After evaluating the condition, the coprocessor returns a true/false indicator to the main processor by placing a null primitive (refer to **7.4.4 Null Primitive**) in the response CIR. The main processor completes the coprocessor instruction execution when it receives the condition indicator from the coprocessor.

MAIN PROCESSOR		COPROCESSOR
M1 RECOGNIZE COPROCESSOR INSTRUCTION F-LINE OPERATION WORD		
M2 WRITE COPROCESSOR CONDITION SELECTOR TO CONDITION CIR	→	C1 DECODE COMMAND WORD AND INITIATE COMMAND EXECUTION
		C2 WHILE (MAIN PROCESSOR SERVICE IS REQUIRED) DO STEPS 1) AND 2) BELOW
M3 READ COPROCESSOR RESPONSE PRIMITIVE CODE FROM RESPONSE CIR 1) PERFORM SERVICE REQUESTED BY RESPONSE PRIMITIVE 2) IF (COPROCESSOR RESPONSE PRIMITIVE INDICATES "COME AGAIN") GO TO M3 (SEE NOTE)	↔	1) REQUEST SERVICE BY PLACING APPROPRIATE RESPONSE PRIMITIVE CODE IN RESPONSE CIR 2) RECEIVE SERVICE FROM MAIN PROCESSOR
		C3 COMPLETE CONDITION EVALUATION
		C4 REFLECT "NO COME AGAIN" STATUS WITH TRUE/FALSE CONDITION INDICATOR IN RESPONSE CIR
M4 COMPLETE EXECUTION OF INSTRUCTION BASED ON THE TRUE/FALSE CONDITION INDICATOR RETURNED IN THE RESPONSE CIR		

NOTE: All coprocessor response primitives, except the Null primitive, that allow the "Come Again" primitive attribute must indicate "Come Again" when used during the execution of a conditional category instruction. If a "Come Again" attribute is not indicated in one of these primitives, the main processor will initiate protocol violation exception processing (see **7.5.2.1 Protocol Violations**).

Figure 7-8. Coprocessor Interface Protocol for Conditional Category Instructions

7

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.2.2.1 BRANCH ON COPROCESSOR CONDITION INSTRUCTION. The conditional instruction category includes two formats of the M68000 family branch instruction. These instructions branch on conditions related to the coprocessor operation. They execute similarly to the conditional branch instructions provided in the M68000 family instruction set.

7.2.2.1.1 Format. Figure 7-9 shows the format of the branch on coprocessor condition instruction that provides a word-length displacement. Figure 7-10 shows the format of this instruction that includes a long-word displacement.

15	14	13	12	11 — 9	8	7	6	5 — 0
1	1	1	1	CpID	0	1	0	CONDITION SELECTOR
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS								
DISPLACEMENT								

Figure 7-9. Branch on Coprocessor Condition Instruction Format (cpBcc.W)

15	14	13	12	11 — 9	8	7	6	5 — 0
1	1	1	1	CpID	0	1	1	CONDITION SELECTOR
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS								
DISPLACEMENT — HIGH								
DISPLACEMENT — LOW								

Figure 7-10. Branch on Coprocessor Condition Instruction Format (cpBcc.L)

The first word of the branch on coprocessor condition instruction is the F-line operation word. Bits 15–12 = 1111 and bits 11–9 contain the CpID code of the coprocessor that is to evaluate the condition. The value in bits 8–6 identifies either the word or the long-word displacement format of the branch instruction, which is specified by the cpBcc.W or cpBcc.L mnemonic, respectively. Bits 5–0 of the F-line operation word contain the coprocessor condition selector field. The MC68020/EC020 writes the entire operation word to the condition CIR to initiate execution of the branch instruction by the coprocessor. The coprocessor uses bits 5–0 to determine which condition to evaluate.

If the coprocessor requires additional information to evaluate the condition, the branch instruction format can include this information in extension words. Following the F-line operation word, the number of extension words is determined by the coprocessor design. The final word(s) of the cpBcc instruction format contains the displacement used by the main processor to calculate the destination address when the branch is taken.

7.2.2.1.2 Protocol. Figure 7-8 shows the protocol for the cpBcc.L and cpBcc.W instructions. The main processor initiates the instruction by writing the F-line operation word to the condition CIR to transfer the condition selector to the coprocessor. The main processor then reads the response CIR to determine its next action. The coprocessor can

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

return a response primitive to request services necessary to evaluate the condition. If the coprocessor returns the false condition indicator, the main processor executes the next instruction in the instruction stream. If the coprocessor returns the true condition indicator, the main processor adds the displacement to the MC68020/EC020 scanPC (refer to **7.4.1 ScanPC**) to determine the address of the next instruction for the main processor to execute. The scanPC must be pointing to the location of the first word of the displacement in the instruction stream when the address is calculated. The displacement is a twos-complement integer that can be either a 16-bit word or a 32-bit long word. The main processor sign-extends the 16-bit displacement to a long-word value for the destination address calculation.

7.2.2.2 SET ON COPROCESSOR CONDITION INSTRUCTION. The set on coprocessor condition instruction sets or resets a flag (a data alterable byte) according to a condition evaluated by the coprocessor. The operation of this instruction type is similar to the operation of the Scc instruction in the M68000 family instruction set. Although the Scc instruction and the cpScc instruction do not explicitly cause a change of program flow, they are often used to set flags that control program flow.

7.2.2.2.1 Format. Figure 7-11 shows the format of the set on coprocessor condition instruction, denoted by the cpScc mnemonic.

15	14	13	12	11 … 9	8	7	6	5 … 0
1	1	1	1	CpID	0	0	1	EFFECTIVE ADDRESS
RESERVED								CONDITION SELECTOR
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS								
OPTIONAL EFFECTIVE ADDRESS EXTENSION WORDS (0–5 WORDS)								

Figure 7-11. Set on Coprocessor Condition Instruction Format (cpScc)

The first word of the cpScc instruction, the F-line operation word, contains the CpID field in bits 11–9 and 001 in bits 8–6 to identify the cpScc instruction. Bits 5–0 of the F-line operation word are used to encode an M68000 family effective addressing mode (refer to M68000PM/AD, *M68000 Family Programmer's Reference Manual*).

The second word of the cpScc instruction format contains the coprocessor condition selector field in bits 5–0. Bits 15–6 of this word are reserved by Motorola and should be zero to ensure compatibility with future M68000 products. This word is written to the condition CIR to initiate execution of the cpScc instruction.

If the coprocessor requires additional information to evaluate the condition, the instruction can include extension words to provide this information. The number of these extension words, which follow the word containing the coprocessor condition selector field, is determined by the coprocessor design.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The final portion of the cpScc instruction format contains zero to five effective address extension words. These words contain any additional information required to calculate the effective address specified by bits 5–0 of the F-line operation word.

7.2.2.2.2 Protocol. Figure 7-8 shows the protocol for the cpScc instruction. The MC68020/EC020 transfers the condition selector to the coprocessor by writing the word following the F-line operation word to the condition CIR. The main processor then reads the response CIR to determine its next action. The coprocessor can return a response primitive to request services necessary to evaluate the condition. The operation of the cpScc instruction depends on the condition evaluation indicator returned to the main processor by the coprocessor. When the coprocessor returns the false condition indicator, the main processor evaluates the effective address specified by bits 5–0 of the F-line operation word and sets the byte at that effective address to FALSE (all bits cleared). When the coprocessor returns the true condition indicator, the main processor sets the byte at the effective address to TRUE (all bits set to one).

7.2.2.3 TEST COPROCESSOR CONDITION, DECREMENT, AND BRANCH INSTRUCTION. The operation of the test coprocessor condition, decrement, and branch instruction is similar to that of the DBcc instruction provided in the M68000 family instruction set. This operation uses a coprocessor-evaluated condition and a loop counter in the main processor. It is useful for implementing DO UNTIL constructs used in many high-level languages.

7.2.2.3.1 Format. Figure 7-12 shows the format of the test coprocessor condition, decrement, and branch instruction, denoted by the cpDBcc mnemonic.

15	14	13	12	11 9	8	7	6	5	4	3	2 0
1	1	1	1	CpID	0	0	1	0	0	1	REGISTER
RESERVED								CONDITION SELECTOR			
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS											
DISPLACEMENT											

Figure 7-12. Test Coprocessor Condition, Decrement, and Branch Instruction Format (cpDBcc)

The first word of the cpDBcc instruction, F-line operation word, contains the CpID field in bits 11–9 and 001001 in bits 8–3 to identify the cpDBcc instruction. Bits 2–0 of this operation word specify the main processor data register used as the loop counter during the execution of the instruction.

The second word of the cpDBcc instruction format contains the coprocessor condition selector field in bits 5–0 and should contain zeros in bits 15–6 (reserved by Motorola) to maintain compatibility with future M68000 products. This word is written to the condition CIR to initiate execution of the cpDBcc instruction.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

If the coprocessor requires additional information to evaluate the condition, the cpDBcc instruction can include this information in extension words. These extension words follow the word containing the coprocessor condition selector field in the cpDBcc instruction format.

The last word of the instruction contains the displacement for the cpDBcc instruction. This displacement is a twos-complement 16-bit value that is sign-extended to long-word size when it is used in a destination address calculation.

7.2.2.3.2 Protocol. Figure 7-8 shows the protocol for the cpDBcc instructions. The MC68020/EC020 transfers the condition selector to the coprocessor by writing the word following the operation word to the condition CIR. The main processor then reads the response CIR to determine its next action. The coprocessor can use a response primitive to request any services necessary to evaluate the condition. If the coprocessor returns the true condition indicator, the main processor executes the next instruction in the instruction stream. If the coprocessor returns the false condition indicator, the main processor decrements the low-order word of the register specified by bits 2–0 of the F-line operation word. If this register contains minus one (–1) after being decremented, the main processor executes the next instruction in the instruction stream. If the register does not contain minus one (–1) after being decremented, the main processor branches to the destination address to continue instruction execution.

The MC68020/EC020 adds the displacement to the scanPC (refer to **7.4.1 ScanPC**) to determine the address of the next instruction. The scanPC must point to the 16-bit displacement in the instruction stream when the destination address is calculated.

7.2.2.4 TRAP ON COPROCESSOR CONDITION INSTRUCTION. The trap on coprocessor condition instruction allows the programmer to initiate exception processing based on conditions related to the coprocessor operation.

7.2.2.4.1 Format. Figure 7-13 shows the format of the trap on coprocessor condition instruction, denoted by the cpTRAPcc mnemonic.

15	14	13	12	11 9	8	7	6	5	4	3	2 0
1	1	1	1	CpID	0	0	1	1	1	1	OPMODE
RESERVED								CONDITION SELECTOR			
OPTIONAL COPROCESSOR-DEFINED EXTENSION WORDS											
OPTIONAL WORD											
OR LONG-WORD OPERAND											

Figure 7-13. Trap on Coprocessor Condition Instruction Format (cpTRAPcc)

The first word of the cpTRAPcc instruction, the F-line operation word contains the CpID field in bits 11–9 and 001111 in bits 8–3 to identify the cpTRAPcc instruction. Bits 2–0 of the cpTRAPcc F-line operation word specify the opmode, which selects the instruction format. The instruction format can include zero, one, or two operand words.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The second word of the cpTRAPcc instruction format contains the coprocessor condition selector in bits 5–0 and should contain zeros in bits 15–6 (these bits are reserved by Motorola) to maintain compatibility with future M68000 products. This word is written to the condition CIR to initiate execution of the cpTRAPcc instruction.

If the coprocessor requires additional information to evaluate a condition, the instruction can include this information in extension words. These extension words follow the word containing the coprocessor condition selector field in the cpTRAPcc instruction format.

The operand words of the cpTRAPcc F-line operation word follow the coprocessor-defined extension words. These operand words are not explicitly used by the MC68020/EC020, but can be used to contain information referenced by the cpTRAPcc exception handling routines. The valid encodings for bits 2–0 of the F-line operation word and the corresponding numbers of operand words are listed in Table 7-1. Other encodings of these bits are invalid for the cpTRAPcc instruction.

Table 7-1. cpTRAPcc Opmode Encodings

Opmode	Operand Words in Instruction Format
010	One
011	Two
100	Zero

7.2.2.4.2 Protocol. Figure 7-8 shows the protocol for the cpTRAPcc instructions. The MC68020/EC020 transfers the condition selector to the coprocessor by writing the word following the operation word to the condition CIR. The main processor then reads the response CIR to determine its next action. The coprocessor can return a response primitive to request any services necessary to evaluate the condition. If the coprocessor returns the true condition indicator, the main processor initiates exception processing for the cpTRAPcc exception (refer to **7.5.2.4 cpTRAPcc Instruction Traps**). If the coprocessor returns the false condition indicator, the main processor executes the next instruction in the instruction stream.

7.2.3 Coprocessor Context Save and Restore Instructions

The coprocessor context save and context restore instruction categories in the M68000 coprocessor interface support multitasking programming environments. In a multitasking environment, the context of a coprocessor may need to be changed asynchronously with respect to the operation of that coprocessor. That is, the coprocessor may be interrupted at any point in the execution of an instruction in the general or conditional category to begin context change operations.

In contrast to the general and conditional instruction categories, the context save and context restore instruction categories do not use the coprocessor response primitives. A set of format codes defined by the M68000 coprocessor interface communicates status

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

information to the main processor during the execution of these instructions. These coprocessor format codes are discussed in detail in **7.2.3.2 Coprocessor Format Words**.

7.2.3.1 COPROCESSOR INTERNAL STATE FRAMES. The context save (cpSAVE) and context restore (cpRESTORE) instructions transfer an internal coprocessor state frame between memory and a coprocessor. This internal coprocessor state frame represents the state of coprocessor operations. Using the cpSAVE and cpRESTORE instructions, it is possible to interrupt coprocessor operation, save the context associated with the current operation, and initiate coprocessor operations with a new context.

A cpSAVE instruction stores a coprocessor internal state frame as a sequence of long-word entries in memory. Figure 7-14 shows the format of a coprocessor state frame. The format and length fields of the coprocessor state frame format comprise the format word. During execution of the cpSAVE instruction, the MC68020/EC020 calculates the state frame effective address from information in the operation word of the instruction and stores a format word at this effective address. The processor writes the long words that form the coprocessor state frame to descending memory addresses, beginning with the address specified by the sum of the effective address and the length field multiplied by four. During execution of the cpRESTORE instruction, the MC68020/EC020 reads the state frame from ascending addresses beginning with the effective address specified in the instruction operation word.

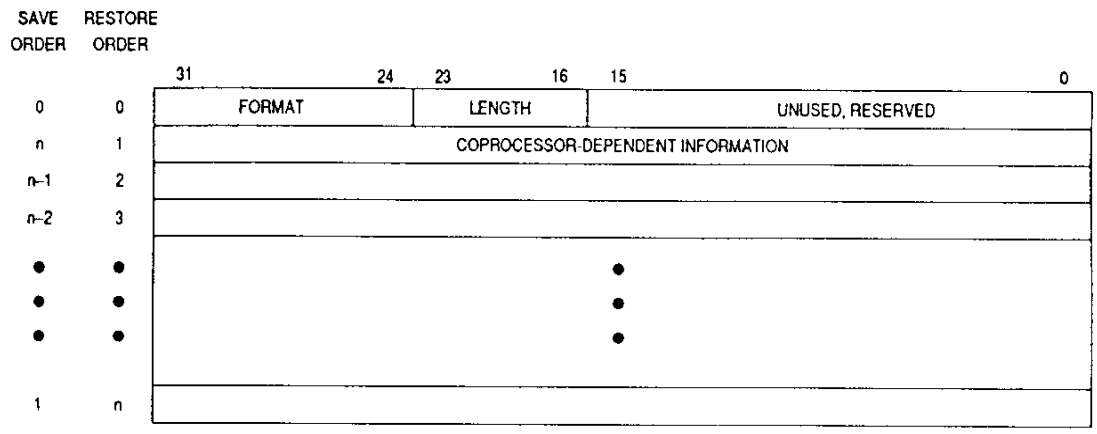


Figure 7-14. Coprocessor State Frame Format in Memory

The processor stores the coprocessor format word at the lowest address of the state frame in memory, and this word is the first word transferred for both the cpSAVE and cpRESTORE instructions. The word following the format word does not contain information relevant to the coprocessor state frame, but serves to keep the information in the state frame a multiple of four bytes in size. The number of entries following the format word (at higher addresses) is determined by the format word length for a given coprocessor state.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The information in a coprocessor state frame describes a context of operation for that coprocessor. This description of a coprocessor context includes the program invisible state information and, optionally, the program visible state information. The program invisible state information consists of any internal registers or status information that cannot be accessed by the program but is necessary for the coprocessor to continue its operation at the point of suspension. Program visible state information includes the contents of all registers that appear in the coprocessor programming model and that can be directly accessed using the coprocessor instruction set. The information saved by the cpSAVE instruction must include the program invisible state information. If cpGEN instructions are provided to save the program visible state of the coprocessor, the cpSAVE and cpRESTORE instructions should only transfer the program invisible state information to minimize interrupt latency during a save or restore operation.

7.2.3.2 COPROCESSOR FORMAT WORDS. The coprocessor communicates status information to the main processor during the execution of cpSAVE and cpRESTORE instructions using coprocessor format words. The format words defined for the M68000 coprocessor interface are listed in Table 7-2.

Table 7-2. Coprocessor Format Word Encodings

Format Code	Length	Meaning
$00	$xx	Empty/Reset
$01	$xx	Not Ready, Come Again
$02	$xx	Invalid Format
$03–$0F	$xx	Undefined, Reserved
$10–$FF	Length	Valid Format, Coprocessor Defined

xx: Don't care

The upper byte of the coprocessor format word contains the code used to communicate coprocessor status information to the main processor. The MC68020/EC020 recognizes four types of format words: empty/reset, not ready, invalid format, and valid format. The MC68020/EC020 interprets the reserved format codes ($03–$0F) as invalid format words. The lower byte of the coprocessor format word specifies the size in bytes (which must be a multiple of four) of the coprocessor state frame. This value is only relevant when the code byte contains the valid format code (refer to **7.2.3.2.4 Valid Format Word**).

7.2.3.2.1 Empty/Reset Format Word. The coprocessor returns the empty/reset format code during a cpSAVE instruction to indicate that the coprocessor contains no user-specific information. That is, no coprocessor instructions have been executed since either a previous cpRESTORE of an empty/reset format code or the previous hardware reset. If the main processor reads the empty/reset format word from the save CIR during the initiation of a cpSAVE instruction, it stores the format word at the effective address specified in the cpSAVE instruction and executes the next instruction.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

When the main processor reads the empty/reset format word from memory during the execution of the cpRESTORE instruction, it writes the format word to the restore CIR. The main processor then reads the restore CIR and, if the coprocessor returns the empty/reset format word, executes the next instruction. The main processor can then initialize the coprocessor by writing the empty/reset format code to restore the CIR. When the coprocessor receives the empty/reset format code, it terminates any current operations and waits for the main processor to initiate the next coprocessor instruction. In particular, after the cpRESTORE of the empty/reset format word, the execution of a cpSAVE should cause the empty/reset format word to be returned when a cpSAVE instruction is executed before any other coprocessor instructions. Thus, an empty/reset state frame consists only of the format word and the following reserved word in memory (refer to Figure 7-14).

7.2.3.2.2 Not-Ready Format Word. When the main processor initiates a cpSAVE instruction by reading the save CIR, the coprocessor can delay the save operation by returning a not-ready format word. The main processor then services any pending interrupts and reads the save CIR again. The not-ready format word delays the save operation until the coprocessor is ready to save its internal state. The cpSAVE instruction can suspend execution of a general or conditional coprocessor instruction; the coprocessor can resume execution of the suspended instruction when the appropriate state is restored with a cpRESTORE. If no further main processor services are required to complete coprocessor instruction execution, it may be more efficient to complete the instruction and thus reduce the size of the saved state. The coprocessor designer should consider the efficiency of completing the instruction or of suspending and later resuming the instruction when the main processor executes a cpSAVE instruction.

When the main processor initiates a cpRESTORE instruction by writing a format word to the restore CIR, the coprocessor should usually terminate any current operations and restore the state frame supplied by the main processor. Thus, the not-ready format word should usually not be returned by the coprocessor during the execution of a cpRESTORE instruction. If the coprocessor must delay the cpRESTORE operation for any reason, it can return the not-ready format word when the main processor reads the restore CIR. If the main processor reads the not-ready format word from the restore CIR during the cpRESTORE instruction, it reads the restore CIR again without servicing any pending interrupts.

7.2.3.2.3 Invalid Format Word. When the format word placed in the restore CIR to initiate a cpRESTORE instruction does not describe a valid coprocessor state frame, the coprocessor returns the invalid format word in the restore CIR. When the main processor reads this format word during the cpRESTORE instruction, it sets the abort bit in the control CIR and initiates format error exception processing.

A coprocessor usually should not place an invalid format word in the save CIR when the main processor initiates a cpSAVE instruction. A coprocessor, however, may not be able to support the initiation of a cpSAVE instruction while it is executing a previously initiated cpSAVE or cpRESTORE instruction. In this situation, the coprocessor can return the invalid format word when the main processor reads the save CIR to initiate the cpSAVE instruction while either another cpSAVE or cpRESTORE instruction is executing. If the

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

main processor reads an invalid format word from the save CIR, it writes the abort mask to the control CIR and initiates format error exception processing (refer to **7.5.1.5 Format Errors**).

7.2.3.2.4 Valid Format Word. When the main processor reads a valid format word from the save CIR during the cpSAVE instruction, it uses the length field to determine the size of the coprocessor state frame to save. The length field in the lower eight bits of a format word is relevant only in a valid format word. During the cpRESTORE instruction, the main processor uses the length field in the format word read from the effective address in the instruction to determine the size of the coprocessor state frame to restore.

The length field of a valid format word, representing the size of the coprocessor state frame, must contain a multiple of four. If the main processor detects a value that is not a multiple of four in a length field during the execution of a cpSAVE or cpRESTORE instruction, the main processor writes the abort mask (refer to **7.2.3.2.3 Invalid Format Word**) to the control CIR and initiates format error exception processing.

7.2.3.3 COPROCESSOR CONTEXT SAVE INSTRUCTION. The M68000 coprocessor context save instruction category consists of one instruction. The coprocessor context save instruction, denoted by the cpSAVE mnemonic, saves the context of a coprocessor dynamically without relation to the execution of coprocessor instructions in the general or conditional instruction categories. During the execution of a cpSAVE instruction, the coprocessor communicates status information to the main processor by using the coprocessor format codes.

7.2.3.3.1 Format. Figure 7-15 shows the format of the cpSAVE instruction. The first word of the instruction, the F-line operation word, contains the CpID code in bits 11–9 and an M68000 effective address code in bits 5–0. The effective address encoded in the cpSAVE instruction is the address at which the state frame associated with the current context of the coprocessor is saved in memory.

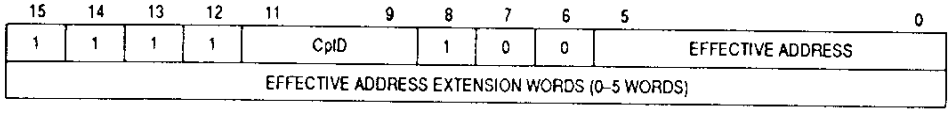


Figure 7-15. Coprocessor Context Save Instruction Format (cpSAVE)

The control alterable and predecrement addressing modes are valid for the cpSAVE instruction. Other addressing modes cause the MC68020/EC020 to initiate F-line emulator exception processing as described in **7.5.2.2 F-Line Emulator Exceptions**.

The instruction can include as many as five effective address extension words following the F-line operation word. These words contain any additional information required to calculate the effective address specified by bits 5–0 of the F-line operation word.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.2.3.3.2 Protocol. Figure 7-16 shows the protocol for the coprocessor context save instruction. The main processor initiates execution of the cpSAVE instruction by reading the save CIR. Thus, the cpSAVE instruction is the only coprocessor instruction that begins by reading from a CIR. All other coprocessor instructions write to a CIR to initiate execution of the instruction by the coprocessor. The coprocessor communicates status information associated with the context save operation to the main processor by placing coprocessor format codes in the save CIR.

MAIN PROCESSOR	COPROCESSOR
M1 RECOGNIZE COPROCESSOR INSTRUCTION F-LINE OPERATION WORD	
M2 READ SAVE CIR TO INITIATE THE cpSAVE INSTRUCTION	C1 IF (NOT READY TO BEGIN CONTEXT SAVE OPERATION) DO STEPS 1) AND 2) BELOW 1) PLACE NOT READY FORMAT CODE IN SAVE CIR 2) SUSPEND OR COMPLETE CURRENT OPERATIONS
M3 IF (FORMAT = NOT READY) DO STEPS 1) AND 2) BELOW ← 1) SERVICE PENDING INTERRUPTS 2) GO TO M2	
M4 EVALUATE EFFECTIVE ADDRESS SPECIFIED IN F-LINE OPWORD AND STORE FORMAT WORD AT EFFECTIVE ADDRESS	C2 PLACE APPROPRIATE FORMAT WORD IN SAVE CIR C3 TRANSFER NUMBER OF BYTES INDICATED IN FORMAT WORD THROUGH OPERAND CIR
M5 IF (FORMAT = EMPTY) GO TO M6; ELSE, TRANSFER NUMBER OF BYTES INDICATED IN FORMAT WORD FROM OPERAND CIR TO EFFECTIVE ADDRESS	
M6 PROCEED WITH EXECUTION OF NEXT INSTRUCTION	

Figure 7-16. Coprocessor Context Save Instruction Protocol

If the coprocessor is not ready to suspend its current operation when the main processor reads the save CIR, it returns a not-ready format code. The main processor services any pending interrupts and then reads the save CIR again. After placing the not-ready format code in the save CIR, the coprocessor should either suspend or complete the instruction it is currently executing.

Once the coprocessor has suspended or completed the instruction it is executing, it places a format code representing the internal coprocessor state in the save CIR. When the main processor reads the save CIR, it transfers the format word to the effective address specified in the cpSAVE instruction. The lower byte of the coprocessor format word specifies the number of bytes of state information, not including the format word and associated null word, to be transferred from the coprocessor to the effective address specified. If the state information is not a multiple of four bytes in size, the MC68020/EC020 initiates format error exception processing (refer to **7.5.1.5 Format Errors**). The coprocessor and main processor coordinate the transfer of the internal state of the coprocessor using the operand CIR. The MC68020/EC020 completes the coprocessor context save by repeatedly reading the operand CIR and writing the

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

information obtained into memory until all the bytes specified in the coprocessor format word have been transferred. Following a cpSAVE instruction, the coprocessor should be in an idle state—that is, not executing any coprocessor instructions.

The cpSAVE instruction is a privileged instruction. When the MC68020/EC020 identifies a cpSAVE instruction, it checks the S-bit in the SR to determine whether it is operating at the supervisor privilege level. If the MC68020/EC020 attempts to execute a cpSAVE instruction while at the user privilege level (S-bit in the SR is clear), it initiates privilege violation exception processing without accessing any of the CIRs (refer to **7.5.2.3 Privilege Violations**).

The MC68020/EC020 initiates format error exception processing if it reads an invalid format word (or a valid format word whose length field is not a multiple of four bytes) from the save CIR during the execution of a cpSAVE instruction (refer to **7.2.3.2.3 Invalid Format Word**). The MC68020/EC020 writes an abort mask (refer to **7.2.3.2.3 Invalid Format Word**) to the control CIR to abort the coprocessor instruction prior to beginning exception processing. Figure 7-16 does not include this case since a coprocessor usually returns either a not-ready or a valid format code in the context of the cpSAVE instruction. The coprocessor can return the invalid format word, however, if a cpSAVE is initiated while the coprocessor is executing a cpSAVE or cpRESTORE instruction and the coprocessor is unable to support the suspension of these two instructions.

7.2.3.4 COPROCESSOR CONTEXT RESTORE INSTRUCTION. The M68000 coprocessor context restore instruction category includes one instruction. The coprocessor context restore instruction, denoted by the cpRESTORE mnemonic, forces a coprocessor to terminate any current operations and to restore a former state. During execution of a cpRESTORE instruction, the coprocessor can communicate status information to the main processor by placing format codes in the restore CIR.

7.2.3.4.1 Format. Figure 7-17 shows the format of the cpRESTORE instruction.

15	14	13	12	11 9	8	7	6	5 0
1	1	1	1	CpID	1	0	1	EFFECTIVE ADDRESS
EFFECTIVE ADDRESS EXTENSION WORDS (0–5 WORDS)								

Figure 7-17. Coprocessor Context Restore Instruction Format (cpRESTORE)

The first word of the instruction, the F-line operation word, contains the CpID code in bits 11–9 and an M68000 effective addressing code in bits 5–0. The effective address encoded in the cpRESTORE instruction is the starting address in memory where the coprocessor context is stored. The effective address is that of the coprocessor format word that applies to the context to be restored to the coprocessor.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The instruction can include as many as five effective address extension words following the F-line operation word in the cpRESTORE instruction format. These words contain any additional information required to calculate the effective address specified by bits 5–0 of the F-line operation word.

All memory addressing modes except the predecrement addressing mode are valid. Invalid effective address encodings cause the MC68020/EC020 to initiate F-line emulator exception processing (refer to **7.5.2.2 F-Line Emulator Exceptions**).

7.2.3.4.2 Protocol. Figure 7-18 shows the protocol for the coprocessor context restore instruction. When the main processor executes a cpRESTORE instruction, it first reads the coprocessor format word from the effective address in the instruction. This format word contains a format code and a length field. During cpRESTORE operation, the main processor retains a copy of the length field to determine the number of bytes to be transferred to the coprocessor during the cpRESTORE operation and writes the format word to the restore CIR to initiate the coprocessor context restore.

MAIN PROCESSOR		COPROCESSOR
M1 RECOGNIZE COPROCESSOR INSTRUCTION F-LINE OPERATION WORD		
M2 READ COPROCESSOR FORMAT CODE FROM EFFECTIVE ADDRESS SPECIFIED IN OPERATION WORD		
M3 WRITE COPROCESSOR FORMAT WORD TO RESTORE CIR	→	C1 TERMINATE CURRENT OPERATIONS AND EVALUATE FORMAT WORD
M4 READ RESTORE CIR	↔	C2 IF (INVALID FORMAT) PLACE INVALID FORMAT CODE IN THE RESTORE CIR
M5 IF (FORMAT = INVALID FORMAT) WRITE $0001 ABORT CODE TO CONTROL CIR AND INITIATE FORMAT ERROR EXCEPTION PROCESSING (SEE NOTE 1)		C3 IF (VALID FORMAT) RECEIVE NUMBER OF BYTES INDICATED IN FORMAT WORD THROUGH OPERAND CIR
M6 IF (FORMAT = EMPTY/RESET) GO TO M7; ELSE, TRANSFER NUMBER OF BYTES SPECIFIED BY FORMAT WORD TO OPERAND CIR (SEE NOTE 2)		
M7 PROCEED WITH EXECUTION OF NEXT INSTRUCTION		

NOTES: 1. See **7.6.1.5 Format Error**.
2. The MC68020/EC020 uses the length field in the format word read during M2 to determine the number of bytes to read from memory and write to the operand CIR.

Figure 7-18. Coprocessor Context Restore Instruction Protocol

When the coprocessor receives the format word in the restore CIR, it must terminate any current operations and evaluate the format word. If the format word represents a valid coprocessor context as determined by the coprocessor design, the coprocessor returns the format word to the main processor through the restore CIR and prepares to receive the number of bytes specified in the format word through its operand CIR.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

After writing the format word to the restore CIR, the main processor continues cpRESTORE dialog by reading that same register. If the coprocessor returns a valid format word, the main processor transfers the number of bytes specified by the format word at the effective address to the operand CIR.

If the format word written to the restore CIR does not represent a valid coprocessor state frame, the coprocessor places an invalid format word in the restore CIR and terminates any current operations. The main processor receives the invalid format code, writes an abort mask (refer to **7.2.3.2.3 Invalid Format Word**) to the control CIR, and initiates format error exception processing (refer to **7.5.1.5 Format Errors**).

The cpRESTORE instruction is a privileged instruction. When the MC68020/EC020 accesses a cpRESTORE instruction, it checks the S-bit in the SR. If the MC68020/EC020 attempts to execute a cpRESTORE instruction while at the user privilege level (S-bit in the SR is clear), it initiates privilege violation exception processing without accessing any of the CIRs (refer to **7.5.2.3 Privilege Violations**).

7.3 COPROCESSOR INTERFACE REGISTER SET

The instructions of the M68000 coprocessor interface use registers of the CIR set to communicate with the coprocessor. These CIRs are not directly related to the coprocessor programming model.

Figure 7-4 is a memory map of the CIR set. The response, control, save, restore, command, condition, and operand registers must be included in a coprocessor interface that implements all four coprocessor instruction categories. The complete register model must be implemented if the system uses all coprocessor response primitives defined for the M68000 coprocessor interface.

The following paragraphs contain detailed descriptions of the registers.

7.3.1 Response CIR

The coprocessor uses the 16-bit response CIR to communicate all service requests (coprocessor response primitives) to the main processor. The main processor reads the response CIR to receive the coprocessor response primitives during the execution of instructions in the general and conditional instruction categories. The offset from the base address of the CIR set for the response CIR is $00. Refer to **7.4 Coprocessor Response Primitives** for additional information.

7.3.2 Control CIR

The main processor writes to the 2-bit control CIR to acknowledge coprocessor-requested exception processing or to abort the execution of a coprocessor instruction. The offset from the base address of the CIR set for the control CIR is $02. The control CIR occupies the two least significant bits of the word at that offset. The 14 most significant bits of the word are undefined and reserved by Motorola. Figure 7-19 shows the format of this register.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Figure 7-19. Control CIR Format

When the MC68020/EC020 receives one of the three take exception coprocessor response primitives, it acknowledges the primitive by setting the exception acknowledge bit (XA) in the control CIR. The MC68020/EC020 sets the abort bit (AB) in the control CIR to abort any coprocessor instruction in progress. (The 14 most significant bits of both masks are undefined.) The MC68020/EC020 aborts a coprocessor instruction when it detects one of the following exception conditions:

- An F-line emulator exception condition after reading a response primitive
- A privilege violation exception as it performs a supervisor check in response to a supervisor check primitive
- A format error exception when it receives an invalid format word or a valid format word that contains an invalid length

7.3.3 Save CIR

The coprocessor uses the 16-bit save CIR to communicate status and state frame format information to the main processor while executing a cpSAVE instruction. The main processor reads the save CIR to initiate execution of the cpSAVE instruction by the coprocessor. The offset from the base address of the CIR set for the save CIR is $04. Refer to **7.2.3.2 Coprocessor Format Words** for more information on the save CIR.

7.3.4 Restore CIR

The main processor initiates the cpRESTORE instruction by writing a coprocessor format word to the 16-bit restore register. During the execution of the cpRESTORE instruction, the coprocessor communicates status and state frame format information to the main processor through the restore CIR. The offset from the base address of the CIR set for the restore CIR is $06. Refer to **7.2.3.2 Coprocessor Format Words** for more information on the restore CIR.

7.3.5 Operation Word CIR

The main processor writes the F-line operation word of the instruction in progress to the 16-bit operation word CIR in response to a transfer operation word coprocessor response primitive (refer to **7.4.6 Transfer Operation Word Primitive**). The offset from the base address of the CIR set for the operation word CIR is $08.

7.3.6 Command CIR

The main processor initiates a coprocessor general category instruction by writing the instruction command word, which follows the instruction F line operation word in the instruction stream, to the 16-bit command CIR. The offset from the base address of the CIR set for the command CIR is $0A.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.3.7 Condition CIR

The main processor initiates a conditional category instruction by writing the condition selector to bits 5–0 of the 16-bit condition CIR. Bits 15–6 are undefined and reserved by Motorola. The offset from the base address of the CIR set for the condition CIR is $0E. Figure 7-20 shows the format of the condition CIR.

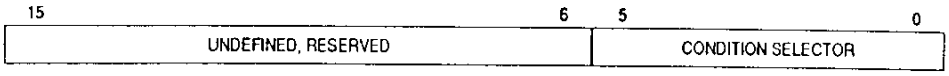


Figure 7-20. Condition CIR Format

7.3.8 Operand CIR

When the coprocessor requests the transfer of an operand, the main processor performs the transfer by reading from or writing to the 32-bit operand CIR. The offset from the base address of the CIR set for the operand CIR is $10.

The MC68020/EC020 aligns all operands transferred to and from the operand CIR to the most significant byte of this CIR. The processor performs a sequence of long-word transfers to read or write any operand larger than four bytes. If the operand size is not a multiple of four bytes, the portion remaining after the initial long-word transfer is aligned to the most significant byte of the operand CIR. Figure 7-21 shows the operand alignment used by the MC68020/EC020 when accessing the operand CIR.

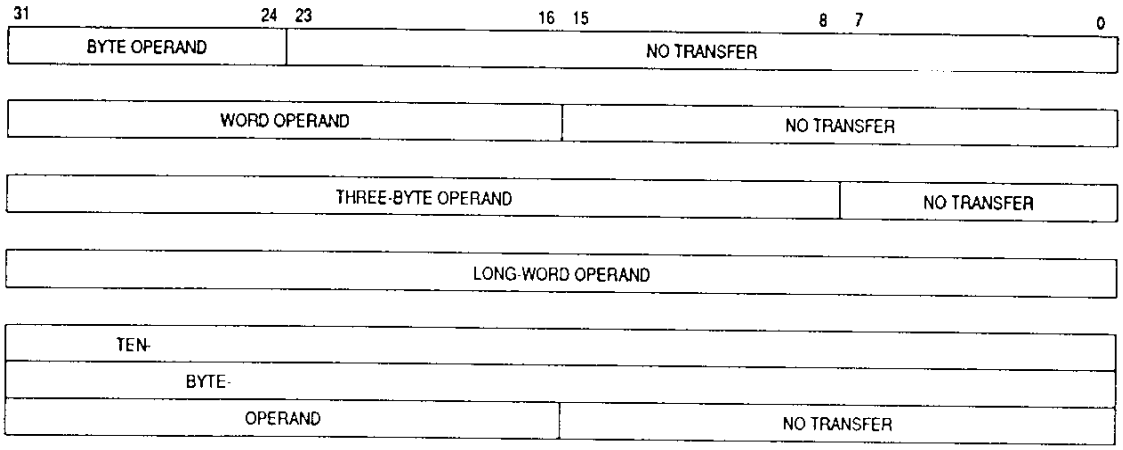


Figure 7-21. Operand Alignment for Operand CIR Accesses

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.3.9 Register Select CIR

When the coprocessor requests the transfer of one or more main processor registers or a group of coprocessor registers, the main processor reads the 16-bit register select CIR to identify the number or type of registers to be transferred. The offset from the base address of the CIR set for the register select CIR is $14. The format of this register depends on the primitive that is currently using it (refer to **7.4 Coprocessor Response Primitives**).

7.3.10 Instruction Address CIR

When the coprocessor requests the address of the instruction it is currently executing, the main processor transfers this address to the 32-bit instruction address CIR. Any transfer of the scanPC is also performed through the instruction address CIR (refer to **7.4.17 Transfer Status Register and ScanPC Primitive**). The offset from the base address of the CIR set for the instruction address CIR is $18.

7.3.11 Operand Address CIR

When a coprocessor requests an operand address transfer between the main processor and the coprocessor, the address is transferred through the 32-bit operand address CIR. The offset from the base address of the CIR set for the operand address CIR is $1C.

7.4 COPROCESSOR RESPONSE PRIMITIVES

The response primitives are primitive instructions that the coprocessor issues to the main processor during the execution of a coprocessor instruction. The coprocessor uses response primitives to communicate status information and service requests to the main processor. In response to an instruction command word written to the command CIR or a condition selector in the condition CIR, the coprocessor returns a response primitive in the response CIR. Within the general and conditional instruction categories, individual instructions are distinguished by the operation of the coprocessor hardware and by services specified by coprocessor response primitives and provided by the main processor.

Subsequent paragraphs, beginning with **7.4.2 Coprocessor Response Primitive General Format**, consist of detailed descriptions of the M68000 coprocessor response primitives supported by the MC68020/EC020. Any response primitive that the MC68020/EC020 does not recognize causes it to initiate protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**). This processing of undefined primitives supports emulation of extensions to the M68000 coprocessor response primitive set by the protocol violation exception handler. Exception processing related to the coprocessor interface is discussed in **7.5 Exceptions**.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.4.1 ScanPC

Several of the response primitives involve the scanPC, and many of them require the main processor to use it while performing services requested. These paragraphs describe the scanPC and its operation.

During the execution of a coprocessor instruction, the PC in the MC68020/EC020 contains the address of the F-line operation word of that instruction. A second register, called the scanPC, sequentially addresses the remaining words of the instruction.

If the main processor requires extension words to calculate an effective address or destination address of a branch operation, it uses the scanPC to address these extension words in the instruction stream. Also, if a coprocessor requests the transfer of extension words, the scanPC addresses the extension words during the transfer. As the processor references each word, it increments the scanPC to point to the next word in the instruction stream. When an instruction has completed, the processor transfers the value in the scanPC to the PC to address the operation word of the next instruction.

The value in the scanPC when the main processor reads the first response primitive after beginning to execute an instruction depends on the instruction being executed. For a cpGEN instruction, the scanPC points to the word following the coprocessor command word. For the cpBcc instructions, the scanPC points to the word following the instruction F-line operation word. For the cpScc, cpTRAPcc, and cpDBcc instructions, the scanPC points to the word following the coprocessor condition specifier word.

If a coprocessor implementation uses optional instruction extension words with a general or conditional instruction, the coprocessor must use these words consistently so that the scanPC is updated accordingly during the instruction execution. Specifically, during the execution of general category instructions, when the coprocessor terminates the instruction protocol, the MC68020/EC020 assumes that the scanPC is pointing to the operation word of the next instruction to be executed. During the execution of conditional category instructions, when the coprocessor terminates the instruction protocol, the MC68020/EC020 assumes that the scanPC is pointing to the word following the last of any coprocessor-defined extension words in the instruction format.

7.4.2 Coprocessor Response Primitive General Format

The M68000 coprocessor response primitives are encoded in a 16-bit word that is transferred to the main processor through the response CIR. Figure 7-22 shows the format of the coprocessor response primitives.

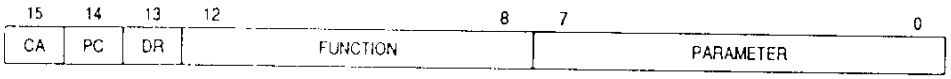


Figure 7-22. Coprocessor Response Primitive Format

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The encoding of bits 12–0 of a coprocessor response primitive depends on the individual primitive. Bits 15–13, however, specify optional additional operations that apply to most of the primitives defined for the M68000 coprocessor interface.

The CA bit specifies the come-again operation of the main processor. When the main processor reads a response primitive from the response CIR with the CA bit set, it performs the service indicated by the primitive and then reads the response CIR again. Using the CA bit, a coprocessor can transfer several response primitives to the main processor during the execution of a single coprocessor instruction.

The PC bit specifies the pass program counter operation. When the main processor reads a primitive with the PC bit set from the response CIR, the main processor immediately passes the current value in its program counter to the instruction address CIR as the first operation in servicing the primitive request. The value in the program counter is the address of the F-line operation word of the coprocessor instruction currently executing. The PC bit is implemented in all coprocessor response primitives currently defined for the M68000 coprocessor interface.

When an undefined primitive or a primitive that requests an illegal operation is passed to the main processor, the main processor initiates exception processing for either an F-line emulator or a protocol violation exception (refer to **7.5.2 Main-Processor-Detected Exceptions**). If the PC bit is set in one of these response primitives, however, the main processor passes the program counter to the instruction address CIR before it initiates exception processing.

When the main processor initiates a cpGEN instruction that can be executed concurrently with main processor instructions, the PC bit is usually set in the first primitive returned by the coprocessor. Since the main processor proceeds with instruction stream execution once the coprocessor releases it, the coprocessor must record the instruction address to support any possible exception processing related to the instruction. Exception processing related to concurrent coprocessor instruction execution is discussed in **7.5.1 Coprocessor-Detected Exceptions**.

The DR bit is the direction bit. It applies to operand transfers between the main processor and the coprocessor. If the DR bit is clear, the direction of transfer is from the main processor to the coprocessor (main processor write). If the DR bit is set, the direction of transfer is from the coprocessor to the main processor (main processor read). If the operation indicated by a given response primitive does not involve an explicit operand transfer, the value of this bit depends on the particular primitive encoding.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.4.3 Busy Primitive

The busy response primitive causes the main processor to reinitiate a coprocessor instruction. This primitive applies to instructions in the general and conditional categories. Figure 7-23 shows the format of the busy primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 7-23. Busy Primitive Format

The busy primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**.

Coprocessors that can operate concurrently with the main processor but cannot buffer write operations to their command or condition CIR use the busy primitive. A coprocessor may execute a cpGEN instruction concurrently with an instruction in the main processor. If the main processor attempts to initiate an instruction in the general or conditional instruction category while the coprocessor is executing a cpGEN instruction, the coprocessor can place the busy primitive in the response CIR. When the main processor reads this primitive, it services pending interrupts using a preinstruction exception stack frame (refer to Figure 7-41). The processor then restarts the general or conditional coprocessor instruction that it had attempted to initiate earlier.

The busy primitive should only be used in response to a write to the command or condition CIR. It should be the first primitive returned after the main processor attempts to initiate a general or conditional category instruction. In particular, the busy primitive should not be issued after program-visible resources have been altered by the instruction. (Program-visible resources include coprocessor and main processor program-visible registers and operands in memory, but not the scanPC.) The restart of an instruction after it has altered program-visible resources causes those resources to have inconsistent values when the processor reinitiates the instruction.

The MC68020/EC020 responds to the busy primitive differently in a special case that can occur during a breakpoint operation (refer to **Section 6 Exception Processing**). This special case occurs when a breakpoint acknowledge cycle initiates a coprocessor F-line instruction, the coprocessor returns the busy primitive in response to the instruction initiation, and an interrupt is pending. When these three conditions are met, the processor reexecutes the breakpoint acknowledge cycle after completion of interrupt exception processing. A design that uses a breakpoint to monitor the number of passes through a loop by incrementing or decrementing a counter may not work correctly under these conditions. This special case may cause several breakpoint acknowledge cycles to be executed during a single pass through a loop.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.4.4 Null Primitive

The null coprocessor response primitive communicates coprocessor status information to the main processor. This primitive applies to instructions in the general and conditional categories. Figure 7-24 shows the format of the null primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	0	IA	0	0	0	0	0	0	PF	TF

Figure 7-24. Null Primitive Format

The null primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The IA bit specifies the interrupts allowed optional operation. This bit determines whether the MC68020/EC020 services pending interrupts prior to rereading the response CIR after receiving a null primitive. Interrupts are allowed when the IA bit is set.

The PF bit shows the processing-finished status of the coprocessor. That is, PF = 1 indicates that the coprocessor has completed all processing associated with an instruction.

The TF bit indicates the true/false condition during execution of a conditional category instruction. TF = 1 is the true condition specifier; TF = 0 is the false condition specifier. The TF bit is only relevant for null primitives with CA = 0 that are used by the coprocessor during the execution of a conditional instruction.

The MC68020/EC020 processes a null primitive with CA = 1 in the same manner whether executing a general or conditional category coprocessor instruction. If the coprocessor sets CA and IA in the null primitive, the main processor services pending interrupts using a midinstruction stack frame (refer to Figure 7-43) and reads the response CIR again. If the coprocessor sets CA and clears IA in the null primitive, the main processor reads the response CIR again without servicing any pending interrupts.

A null primitive with CA = 0 provides a condition evaluation indicator to the main processor during the execution of a conditional instruction and ends the dialogue between the main processor and coprocessor for that instruction. The main processor completes the execution of a conditional category coprocessor instruction when it receives the primitive. The PF bit is not relevant during conditional instruction execution since the primitive itself implies completion of processing.

Usually, when the main processor reads any primitive that does not have CA = 1 while executing a general category instruction, it terminates the dialogue between the main processor and coprocessor. If a trace exception is pending, however, the main processor does not terminate the instruction dialogue until it reads a null primitive with CA = 0 and PF = 1 from the response CIR (refer to **7.5.2.5 Trace Exceptions**). Thus, the main processor continues to read the response CIR until it receives a null primitive with CA = 0

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

and PF = 1, and then performs trace exception processing. When IA = 1, the main processor services pending interrupts before reading the response CIR again.

A coprocessor can be designed to execute a cpGEN instruction concurrently with the execution of main processor instructions and, also, buffer one write operation to either its command or condition CIR. This type of coprocessor issues a null primitive with CA = 1 when it is concurrently executing a cpGEN instruction, and the main processor initiates another general or conditional coprocessor instruction. This primitive indicates that the coprocessor is busy and the main processor should read the response CIR again without reinitiating the instruction. The IA bit of this null primitive usually should be set to minimize interrupt latency while the main processor is waiting for the coprocessor to complete the general category instruction.

Table 7-3 summarizes the encodings of the null primitive.

Table 7-3. Null Coprocessor Response Primitive Encodings

CA	PC	IA	PF	TF	General Instructions	Conditional Instructions
x	1	x	x	x	Pass Program Counter to Instruction Address CIR, Clear PC Bit, and Proceed with Operation Specified by CA, IA, PF, and TF Bits	Same as General Category
1	0	0	x	x	Reread Response CIR, Do Not Service Pending Interrupts	Same as General Category
1	0	1	x	x	Service Pending Interrupts and Reread the Response CIR	Same as General Category
0	0	0	0	c	If (Trace Pending) Reread Response CIR; Else, Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c
0	0	1	0	c	If (Trace Pending) Service Pending Interrupts and Reread Response CIR; Else, Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c
0	0	x	1	c	Coprocessor Instruction Completed; Service Pending Exceptions or Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c.

x = Don't Care

c = 1 or 0 Depending on Coprocessor Condition Evaluation

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.4.5 Supervisor Check Primitive

The supervisor check primitive verifies that the main processor is operating in the supervisor privilege level while executing a coprocessor instruction. This primitive applies to instructions in the general and conditional coprocessor instruction categories. Figure 7-25 shows the format of the supervisor check primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 7-25. Supervisor Check Primitive Format

The supervisor check primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. Bit 15 is shown as one, but during execution of a general category instruction, this primitive performs the same operations, regardless of the value of bit 15. However, if this primitive is issued with bit 15 = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the MC68020/EC020 reads the supervisor check primitive from the response CIR, it checks the value of the S-bit in the SR. If S = 0 (main processor operating at user privilege level), the main processor aborts the coprocessor instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**). The main processor then initiates privilege violation exception processing (refer to **7.5.2.3 Privilege Violations**). If the main processor is at the supervisor privilege level when it receives this primitive, it reads the response CIR again.

The supervisor check primitive allows privileged instructions to be defined in the coprocessor general and conditional instruction categories. This primitive should be the first one issued by the coprocessor during the dialog for an instruction that is implemented as privileged.

7.4.6 Transfer Operation Word Primitive

The transfer operation word primitive requests a copy of the coprocessor instruction operation word for the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-26 shows the format of the transfer operation word primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	1	1	0	0	0	0	0	0	0	0

Figure 7-26. Transfer Operation Word Primitive Format

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The transfer operation word primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If this primitive is issued with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the main processor reads this primitive from the response CIR, it transfers the F-line operation word of the currently executing coprocessor instruction to the operation word CIR. The value of the scanPC is not affected by this primitive.

7.4.7 Transfer from Instruction Stream Primitive

The transfer from instruction stream primitive initiates transfers of operands from the instruction stream to the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-27 shows the format of the transfer from instruction stream primitive.

15	14	13	12	11	10	9	8	7 … 0
CA	PC	0	0	1	1	1	1	LENGTH

Figure 7-27. Transfer from Instruction Stream Primitive Format

The transfer from instruction stream primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If this primitive is issued with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

The length field of this primitive specifies the length, in bytes, of the operand to be transferred from the instruction stream to the coprocessor. The length must be an even number of bytes. If an odd length is specified, the main processor initiates protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**).

This primitive transfers coprocessor-defined extension words to the coprocessor. When the main processor reads this primitive from the response CIR, it copies the number of bytes indicated by the length field from the instruction stream to the operand CIR. The first word or long word transferred is at the location pointed to by the scanPC when the primitive is read by the main processor. The scanPC is incremented after each word or long word is transferred. When execution of the primitive has completed, the scanPC has been incremented by the total number of bytes transferred and points to the word following the last word transferred. The main processor transfers the operands from the instruction stream, using a sequence of long-word writes, to the operand CIR. If the length field is not an even multiple of four bytes, the last two bytes from the instruction stream are transferred using a word write to the operand CIR.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.4.8 Evaluate and Transfer Effective Address Primitive

The evaluate and transfer effective address primitive evaluates the effective address specified in the coprocessor instruction operation word and transfers the result to the coprocessor. This primitive applies to general category instructions. If this primitive is issued by the coprocessor during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-28 shows the format of the evaluate and transfer effective address primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	1	0	0	0	0	0	0	0	0	0

Figure 7-28. Evaluate and Transfer Effective Address Primitive Format

The evaluate and transfer effective address primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

When the main processor reads this primitive while executing a general category instruction, it evaluates the effective address specified in the instruction. At this point, the scanPC contains the address of the first of any required effective address extension words. The main processor increments the scanPC by two after it references each of these extension words. After the effective address is calculated, the resulting 32-bit value is written to the operand address CIR.

The MC68020/EC020 only calculates effective addresses for control alterable addressing modes in response to this primitive. If the addressing mode in the operation word is not a control alterable mode, the main processor aborts the instruction by writing a $0001 to the control CIR and initiates F-line emulation exception processing (refer to **7.5.2.2 F-Line Emulator Exceptions**).

7.4.9 Evaluate Effective Address and Transfer Data Primitive

The evaluate effective address and transfer data primitive transfers an operand between the coprocessor and the effective address specified in the coprocessor instruction operation word. This primitive applies to general category instructions. If the coprocessor issues this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-29 shows the format of the evaluate effective address and transfer data primitive.

15	14	13	12	11	10 … 8	7 … 0
CA	PC	DR	1	0	VALID EA	LENGTH

Figure 7-29. Evaluate Effective Address and Transfer Data Primitive Format

This primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The valid EA field of the primitive format specifies the valid effective address categories for this primitive. If the effective address specified in the instruction operation word is not a member of the class specified by the valid EA field, the main processor aborts the coprocessor instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**) and by initiating F-line emulation exception processing. Table 7-4 lists the valid effective address field encodings.

Table 7-4. Valid Effective Address Field Codes

Field	Category
000	Control Alterable
001	Data Alterable
010	Memory Alterable
011	Alterable
100	Control
101	Data
110	Memory
111	Any Effective Address (No Restriction)

Even when the valid EA fields specified in the primitive and in the instruction operation word match, the MC68020/EC020 initiates protocol violation exception processing if the primitive requests a write to an unalterable effective address.

The length in bytes of the operand to be transferred is specified by the length field of the primitive format. Several restrictions apply to operand lengths for certain effective addressing modes. If the effective address is a main processor register (register direct mode), only operand lengths of one, two, or four bytes are valid; all other lengths cause the main processor to initiate protocol violation exception processing. Operand lengths of 0–255 bytes are valid for the memory addressing modes.

The length of 0–255 bytes does not apply to an immediate operand. The length of an immediate operand must be one byte or an even number of bytes (less than 256), and the direction of transfer must be to the coprocessor; otherwise, the main processor initiates protocol violation exception processing.

When the main processor receives the evaluate effective address and transfer data primitive during the execution of a general category instruction, it verifies that the effective address encoded in the instruction operation word is in the category specified by the primitive. If so, the processor calculates the effective address using the appropriate effective address extension words at the current scanPC address and increments the scanPC by two for each word referenced. Using long-word transfers whenever possible, the main processor then transfers the number of bytes specified in the primitive between the operand CIR and the effective address. Refer to **7.3.8 Operand CIR** for information concerning operand alignment for transfers involving the operand CIR.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The DR bit specifies the direction of the operand transfer. DR = 0 requests a transfer from the main processor to the coprocessor, and DR = 1 specifies a transfer from the coprocessor to the main processor.

If the effective addressing mode specifies the predecrement mode, the address register used is decremented by the size of the operand before the transfer. The bytes within the operand are then transferred to or from ascending addresses beginning with the location specified by the decremented address register. In this mode, if A7 is used as the address register and the operand length is one byte, A7 is decremented by two to maintain a word-aligned stack.

For the postincrement effective addressing mode, the address register used is incremented by the size of the operand after the transfer. The bytes within the operand are transferred to or from ascending addresses beginning with the location specified by the address register. In this mode, if A7 is used as the address register and the operand length is one byte, A7 is incremented by two after the transfer to maintain a word-aligned stack. Transferring odd length operands longer than one byte using the –(A7) or (A7)+ addressing modes can result in a stack pointer that is not word aligned.

The processor repeats the effective address calculation each time this primitive is issued during the execution of a given instruction. The calculation uses the current contents of any required address and data registers. The instruction must include a set of effective address extension words for each repetition of a calculation that requires them. The processor locates these words at the current scanPC location and increments the scanPC by two for each word referenced in the instruction stream.

The MC68020/EC020 sign-extends a byte or word-sized operand to a long-word value when it is transferred to an address register (A7–A0) using this primitive with the register direct effective addressing mode. A byte or word-sized operand transferred to a data register (D7–D0) only overwrites the lower byte or word of the data register.

7.4.10 Write to Previously Evaluated Effective Address Primitive

The write to previously evaluated effective address primitive transfers an operand from the coprocessor to a previously evaluated effective address. This primitive applies to general category instructions. If the coprocessor uses this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-30 shows the format of the write to previously evaluated effective address primitive.

15	14	13	12	11	10	9	8	7 … 0
CA	PC	1	0	0	0	0	0	LENGTH

Figure 7-30. Write to Previously Evaluated Effective Address Primitive Format

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The write to previously evaluated effective address primitive uses the CA and PC bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The length field of the primitive format specifies the length of the operand in bytes. The MC68020/EC020 transfers operands of 0–255 bytes in length.

When the main processor receives this primitive during the execution of a general category instruction, it transfers an operand from the operand CIR to an effective address specified by a temporary register within the MC68020/EC020. When a previous primitive for the current instruction has evaluated the effective address, this temporary register contains the evaluated effective address. Primitives that store an evaluated effective address in a temporary register of the main processor are the evaluate and transfer effective address, evaluate effective address and transfer data, and transfer multiple coprocessor registers primitive. If this primitive is used during an instruction in which the effective address specified in the instruction operation word has not been calculated, the effective address used for the write is undefined. Also, if the previously evaluated effective address was register direct, the address written to in response to this primitive is undefined.

The function code value during the write operation indicates either supervisor or user data space, depending on the value of the S-bit in the MC68020/EC020 SR when the processor reads this primitive. While a coprocessor should request writes to only alterable effective addressing modes, the MC68020/EC020 does not check the type of effective address used with this primitive. For example, if the previously evaluated effective address was PC relative and the MC68020/EC020 is at the user privilege level (S = 0 in SR), the MC68020/EC020 writes to user data space at the previously calculated program relative address (the 32-bit value in the temporary internal register of the processor).

Operands longer than four bytes are transferred in increments of four bytes (operand parts) when possible. The main processor reads a long-word operand part from the operand CIR and transfers this part to the current effective address. The transfers continue in this manner using ascending memory locations until all of the long-word operand parts are transferred, and any remaining operand part is then transferred using a one-, two-, or three-byte transfer as required. The operand parts are stored in memory using ascending addresses beginning with the address in the MC68020/EC020 temporary register, which is internal to the processor and not for user use.

The execution of this primitive does not modify any of the registers in the MC68020/EC020 programming model, even if the previously evaluated effective address mode is the predecrement or postincrement mode. If the previously evaluated effective addressing mode used any of the MC68020/EC020 internal address or data registers, the effective address value used is the final value from the preceding primitive. That is, this primitive uses the value from an evaluate and transfer effective address, evaluate effective address and transfer data, or transfer multiple coprocessor registers primitive without modification.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The take address and transfer data primitive described in **7.4.11 Take Address and Transfer Data Primitive** does not replace the effective address value that has been calculated by the MC68020/EC020. The address that the main processor obtains in response to the take address and transfer data primitive is not available to the write to previously evaluated effective address primitive.

A coprocessor can issue an evaluate effective address and transfer data primitive followed by this primitive to perform a read-modify-write operation that is not indivisible. The bus cycles for this operation are normal bus cycles that can be interrupted, and the bus can be arbitrated between the cycles.

7.4.11 Take Address and Transfer Data Primitive

The take address and transfer data primitive transfers an operand between the coprocessor and an address supplied by the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-31 shows the format of the take address and transfer data primitive.

15	14	13	12	11	10	9	8	7 … 0
CA	PC	DR	0	0	1	0	1	LENGTH

Figure 7-31. Take Address and Transfer Data Primitive Format

The take address and transfer data primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If the coprocessor issues this primitive with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

The length field of the primitive format specifies the operand length, which can be from 0–255 bytes.

The main processor reads a 32-bit address from the operand address CIR. Using a series of long-word transfers, the processor transfers the operand between this address and the operand CIR. The DR bit determines the direction of the transfer. The processor reads or writes the operand parts to ascending addresses, starting at the address from the operand address CIR. If the operand length is not a multiple of four bytes, the final operand part is transferred using a one-, two-, or three-byte transfer as required.

The function code used with the address read from the operand address CIR indicates either supervisor or user data space according to the value of the S-bit in the MC68020/EC020 SR.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.4.12 Transfer to/from Top of Stack Primitive

The transfer to/from top of stack primitive transfers an operand between the coprocessor and the top of the active system stack of the main processor. This primitive applies to general and conditional category instructions. Figure 7-32 shows the format of the transfer to/from top of stack primitive.

15	14	13	12	11	10	9	8	7 … 0
CA	PC	DR	0	1	1	1	0	LENGTH

Figure 7-32. Transfer to/from Top of Stack Primitive Format

The transfer to/from top of stack primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If the coprocessor issues this primitive with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

The length field of the primitive format specifies the length in bytes of the operand to be transferred. The operand may be one, two, or four bytes in length; other length values cause the main processor to initiate protocol violation exception processing.

If DR = 0, the main processor transfers the operand from the active system stack to the operand CIR. The implied effective address mode used for the transfer is the (A7)+ addressing mode. A one-byte operand causes the stack pointer to be incremented by two after the transfer to maintain word alignment of the stack.

If DR = 1, the main processor transfers the operand from the operand CIR to the active system stack. The implied effective address mode used for the transfer is the –(A7) addressing mode. A one-byte operand causes the stack pointer to be decremented by two before the transfer to maintain word alignment of the stack.

7.4.13 Transfer Single Main Processor Register Primitive

The transfer single main processor register primitive transfers an operand between one of the main processor's data or address registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-33 shows the format of the transfer single main processor register primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2 … 0
CA	PC	DR	0	1	1	0	0	0	0	0	0	D/A	REGISTER

Figure 7-33. Transfer Single Main Processor Register Primitive Format

The transfer single main processor register primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If the coprocessor issues this primitive with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The D/A bit specifies whether the primitive transfers an address or data register. D/A = 0 indicates a data register, and D/A = 1 indicates an address register. The register field contains the register number.

If DR = 0, the main processor writes the long-word operand in the specified register to the operand CIR. If DR = 1, the main processor reads a long-word operand from the operand CIR and transfers it to the specified data or address register.

7.4.14 Transfer Main Processor Control Register Primitive

The transfer main processor control register primitive transfers a long-word operand between one of its control registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-34 shows the format of the transfer main processor control register primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	0	1	0	0	0	0	0	0	0	0

Figure 7-34. Transfer Main Processor Control Register Primitive Format

The transfer main processor control register primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If the coprocessor issues this primitive with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the main processor receives this primitive, it reads a control register select code from the register select CIR. This code determines which main processor control register is transferred. Table 7-5 lists the valid control register select codes. If the control register select code is not valid, the MC68020/EC020 initiates protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**).

Table 7-5. Main Processor Control Register Select Codes

Select Code	Control Register
$x000	SFC
$x001	DFC
$x002	CACR
$x800	USP
$x801	VBR
$x802	CAAR
$x803	MSP
$x804	ISP

All other codes cause a protocol violation exception.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

After reading a valid code from the register select CIR, if DR = 0, the main processor writes the long-word operand from the specified control register to the operand CIR. If DR = 1, the main processor reads a long-word operand from the operand CIR and places it in the specified control register.

7.4.15 Transfer Multiple Main Processor Registers Primitive

The transfer multiple main processor registers primitive transfers long-word operands between one or more of its data or address registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-35 shows the format of the transfer multiple main processor registers primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 7-35. Transfer Multiple Main Processor Registers Primitive Format

The transfer multiple main processor registers primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If the coprocessor issues this primitive with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the main processor receives this primitive, it reads a 16-bit register select mask from the register select CIR. The format of the register select mask is shown in Figure 7-36. A register is transferred if the bit corresponding to the register in the register select mask is set. The selected registers are transferred in the order D7–D0 and then A7–A0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

Figure 7-36. Register Select Mask Format

If DR = 0, the main processor writes the contents of each register indicated in the register select mask to the operand CIR using a sequence of long-word transfers. If DR = 1, the main processor reads a long-word operand from the operand CIR into each register indicated in the register select mask. The registers are transferred in the same order, regardless of the direction of transfer indicated by the DR bit.

7.4.16 Transfer Multiple Coprocessor Registers Primitive

The transfer multiple coprocessor registers primitive transfers from 0–16 operands between the effective address specified in the coprocessor instruction and the coprocessor. This primitive applies to general category instructions. If the coprocessor issues this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-37 shows the format of the transfer multiple coprocessor registers primitive.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

15	14	13	12	11	10	9	8	7 … 0
CA	PC	DR	0	0	0	0	1	LENGTH

Figure 7-37. Transfer Multiple Coprocessor Registers Primitive Format

The transfer multiple coprocessor registers primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The length field of the primitive format indicates the length in bytes of each operand transferred. The operand length must be an even number of bytes; odd length operands cause the MC68020/EC020 to initiate protocol violation exception processing (refer to **7.5.2.1 Protocol Violations**).

When the main processor reads this primitive, it calculates the effective address specified in the coprocessor instruction. The scanPC should be pointing to the first of any necessary effective address extension words when this primitive is read from the response CIR; the scanPC is incremented by two for each extension word referenced during the effective address calculation. For transfers from the effective address to the coprocessor (DR = 0), the control addressing modes and the postincrement addressing mode are valid. For transfers from the coprocessor to the effective address (DR = 1), the control alterable and predecrement addressing modes are valid. Invalid addressing modes cause the MC68020/EC020 to abort the instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**) and to initiate F-line emulator exception processing (refer to **7.5.2.2 F-Line Emulator Exceptions**).

After performing the effective address calculation, the MC68020/EC020 reads a 16-bit register select mask from the register select CIR. The coprocessor uses the register select mask to specify the number of operands to transfer; the MC68020/EC020 counts the number of ones in the register select mask to determine the number of operands. The order of the ones in the register select mask is not relevant to the operation of the main processor. As many as 16 operands can be transferred by the main processor in response to this primitive. The total number of bytes transferred is the product of the number of operands transferred and the length of each operand specified in the length field of the primitive format.

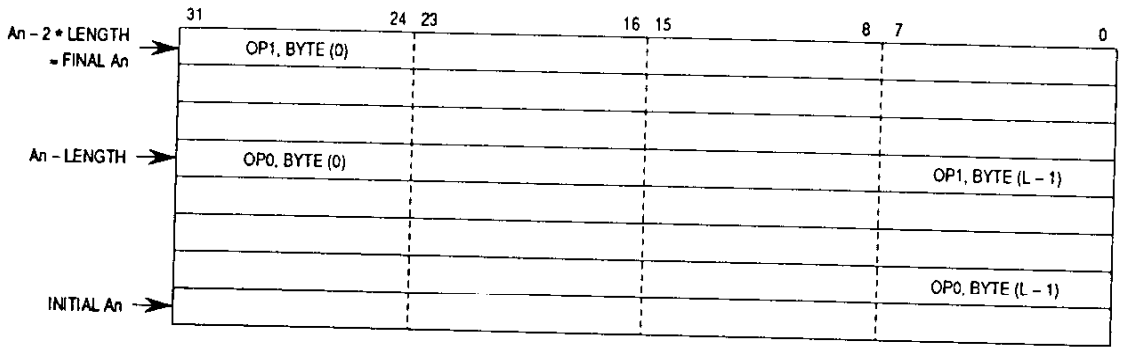
If DR = 1, the main processor reads the number of operands specified in the register select mask from the operand CIR and writes these operands to the effective address specified in the instruction using long-word transfers whenever possible. If DR = 0, the main processor reads the number of operands specified in the register select mask from the effective address and writes them to the operand CIR.

For the control addressing modes, the operands are transferred to or from memory using ascending addresses. For the postincrement addressing mode, the operands are read from memory with ascending addresses also, and the address register used is incremented by the size of an operand after each operand is transferred. The address register used with the (An)+ addressing mode is incremented by the total number of bytes transferred during the primitive execution.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

For the predecrement addressing mode, the operands are written to memory with descending addresses, but the bytes within each operand are written to memory with ascending addresses. As an example, Figure 7-38 shows the format in long-word-oriented memory for two 12-byte operands transferred from the coprocessor to the effective address using the –(An) addressing mode. The processor decrements the address register by the size of an operand before the operand is transferred. It writes the bytes of the operand to ascending memory addresses. When the transfer is complete, the address register has been decremented by the total number of bytes transferred. The MC68020/EC020 transfers the data using long-word transfers whenever possible.



NOTE: OP0, Byte (0) is the first byte written to memory
OP0, Byte (L–1) is the last byte of the first operand written to memory
OP1, Byte (0) is the first byte of the second operand written to memory
OP1, Byte (L–1) is the last byte written to memory

Figure 7-38. Operand Format in Memory for Transfer to –(An)

7.4.17 Transfer Status Register and ScanPC Primitive

The transfer status register and the scanPC primitive transfers values between the coprocessor and the MC68020/EC020 SR. On an optional basis, the scanPC also makes transfers. This primitive applies to general category instructions. If the coprocessor issues this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-39 shows the format of the transfer status register and scanPC primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	0	1	SP	0	0	0	0	0	0	0	0

Figure 7-39. Transfer Status Register and ScanPC Primitive Format

The transfer status register and scanPC primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The SP bit selects the scanPC option. If SP = 1, the primitive transfers both the scanPC and SR. If SP = 0, only the SR is transferred.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

If SP = 0 and DR = 0, the main processor writes the 16-bit SR value to the operand CIR. If SP = 0 and DR = 1, the main processor reads a 16-bit value from the operand CIR into the main processor SR.

If SP = 1 and DR = 0, the main processor writes the long-word value in the scanPC to the instruction address CIR and then writes the SR value to the operand CIR. If SP = 1 and DR = 1, the main processor reads a 16-bit value from the operand CIR into the SR and then reads a long-word value from the instruction address CIR into the scanPC.

With this primitive, a general category instruction can change the main processor program flow by placing a new value in the SR, in the scanPC, or new values in both the SR and the scanPC. By accessing the SR, the coprocessor can determine and manipulate the main processor condition codes, supervisor status, trace modes, selection of the active stack, and interrupt mask level.

The MC68020/EC020 discards any instruction words that have been prefetched beyond the current scanPC location when this primitive is issued with DR = 1 (transfer to main processor). The MC68020/EC020 then refills the instruction pipe from the scanPC address in the address space indicated by the S-bit of the SR.

If the MC68020/EC020 is operating in the trace on change of flow mode (T1, T0 in the SR = 01) when the coprocessor instruction begins to execute and if this primitive is issued with DR = 1 (from coprocessor to main processor), the MC68020/EC020 prepares to take a trace exception. The trace exception occurs when the coprocessor signals that it has completed all processing associated with the instruction. Changes in the trace modes due to the transfer of the SR to the main processor take effect on execution of the next instruction.

7.4.18 Take Preinstruction Exception Primitive

The take preinstruction exception primitive initiates exception processing using a coprocessor-supplied exception vector number and the preinstruction exception stack frame format. This primitive applies to general and conditional category instructions. Figure 7-40 shows the format of the take preinstruction exception primitive.

15	14	13	12	11	10	9	8	7 … 0
0	PC	0	1	1	1	0	0	VECTOR NUMBER

Figure 7-40. Take Preinstruction Exception Primitive Format

The take preinstruction exception primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. The vector number field contains the exception vector number used by the main processor to initiate exception processing.

When the main processor receives this primitive, it acknowledges the coprocessor exception request by writing an exception acknowledge mask to the control CIR (refer to **7.3.2 Control CIR**). The MC68020/EC020 then proceeds with exception processing as

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

described in **Section 6 Exception Processing**. The vector number for the exception is taken from the vector number field of the primitive, and the MC68020/EC020 uses the four-word stack frame format shown in Figure 7-41.

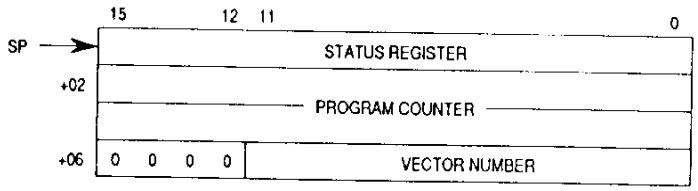


Figure 7-41. MC68020/EC020 Preinstruction Stack Frame

The value of the PC saved in this stack frame is the F-line operation word address of the coprocessor instruction during which the primitive was received. Thus, if the exception handler routine does not modify the stack frame, an RTE instruction causes the MC68020/EC020 to return and reinitiate execution of the coprocessor instruction.

The take preinstruction exception primitive can be used when the coprocessor does not recognize a value written to either its command CIR or condition CIR to initiate a coprocessor instruction. This primitive can also be used if an exception occurs in the coprocessor instruction before any program-visible resources are modified by the instruction operation. This primitive should not be used during a coprocessor instruction if program-visible resources have been modified by that instruction. Otherwise, since the MC68020/EC020 reinitiates the instruction when it returns from exception processing, the restarted instruction receives the previously modified resources in an inconsistent state.

One of the most important uses of the take preinstruction exception primitive is to signal an exception condition in a cpGEN instruction that was executing concurrently with the main processor's instruction execution. If the coprocessor no longer requires the services of the main processor to complete a cpGEN instruction and if the concurrent instruction completion is transparent to the programming model, the coprocessor can release the main processor by issuing a primitive with CA = 0. The main processor usually executes the next instruction in the instruction stream, and the coprocessor completes its operations concurrently with the main processor operation. If an exception occurs while the coprocessor is executing an instruction concurrently, the exception is not processed until the main processor attempts to initiate the next general or conditional instruction. After the main processor writes to the command or condition CIR to initiate a general or conditional instruction, it then reads the response CIR. At this time, the coprocessor can return the take preinstruction exception primitive. This protocol allows the main processor to proceed with exception processing related to the previous concurrently executing coprocessor instruction and then return and reinitiate the coprocessor instruction during which the exception was signaled. The coprocessor should record the addresses of all general category instructions that can be executed concurrently with the main processor and that support exception recovery. Since the exception is not reported until the next coprocessor instruction is initiated, the processor usually requires the instruction address to determine

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

which instruction the coprocessor was executing when the exception occurred. A coprocessor can record the instruction address by setting PC = 1 in one of the primitives it uses before releasing the main processor.

7.4.19 Take Midinstruction Exception Primitive

The take midinstruction exception primitive initiates exception processing using a coprocessor-supplied exception vector number and the midinstruction exception stack frame format. This primitive applies to general and conditional category instructions. Figure 7-42 shows the format of the take midinstruction exception primitive.

15	14	13	12	11	10	9	8	7 … 0
0	PC	0	1	1	1	0	1	VECTOR NUMBER

Figure 7-42. Take Midinstruction Exception Primitive Format

The take midinstruction exception primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. The vector number field contains the exception vector number used by the main processor to initiate exception processing.

When the main processor receives this primitive, it acknowledges the coprocessor exception request by writing an exception acknowledge mask (refer to **7.3.2 Control CIR**) to the control CIR. The MC68020/EC020 then performs exception processing as described in **Section 6 Exception Processing**. The vector number for the exception is taken from the vector number field of the primitive, and the MC68020/EC020 uses the 10-word stack frame format shown in Figure 7-43.

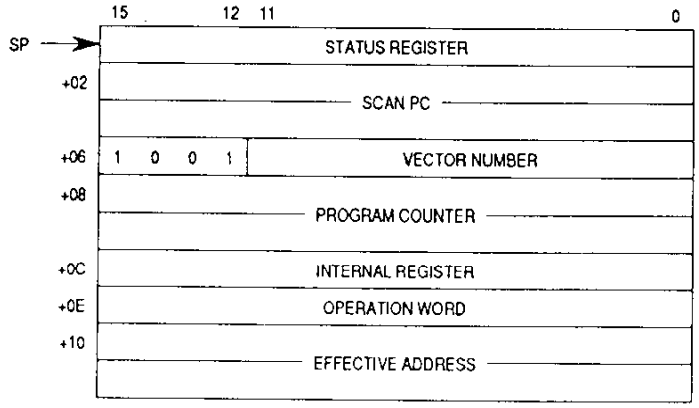


Figure 7-43. MC68020/EC020 Midinstruction Stack Frame

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The PC value saved in this stack frame is the operation word address of the coprocessor instruction during which the primitive is received. The scanPC field contains the value of the MC68020/EC020 scanPC when the primitive is received. If the current instruction does not evaluate an effective address prior to the exception request primitive, the value of the effective address field in the stack frame is undefined.

The coprocessor uses this primitive to request exception processing for an exception during the instruction dialog with the main processor. If the exception handler does not modify the stack frame, the MC68020/EC020 returns from the exception handler and reads the response CIR. Thus, the main processor attempts to continue executing the suspended instruction by reading the response CIR and processing the primitive it receives.

7.4.20 Take Postinstruction Exception Primitive

The take postinstruction exception primitive initiates exception processing using a coprocessor-supplied exception vector number and the postinstruction exception stack frame format. This primitive applies to general and conditional category instructions. Figure 7-44 shows the format of the take postinstruction exception primitive.

15	14	13	12	11	10	9	8	7 … 0
0	PC	0	1	1	1	1	0	VECTOR NUMBER

Figure 7-44. Take Postinstruction Exception Primitive Format

The take postinstruction exception primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**. The vector number field contains the exception vector number used by the main processor to initiate exception processing.

When the main processor receives this primitive, it acknowledges the coprocessor exception request by writing an exception acknowledge mask to the control CIR (refer to **7.3.2 Control CIR**). The MC68020/EC020 then performs exception processing as described in **Section 6 Exception Processing**. The vector number for the exception is taken from the vector number field of the primitive, and the MC68020/EC020 uses the six-word stack frame format shown in Figure 7-45.

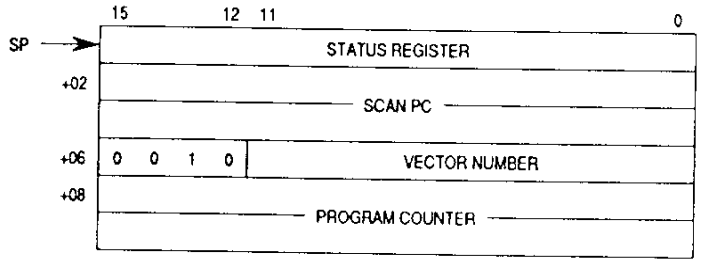


Figure 7-45. MC68020/EC020 Postinstruction Stack Frame

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The value in the main processor scanPC at the time this primitive is received is saved in the scanPC field of the postinstruction exception stack frame. The value of the PC saved is the F-line operation word address of the coprocessor instruction during which the primitive is received.

When the MC68020/EC020 receives the take postinstruction exception primitive, it assumes that the coprocessor either completed or aborted the instruction with an exception. If the exception handler does not modify the stack frame, the MC68020/EC020 returns from the exception handler to begin execution at the location specified by the scanPC field of the stack frame. This location should be the address of the next instruction to be executed.

The coprocessor uses this primitive to request exception processing when it completes or aborts an instruction while the main processor is awaiting a normal response. For a general category instruction, the response is a release; for a conditional category instruction, it is an evaluated true/false condition indicator. Thus, the operation of the MC68020/EC020 in response to this primitive is compatible with standard M68000 family instruction related exception processing (for example, the divide-by-zero exception).

7.5 EXCEPTIONS

Various exception conditions related to the execution of coprocessor instructions may occur. Whether an exception is detected by the main processor or by the coprocessor, the main processor coordinates and performs exception processing. Servicing these coprocessor-related exceptions is an extension of the protocol used to service standard M68000 family exceptions. That is, when either the main processor detects an exception or is signaled by the coprocessor that an exception condition has occurred, the main processor proceeds with exception processing as described in **Section 6 Exception Processing**.

7.5.1 Coprocessor-Detected Exceptions

Coprocessor interface exceptions that the coprocessor detects, as well as those that the main processor detects, are usually classified as coprocessor-detected exceptions. Coprocessor-detected exceptions can occur during M68000 coprocessor interface operations, internal operations, or other system-related operations of the coprocessor.

Most coprocessor-detected exceptions are signaled to the main processor through the use of one of the three take exception primitives defined for the M68000 coprocessor interface. The main processor responds to these primitives as described in **7.4.18 Take Preinstruction Exception Primitive**, **7.4.19 Take Midinstruction Exception Primitive**, and **7.4.20 Take Postinstruction Exception Primitive**. However, not all coprocessor-detected exceptions are signaled by response primitives. Coprocessor-detected format errors during the cpSAVE or cpRESTORE instruction are signaled to the main processor using the invalid format word described in **7.2.3.2.3 Invalid Format Words**.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.5.1.1 COPROCESSOR-DETECTED PROTOCOL VIOLATIONS. Protocol violation exceptions are communication failures between the main processor and coprocessor across the M68000 coprocessor interface. Coprocessor-detected protocol violations occur when the main processor accesses entries in the CIR set in an unexpected sequence. The sequence of operations that the main processor performs for a given coprocessor instruction or coprocessor response primitive has been described previously in this section.

A coprocessor can detect protocol violations in various ways. According to the M68000 coprocessor interface protocol, the main processor always accesses the operation word, operand, register select, instruction address, or operand address CIRs synchronously with respect to the operation of the coprocessor. That is, the main processor accesses these five registers in a certain sequence, and the coprocessor expects them to be accessed in that sequence. As a minimum, all M68000 coprocessors should detect a protocol violation if the main processor accesses any of these five registers when the coprocessor is expecting an access to either the command or condition CIR. Likewise, if the coprocessor is expecting an access to the command or condition CIR and the main processor accesses one of these five registers, the coprocessor should detect and signal a protocol violation.

According to the M68000 coprocessor interface protocol, the main processor can perform a read of either the save CIR or response CIR or a write of either the restore CIR or control CIR asynchronously with respect to the operation of the coprocessor. That is, an access to one of these registers without the coprocessor explicitly expecting that access at that point can be a valid access. Although the coprocessor can anticipate certain accesses to the restore, response, and control CIRs, these registers can be accessed at other times also.

The coprocessor cannot signal a protocol violation to the main processor during execution of a cpSAVE or cpRESTORE instruction. If a coprocessor detects a protocol violation during execution of the cpSAVE or cpRESTORE instruction, it should signal the exception to the main processor when the next coprocessor instruction is initiated.

The main philosophy of the coprocessor-detected protocol violation is that the coprocessor should always acknowledge an access to one of its interface registers. If the coprocessor determines that the access is not valid, it should assert $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ to the main processor and signal a protocol violation when the main processor next reads the response CIR. If the coprocessor fails to assert $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$, the main processor waits for the assertion of that signal (or some other bus termination signal) indefinitely. The protocol previously described ensures that the coprocessor cannot halt the main processor.

The coprocessor can signal a protocol violation to the main processor with the take midinstruction exception primitive. To maintain consistency, the vector number should be 13, as it is for a protocol violation detected by the main processor. When the main processor reads this primitive, it proceeds as described in **7.4.19 Take Midinstruction Exception Primitive**. If the exception handler does not modify the stack frame, the MC68020/EC020 returns from the exception handler and reads the response CIR.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.5.1.2 COPROCESSOR-DETECTED ILLEGAL COMMAND OR CONDITION WORDS. Illegal coprocessor command or condition words are values written to the command CIR or condition CIR that the coprocessor does not recognize. If a value written to either of these registers is not valid, the coprocessor should return the take preinstruction exception primitive in the response CIR. When it receives this primitive, the main processor takes a preinstruction exception as described in **7.4.18 Take Preinstruction Exception Primitive**. If the exception handler does not modify the main processor stack frame, an RTE instruction causes the MC68020/EC020 to reinitiate the instruction that took the exception. The coprocessor designer should ensure that the state of the coprocessor is not irrecoverably altered by an illegal command or condition exception if the system supports emulation of the unrecognized command or condition word.

All M68000 coprocessors signal illegal command and condition words by returning the take preinstruction exception primitive with the F-line emulator exception vector number 11.

7.5.1.3 COPROCESSOR DATA-PROCESSING-RELATED EXCEPTIONS. Exceptions related to the internal operation of a coprocessor are classified as data-processing-related exceptions. These exceptions are analogous to the divide-by-zero exception defined by M68000 microprocessors and should be signaled to the main processor using one of the three take exception primitives containing an appropriate exception vector number. Which of these three primitives is used to signal the exception is usually determined by the point in the instruction operation where the main processor should continue the program flow after exception processing. Refer to **7.4.18 Take Preinstruction Exception Primitives**, **7.4.19 Take Midinstruction Exception Primitive**, and **7.4.20 Take Postinstruction Exception Primitive**.

7.5.1.4 COPROCESSOR SYSTEM-RELATED EXCEPTIONS. System-related exceptions detected by a DMA coprocessor include those associated with bus activity and any other exceptions (interrupts, for example) occurring external to the coprocessor. The actions taken by the coprocessor and the main processor depend on the type of exception that occurs.

When an address or bus error is detected by a DMA coprocessor, the coprocessor should store any information necessary for the main processor exception handling routines in system-accessible registers. The coprocessor should place one of the three take exception primitives encoded with an appropriate exception vector number in the response CIR. Which of the three primitives is used depends upon the point in the coprocessor instruction at which the exception was detected and the point in the instruction execution at which the main processor should continue after exception processing. Refer to **7.4.18 Take Preinstruction Exception Primitives**, **7.4.19 Take Midinstruction Exception Primitive**, and **7.4.20 Take Postinstruction Exception Primitive**.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.5.1.5 FORMAT ERRORS. Format errors are the only coprocessor-detected exceptions that are not signaled to the main processor with a response primitive. When the main processor writes a format word to the restore CIR during the execution of a cpRESTORE instruction, the coprocessor decodes this word to determine if it is valid (refer to **7.2.3.3 Coprocessor Context Save Instruction**). If the format word is not valid, the coprocessor places the invalid format code in the restore CIR. When the main processor reads the invalid format code, it aborts the coprocessor instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**). The main processor then performs exception processing using a four-word preinstruction stack frame and the format error exception vector number 14. Thus, if the exception handler does not modify the stack frame, the MC68020/EC020 restarts the cpRESTORE instruction when the RTE instruction in the handler is executed. If the coprocessor returns the invalid format code when the main processor reads the save CIR to initiate a cpSAVE instruction, the main processor performs format error exception processing as outlined for the cpRESTORE instruction.

7.5.2 Main-Processor-Detected Exceptions

A number of exceptions related to coprocessor instruction execution are detected and serviced by the main processor instead of the coprocessor. These exceptions can be related to the execution of coprocessor response primitives, communication across the M68000 coprocessor interface, or completion of conditional coprocessor instructions by the main processor.

7.5.2.1 PROTOCOL VIOLATIONS. The main processor detects a protocol violation when it reads a primitive from the response CIR that is not a valid primitive. The protocol violations that can occur in response to the primitives defined for the M68000 coprocessor interface are summarized in Table 7-6.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 7-6. Exceptions Related to Primitive Processing

Primitive	Protocol	F-Line	Other
Busy			
Null			
Supervisory Check* Other: Privilege Violation if S-Bit in the SR = 0			X
Transfer Operation Word*			
Transfer from Instruction Stream* Protocol: If Length Field Is Odd (Zero Length Legal)	X		
Evaluate and Transfer Effective Address Protocol: If Used with Conditional Instruction F-Line: If EA in Opword Is NOT Control Alterable	X	X	
Evaluate Effective Address and Transfer Data Protocol: 1. If Used with Conditional Instructions 2. Length Is Not 1, 2, or 4 and EA = Register Direct 3. If EA = Immediate and Length Odd and Greater Than 1 4. Attempt to Write to Unalterable Address Even if Address Declared Legal in Primitive F-Line: Valid EA Field Does Not Match EA in Opword	X	X	
Write to Previously Evaluated Effective Address Protocol: If Used with Conditional Instruction	X		
Take Address and Transfer Data*			
Transfer to/from Top of Stack* Protocol: Length Field Other Than 1, 2, or 4	X		
Transfer Single Main Processor Register*			
Transfer Main Processor Control Register Protocol: Invalid Control Register Select Code	X		
Transfer Multiple Main Processor Registers*			
Transfer Multiple Coprocessor Registers Protocol: 1. If Used with Conditional Instructions 2. Odd Length Value F-Line: 1. EA Not Control Alterable or (An)+ for CP to Memory Transfer 2. EA Not Control Alterable or (An) for Memory to CP Transfer	X	X	
Transfer Status and ScanPC Protocol: If Used with Conditional Instruction Other: 1. Trace—Trace Made Pending if MC68020/EC020 in "Trace on Change of Flow" Mode and DR = 1 2. Address Error—If Odd Value Written to ScanPC	X		X
Take Preinstruction, Midinstruction, or Postinstruction Exception Exception Depends on Vector Supplies in Primitive	X	X	X

*Use of this primitive with CA = 0 will cause protocol violation on conditional instructions.

Abbreviations:
EA—Effective Address
CP—Coprocessor

7

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

When the MC68020/EC020 detects a protocol violation, it does not automatically notify the coprocessor of the resulting exception by writing to the control CIR. However, the exception handling routine may use the MOVES instruction to read the response CIR and thus determine the primitive that caused the MC68020/EC020 to initiate protocol violation exception processing. The main processor initiates exception processing using the midinstruction stack frame (refer to Figure 7-43) and the coprocessor protocol violation exception vector number 13. If the exception handler does not modify the stack frame, the main processor reads the response CIR again following the execution of an RTE instruction to return from the exception handler. This protocol allows extensions to the M68000 coprocessor interface to be emulated in software by a main processor that does not provide hardware support for these extensions. Thus, the protocol violation is transparent to the coprocessor if the primitive execution can be emulated in software by the main processor.

7.5.2.2 F-LINE EMULATOR EXCEPTIONS. The F-line emulator exceptions detected by the MC68020/EC020 are either explicitly or implicitly related to the encodings of F-line operation words in the instruction stream. If the main processor determines that an F-line operation word is not valid, it initiates F-line emulator exception processing. Any F-line operation word with bits 8–6 = 110 or 111 causes the MC68020/EC020 to initiate exception processing without initiating any communication with the coprocessor for that instruction. Also, an operation word with bits 8–6 = 000–101 that does not map to one of the valid coprocessor instructions in the instruction set causes the MC68020/EC020 to initiate F-line emulator exception processing. If the F-line emulator exception is either of these two situations, the main processor does not write to the control CIR prior to initiating exception processing.

F-line exceptions can also occur if the operations requested by a coprocessor response primitive are not compatible with the effective address type in bits 5–0 of the coprocessor instruction operation word. The F-line emulator exceptions that can result from the use of the M68000 coprocessor response primitives are summarized in Table 7-6. If the exception is caused by receiving an invalid primitive, the main processor aborts the coprocessor instruction in progress by writing an abort mask (refer to **7.3.2 Control CIR**) to the control CIR prior to F-line emulator exception processing.

Another type of F-line emulator exception occurs when a bus error occurs during the CIR access that initiates a coprocessor instruction. The main processor assumes that the coprocessor is not present and takes the exception.

When the main processor initiates F-line emulator exception processing, it uses the four-word preinstruction exception stack frame (refer to Figure 7-41) and the F-line emulator exception vector number 11. Thus, if the exception handler does not modify the stack frame, the main processor attempts to restart the instruction that caused the exception after it executes an RTE instruction to return from the exception handler.

If the cause of the F-line exception can be emulated in software, the handler stores the results of the emulation in the appropriate registers of the programming model and in the status register field of the saved stack frame. The exception handler adjusts the program

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

counter field of the saved stack frame to point to the next instruction operation word and executes the RTE instruction. The MC68020/EC020 then executes the instruction following the instruction that was emulated.

The exception handler should also check the copy of the SR on the stack to determine whether tracing is enabled. If tracing is enabled, the trace exception processing should also be emulated. Refer to **Section 6 Exception Processing** for additional information.

7.5.2.3 PRIVILEGE VIOLATIONS. Privilege violations can result from the cpSAVE and cpRESTORE instructions and from the supervisor check coprocessor response primitive. The MC68020/EC020 initiates privilege violation exception processing if it attempts to execute either the cpSAVE or cpRESTORE instruction when it is in the user state (S = 0 in the SR). The main processor initiates this exception processing prior to any communication with the coprocessor associated with the cpSAVE or cpRESTORE instructions.

If the main processor is executing a coprocessor instruction in the user state when it reads the supervisor check primitive, it aborts the coprocessor instruction in progress by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**). The main processor then performs privilege violation exception processing.

If a privilege violation occurs, the main processor initiates exception processing using the four-word preinstruction stack frame (refer to Figure 7-41) and the privilege violation exception vector number 8. Thus, if the exception handler does not modify the stack frame, the main processor attempts to restart the instruction during which the exception occurred after it executes an RTE to return from the handler.

7.5.2.4 cpTRAPcc INSTRUCTION TRAPS. If, during the execution of a cpTRAPcc instruction, the coprocessor returns the TRUE condition indicator to the main processor with a null primitive, the main processor initiates trap exception processing. The main processor uses the six-word postinstruction exception stack frame (refer to Figure 7-45) and the trap exception vector number 7. The scanPC field of this stack frame contains the address of the instruction following the cpTRAPcc instruction. The processing associated with the cpTRAPcc instruction can then proceed, and the exception handler can locate any immediate operand words encoded in the cpTRAPcc instruction using the information contained in the six-word stack frame. If the exception handler does not modify the stack frame, the main processor executes the instruction following the cpTRAPcc instruction after it executes an RTE instruction to exit from the handler.

7.5.2.5 TRACE EXCEPTIONS. The MC68020/EC020 supports two modes of instruction tracing, as discussed in **Section 6 Exception Processing**. In the trace on instruction execution mode, the MC68020/EC020 takes a trace exception after completing each instruction. In the trace on change of flow mode, the MC68020/EC020 takes a trace exception after each instruction that alters the SR or places an address other than the address of the next instruction in the PC.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The protocol used to execute coprocessor cpSAVE, cpRESTORE, or conditional category instructions does not change when a trace exception is pending in the main processor. The main processor performs a pending trace on instruction execution exception after completing the execution of that instruction. If the main processor is in the trace on change of flow mode and an instruction places an address other than that of the next instruction in the PC, the processor takes a trace exception after it executes the instruction.

If a trace exception is not pending during a general category instruction, the main processor terminates communication with the coprocessor after reading any primitive with CA = 0. Thus, the coprocessor can complete a cpGEN instruction concurrently with the execution of instructions by the main processor. When a trace exception is pending, however, the main processor must ensure that all processing associated with a cpGEN instruction has been completed before it takes the trace exception. In this case, the main processor continues to read the response CIR and to service the primitives until it receives either a null primitive with CA = 0 and PF = 1 or until exception processing caused by a take postinstruction exception primitive has completed. The coprocessor should return the null primitive with CA = 0 and PF = 0 while it is completing the execution of the cpGEN instruction. The main processor may service pending interrupts between reads of the response CIR if IA = 1 in these primitives (refer to Table 7-3). This protocol ensures that a trace exception is not taken until all processing associated with a cpGEN instruction has completed.

If T1, T0 = 01 in the MC68020/EC020 SR (trace on change of flow mode) when a general category instruction is initiated, a trace exception is taken for the instruction only when the coprocessor issues a transfer status register and scanPC primitive with DR = 1 during the execution of that instruction. In this case, it is possible that the coprocessor is still executing the cpGEN instruction concurrently when the main processor begins execution of the trace exception handler. A cpSAVE instruction executed during the trace on change of flow exception handler could thus suspend the execution of a concurrently operating cpGEN instruction.

7.5.2.6 INTERRUPTS. Interrupt processing, discussed in **Section 6 Exception Processing**, can occur at any instruction boundary. Interrupts are also serviced during the execution of a general or conditional category instruction under either of two conditions. If the main processor reads a null primitive with CA = 1 and IA = 1, it services any pending interrupts prior to reading the response CIR. Similarly, if a trace exception is pending during cpGEN instruction execution and the main processor reads a null primitive with CA = 0, IA = 1, and PF = 0 (refer to **7.5.2.5 Trace Exceptions**), the main processor services pending interrupts prior to reading the response CIR again.

The MC68020/EC020 uses the 10-word midinstruction stack frame (see Figure 7-43) when it services interrupts during the execution of a general or conditional category coprocessor instruction. Since it uses this stack frame, the main processor can perform all necessary processing and then return to read the response CIR. Thus, it can continue execution of the coprocessor instruction during which the interrupt exception occurred.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The MC68020/EC020 also services interrupts if it reads the not-ready format word from the save CIR during a cpSAVE instruction. The MC68020/EC020 uses the normal four-word preinstruction stack frame (see Figure 7-41) when it services interrupts after reading the not-ready format word. Thus, the processor can service any pending interrupts and execute an RTE to return and reinitiate the cpSAVE instruction by reading the save CIR.

7.5.2.7 FORMAT ERRORS. The MC68020/EC020 can detect a format error while executing a cpSAVE or cpRESTORE instruction if the length field of a valid format word is not a multiple of four bytes. If the MC68020/EC020 reads a format word with an invalid length field from the save CIR during the cpSAVE instruction, it aborts the coprocessor instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**) and initiates format error exception processing. If the MC68020/EC020 reads a format word with an invalid length field from the effective address specified in the cpRESTORE instruction, the MC68020/EC020 writes that format word to the restore CIR and then reads the coprocessor response from the restore CIR. The MC68020/EC020 then aborts the cpRESTORE instruction by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**) and initiates format error exception processing.

The MC68020/EC020 uses the four-word preinstruction stack frame (see Figure 7-41) and the format error vector number 14 when it initiates format error exception processing. Thus, if the exception handler does not modify the stack frame, the main processor, after it executes an RTE to return from the handler, attempts to restart the instruction during which the exception occurred.

7.5.2.8 ADDRESS AND BUS ERRORS. Coprocessor-instruction-related bus faults can occur during main processor bus cycles to CPU space to communicate with a coprocessor or during memory cycles run as part of the coprocessor instruction execution. If a bus error occurs during the CIR access that is used to initiate a coprocessor instruction, the main processor assumes that the coprocessor is not present and takes an F-line emulator exception as described in **7.5.2.2 F-Line Emulator Exceptions**. That is, the processor takes an F-line emulator exception when a bus error occurs during the initial access to a CIR by a coprocessor instruction. If a bus error occurs on any other coprocessor access or on a memory access made during the execution of a coprocessor instruction, the main processor performs bus error exception processing as described in **Section 6 Exception Processing**. After the exception handler has corrected the cause of the bus error, the main processor can return to the point in the coprocessor instruction at which the fault occurred.

An address error occurs if the MC68020/EC020 attempts to prefetch an instruction from an odd address. This can occur if the calculated destination address of a cpBcc or cpDBcc instruction is odd or if an odd value is transferred to the scanPC with the transfer status register and the scanPC response primitive. If an address error occurs, the MC68020/EC020 performs exception processing for a bus fault as described in **Section 6 Exception Processing**.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.5.3 Coprocessor Reset

Either an external reset signal or a RESET instruction can reset the external devices of a system. The system designer can design a coprocessor to be reset and initialized by both reset types or by external reset signals only. To be consistent with the MC68020/EC020 design, the coprocessor should be affected by external reset signals only and not by RESET instructions, because the coprocessor is an extension to the main processor programming model and to the internal state of the MC68020/EC020.

7.6 COPROCESSOR SUMMARY

Coprocessor instruction formats are included with the instruction formats in the M68000PM/AD, *M68000 Family Programmer's Reference Manual*.

The M68000 coprocessor response primitive formats are shown in this section. Any response primitive with bits 13–8 = $00 or $3F causes a protocol violation exception. Response primitives with bits 13–8 = $0B, $18–$1B, $1F, $28–$2B, and $38–3B currently cause protocol violation exceptions; they are undefined and reserved for future use by Motorola.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Busy

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Transfer Multiple Coprocessor Registers

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	0	0	0	1	LENGTH	

Transfer Status Register and ScanPC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	0	1	SP	0	0	0	0	0	0	0	0

Supervisor Check

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	PC	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Take Address and Transfer Data

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	0	1	0	1	LENGTH	

Transfer Multiple Main Processor Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	1	1	0	0	0	0	0	0	0	0	0

Transfer Operation Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	0	1	1	1	0	0	0	0	0	0	0	0

Null

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	0	IA	0	0	0	0	0	0	PF	TF

Evaluate and Transfer Effective Address

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	0	0	1	0	1	0	0	0	0	0	0	0	0	0

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Transfer Single Main Processor Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
CA	PC	DR	0	1	1	0	0	0	0	0	0	D/A	REGISTER	

Transfer Main Processor Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	0	1	0	0	0	0	0	0	0	0

Transfer to/from Top of Stack

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	1	1	1	0	LENGTH	

Transfer from Instruction Stream

15	14	13	12	11	10	9	8	7	0
CA	PC	0	0	1	1	1	1	LENGTH	

Evaluate Effective Address and Transfer Data

15	14	13	12	11	10	8	7	0
CA	PC	DR	1	0	VALID EA		LENGTH	

Take Preinstruction Exception

15	14	13	12	11	10	9	8	7	0
0	PC	0	1	1	1	0	0	VECTOR NUMBER	

Take Midinstruction Exception

15	14	13	12	11	10	9	8	7	0
0	PC	0	1	1	1	0	1	VECTOR NUMBER	

Take Postinstruction Exception

15	14	13	12	11	10	9	8	7	0
0	PC	0	1	1	1	1	0	VECTOR NUMBER	

Write to Previously Evaluated Effective Address

15	14	13	12	11	10	9	8	7	0
CA	PC	1	0	0	0	0	0	LENGTH	

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 8
INSTRUCTION EXECUTION TIMING

This section describes the instruction execution and operations (table searches, etc.) of the MC68020/EC020 in terms of external clock cycles. It provides accurate execution and operation timing guidelines but not exact timings for every possible circumstance. This approach is used since exact execution time for an instruction or operation is highly dependent on memory speeds and other variables. The timing numbers presented in this section allow the assembly language programmer or compiler writer to predict timings needed to evaluate the performance of the MC68020/EC020.

In this section, instruction and operation times are shown in clock cycles, which eliminates clock frequency dependencies.

8.1 TIMING ESTIMATION FACTORS

The advanced architecture of the MC68020/EC020 makes exact instruction timing calculations difficult due to the effects of:

1. An On-Chip Instruction Cache and Instruction Prefetch
2. Operand Misalignment
3. Bus Controller/Sequence Concurrency
4. Instruction Execution Overlap

These factors make MC68020/EC020 instruction set timing difficult to calculate on a single instruction basis since instructions vary in execution time from one context to another. A detailed explanation of each of these factors follows.

8.1.1 Instruction Cache and Prefetch

The on-chip cache of the MC68020/EC020 is an instruction-only cache. Its purpose is to increase execution efficiency by providing a quick store for instructions.

Instruction prefetches that hit in the cache will occur with no delay in instruction execution. Instruction prefetches that miss in the cache will cause an external memory cycle to be performed, which may overlap with internal instruction execution. Thus, while the execution unit of the microprocessor is busy, the bus controller prefetches the next instruction from external memory. Both cases are illustrated in later examples.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

When prefetching instructions from external memory, the microprocessor will utilize long word read cycles. When the read is aligned on a long-word address boundary, the processor reads two words, which may load two instructions at once or two words of a multiword instruction. The subsequent instruction prefetch will find the second word is already available, and there is no need to run an external bus cycle (read).

The MC68020/EC020 always prefetches long words. When an instruction prefetch falls on an odd-word boundary (e.g., due to a branch to an odd-word location), the MC68020/EC020 will read the even word associated with the long-word base address at the same time as (32-bit memory) or before (8- or 16-bit memory) the odd word is read. When an instruction prefetch falls on an even-word boundary (as would be the normal case), the MC68020/EC020 reads both words at the long-word address, thus effectively prefetching the next two words.

8.1.2 Operand Misalignment

Another significant factor affecting instruction timing is operand misalignment. Operand misalignment has impact on performance when the microprocessor is reading or writing external memory. In this case, the address of a word operand falls across a long-word boundary, or a long-word operand falls on a byte or word address that is not a long-word boundary. Although the MC68020/EC020 will automatically handle all occurrences of operand misalignment, it must use multiple bus cycles to complete such transfers.

8.1.3 Bus/Sequencer Concurrency

The bus controller is responsible for all bus activity. The sequencer controls the bus controller, instruction execution, and internal processor operation, such as calculation of effective addresses and setting of condition codes.

The bus controller and sequencer can operate on an instruction concurrently. The bus controller can perform a read or write while the sequencer controls an effective address calculation or sets the condition codes. The sequencer may also request a bus cycle that the bus controller cannot immediately perform. In this case, the bus cycle is queued and the bus controller runs the cycle when the current cycle is complete.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.1.4 Instruction Execution Overlap

Overlap is the time, measured in clocks, when two instructions execute concurrently. In Figure 8-1, instructions A and B execute concurrently, and the overlapped portion of instruction B is absorbed in the instruction execution time of A (the previous instruction). The overlap time is deducted from the execution time of instruction B. Similarly, there is an overlap period between instruction B and instruction C, which reduces the attributed execution time for C.

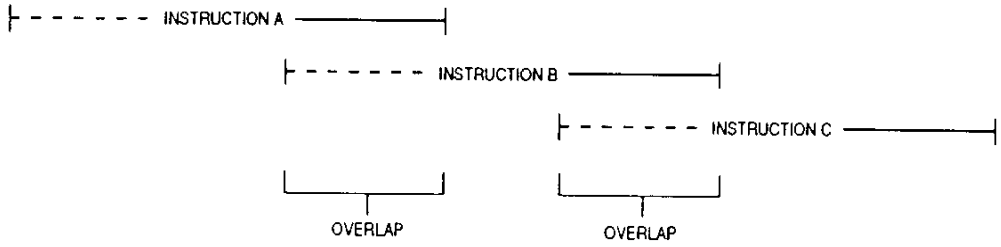


Figure 8-1. Concurrent Instruction Execution

The execution time attributed to instructions A, B, and C (after considering the overlap) is depicted in Figure 8-2.

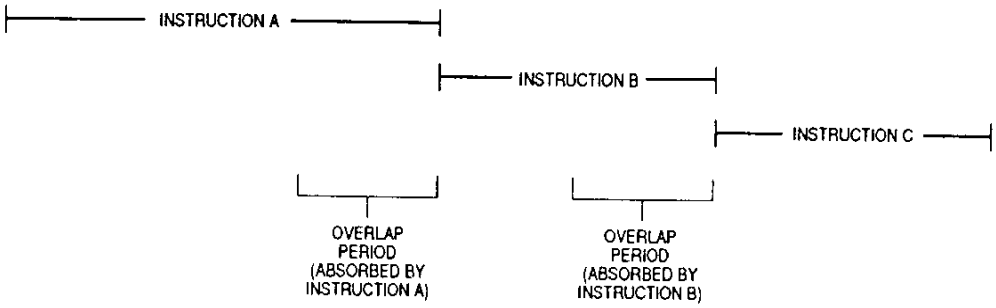


Figure 8-2. Instruction Execution for Instruction Timing Purposes

It is possible that the execution time of an instruction will be absorbed by the overlap with a previous instruction for a net execution time of zero clocks.

Because of this overlap, a NOP is required between a write to a peripheral to clear an interrupt request and a subsequent MOVE to SR instruction to lower the interrupt mask level. Otherwise, the MOVE to SR instruction may complete before the write is accomplished, and a new interrupt exception will be generated for an old interrupt request.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.1.5 Instruction Stream Timing Examples

A programming example allows a more detailed examination of these effects. The effect of instruction execution overlap on instruction timing is illustrated by the following example instruction stream.

Instruction	
#1) MOVE.L	D4,(A1)+
#2) ADD.L	D4,D5
#3) MOVE.L	(A1), –(A2)
#4) ADD.L	D5,D6

Example 1

For the first example, the assumptions are:

1. The data bus is 32 bits,
2. The first instruction is prefetched from an odd-word address,
3. Memory access occurs with no wait states, and
4. The instruction cache is disabled.

For example 1, the instruction stream is positioned in 32-bit memory as follows:

Address			
	n	•••	MOVE #1
	n + 4	ADD #2	MOVE #3
	n + 8	ADD #4	•••

Figure 8-3 shows processor activity on the first example instruction stream. It shows the activity of the external bus, the bus controller, the sequencer, and the attributed instruction execution time.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

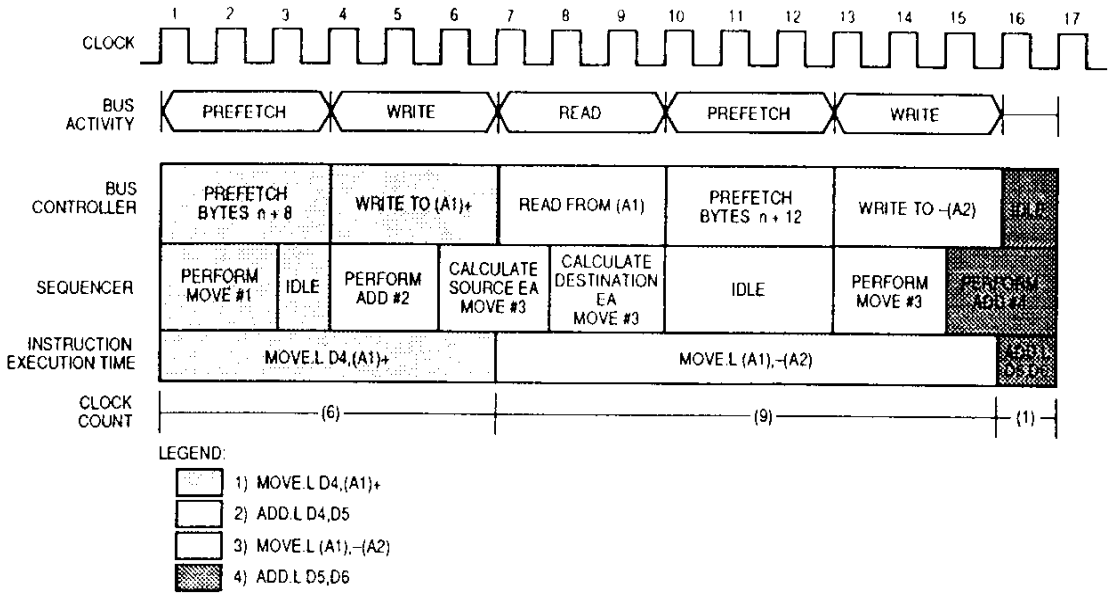


Figure 8-3. Processor Activity for Example 1

For the first three clocks of this example, the bus controller and sequencer are both performing tasks associated with the MOVE #1 instruction. The next three clocks (clocks 4, 5, and 6) demonstrate instruction overlap. The bus controller is performing a write to memory as part of the MOVE #1 instruction. The sequencer, on the other hand, is performing the ADD #2 instruction for two clocks (clocks 4 and 5) and beginning source effective address (EA) calculations for the MOVE #3 instruction. The bus controller activity completely overlaps the execution of the ADD #2 instruction, causing the ADD #2 attributed execution time to be zero clocks. The overlap also shortens the effective execution time of the MOVE #3 instruction by one clock because the bus controller completes the MOVE #1 write operation while the sequencer begins the MOVE #3 EA calculation.

The sequencer continues the source EA calculation for one more clock period (clock 7) while the bus controller begins a read for MOVE #3. When counting instruction execution time in bus clocks, the MOVE #1 completes at the end of clock 6, and the execution of MOVE #3 begins on clock 7.

Both the sequencer and bus controller continue with MOVE #3 until the end of clock 14, when the sequencer begins to perform ADD #4. Timing for MOVE #3 continues because the bus controller is still performing the write to the destination of MOVE #3. The bus activity for MOVE #3 completes at the end of clock 15. The effective execution time for MOVE #3 is nine clocks.

The one clock cycle (clock 15) when the sequencer is performing ADD #4 and the bus controller is writing to the destination of MOVE #3 is absorbed by the execution time of MOVE #3. This overlap shortens the effective execution time of ADD #4 by one clock, giving it an attributed execution time of one clock.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Example 2

Using the same instruction stream, the second example demonstrates the different effects of instruction execution overlap on instruction timing when the same instructions are positioned slightly differently in 32-bit memory:

Address			
	n	MOVE #1	ADD #2
	n + 4	MOVE #3	ADD #4
	n + 8	•••	•••

The assumptions for example 2 (see Figure 8-4) are:

1. The data bus is 32 bits,
2. The first instruction is prefetched from an even-word address,
3. Memory access occurs with no wait states, and
4. The cache is disabled.

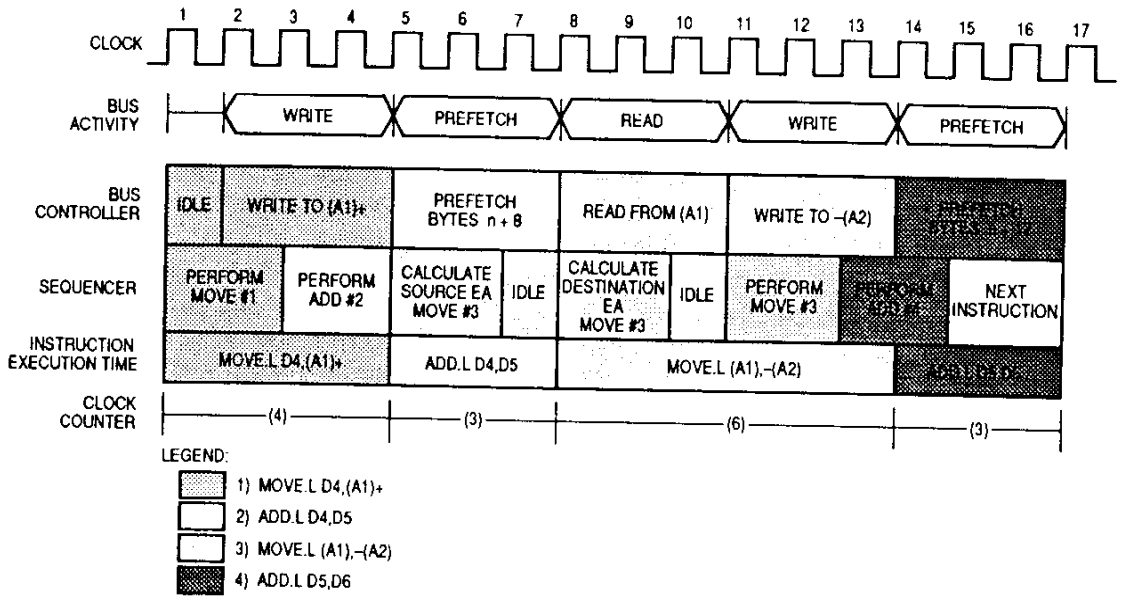


Figure 8-4. Processor Activity for Example 2

Although the total execution time of the instruction segment does not change in this example, the individual instruction times are significantly different. This example demonstrates that the effects of overlap are not only instruction-sequence dependent but are also dependent upon the alignment of the instruction stream in memory.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Example 3

Both Figures 8-3 and 8-4 show instruction execution without benefit of the MC68020/EC020 instruction cache. Figure 8-5 shows a third example for the same instruction stream executing in the cache. Note that once the instructions are in the cache, the original location in external memory is no longer a factor in timing.

The assumptions for Example 3 are:

1. The data bus is 32 bits,
2. The cache is enabled and instructions are in the cache, and
3. Memory access occurs with no wait states.

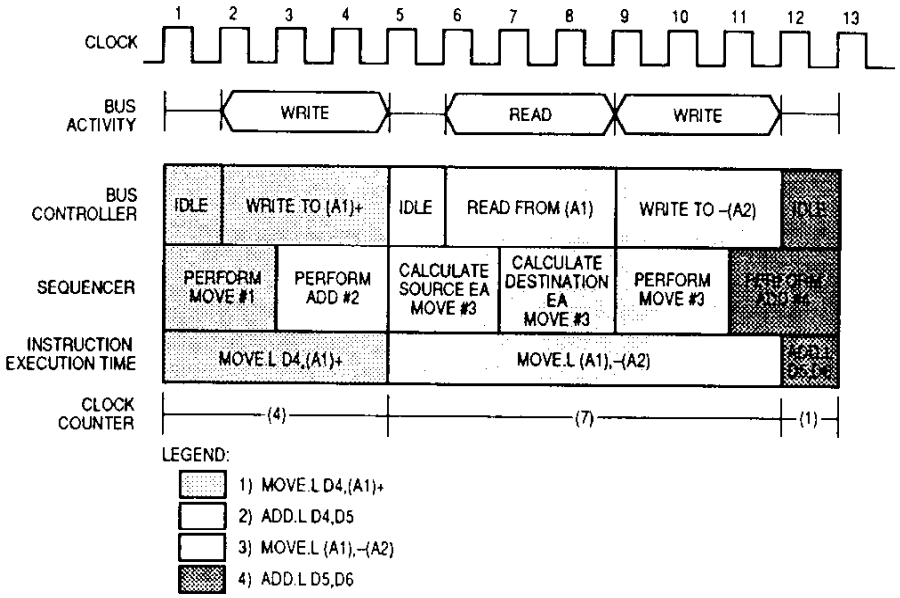


Figure 8-5. Processor Activity for Example 3

Figure 8-5 illustrates the benefits of the instruction cache. The total number of clock cycles is reduced from 16 to 12 clocks. Since the instructions are resident in the cache, the instruction prefetch activity does not require the bus controller to perform external bus cycles. Since prefetch occurs with no delay, the bus controller is idle more often.

Example 4

Idle clock cycles, such as those shown in example 3, are useful in MC68020/EC020 systems that require wait states when accessing external memory. This fact is illustrated in example 4 (see Figure 8-6) with the following assumptions:

1. The data bus is 32 bits,
2. The cache is enabled and instructions are in the cache, and
3. Memory access occurs with one wait state.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

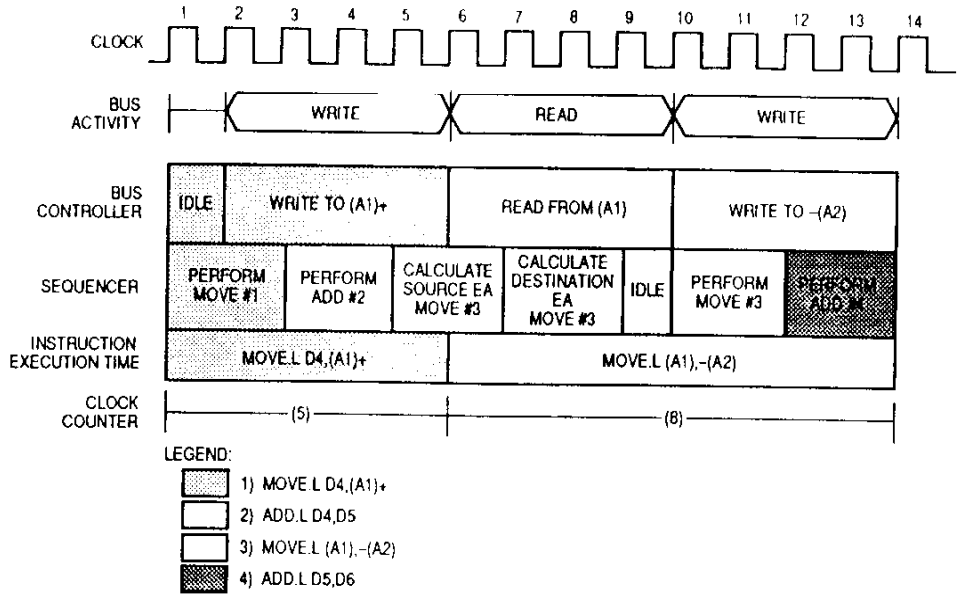


Figure 8-6. Processor Activity for Example 4

Figure 8-6 shows the same instruction stream executing with four clocks for every read and write. The idle bus cycles coincide with the wait states of the memory access; therefore, the total execution time is only 13 clocks.

Examples 1–4 demonstrate the complexity of instruction timing calculation for the MC68020/EC020. It is impossible to anticipate individual instruction timing as an absolute number of clock cycles due to the dependency of overlap on the instruction sequence and alignment as well as the number of wait states in memory. This can be seen by comparing individual and composite time for Figures 8-3 through 8-6. These instruction timings are compared in Table 8-1, where timing varies for each instruction as the context varies.

Table 8-1. Examples 1–4 Instruction Stream Execution Comparison

Instruction	Example 1 (Odd Alignment)	Example 2 (Even Alignment)	Example 3 (Cache)	Example 4 (Cache with Wait States)
#1) MOVE.L D4,(A1)+	6	4	4	5
#2) ADD.L D4,D5	0	3	0	0
#3) MOVE.L (A1),-(A2)	9	6	7	8
#4) ADD.L D5,D6	1	3	1	0
Total Clock Cycles	16	16	12	13

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2 INSTRUCTION TIMING TABLES

The instruction times given in the following illustration include the following assumptions about the MC68020/EC020 system:

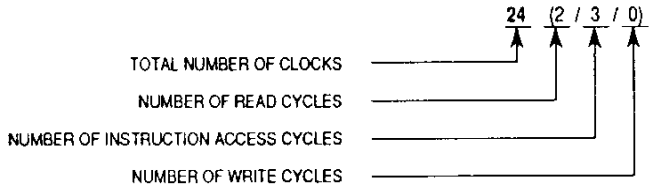
1. All operands are long-word aligned as is the stack,
2. The data bus is 32 bits, and
3. Memory access occurs with no wait states (three-cycle read/write).

There are three values given for each instruction and addressing mode:

1. The best case (BC), which reflects the time (in clocks) when the instruction is in the cache and benefits from maximum overlap due to other instructions,
2. The cache-only case (CC) when the instruction is in the cache but has no overlap, and
3. The worst case (WC) when the instruction is not in the cache or the cache is disabled and there is no instruction overlap.

The only instances for which the size of the operand has any effect are the instructions with immediate operands. Unless specified otherwise, immediate byte and word operands have identical execution times.

Within each set or column of instruction timings are four sets of numbers, three of which are enclosed in parentheses. The bolded outer number is the total number of clocks for the instruction. The first number inside the parentheses is the number of operand read cycles performed by the instruction. The second value inside parentheses is the number of instruction accesses performed by the instruction, including all prefetches to keep the instruction pipe filled. The third value within parentheses is the number of write cycles performed by the instruction. One example from the instruction timing table is:



The total number of bus-activity clocks for the previous example is derived in the following way:

(2 Reads * 3 Clocks/Read) + (3 Instruction Accesses * 3 Clocks/Access)
+ (0 Writes * 3 Clocks/Write) = 15 Clocks of Bus Activity
24 Total Clocks – 15 Clocks (Bus Activity) = 9 Internal Clocks

The example used here was taken from a worst-case fetch effective address time. The addressing mode was ([d_{32},B],I,d_{32}). The same addressing mode under the best-case

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

entry is 17 (2/0/0). For the best case, there are no instruction accesses because the cache is enabled and the sequencer does not have to go to external memory for the instruction words.

The first tables deal exclusively with fetching and calculating effective addresses and immediate operands. The tables are arranged in this manner because some instructions do not require effective address calculation or fetching. For example, the instruction CLR <ea> (found in the table under **8.2.11 Single Operand Instructions**) only needs to have a calculated effective address time added to its table entry because no fetch of an operand is required. This instruction only writes to memory or a register. Some instructions use specific addressing modes which exclude timing for calculation or fetching of an operand. When these instances arise, they are footnoted to indicate which other tables are needed in the timing calculation.

Many two-word instructions (e.g., MULU.L, DIV.L, BFSET, etc.) include the fetch immediate effective address time or the calculate immediate effective address time in the execution time calculation. The timing for immediate data of word length (#<data>.W) is used for these calculations. If the instruction has a source and a destination, the source effective address is used for the table lookup. If the instruction is single operand, the effective address of that operand is used.

The following example includes multiword instructions that refer to the fetch immediate effective address and calculate immediate effective address tables in **8.2 Instruction Timing Tables**.

Instruction

#1) MULU.L D7,D1:D2
#2) BFCLR $6000{0:8}
#3) DIVS.L #$10000,D3:D4

	CC
1.MULU.L (D7),D1:D2	
#<data>.W,Dn	2
MUL.L EA,Dn	43
2.BFCLR $6000{0:8}	
#<data>.W.,$XXX.W	5
BFCLR Mem (<5 bytes)	16
3.DIVS.L #$10000,D3:D4	
#<data>.W,#<data>.L	6
DIVS.L EA, Dn	90

Execution time = 2 + 43 + 5 + 16 + 6 + 90
= 102 clock periods

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

NOTE

This CC time is a maximum since the times given for the MULU.L and DIVS.L are maximums.

The MOVE instruction timing tables include all necessary timing for extension word fetch, address calculation, and operand fetch.

The instruction timing tables are used to calculate a best-case and worst-case bounds for some target instruction stream. Calculating exact timing from the timing tables is impossible because the tables cannot anticipate how the combination of factors will influence every particular sequence of instructions. This is illustrated by comparing the observed instruction timing from the prior four examples with instruction timing derived from the instruction timing tables.

Table 8-2 lists the original instruction stream and the corresponding clock timing from the appropriate timing tables for the best case, cache-only case, and worst case.

Table 8-2. Instruction Timings from Timing Tables

Instruction		Best Case	Cache Case	Worst Case
#1) MOVE.L	D4,(A1)+	4	4	6
#2) ADD.L	D4,D5	0	2	3
#3) MOVE.L	(A1),-(A2)	6	7	9
#4) ADD.L	D5,D6	0	2	3
Total		10	15	21

Table 8-3 summarizes the observed instruction timings for the same instruction stream as executed according to the assumptions of the four examples. For each example, Table 8-3 shows which entry (BC/CC/WC) from the timing tables corresponds to the observed timing for each of the four instructions. Some of the observed instruction timings cannot be found in the timing tables and appear in Table 8-3 within parentheses in the most appropriate column. These timings occur when instruction execution overlap dynamically alters what would otherwise be a BC, CC, or WC timing.

Table 8-3. Observed Instruction Timings

Instruction		Example 1			Example 2			Example 3			Example 4		
		BC	CC	WC	BC	CC	WC	BC	CC	WC	BC	CC	WC
#1) MOVE.L	D4,(A1)+			6	4				4			(5)	
#2) ADD.L	D4,D5	0					3	0			0		
#3) MOVE.L	(A1),-(A2)			9	6				7			(8)	
#4) ADD.L	D5,D6	(1)					3	(1)			0		
Total			(16)			(16)			(12)			(13)	

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Comparing Tables 8-2 and 8-3 demonstrates that calculation of instruction timing cannot be a simple lookup of only BC or only WC timings. Even when the assumptions are known and fixed, as in the four examples summarized in Table 8-3, the microprocessor can sometimes achieve best-case timings under worst-case assumptions.

Looking across the four examples in Table 8-3 for an individual instruction, it is difficult to predict which timing table entry is used, since the influence of instruction overlap may or may not improve the BC, WC, or CC timings. When looking at the observed instruction timings for one example, it is also difficult to determine which combination of BC/CC/WC timing is required. Just how the instruction stream will fit and run with the cache enabled, how instructions are positioned in memory, and the degree of instruction overlap are factors that are impossible to account for in all combinations of the timing tables.

Although the timing tables cannot accurately predict the instruction timing that would be observed when executing an instruction stream on the MC68020/EC020, the tables can be used to calculate best-case and worst-case bounds for instruction timing. Absolute instruction timing must be measured by using the microprocessor itself to execute the target instruction stream.

For More Information On This Product,
Go to: www.freescale.com

8.2.1 Fetch Effective Address

The fetch effective address table indicates the number of clock periods needed for the processor to calculate and fetch the specified effective address. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Address Mode	Best Case	Cache Case	Worst Case
Dn	**0**(0/0/0)	**0**(0/0/0)	**0**(0/0/0)
An	**0**(0/0/0)	**0**(0/0/0)	**0**(0/0/0)
(An)	**3**(1/0/0)	**4**(1/0/0)	**4**(1/0/0)
(An)+	**4**(1/0/0)	**4**(1/0/0)	**4**(1/0/0)
–(An)	**3**(1/0/0)	**5**(1/0/0)	**5**(1/0/0)
(d_{16},An) of (d_{16},PC)	**3**(1/0/0)	**5**(1/0/0)	**6**(1/1/0)
(xxx).W	**3**(1/0/0)	**4**(1/0/0)	**6**(1/1/0)
(xxx).L	**3**(1/0/0)	**4**(1/0/0)	**7**(1/1/0)
#<data>.B	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
#<data>.W	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
#<data>.L	**0**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
(d_8,An,Xn) or (d_8,PC,Xn)	**4**(1/0/0)	**7**(1/0/0)	**8**(1/1/0)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**4**(1/0/0)	**7**(1/0/0)	**9**(1/1/0)
(B)	**4**(1/0/0)	**7**(1/0/0)	**9**(1/1/0)
(d_{16},B)	**6**(1/0/0)	**9**(1/0/0)	**12**(1/1/0)
(d_{32},B)	**10**(1/0/0)	**13**(1/0/0)	**16**(1/2/0)
([B],I)	**9**(2/0/0)	**12**(2/0/0)	**13**(2/1/0)
([B],I,d_{16})	**11**(2/0/0)	**14**(2/0/0)	**16**(2/1/0)
([B],I,d_{32})	**11**(2/0/0)	**14**(2/0/0)	**17**(2/2/0)
([d_{16},B],I)	**11**(2/0/0)	**14**(2/0/0)	**16**(2/1/0)
([d_{16},B],I,d_{16})	**13**(2/0/0)	**16**(2/0/0)	**19**(2/2/0)
([d_{16},B],I,d_{32})	**13**(2/0/0)	**16**(2/0/0)	**20**(2/2/0)
([d_{32},B],I)	**15**(2/0/0)	**18**(2/0/0)	**20**(2/2/0)
([d_{32},B],I,d_{16})	**17**(2/0/0)	**20**(2/0/0)	**22**(2/2/0)
([d_{32},B],I,d_{32})	**17**(2/0/0)	**20**(2/0/0)	**24**(2/3/0)

B = Base address; 0, An, PC, Xn, An + Xn. Form does not affect timing.

I = Index; 0, Xn

NOTE: Xn cannot be in B and I at the same time. Scaling and size of Xn do not affect timing.

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.2 Fetch Immediate Effective Address

The fetch immediate effective address table indicates the number of clock periods needed for the processor to fetch the immediate source operand and calculate and fetch the specified destination operand. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Address Mode	Best Case	Cache Case	Worst Case
#<data>.W,Dn	0(0/0/0)	2(0/0/0)	3(0/1/0)
#<data>.L,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
#<data>.W,(An)	3(1/0/0)	4(1/0/0)	4(1/1/0)
#<data>.L,(An)	3(1/0/0)	4(1/0/0)	7(1/1/0)
#<data>.W,(An)+	4(1/0/0)	6(1/0/0)	7(1/1/0)
#<data>.L,(An)+	5(1/0/0)	8(1/0/0)	9(1/1/0)
#<data>.W,–(An)	3(1/0/0)	5(1/0/0)	6(1/1/0)
#<data>.L,–(An)	4(1/0/0)	7(1/0/0)	8(1/1/0)
#<data>.W,(bd,An)	3(1/0/0)	5(1/0/0)	7(1/1/0)
#<data>.L,(bd,An)	4(1/0/0)	7(1/0/0)	10(1/2/0)
#<data>.W,xxx.W	3(1/0/0)	5(1/0/0)	7(1/1/0)
#<data>.L,xxx.W	4(1/0/0)	7(1/0/0)	10(1/2/0)
#<data>.W,xxx.L	3(1/0/0)	6(1/0/0)	10(1/2/0)
#<data>.L,xxx.L	4(1/0/0)	8(1/0/0)	12(1/2/0)
#<data>.W,#<data>.B,W	0(0/0/0)	4(0/0/0)	6(0/2/0)
#<data>.L,#<data>.B,W	1(0/0/0)	6(0/0/0)	8(0/2/0)
#<data>.W,#<data>.L	0(0/0/0)	6(0/0/0)	8(0/2/0)
#<data>.L,#(data>.L	1(0/0/0)	8(0/0/0)	10(0/2/0)
#<data>.W,(d_8,An,Xn) or (d_8,PC,Xn)	4(1/0/0)	9(1/0/0)	11(1/2/0)
#<data>.L,(d_8,An,Xn) or (d_8,PC,Xn)	5(1/0/0)	11(1/0/0)	13(1/2/0)
#<data>.W,(d_{16},An,Xn) or (d_{16},PC,Xn)	4(1/0/0)	9(1/0/0)	12(1/2/0)
#<data>.L,(d_{16},An,Xn) or (d_{16},PC,Xn)	5(1/0/0)	11(1/0/0)	15(1/2/0)
#<data>.W,(B)	4(1/0/0)	9(1/0/0)	12(1/1/0)
#<data>.L,(B)	5(1/0/0)	11(1/0/0)	14(1/2/0)
#<data>.W,(bd,PC)	10(1/0/0)	15(1/0/0)	19(1/3/0)
#<data>.L,(bd,PC)	11(1/0/0)	17(1/0/0)	21(1/3/0)
#<data>.W,(d_{16},B)	6(1/0/0)	11(1/0/0)	15(1/2/0)
#<data>.L,(d_{16},B)	7(1/0/0)	13(1/0/0)	17(1/2/0)
#<data>.W,(d_{32},B)	10(1/0/0)	15(1/0/0)	19(1/3/0)
#<data>.L,(d_{32},B)	11(1/0/0)	17(1/0/0)	21(1/3/0)
#<data>.W,([B],I)	9(2/0/0)	14(2/0/0)	16(2/2/0)
#<data>.L,([B],I)	10(2/0/0)	16(2/0/0)	18(2/2/0)
#<data>.W,([B],I,d_{16})	11(2/0/0)	16(2/0/0)	19(2/2/0)
#<data>.L,([B],I,d_{16})	12(2/0/0)	18(2/0/0)	21(2/2/0)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Address Mode	Best Case	Cache Case	Worst Case
#<data>.W,([B],I,d_{32})	**11**(2/0/0)	**16**(2/0/0)	**20**(2/2/0)
#<data>.L,([d_{16},B],I,d_{32})	**12**(2/0/0)	**18**(2/0/0)	**22**(2/3/0)
#<data>.W,([d_{16},B],I)	**11**(2/0/0)	**16**(2/0/0)	**19**(2/2/0)
#<data>.L,([d_{16},B],I)	**12**(2/0/0)	**18**(2/0/0)	**21**(2/2/0)
#<data>.W,([d_{16},B],I,d_{16})	**13**(2/0/0)	**18**(2/0/0)	**22**(2/2/0)
#<data>.L,([d_{16},B],I,d_{16})	**14**(2/0/0)	**20**(2/0/0)	**24**(2/3/0)
#<data>.W,([d_{32},B],I)	**15**(2/0/0)	**20**(2/0/0)	**23**(2/3/0)
#<data>.L,([d_{32},B],I)	**16**(2/0/0)	**22**(2/0/0)	**25**(2/3/0)
#<data>.W,([d_{32},B],I,d_{16})	**17**(2/0/0)	**22**(2/0/0)	**25**(2/3/0)
#<data>.L,([d_{32},B],I,d_{16})	**18**(2/0/0)	**24**(2/0/0)	**27**(2/3/0)
#<data>.W,([d_{32},b],I,d_{32})	**17**(2/0/0)	**22**(2/0/0)	**27**(2/3/0)
#<data>.L,([d_{32},b],I,d_{32})	**18**(2/0/0)	**24**(2/0/0)	**29**(2/4/0)

B = Base address; 0, An, PC, Xn, An + Xn. Form does not affect timing.

I = Index; 0, Xn

NOTE: Xn cannot be in B and I at the same time. Scaling and size of Xn do not affect timing.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.3 Calculate Effective Address

The calculate immediate effective address table indicates the number of clock periods needed for the processor to calculate the specified effective address. Fetch time is only included for the first level of indirection on memory indirect addressing modes. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Address Mode	Best Case	Cache Case	Worst Case
Dn	**0**(0/0/0)	**0**(0/0/0)	**0**(0/0/0)
An	**0**(0/0/0)	**0**(0/0/0)	**0**(0/0/0)
(An)	**2**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
(An)+	**2**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
−(An)	**2**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
(d_{16},An) or (d_{16},PC)	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
<data>.W	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
<data>.L	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
(d_8,An,Xn) or (d_8,PC,Xn)	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
(B)	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
(d_{16},B)	**5**(0/0/0)	**8**(0/0/0)	**10**(0/1/0)
(d_{32},B)	**9**(0/0/0)	**12**(0/0/0)	**15**(0/2/0)
([B],I)	**8**(1/0/0)	**11**(1/0/0)	**12**(1/1/0)
([B],I,d_{16})	**10**(1/0/0)	**13**(1/0/0)	**15**(1/1/0)
([B],I,d_{32})	**10**(1/0/0)	**13**(1/0/0)	**16**(1/2/0)
([d_{16},B],I)	**10**(1/0/0)	**13**(1/0/0)	**15**(1/1/0)
([d_{16},B],I,d_{16})	**12**(1/0/0)	**15**(1/0/0)	**18**(1/2/0)
([d_{16},B],I,d_{32})	**12**(1/0/0)	**15**(1/0/0)	**19**(1/2/0)
([d_{32},B],I)	**14**(1/0/0)	**17**(1/0/0)	**19**(1/2/0)
([d_{32},B],I,d_{16})	**16**(1/0/0)	**19**(1/0/0)	**21**(1/2/0)
([d_{32},B],I,d_{32})	**16**(1/0/0)	**19**(1/0/0)	**24**(1/3/0)

B = Base address; 0, An, PC, Xn, An + Xn. Form does not affect timing.

I = Index; 0, Xn

NOTE: Xn cannot be in B and I at the same time. Scaling and size of Xn do not affect timing.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.4 Calculate Immediate Effective Address

The calculate immediate effective address table indicates the number of clock periods needed for the processor to fetch the immediate source operand and calculate the specified destination effective address. Fetch time is only included for the first level of indirection on memory indirect addressing modes. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Address Mode	Best Case	Cache Case	Worst Case
#<data>.W,Dn	0(0/0/0)	2(0/0/0)	3(0/1/0)
#<data>.L,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
#<data>.W,(An)	0(0/0/0)	2(0/0/0)	3(0/1/0)
#<data>.L,(An)	1(0/0/0)	4(0/0/0)	5(0/1/0)
#<data>.W,(An)+	2(0/0/0)	4(0/0/0)	5(0/1/0)
#<data>.L,(An)+	3(0/0/0)	6(0/0/0)	7(0/1/0)
#<data>.W,(bd,An)	1(0/0/0)	4(0/0/0)	5(0/1/0)
#<data>.L,(bd,An)	3(0/0/0)	6(0/0/0)	8(0/2/0)
#<data>.W,xxx.W	1(0/0/0)	4(0/0/0)	5(0/1/0)
#<data>.L,xxx.W	3(0/0/0)	6(0/0/0)	8(0/2/0)
#<data>.W,xxx.L	2(0/0/0)	4(0/0/0)	6(0/2/0)
#<data>.L,xxx.L	3(0/0/0)	8(0/0/0)	10(0/2/0)
#<data>.W,(d_8,An,Xn) or (d_8,PC,Xn)	0(0/0/0)	6(0/0/0)	8(0/2/0)
#<data>.L,(d_8,An,Xn) or (d_8,PC,Xn)	2(0/0/0)	8(0/0/0)	10(0/2/0)
#<data>.W,(d_{16},An,Xn) or (d_{16},PC,Xn)	3(0/0/0)	8(0/0/0)	10(0/2/0)
#<data>.L,(d_{16},An,Xn) or (d_{16},PC,Xn)	4(0/0/0)	10(0/0/0)	12(0/2/0)
#<data>.W,(B)	3(0/0/0)	8(0/0/0)	10(0/1/0)
#<data>.L,(B)	4(0/0/0)	10(0/0/0)	12(0/2/0)
#<data>.W,(bd,PC)	9(0/0/0)	14(0/0/0)	18(0/3/0)
#<data>.L,(bd,PC)	10(0/0/0)	16(0/0/0)	20(0/3/0)
#<data>.W,(d_{16},B)	5(0/0/0)	10(0/0/0)	13(0/2/0)
#<data>.L,(d_{16},B)	6(0/0/0)	12(0/0/0)	15(0/2/0)
#<data>.W,(d_{32},B)	9(0/0/0)	14(0/0/0)	18(0/2/0)
#<data>.L,(d_{32},B)	10(0/0/0)	16(0/0/0)	20(0/3/0)
#<data>.W,([B],I)	8(1/0/0)	13(1/0/0)	15(1/2/0)
#<data>.L,([B],I)	9(1/0/0)	15(1/0/0)	17(1/2/0)
#<data>.W,([B],I,d_{16})	10(1/0/0)	15(1/0/0)	18(1/2/0)
#<data>.L,([B],I,d_{16})	11(1/0/0)	17(1/0/0)	20(1/2/0)
#<data>.W,([B],I,d_{32})	10(1/0/0)	15(1/0/0)	19(1/2/0)
#<data>.L,([d_{16},B],I,d_{32})	11(1/0/0)	17(1/0/0)	21(1/3/0)
#<data>.W,([d_{16},B],I)	10(1/0/0)	15(1/0/0)	18(1/2/0)
#<data>.L,([d_{16},B],I)	11(1/0/0)	17(1/0/0)	20(1/2/0)
#<data>.W,([d_{16},B],I,d_{16})	12(1/0/0)	17(1/0/0)	21(1/2/0)

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Address Mode	Best Case	Cache Case	Worst Case
#<data>.L,([d_{16},B],I,d_{16})	**13**(1/0/0)	**19**(1/0/0)	**23**(1/3/0)
#<data>.([d_{16},B],I,d_{32})	**12**(1/0/0)	**17**(1/0/0)	**22**(1/3/0)
#<data>.([d_{16},B],I,d_{32})	**13**(1/0/0)	**19**(1/0/0)	**24**(1/3/0)
#<data>.W,([d_{32},B],I)	**14**(1/0/0)	**19**(1/0/0)	**22**(1/3/0)
#<data>.L,([d_{32},B],I)	**15**(1/0/0)	**21**(1/0/0)	**24**(1/3/0)
#<data>.W,([d_{32},B],I,d_{16})	**16**(1/0/0)	**21**(1/0/0)	**24**(1/3/0)
#<data>.L,([d_{32},B],I,d_{16})	**17**(1/0/0)	**23**(1/0/0)	**26**(1/3/0)
#<data>.W,([d_{32},B],I,d_{32})	**16**(1/0/0)	**21**(1/0/0)	**24**(1/3/0)
#<data>.L,([d_{32},B],I,d_{32})	**17**(1/0/0)	**23**(1/0/0)	**29**(1/4/0)

B = Base address; 0, An, PC, Xn, An + Xn. Form does not affect timing.

I = Index; 0, Xn

NOTE: Xn cannot be in B and I at the same time. Scaling and size of Xn do not affect timing.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2.5 Jump Effective Address

The jump effective address table indicates the number of clock periods needed for the processor to calculate the specified effective address. Fetch time is only included for the first level of indirection on memory indirect addressing modes. The total number of clock cycles is outside the parentheses; the number or read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Address Mode	Best Case	Cache Case	Worst Case
(An)	**0**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
(d_{16},An)	**1**(0/0/0)	**4**(0/0/0)	**4**(0/0/0)
(xxx).W	**0**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
(xxx).L	**0**(0/0/0)	**2**(0/0/0)	**2**(0/0/0)
(d_8,An,Xn) or (d_8,PC,Xn)	**3**(0/0/0)	**6**(0/0/0)	**6**(0/0/0)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**3**(0/0/0)	**6**(0/0/0)	**6**(0/0/0)
(B)	**3**(0/0/0)	**6**(0/0/0)	**6**(0/0/0)
(B,d_{16})	**5**(0/0/0)	**8**(0/0/0)	**8**(0/1/0)
(B,d_{32})	**9**(0/0/0)	**12**(0/0/0)	**12**(0/1/0)
([B],I)	**8**(1/0/0)	**11**(1/0/0)	**11**(1/1/0)
([B],I,d_{16})	**10**(1/0/0)	**13**(1/0/0)	**14**(1/1/0)
([B],I,d_{32})	**10**(1/0/0)	**13**(1/0/0)	**14**(1/1/0)
([d_{16},B],I)	**10**(1/0/0)	**13**(1/0/0)	**14**(1/1/0)
([d_{16},B],I,d_{16})	**12**(1/0/0)	**15**(1/0/0)	**17**(1/1/0)
([d_{16},B],I,d_{32})	**12**(1/0/0)	**15**(1/0/0)	**17**(1/1/0)
([d_{32},B},I)	**14**(1/0/0)	**17**(1/0/0)	**19**(1/2/0)
([d_{32},B],I,d_{16})	**16**(1/0/0)	**19**(1/0/0)	**21**(1/2/0)
([d_{32},B],I,d_{32})	**16**(1/0/0)	**19**(1/0/0)	**23**(1/2/0)

B = Base address; 0, An, PC, Xn, An + Xn, PC + Xn. Form does not affect timing.

I = Index; 0, Xn

NOTE: Xn cannot be in B and I at the same time. Scaling and size of Xn do not affect timing.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.6 MOVE Instruction

The MOVE instruction table indicates the number of clock periods needed for the processor to fetch, calculate, and perform the MOVE or MOVEA with the specified source and destination effective addresses, including both levels of indirection on memory indirect addressing modes. No additional tables are needed to calculate the total effective execution time for the MOVE or MOVEA instruction. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

BEST CASE

Source Address Mode	Destination							
	An	Dn	(An)	(An)+	–(An)	(d_{16},An)	(xxx).W	(xxx).L
Rn	0(0/0/0)	0(0/0/0)	3(0/0/1)	4(0/0/1)	3(0/0/1)	3(0/0/1)	3(0/0/1)	5(0/0/1)
#<data>.B,W	0(0/0/0)	0(0/0/0)	3(0/0/1)	4(0/0/1)	3(0/0/1)	3(0/0/1)	3(0/0/1)	5(0/0/1)
#<data>.L	0(0/0/0)	0(0/0/0)	3(0/0/1)	4(0/0/1)	3(0/0/1)	3(0/0/1)	3(0/0/1)	5(0/0/1)
(An)	3(1/0/0)	3(1/0/0)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	8(1/0/1)
(An)+	4(1/0/0)	4(1/0/0)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	9(1/0/1)
–(An)	3(1/0/0)	3(1/0/0)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	8(1/0/1)
(d_{16},An) or (d_{16},PC)	3(1/0/0)	3(1/0/0)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	8(1/0/1)
(xxx).W	3(1/0/0)	3(1/0/0)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	8(1/0/1)
(xxx).L	3(1/0/0)	3(1/0/0)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	6(1/0/1)	8(1/0/1)
(d_8,An,Xn) or (d_8,PC,Xn)	4(1/0/0)	4(1/0/0)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	9(1/0/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	4(1/0/0)	4(1/0/0)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	9(1/0/1)
(B)	4(1/0/0)	4(1/0/0)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	9(1/0/1)
(d_{16},B)	6(1/0/0)	6(1/0/0)	9(1/0/1)	9(1/0/1)	9(1/0/1)	9(1/0/1)	9(1/0/1)	11(1/0/1)
(d_{32},B)	10(1/0/0)	10(1/0/0)	13(1/0/1)	13(1/0/1)	13(1/0/1)	13(1/0/1)	13(1/0/1)	15(1/0/1)
([B],I)	9(2/0/0)	9(2/0/0)	12(2/0/1)	12(2/0/1)	12(2/0/1)	12(2/0/1)	12(2/0/1)	14(2/0/1)
([B],I,d_{16})	11(2/0/0)	11(2/0/0)	14(2/0/1)	14(2/0/1)	14(2/0/1)	14(2/0/1)	14(2/0/1)	16(2/0/1)
([B],I,d_{32})	11(2/0/0)	11(2/0/0)	14(2/0/1)	14(2/0/1)	14(2/0/1)	14(2/0/1)	14(2/0/1)	16(2/0/1)
([d_{16},B],I)	11(2/0/0)	11(2/0/0)	14(2/0/1)	14(2/0/1)	14(2/0/1)	14(2/0/1)	14(2/0/1)	16(2/0/1)
([d_{16},B],I,d_{16})	13(2/0/0)	13(2/0/0)	16(2/0/1)	16(2/0/1)	16(2/0/1)	16(2/0/1)	16(2/0/1)	18(2/0/1)
([d_{16},B],d_{32})	13(2/0/0)	13(2/0/0)	16(2/0/1)	16(2/0/1)	16(2/0/1)	16(2/0/1)	16(2/0/1)	18(2/0/1)
([d_{32},B],I)	15(2/0/0)	15(2/0/0)	18(2/0/1)	18(2/0/1)	18(2/0/1)	18(2/0/1)	18(2/0/1)	20(2/0/1)
([d_{32},B],I,d_{16})	17(2/0/0)	17(2/0/0)	20(2/0/1)	20(2/0/1)	20(2/0/1)	20(2/0/1)	20(2/0/1)	22(2/0/1)
([d_{32},B],I,d_{32})	17(2/0/0)	17(2/0/0)	20(2/0/1)	20(2/0/1)	20(2/0/1)	20(2/0/1)	20(2/0/1)	22(2/0/1)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

BEST CASE (Continued)

Source Address Mode	Destination							
	(d_8,An,Xn)	(d_{16},An,Xn)	(B)	(d_{16},B)	(d_{32},B)	([B],I)	([B],I,d_{16})	([B],I,d_{32})
Rn	**4**(0/0/1)	**6**(0/0/1)	**5**(0/0/1)	**7**(0/0/1)	**11**(0/0/1)	**9**(1/0/1)	**11**(1/0/1)	**12**(1/0/1)
#<data>.B,W	**4**(0/0/1)	**6**(0/0/1)	**5**(0/0/1)	**7**(0/0/1)	**11**(0/0/1)	**9**(1/0/1)	**11**(1/0/1)	**12**(1/0/1)
#<data>.L	**4**(0/0/1)	**6**(0/0/1)	**5**(0/0/1)	**7**(0/0/1)	**11**(0/0/1)	**9**(1/0/1)	**11**(1/0/1)	**12**(1/0/1)
(An)	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(An)+	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
−(An)	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(d_{16},An) or (d_{16},PC)	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(xxx).W	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(xxx).L	**8**(1/0/1)	**10**(1/0/1)	**9**(1/0/1)	**11**(1/0/1)	**15**(1/0/1)	**13**(2/0/1)	**15**(2/0/1)	**16**(2/0/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**9**(1/0/1)	**10**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(B)	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(d_{16},B)	**11**(1/0/1)	**13**(1/0/1)	**12**(1/0/1)	**14**(1/0/1)	**18**(1/0/1)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)
(d_{32},B)	**15**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)	**18**(1/0/1)	**22**(1/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
([B],I)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)	**17**(2/0/1)	**21**(2/0/1)	**19**(3/0/1)	**21**(3/0/1)	**22**(3/0/1)
([B],I,d_{16})	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**19**(2/0/1)	**23**(2/0/1)	**21**(3/0/1)	**23**(3/0/1)	**24**(3/0/1)
([B],I,d_{32})	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**19**(2/0/1)	**23**(2/0/1)	**21**(3/0/1)	**23**(3/0/1)	**24**(3/0/1)
([d_{16},B],I)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**19**(2/0/1)	**23**(2/0/1)	**21**(3/0/1)	**23**(3/0/1)	**24**(3/0/1)
([d_{16},B],I,d_{16})	**18**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**21**(2/0/1)	**25**(2/0/1)	**23**(3/0/1)	**25**(3/0/1)	**26**(3/0/1)
([d_{16},B],I,d_{32})	**18**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**21**(2/0/1)	**25**(2/0/1)	**23**(3/0/1)	**25**(3/0/1)	**26**(3/0/1)
([d_{32},B],I)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)	**23**(2/0/1)	**27**(2/0/1)	**25**(3/0/1)	**27**(3/0/1)	**28**(3/0/1)
([d_{32},B],I,d_{16})	**22**(2/0/1)	**24**(2/0/1)	**25**(2/0/1)	**25**(2/0/1)	**29**(2/0/1)	**27**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

BEST CASE (Concluded)

Source Address Mode	Destination					
	([d_{16},B],I)	([d_{16},B],I,d_{16})	([d_{16},B],I,d_{32})	([d_{32},B],I)	([d_{32},B],I,d_{16})	([d_{32},B],I,d_{32})
Rn	**11**(1/0/1)	**13**(1/0/1)	**14**(1/0/1)	**15**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)
#<data>.B,W	**11**(1/0/1)	**13**(1/0/1)	**14**(1/0/1)	**15**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)
#<data>.L	**11**(1/0/1)	**13**(1/0/1)	**14**(1/0/1)	**15**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)
(An)	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)
(An)+	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
−(An)	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)
(d_{16},An) or (d_{16},PC)	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)
(xxx).W	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)
(xxx).L	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(B)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(d_{16},B)	**18**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)	**24**(2/0/1)	**25**(2/0/1)
(d_{32},B)	**22**(2/0/1)	**24**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)	**28**(2/0/1)	**29**(2/0/1)
([B],I)	**21**(3/0/1)	**23**(3/0/1)	**24**(3/0/1)	**25**(3/0/1)	**27**(3/0/1)	**28**(3/0/1)
([B],I,d_{16})	**23**(3/0/1)	**25**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)
([B],I,d_{32})	**23**(3/0/1)	**25**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)
([d_{16},B],I)	**23**(3/0/1)	**25**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)
([d_{16},B],I,d_{16})	**25**(3/0/1)	**27**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)	**31**(3/0/1)	**32**(3/0/1)
([d_{16},B],I,d_{32})	**25**(3/0/1)	**27**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)	**31**(3/0/1)	**32**(0/0/1)
([d_{32},B],I)	**27**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)	**33**(3/0/1)	**34**(3/0/1)
([d_{32},B],I,d_{16})	**29**(3/0/1)	**31**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)	**35**(3/0/1)	**36**(3/0/1)
([d_{32},B],I,d_{32})	**29**(3/0/1)	**31**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)	**35**(3/0/1)	**36**(3/0/1)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

CACHE CASE

Source Address Mode	Destination							
	An	Dn	(An)	(An)+	–(An)	(d_{16},An)	(xxx).W	(xxx).L
Rn	2(0/0/0)	2(0/0/0)	4(0/0/1)	4(0/0/1)	5(0/0/1)	5(0/0/1)	4(0/0/1)	6(0/0/1)
#<data>.B,W	4(0/0/0)	4(0/0/0)	6(0/0/1)	6(0/0/1)	7(0/0/1)	7(0/0/1)	6(0/0/1)	8(0/0/1)
#<data>.L	6(0/0/0)	6(0/0/0)	8(0/0/1)	8(0/0/1)	9(0/0/1)	9(0/0/1)	8(0/0/1)	10(0/0/1)
(An)	6(1/0/0)	6(1/0/0)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	9(1/0/1)
(An)+	6(1/0/0)	6(1/0/0)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	9(1/0/1)
–(An)	7(1/0/0)	7(1/0/0)	8(1/0/1)	8(1/0/1)	8(1/0/1)	8(1/0/1)	8(1/0/1)	10(1/0/1)
(d_{16},An) or (d_{16},PC)	7(1/0/0)	7(1/0/0)	8(1/0/1)	8(1/0/1)	8(1/0/1)	8(1/0/1)	8(1/0/1)	10(1/0/1)
(xxx).W	6(1/0/0)	6(1/0/0)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	9(1/0/1)
(xxx).L	6(1/0/0)	6(1/0/0)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	7(1/0/1)	9(1/0/1)
(d_8,An,Xn) or (d_8,PC,Xn)	9(1/0/0)	9(1/0/0)	10(1/0/1)	10(1/0/1)	10(1/0/1)	10(1/0/1)	10(1/0/1)	12(1/0/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	9(1/0/0)	9(1/0/0)	10(1/0/1)	10(1/0/1)	10(1/0/1)	10(1/0/1)	10(1/0/1)	12(1/0/1)
(B)	9(1/0/0)	9(1/0/0)	10(1/0/1)	10(1/0/1)	10(1/0/1)	10(1/0/1)	10(1/0/1)	12(1/0/1)
(d_{16},B)	11(1/0/0)	11(1/0/0)	12(1/0/1)	12(1/0/1)	12(1/0/1)	12(1/0/1)	12(1/0/1)	14(1/0/1)
(d_{32},B)	15(1/0/0)	15(1/0/0)	16(1/0/1)	16(1/0/1)	16(1/0/1)	16(1/0/1)	16(1/0/1)	18(1/0/1)
([B],I)	14(2/0/0)	14(2/0/0)	15(2/0/1)	15(2/0/1)	15(2/0/1)	15(2/0/1)	15(2/0/1)	17(2/0/1)
([B],I,d_{16})	16(2/0/0)	16(2/0/0)	17(2/0/1)	17(2/0/1)	17(2/0/1)	17(2/0/1)	17(2/0/1)	19(2/0/1)
([B],I,d_{32})	16(2/0/0)	16(2/0/0)	17(2/0/1)	17(2/0/1)	17(2/0/1)	17(2/0/1)	17(2/0/1)	19(2/0/1)
([d_{16},B],I)	16(2/0/0)	16(2/0/0)	17(2/0/1)	17(2/0/1)	17(2/0/1)	17(2/0/1)	17(2/0/1)	19(2/0/1)
([d_{16},B],I,d_{16})	18(2/0/0)	18(2/0/0)	19(2/0/1)	19(2/0/1)	19(2/0/1)	19(2/0/1)	19(2/0/1)	21(2/0/1)
([d_{16},B],d_{32})	18(2/0/0)	18(2/0/0)	19(2/0/1)	19(2/0/1)	19(2/0/1)	19(2/0/1)	19(2/0/1)	21(2/0/1)
([d_{32},B],I)	20(2/0/0)	20(2/0/0)	21(2/0/1)	21(2/0/1)	21(2/0/1)	21(2/0/1)	21(2/0/1)	23(2/0/1)
([d_{32},B],I,d_{16})	22(2/0/0)	22(2/0/0)	23(2/0/1)	23(2/0/1)	23(2/0/1)	23(2/0/1)	23(2/0/1)	25(2/0/1)
([d_{32},B],I,d_{32})	22(2/0/0)	22(2/0/0)	23(2/0/1)	23(2/0/1)	23(2/0/1)	23(2/0/1)	23(2/0/1)	25(2/0/1)

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

CACHE CASE (Continued)

Source Address Mode	Destination							
	(d_8,An,Xn)	(d_{16},An,Xn)	(B)	(d_{16},B)	(d_{32},B)	([B],I)	$([B],I,d_{16})$	$([B],I,d_{32})$
Rn	7(0/0/1)	9(0/0/1)	8(0/0/1)	10(0/0/1)	14(0/0/1)	12(1/0/1)	14(1/0/1)	15(1/0/1)
#<data>.B,W	7(0/0/1)	9(0/0/1)	8(0/0/1)	10(0/0/1)	14(0/0/1)	12(1/0/1)	14(1/0/1)	15(1/0/1)
#<data>.L	9(0/0/1)	11(0/0/1)	10(0/0/1)	12(0/0/1)	16(0/0/1)	14(1/0/1)	16(1/0/1)	17(1/0/1)
(An)	9(1/0/1)	11(1/0/1)	10(1/0/1)	12(1/0/1)	16(1/0/1)	14(2/0/1)	16(2/0/1)	17(2/0/1)
(An)+	9(1/0/1)	11(1/0/1)	10(1/0/1)	12(1/0/1)	16(1/0/1)	14(2/0/1)	16(2/0/1)	17(2/0/1)
−(An)	10(1/0/1)	12(1/0/1)	11(1/0/1)	13(1/0/1)	17(1/0/1)	15(2/0/1)	17(2/0/1)	18(2/0/1)
(d_{16},An) or (d_{16},PC)	10(1/0/1)	12(2/0/1)	11(1/0/1)	13(1/0/1)	17(1/0/1)	15(2/0/1)	17(2/0/1)	18(2/0/1)
(xxx).W	9(1/0/1)	11(1/0/1)	10(1/0/1)	12(1/0/1)	16(1/0/1)	14(2/0/1)	16(2/0/1)	17(2/0/1)
(xxx).L	9(1/0/1)	11(1/0/1)	10(1/0/1)	12(1/0/1)	16(1/0/1)	14(2/0/1)	16(2/0/1)	17(2/0/1)
(d_8,An,Xn) or (d_8,PC,Xn)	12(1/0/1)	14(1/0/1)	13(1/0/1)	15(1/0/1)	19(1/0/1)	17(2/0/1)	19(2/0/1)	20(2/0/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	12(1/0/1)	14(1/0/1)	13(1/0/1)	15(1/0/1)	19(1/0/1)	17(2/0/1)	19(2/0/1)	20(2/0/1)
(B)	12(1/0/1)	14(1/0/1)	13(1/0/1)	15(1/0/1)	19(1/0/1)	17(2/0/1)	19(2/0/1)	20(2/0/1)
(d_{16},B)	14(1/0/1)	16(1/0/1)	15(1/0/1)	17(1/0/1)	21(1/0/1)	19(2/0/1)	21(2/0/1)	22(2/0/1)
(d_{32},B)	18(1/0/1)	20(1/0/1)	19(1/0/1)	21(1/0/1)	25(1/0/1)	23(2/0/1)	25(2/0/1)	26(2/0/1)
([B],I)	17(2/0/1)	19(2/0/1)	18(2/0/1)	20(2/0/1)	24(2/0/1)	22(3/0/1)	24(3/0/1)	25(3/0/1)
$([B],I,d_{16})$	19(2/0/1)	21(2/0/1)	20(2/0/1)	22(2/0/1)	26(2/0/1)	24(3/0/1)	26(3/0/1)	27(3/0/1)
$([B],I,d_{32})$	19(2/0/1)	21(2/0/1)	20(2/0/1)	22(2/0/1)	26(2/0/1)	24(3/0/1)	26(3/0/1)	27(3/0/1)
$([d_{16},B],I)$	19(2/0/1)	21(2/0/1)	20(2/0/1)	22(2/0/1)	26(2/0/1)	24(3/0/1)	26(3/0/1)	27(3/0/1)
$([d_{16},B],I,d_{16})$	21(2/0/1)	23(2/0/1)	22(2/0/1)	24(2/0/1)	28(2/0/1)	26(3/0/1)	28(3/0/1)	29(3/0/1)
$([d_{16},B],I,d_{32})$	21(2/0/1)	23(2/0/1)	22(2/0/1)	24(2/0/1)	28(2/0/1)	26(3/0/1)	28(3/0/1)	29(3/0/1)
$([d_{32},B],I)$	23(2/0/1)	25(2/0/1)	24(2/0/1)	26(2/0/1)	30(2/0/1)	28(3/0/1)	30(3/0/1)	31(3/0/1)
$([d_{32},B],I,d_{16})$	25(2/0/1)	27(2/0/1)	26(2/0/1)	28(2/0/1)	32(2/0/1)	30(3/0/1)	32(3/0/1)	33(3/0/1)
$([d_{32},B],I,d_{32})$	25(2/0/1)	27(2/0/1)	26(2/0/1)	28(2/0/1)	32(2/0/1)	30(3/0/1)	32(3/0/1)	33(3/0/1)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

CACHE CASE (Concluded)

Source Address Mode	Destination					
	([d_{16},B],I)	([d_{16},B],I,d_{16})	([d_{16},B],I,d_{32})	([d_{32},B],I)	([d_{32},B],I,d_{16})	([d_{32},B],I,d_{32})
Rn	**14**(1/0/1)	**16**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)	**20**(1/0/1)	**21**(1/0/1)
#<data>.B,W	**14**(1/0/1)	**16**(1/0/1)	**17**(1/0/1)	**18**(1/0/1)	**20**(1/0/1)	**21**(1/0/1)
#<data>.L	**16**(1/0/1)	**18**(1/0/1)	**19**(1/0/1)	**20**(1/0/1)	**22**(1/0/1)	**23**(1/0/1)
(An)	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(An)+	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
-(An)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**23**(2/0/1)	**24**(2/0/1)
(d_{16},An) or (d_{16},PC)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**21**(2/0/1)	**23**(2/0/1)	**24**(2/0/1)
(xxx).W	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(xxx).L	**16**(2/0/1)	**18**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)
(B)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)	**23**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)
(d_{16},B)	**21**(2/0/1)	**23**(2/0/1)	**24**(2/0/1)	**25**(2/0/1)	**27**(2/0/1)	**28**(2/0/1)
(d_{32},B)	**25**(2/0/1)	**27**(2/0/1)	**28**(2/0/1)	**29**(2/0/1)	**31**(2/0/1)	**32**(2/0/1)
([B],I)	**24**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)	**28**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)
([B],I,d_{16})	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)
([B],I,d_{32})	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)
([d_{16},B],I)	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)
([d_{16},B],I,d_{16})	**28**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)	**32**(3/0/1)	**34**(3/0/1)	**35**(3/0/1)
([d_{16},B],I,d_{32})	**28**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)	**32**(3/0/1)	**34**(3/0/1)	**35**(3/0/1)
([d_{32},B],I)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)	**34**(3/0/1)	**36**(3/0/1)	**37**(3/0/1)
([d_{32},B],I,d_{16})	**32**(3/0/1)	**34**(3/0/1)	**35**(3/0/1)	**36**(3/0/1)	**38**(3/0/1)	**39**(3/0/1)
([d_{32},B],I,d_{32})	**32**(3/0/1)	**34**(3/0/1)	**35**(3/0/1)	**36**(3/0/1)	**38**(3/0/1)	**39**(3/0/1)

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

WORST CASE

Source Address Mode	Destination							
	An	Dn	(An)	(An)+	–(An)	(d_{16},An)	(xxx).W	(xxx).L
Rn	3(0/1/0)	3(0/1/0)	5(0/1/0)	5(0/1/1)	6(0/1/1)	7(0/1/1)	7(0/1/1)	9(0/2/1)
#<data>.B,W	3(0/1/0)	3(0/1/0)	5(0/1/0)	8(0/1/1)	6(0/1/1)	7(0/1/1)	7(0/1/1)	9(0/2/1)
#<data>.L	5(0/1/0)	5(0/1/0)	7(0/0/1)	7(0/1/1)	8(0/1/1)	9(0/1/1)	9(0/1/1)	11(0/2/1)
(An)	7(1/1/0)	7(1/1/0)	9(1/1/1)	9(1/1/1)	9(1/1/1)	11(1/1/1)	11(1/1/1)	13(1/2/1)
(An)+	7(1/1/0)	7(1/1/0)	9(1/1/1)	9(1/1/1)	9(1/1/1)	11(1/1/1)	11(1/1/1)	13(1/2/1)
–(An)	8(1/1/0)	8(1/1/0)	10(1/1/1)	10(1/1/1)	10(1/1/1)	12(1/1/1)	12(1/1/1)	14(1/2/1)
(d_{16},An) or (d_{16},PC)	9(1/2/0)	9(1/2/0)	11(1/2/1)	11(1/2/1)	11(1/2/1)	13(1/2/1)	13(1/2/1)	15(1/3/1)
(xxx).W	8(1/2/0)	8(1/2/0)	10(1/2/1)	10(1/2/1)	10(1/2/1)	12(1/2/1)	12(1/2/1)	14(1/3/1)
(xxx).L	10(1/2/0)	10(1/2/0)	12(1/2/1)	12(1/2/1)	12(1/2/1)	14(1/2/1)	14(1/2/1)	16(1/3/1)
(d_8,An,Xn) or (d_8,PC,Xn)	11(1/2/0)	11(1/2/0)	13(1/2/1)	13(1/2/1)	13(1/2/1)	15(1/2/1)	15(1/2/1)	17(1/3/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	12(1/2/0)	12(1/2/0)	14(1/2/1)	14(1/2/1)	14(1/2/1)	16(1/2/1)	16(1/2/1)	18(1/3/1)
(B)	12(1/2/0)	12(1/2/0)	14(1/2/1)	14(1/2/1)	14(1/2/1)	16(1/2/1)	16(1/2/1)	18(1/3/1)
(d_{16},B)	15(1/2/0)	15(1/2/0)	17(1/2/1)	17(1/2/1)	17(1/3/1)	19(1/2/1)	19(1/2/1)	21(1/3/1)
(d_{32},B)	19(1/3/0)	19(1/3/0)	21(1/3/1)	21(1/3/1)	21(1/3/1)	23(1/3/1)	23(1/3/1)	25(1/4/1)
([B],I)	16(2/2/0)	16(2/2/0)	18(2/2/1)	18(2/2/1)	18(2/2/1)	20(2/2/1)	20(2/2/1)	22(2/3/1)
([B],I,d_{16})	19(2/2/0)	19(2/2/0)	21(2/2/1)	21(2/2/1)	21(2/2/1)	23(2/2/1)	23(2/2/1)	25(2/3/1)
([B],I,d_{32})	20(2/3/0)	20(2/3/0)	22(2/3/1)	22(2/3/1)	22(2/3/1)	24(2/3/1)	24(2/3/1)	26(2/4/1)
([d_{16},B],I)	19(2/2/0)	19(2/2/0)	21(2/2/1)	21(2/2/1)	21(2/2/1)	23(2/2/1)	23(2/2/1)	25(2/3/1)
([d_{16},B],I,d_{16})	22(2/3/0)	22(2/3/0)	24(2/3/1)	24(2/3/1)	24(2/3/1)	26(2/3/1)	26(2/3/1)	28(2/4/1)
([d_{16},B],d_{32})	23(2/3/0)	23(2/3/0)	25(2/3/1)	25(2/3/1)	25(2/3/1)	27(2/3/1)	27(2/3/1)	29(2/4/1)
([d_{32},B],I)	23(2/3/0)	23(2/3/0)	25(2/3/1)	25(2/3/1)	25(2/3/1)	27(2/3/1)	27(2/3/1)	29(2/4/1)
([d_{32},B],I,d_{16})	25(2/3/0)	25(2/3/0)	27(2/3/1)	27(2/3/1)	27(2/3/1)	29(2/3/1)	29(2/3/1)	31(2/4/1)
([d_{32},B],I,d_{32})	27(2/4/0)	27(2/4/0)	29(2/4/1)	29(2/4/1)	29(2/4/1)	31(2/4/1)	31(2/4/1)	33(2/5/1)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

WORST CASE (Continued)

Source Address Mode	Destination							
	(d_8,An,Xn)	(d_{16},An,Xn)	(B)	(d_{16},B)	(d_{32},B)	([B],I)	([B],I,d_{16})	([B],I,D_{32})
Rn	9(0/1/1)	12(0/2/1)	10(0/1/1)	14(0/2/1)	19(0/2/1)	14(1/1/1)	17(1/2/1)	20(1/2/1)
#<data>.B,W	9(0/1/1)	12(0/2/1)	10(0/1/1)	14(0/2/1)	19(0/2/1)	14(1/1/1)	17(1/2/1)	20(1/2/1)
#<data>.L	11(0/1/1)	14(0/2/1)	12(0/1/1)	16(0/2/1)	21(0/2/1)	16(1/1/1)	19(1/2/1)	22(1/2/1)
(An)	11(1/1/1)	14(1/2/1)	12(1/1/1)	16(1/2/1)	21(1/2/1)	12(2/1/1)	19(2/2/1)	22(2/2/1)
(An)+	11(1/1/1)	14(1/2/1)	12(1/1/1)	16(1/2/1)	21(1/2/1)	12(2/1/1)	19(2/2/1)	22(2/2/1)
−(An)	12(1/1/1)	15(1/2/1)	13(1/1/1)	17(1/2/1)	22(1/2/1)	13(2/1/1)	20(2/2/1)	23(2/2/1)
(d_{16},An) or (d_{16},PC)	13(1/2/1)	16(1/3/1)	14(1/2/1)	18(1/3/1)	23(1/3/1)	14(2/2/1)	21(2/3/1)	24(2/3/1)
(xxx).W	12(1/2/1)	15(1/3/1)	13(1/2/1)	17(1/3/1)	22(1/3/1)	13(2/2/1)	20(2/3/1)	23(2/3/1)
(xxx).L	14(1/2/1)	17(1/3/1)	15(1/2/1)	19(1/3/1)	24(1/3/1)	15(2/2/1)	22(2/3/1)	25(2/3/1)
(d_8,An,Xn) or (d_8,PC,Xn)	15(1/2/1)	18(1/3/1)	16(1/2/1)	20(1/3/1)	25(1/3/1)	16(2/2/1)	23(2/3/1)	26(2/3/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	16(1/2/1)	19(1/3/1)	17(1/2/1)	21(1/3/1)	26(1/3/1)	17(2/2/1)	24(2/3/1)	27(2/3/1)
(B)	16(1/2/1)	19(1/3/1)	17(1/2/1)	21(1/3/1)	26(1/3/1)	17(2/2/1)	24(2/3/1)	27(2/3/1)
(d_{16},B)	19(1/2/1)	22(1/3/1)	20(1/2/1)	24(1/3/1)	29(1/3/1)	20(2/2/1)	27(2/3/1)	30(2/3/1)
(d_{32},B)	23(1/3/1)	26(1/4/1)	24(1/3/1)	28(1/4/1)	33(1/4/1)	24(2/3/1)	31(2/4/1)	34(2/4/1)
([B],I)	20(2/2/1)	23(2/3/1)	21(2/2/1)	25(2/3/1)	30(2/3/1)	21(3/2/1)	28(3/3/1)	31(3/3/1)
([B],I,d_{16})	23(2/2/1)	26(2/3/1)	24(2/2/1)	28(2/3/1)	33(2/3/1)	24(3/2/1)	31(3/3/1)	34(3/3/1)
([B],I,d_{32})	24(2/3/1)	27(2/4/1)	25(2/3/1)	29(2/4/1)	34(2/4/1)	25(3/3/1)	32(3/4/1)	35(3/4/1)
([d_{16},B],I)	23(2/2/1)	26(2/3/1)	24(2/2/1)	28(2/3/1)	33(2/3/1)	24(3/2/1)	31(3/3/1)	34(3/3/1)
([d_{16},B],I,d_{16})	26(2/3/1)	29(2/4/1)	27(2/3/1)	31(2/4/1)	36(2/4/1)	27(3/3/1)	34(3/4/1)	37(3/4/1)
([d_{16},B],I,d_{32})	27(2/3/1)	30(2/4/1)	28(2/3/1)	32(2/4/1)	37(2/4/1)	28(3/3/1)	35(3/4/1)	38(3/4/1)
([d_{32},B],I)	27(2/3/1)	30(2/4/1)	28(2/3/1)	32(2/4/1)	37(2/4/1)	28(3/3/1)	35(3/4/1)	38(3/4/1)
([d_{32},B],I,d_{16})	29(2/3/1)	32(2/4/1)	30(2/3/1)	34(2/4/1)	39(2/4/1)	30(3/3/1)	37(3/4/1)	40(3/4/1)
([d_{32},B],I,d_{32})	31(2/4/1)	34(2/5/1)	32(2/4/1)	36(2/5/1)	41(2/5/1)	32(3/4/1)	39(3/5/1)	42(3/5/1)

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

WORST CASE (Concluded)

Source Address Mode	Destination					
	([d_{16},B],I)	([d_{16},B],I,d16)	([d_{16},B],I,d_{32})	([d_{32},B],I)	([d_{32},B],I,d_{16})	([d_{32},B],I,d_{32})
Rn	**17**(1/2/1)	**20**(1/2/1)	**23**(1/3/1)	**22**(1/2/1)	**25**(1/3/1)	**27**(1/3/1)
#<data>.B,W	**17**(1/2/1)	**20**(1/2/1)	**23**(1/3/1)	**22**(1/2/1)	**25**(1/3/1)	**27**(1/3/1)
#<data>.L	**19**(1/2/1)	**22**(1/2/1)	**25**(1/3/1)	**24**(1/2/1)	**27**(1/3/1)	**29**(1/3/1)
(An)	**19**(2/2/1)	**22**(2/2/1)	**25**(2/3/1)	**24**(2/2/1)	**27**(2/3/1)	**29**(2/3/1)
(An)+	**19**(2/2/1)	**22**(2/2/1)	**25**(2/3/1)	**24**(2/2/1)	**27**(2/3/1)	**29**(2/3/1)
–(An)	**20**(2/2/1)	**23**(2/2/1)	**26**(2/3/1)	**25**(2/2/1)	**28**(2/3/1)	**30**(2/3/1)
(d_{16},An) or (d_{16},PC)	**21**(2/3/1)	**24**(2/3/1)	**27**(2/4/1)	**26**(2/3/1)	**29**(2/4/1)	**31**(2/4/1)
(xxx).W	**20**(2/3/1)	**23**(2/3/1)	**26**(2/4/1)	**27**(2/3/1)	**28**(2/4/1)	**30**(2/4/1)
(xxx).L	**22**(2/3/1)	**25**(2/3/1)	**28**(2/4/1)	**29**(2/3/1)	**30**(2/4/1)	**32**(2/4/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**23**(2/3/1)	**26**(2/3/1)	**29**(2/4/1)	**30**(2/3/1)	**31**(2/4/1)	**33**(2/4/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**24**(2/3/1)	**27**(2/3/1)	**30**(2/4/1)	**31**(2/3/1)	**32**(2/4/1)	**34**(2/4/1)
(B)	**24**(2/3/1)	**27**(2/3/1)	**30**(2/4/1)	**31**(2/3/1)	**32**(2/4/1)	**34**(2/4/1)
(d_{16},B)	**27**(2/3/1)	**30**(2/3/1)	**33**(2/4/1)	**34**(2/3/1)	**35**(2/4/1)	**37**(2/4/1)
(d_{32},B)	**31**(2/4/1)	**34**(2/4/1)	**37**(2/5/1)	**38**(2/4/1)	**39**(2/5/1)	**41**(2/5/1)
([B],I)	**28**(3/3/1)	**31**(3/3/1)	**34**(3/4/1)	**35**(3/3/1)	**36**(3/4/1)	**38**(3/4/1)
([B],I,d_{16})	**31**(3/3/1)	**34**(3/3/1)	**37**(3/4/1)	**38**(3/3/1)	**39**(3/4/1)	**41**(3/4/1)
([B],I,d_{32})	**32**(3/4/1)	**35**(3/4/1)	**38**(3/5/1)	**39**(3/4/1)	**40**(3/5/1)	**42**(3/5/1)
([d_{16},B],I)	**31**(3/3/1)	**34**(3/3/1)	**37**(3/4/1)	**38**(3/3/1)	**39**(3/4/1)	**41**(3/4/1)
([d_{16},B],I,d_{16})	**34**(3/4/1)	**37**(3/4/1)	**40**(3/5/1)	**41**(3/4/1)	**42**(3/5/1)	**44**(3/5/1)
([d_{16},B],I,d_{32})	**35**(3/4/1)	**38**(3/4/1)	**41**(3/5/1)	**42**(3/4/1)	**43**(3/5/1)	**45**(3/5/1)
([d_{32},B],I)	**35**(3/4/1)	**38**(3/4/1)	**41**(3/5/1)	**42**(3/4/1)	**43**(3/5/1)	**45**(3/5/1)
([d_{32},B],I,d_{16})	**37**(3/4/1)	**40**(3/4/1)	**43**(3/5/1)	**44**(3/4/1)	**45**(3/5/1)	**47**(3/5/1)
([d_{32},B],I,d_{32})	**39**(3/5/1)	**42**(3/5/1)	**45**(3/6/1)	**46**(3/5/1)	**47**(3/6/1)	**49**(3/6/1)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2.7 Special-Purpose MOVE Instruction

The special-purpose MOVE instruction table indicates the number of clock periods needed for the processor to fetch, calculate, and perform the special-purpose MOVE operation on the control registers or specified effective address. The total number of clock cycles is outside the parentheses, the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction			Best Case	Cache Case	Worst Case
	EXG	Ry,Rx	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
	MOVEC	Cr,Rn	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
	MOVEC	Rn,Cr	**9**(0/0/0)	**12**(0/0/0)	**13**(0/1/0)
	MOVE	PSW,Rn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
†	MOVE	PSW,Mem	**5**(0/0/1)	**5**(0/0/1)	**7**(0/1/1)
*	MOVE	EA,CCR	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
*	MOVE	EA,SR	**8**(0/0/0)	**8**(0/0/0)	**11**(0/2/0)
‡	MOVEM	EA,RL	**8 + 4n**(n/0/0)	**8 + 4n**(n/0/0)	**9 + 4n**(n/1/0)
‡	MOVEM	RL,EA	**4 + 3n**(0/0/n)	**4 + 3n**(0/0/n)	**5 + 3n**(0/1/n)
	MOVEP.W	Dn,(d_{16},An)	**8**(0/0/2)	**11**(0/0/2)	**11**(0/1/2)
	MOVEP.L	Dn,(d_{16},An)	**14**(0/0/4)	**17**(0/0/4)	**17**(0/1/4)
	MOVEP.W	(d_{16},An),Dn	**10**(2/0/0)	**12**(2/0/0)	**12**(2/1/0)
	MOVEP.L	(d_{16},An),Dn	**16**(4/0/0)	**18**(4/0/0)	**18**(4/1/0)
‡	MOVES	EA,Rn	**7**(1/0/0)	**7**(1/0/0)	**8**(1/1/0)
‡	MOVES	Rn,EA	**5**(0/0/1)	**5**(0/0/1)	**7**(0/1/1)
	MOVE	USP	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
	SWAP	Rx,Ry	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)

n—Number of Registers to Transfer
RL—Register List
*Add Fetch Effective Address Time
†Add Calculate Effective Address Time
‡Add Calculate Immediate Address Time

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.8 Arithmetic/Logical Instructions

The arithmetic/logical instructions table indicates the number of clock periods needed for the processor to perform the specified arithmetic/logical operation using the specified addressing mode. It also includes, in worst case, the amount of time needed to prefetch the next instruction. Footnotes specify when to add either fetch address or fetch immediate effective address time. This sum gives the total effective execution time for the operation using the specified addressing mode. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction			Best Case	Cache Case	Worst Case
*	ADD	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	ADDA	EA,An	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	ADD	Dn,EA	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	AND	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	AND	Dn,EA	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	EOR	Dn,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	EOR	Dn,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	OR	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	OR	Dn,EA	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	SUB	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	SUBA	EA,An	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	SUB	Dn,EA	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
*	CMP	EA,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	CMPA	EA,An	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)
**	CMP2	EA,Rn	**16**(1/0/0)	**18**(1/0/0)	**18**(1/1/0)
*	MUL.W	EA,Dn	**25**(0/0/0)	**27**(0/0/0)	**28**(0/1/0)
**	MUL.L	EA,Dn	**41**(0/0/0)	**43**(0/0/0)	**44**(0/1/0)
*	DIVU.W	EA,Dn	**42**(0/0/0)	**44**(0/0/0)	**44**(0/1/0)
**	DIVU.L	EA,Dn	**76**(0/0/0)	**78**(0/0/0)	**79**(0/1/0)
*	DIVS.W	EA,Dn	**54**(0/0/0)	**56**(0/0/0)	**57**(0/1/0)
**	DIVS.L	EA,Dn	**88**(0/0/0)	**90**(0/0/0)	**91**(0/1/0)

*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2.9 Immediate Arithmetic/Logical Instructions

The immediate arithmetic/logical instructions table indicates the number of clock periods needed for the processor to fetch the source immediate data value and perform the specified arithmetic/logical operation using the specified destination addressing mode. Footnotes indicate when to add appropriate fetch effective or fetch immediate effective address time. This computation will give the total execution time needed to perform the appropriate immediate arithmetic/logical operation. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

	Instruction		Best Case	Cache Case	Worst Case
	MOVEQ	#<data>,Dn	0(0/0/0)	2(0/0/0)	3(0/1/0)
	ADDQ	#<data>,Rn	0(0/0/0)	2(0/0/0)	3(0/1/0)
*	ADDQ	#<data>,Mem	3(0/0/1)	4(0/0/1)	6(0/1/1)
	SUBQ	#<data>,Rn	0(0/0/0)	2(0/0/0)	3(0/1/0)
*	SUBQ	#<data>,Mem	3(0/0/1)	4(0/0/1)	6(0/1/1)
**	ADDI	#<data>,Dn	0(0/0/0)	2(0/0/0)	3(0/1/0)
**	ADDI	#<data>,Mem	3(0/0/1)	4(0/0/1)	6(0/1/1)
**	ANDI	#<data>,Dn	0(0/0/0)	2(0/0/0)	3(0/1/0)
**	ANDI	#<data>,Mem	3(0/0/1)	4(0/0/1)	6(0/1/1)
**	EORI	#<data>,Dn	0(0/0/0)	2(0/0/0)	3(0/1/0)
**	EORI	#<data>,Mem	3(0/0/1)	4(0/0/1)	6(0/1/1)
**	ORI	#<data>,Dn	0(0/0/0)	2(0/0/0)	3(0/1/0)
**	ORI	#<data>,Mem	3(0/0/1)	4(0/0/1)	6(0/1/1)
**	SUBI	#<data>,Dn	0(0/0/0)	2(0/0/0)	3(0/1/0)
**	SUBI	#<data>,Mem	3(0/0/1)	4(0/0/1)	6(0/1/1)
**	CMPI	#<data>,EA	0(0/0/0)	2(0/0/0)	3(0/1/0)

*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.10 Binary-Coded Decimal Operations

The binary-coded decimal operations table indicates the number of clock periods needed for the processor to perform the specified operation using the given addressing modes, with complete execution times given. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction		Best Case	Cache Case	Worst Case
ABCD	Dn,Dn	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
ABCD	–(An),–(An)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/1/1)
SBCD	Dn,Dn	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
SBCD	–(An),–(An)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/1/1)
ADDX	Dn,Dn	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
ADDX	–(An),–(An)	**10**(2/0/1)	**12**(2/0/1)	**13**(2/1/1)
SUBX	Dn,Dn	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
SUBX	–(An),–(An)	**10**(2/0/1)	**12**(2/0/1)	**13**(2/1/1)
CMPM	(An)+,(An)+	**8**(2/0/0)	**9**(2/0/0)	**10**(2/1/0)
PACK	Dn,Dn,#<data>	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
PACK	–(An),–(An),#<data>	**11**(1/0/1)	**13**(1/0/1)	**13**(1/1/1)
UNPK	Dn,Dn,#<data>	**5**(0/0/0)	**8**(0/0/0)	**9**(0/1/0)
UNPK	–(An),–(An),#<data>	**11**(1/0/1)	**13**(1/0/1)	**13**(1/1/1)

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.11 Single-Operand Instructions

The single-operand instructions table indicates the number of clock periods needed for the processor to perform the specified operation on the given addressing mode. Footnotes indicate when it is necessary to add another table entry to calculate the total effective execution time for the instruction. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction			Best Case	Cache Case	Worst Case
	CLR	Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
†	CLR	Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	NEG	Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	NEG	Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	NEGX	Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	NEGX	Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	NOT	Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	NOT	Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	EXT	Dn	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)
	NBCD	Dn	**6**(0/0/0)	**6**(0/0/0)	**6**(0/1/0)
	Scc	Dn	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)
†	Scc	Mem	**6**(0/0/1)	**6**(0/0/1)	**6**(0/1/1)
	TAS	Dn	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)
†	TAS	Mem	**12**(1/0/1)	**12**(1/0/1)	**13**(1/1/1)
*	TST	EA	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)

*Add Fetch Effective Address Time

†Add Calculate Effective Address Time

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.12 Shift/Rotate Instructions

The shift/rotate instructions table indicates the number of clock periods needed for the processor to perform the specified operation on the given addressing mode. Footnotes indicate when it is necessary to add another table entry to calculate the total effective execution time for the instruction. The number of bits shifted does not affect execution time. The total number of clock cycles is outside the parentheses, the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction			Best Case	Cache Case	Worst Case
	LSL	Dn (Static)	1(0/0/0)	4(0/0/0)	4(0/1/0)
	LSR	Dn (Static)	1(0/0/0)	4(0/0/0)	4(0/1/0)
	LSL	Dn (Dynamic)	3(0/0/0)	6(0/0/0)	6(0/1/0)
	LSR	Dn (Dynamic)	3(0/0/0)	6(0/0/0)	6(0/1/0)
*	LSL	Mem by 1	5(0/0/1)	5(0/0/1)	6(0/1/1)
*	LSR	Mem by 1	5(0/0/1)	5(0/0/1)	6(0/1/1)
	ASL	Dn	5(0/0/0)	8(0/0/0)	8(0/1/0)
	ASR	Dn	3(0/0/0)	6(0/0/0)	6(0/1/0)
*	ASL	Mem by 1	6(0/0/1)	6(0/0/1)	7(0/1/1)
*	ASR	Mem by 1	5(0/0/1)	5(0/0/1)	6(0/1/1)
	ROL	Dn	5(0/0/0)	8(0/0/0)	8(0/1/0)
	ROR	Dn	5(0/0/0)	8(0/0/0)	8(0/1/0)
*	ROL	Mem by 1	7(0/0/1)	7(0/0/1)	7(0/1/1)
*	ROR	Mem by 1	7(0/0/1)	7(0/0/1)	7(0/1/1)
	ROXL	Dn	9(0/0/0)	12(0/0/0)	12(0/1/0)
	ROXR	Dn	9(0/0/0)	12(0/0/0)	12(0/1/0)
*	ROXd	Mem by 1	5(0/0/1)	5(0/0/1)	6(0/1/1)

*Add Fetch Effective Address Time

d—Direction of Shift/Rotate, L or R

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2.13 Bit Manipulation Instructions

The bit manipulation instructions table indicates the number of clock periods needed for the processor to perform the specified bit operation on the given addressing mode. Footnotes indicate when it is necessary to add another table entry to calculate the total effective execution time for the instruction. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction			Best Case	Cache Case	Worst Case
	BTST	#<data>,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
	BTST	Dn,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
**	BTST	#<data>,Mem	4(0/0/0)	4(0/0/0)	5(0/1/0)
*	BTST	Dn,Mem	4(0/0/0)	4(0/0/0)	5(0/1/0)
	BCHG	#<data>,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
	BCHG	Dn,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
**	BCHG	#<data>,Mem	4(0/0/1)	4(0/0/1)	5(0/1/1)
*	BCHG	Dn,Mem	4(0/0/1)	4(0/0/1)	5(0/1/1)
	BCLR	#<data>,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
	BCLR	Dn,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
**	BCLR	#<data>,Mem	4(0/0/1)	4(0/0/1)	5(0/1/1)
*	BCLR	Dn,Mem	4(0/0/1)	4(0/0/1)	5(0/1/1)
	BSET	#<data>,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
	BSET	Dn,Dn	1(0/0/0)	4(0/0/0)	5(0/1/0)
**	BSET	#<data>,Mem	4(0/0/1)	4(0/0/1)	5(0/1/1)
*	BSET	Dn,Mem	4(0/0/1)	4(0/0/1)	5(0/1/1)

*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

8

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.14 Bit Field Manipulation Instructions

The bit field manipulation instructions table indicates the number of clock periods needed for the processor to perform the specified bit field operation using the given addressing mode. Footnotes indicate when it is necessary to add another table entry to calculate the total effective execution time for the instruction. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction			Best Case	Cache Case	Worst Case
	BFTST	Dn	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
‡	BFTST	Mem (< 5 Bytes)	**11**(1/0/0)	**11**(1/0/0)	**12**(1/1/0)
‡	BFTST	Mem (5 Bytes)	**15**(2/0/0)	**15**(2/0/0)	**16**(2/1/0)
	BFCHG	Dn	**9**(0/0/0)	**12**(0/0/0)	**12**(0/1/0)
‡	BFCHG	Mem (< 5 Bytes)	**16**(1/0/1)	**16**(1/0/1)	**16**(1/1/1)
‡	BFCHG	Mem (5 Bytes)	**24**(2/0/2)	**24**(2/0/2)	**24**(2/1/2)
	BFCLR	Dn	**9**(0/0/0)	**12**(0/0/0)	**12**(0/1/0)
‡	BFCLR	Mem (< 5 Bytes)	**16**(1/0/1)	**16**(1/0/1)	**16**(1/1/1)
‡	BFCLR	Mem (5 Bytes)	**24**(2/0/2)	**24**(2/0/2)	**24**(2/1/2)
	BFSET	Dn	**9**(0/0/0)	**12**(0/0/0)	**12**(0/1/0)
‡	BFSET	Mem (< 5 Bytes)	**16**(1/0/1)	**16**(1/0/1)	**16**(1/1/1)
‡	BFSET	Mem (5 Bytes)	**24**(2/0/2)	**24**(2/0/2)	**24**(2/1/2)
	BFEXTS	Dn	**5**(0/0/0)	**8**(0/0/0)	**8**(0/1/0)
‡	BFEXTS	Mem (< 5 Bytes)	**13**(1/0/0)	**13**(1/0/0)	**13**(1/1/0)
‡	BFEXTS	Mem (5 Bytes)	**18**(2/0/0)	**18**(2/0/0)	**18**(2/1/0)
	BFEXTU	Dn	**5**(0/0/0)	**8**(0/0/0)	**8**(0/1/0)
‡	BFEXTU	Mem (< 5 Bytes)	**13**(1/0/0)	**13**(1/0/0)	**13**(1/1/0)
‡	BFEXTU	Mem (5 Bytes)	**18**(2/0/0)	**18**(2/0/0)	**18**(2/1/0)
	BFINS	Dn	**7**(0/0/0)	**10**(0/0/0)	**10**(0/1/0)
‡	BFINS	Mem (< 5 Bytes)	**14**(1/0/1)	**14**(1/0/1)	**15**(1/1/1)
‡	BFINS	Mem (5 Bytes)	**20**(2/0/2)	**20**(2/0/2)	**21**(2/1/2)
	BFFFO	Dn	**15**(0/0/0)	**18**(0/0/0)	**18**(0/1/0)
‡	BFFFO	Mem (< 5 Bytes)	**24**(1/0/0)	**24**(1/0/0)	**24**(1/1/0)
‡	BFFFO	Mem (5 Bytes)	**32**(2/0/0)	**32**(2/0/0)	**32**(2/1/0)

‡Add Calculate Immediate Address Time

NOTE: A bit field of 32 bits may span five bytes that require two operand cycles to access or may span four bytes that require only one operand cycle to access.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2.15 Conditional Branch Instructions

The conditional branch instructions table indicates the number of clock periods needed for the processor to perform the specified branch on the given branch size, with complete execution times given. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction	Best Case	Cache Case	Worst Case
Bcc (Taken)	3(0/0/0)	6(0/0/0)	9(0/2/0)
Bcc.B (Not Taken)	1(0/0/0)	4(0/0/0)	5(0/1/0)
Bcc.W (Not Taken)	3(0/0/0)	6(0/0/0)	7(0/1/0)
Bcc.L (Not Taken)	3(0/0/0)	6(0/0/0)	9(0/2/0)
DBcc (cc = False, Count Not Expired)	3(0/0/0)	6(0/0/0)	9(0/2/0)
DBcc (cc = False, Count Expired)	7(0/0/0)	10(0/0/0)	10(0/3/0)
DBcc (cc = True)	3(0/0/0)	6(0/0/0)	7(0/1/0)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2.16 Control Instructions

The control instructions table indicates the number of clock periods needed for the processor to perform the specified operation. Footnotes specify when it is necessary to add an entry from another table to calculate the total effective execution time for the given instruction. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

	Instruction	Best Case	Cache Case	Worst Case
	ANDI to SR	**9**(0/0/0)	**12**(0/0/0)	**15**(0/2/0)
	EORI to SR	**9**(0/0/0)	**12**(0/0/0)	**15**(0/2/0)
	ORI to SR	**9**(0/0/0)	**12**(0/0/0)	**15**(0/2/0)
	ANDI to CCR	**9**(0/0/0)	**12**(0/0/0)	**15**(0/2/0)
	EORI to CCR	**9**(0/0/0)	**12**(0/0/0)	**15**(0/2/0)
	ORI to CCR	**9**(0/0/0)	**12**(0/0/0)	**15**(0/2/0)
	BSR	**5**(0/0/1)	**7**(0/0/1)	**13**(0/2/1)
**	CALLM (Type 0)	**28**(2/0/6)	**30**(2/0/6)	**36**(2/2/6)
**	CALLM (Type 1)—No Stack Copy	**48**(5/0/8)	**50**(5/0/8)	**56**(5/2/8)
**	CALLM (Type 1)—No Stack Copy	**55**(6/0/8)	**57**(6/0/8)	**64**(6/2/8)
**	CALLM (Type 1)—Stack Copy	**63 + 6n**(7 + n/0/8 + n)	**65 + 6n**(7 + n/0/8 + n)	**71 + 6n**(7 + n/2/8 + n)
‡	CAS (Successful Compare)	**15**(1/0/1)	**15**(1/0/1)	**16**(1/1/1)
‡	CAS (Unsuccessful Compare)	**12**(1/0/0)	**12**(1/0/0)	**13**(1/1/0)
	CAS2 (Successful Compare)	**23**(2/0/2)	**25**(2/0/2)	**28**(2/2/2)
	CAS2 (Unsuccessful Compare)	**19**(2/0/0)	**22**(2/0/0)	**25**(2/2/0)
*	CHK	**8**(0/0/0)	**8**(0/0/0)	**8**(0/1/0)
**	CHK2 EA,Rn	**16**(2/0/0)	**18**(2/0/0)	**18**(2/1/0)
%	JMP	**1**(0/0/0)	**4**(0/0/0)	**7**(0/2/0)
%	JSR	**3**(0/0/1)	**5**(0/0/1)	**11**(0/2/1)
†	LEA	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
	LINK.W	**3**(0/0/1)	**5**(0/0/1)	**7**(0/1/1)
	LINK.L	**4**(0/0/1)	**6**(0/0/1)	**10**(0/2/1)
	NOP	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
†	PEA	**3**(0/0/1)	**5**(0/0/1)	**6**(0/1/1)
	RTD	**9**(1/0/0)	**10**(1/0/0)	**12**(1/2/0)
	RTM (Type 0)	**18**(4/0/0)	**19**(4/0/0)	**22**(4/2/0)
	RTM (Type 1)	**31**(6/0/1)	**32**(6/0/1)	**35**(6/2/1)
	RTR	**13**(2/0/0)	**14**(2/0/0)	**15**(2/2/0)
	RTS	**9**(1/0/0)	**10**(1/0/0)	**12**(1/2/0)
	UNLK	**5**(1/0/0)	**6**(1/0/0)	**7**(1/1/0)

n—Number of Operand Transfers Required
*Add Fetch Effective Address Time
†Add Calculate Effective Address Time
%—Add Jump Effective Address Time
**Add Fetch Immediate Address Time
‡Add Calculate Immediate Address Time

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

8.2.17 Exception-Related Instructions

The exception-related instructions table indicates the number of clock periods needed for the processor to perform the specified exception-related action. Footnotes specify when it is necessary to add the entry from another table to calculate the total effective execution time for the given instruction. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction	Best Case	Cache Case	Worst Case
BKPT	**9**(1/0/0)	**10**(1/0/0)	**10**(1/0/0)
Interrupt (I-Stack)	**26**(2/0/4)	**26**(2/0/4)	**33**(2/2/4)
Interrupt (M-Stack)	**41**(2/0/8)	**41**(2/0/8)	**48**(2/2/8)
RESET Instruction	**518**(0/0/0)	**518**(0/0/0)	**519**(0/1/0)
STOP	**8**(0/0/0)	**8**(0/0/0)	**8**(0/0/0)
Trace	**25**(1/0/5)	**25**(1/0/5)	**32**(1/2/5)
TRAP #n	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
Illegal Instruction	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
A-Line Trap	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
F-Line Trap	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
Privilege Violation	**20**(1/0/4)	**20**(1/0/4)	**27**(1/2/4)
TRAPcc (Trap)	**23**(1/0/5)	**25**(1/0/5)	**32**(1/2/5)
TRAPcc (No Trap)	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
TRAPcc.W (Trap)	**23**(1/0/5)	**25**(1/0/5)	**33**(1/3/5)
TRAPcc.W (No Trap)	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
TRAPcc.L (Trap)	**23**(1/0/5)	**25**(1/0/5)	**33**(1/3/5)
TRAPcc.L (No Trap)	**5**(0/0/0)	**8**(0/0/0)	**10**(0/2/0)
TRAPV (Trap)	**23**(1/0/5)	**25**(1/0/5)	**32**(1/2/5)
TRAPV (No Trap)	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

8.2.18 Save and Restore Operations

The save and restore operations table indicates the number of clock periods needed for the processor to perform the specified state save or return from exception, with complete execution times and stack length given. No additional tables are needed to calculate total effective execution time for these operations. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Operation	Best Case	Cache Case	Worst Case
Bus Cycle Fault (Short)	**42**(1/0/10)	**43**(1/0/10)	**50**(1/2/10)
Bus Cycle Fault (Long)	**79**(1/0/24)	**79**(1/0/24)	**86**(1/2/24)
RTE (Normal)	**20**(4/0/0)	**21**(4/0/0)	**24**(4/2/0)
RTE (Six Word)	**20**(4/0/0)	**21**(4/0/0)	**24**(4/2/0)
RTE (Throwaway)*	**15**(4/0/0)	**16**(4/0/0)	**39**(4/0/0)
RTE (Coprocessor)	**31**(7/0/0)	**32**(7/0/0)	**33**(7/1/0)
RTE (Short Fault)	**42**(10/0/0)	**43**(10/0/0)	**45**(10/2/0)
RTE (Long Fault)	**91**(24/0/0)	**92**(24/0/0)	**94**(24/2/0)

*Add the time for RTE on second stack frame.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 9
APPLICATIONS INFORMATION

This section, which provides guidelines for using the MC68020/EC020, contains information on floating-point units, byte select logic, power and ground considerations, clock driver, memory interface, access time calculations, module support, and access levels.

9.1 FLOATING-POINT UNITS

Floating-point support for the MC68020/EC020 is provided by the MC68881 floating-point coprocessor or the MC68882 enhanced floating-point coprocessor. Both devices offer a full implementation of the *IEEE Standard for Binary Floating-Point Arithmetic* (754). The MC68882 is a pin- and software-compatible upgrade of the MC68881, with an optimized MPU interface that provides over 1.5 times the performance of the MC68881 at the same clock frequency.

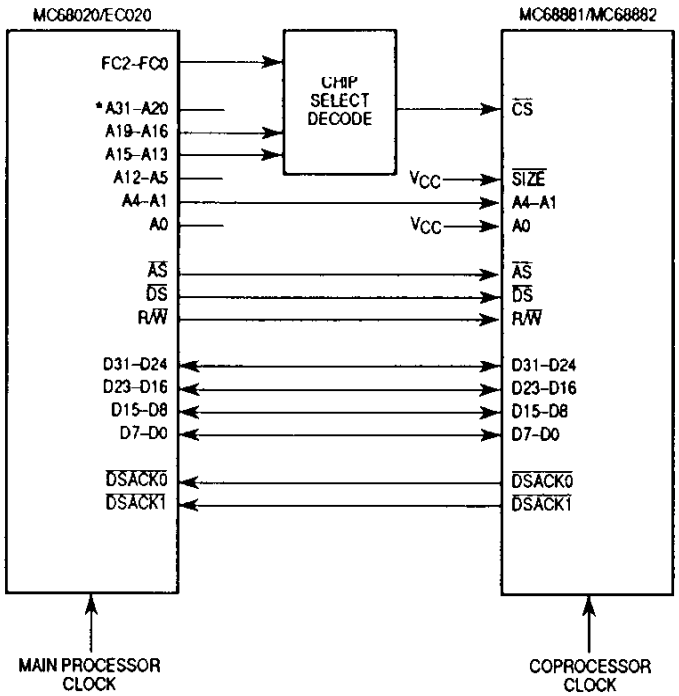
Both coprocessors provide a logical extension to the integer data processing capabilities of the main processor. They contain a high-performance floating-point arithmetic unit and a set of floating-point data registers that are utilized in a manner that is analogous to the use of the integer data registers of the processor. The MC68881/MC68882 instruction set, a natural extension of all earlier members of the M68000 family, supports all addressing modes and data types of the host MC68020/EC020. The programmer perceives the MC68020/EC020 coprocessor execution model as if both devices are implemented on one chip. In addition to supporting the full IEEE standard, the MC68881 and MC68882 provide a full set of trigonometric and transcendental functions, on-chip constants, and a full 80-bit extended-precision real data format.

The interface of the MC68020/EC020 to the MC68881 or MC68882 is easily tailored to system cost/performance needs. The MC68020/EC020 and the MC68881/MC68882 communicate via standard asynchronous M68000 bus cycles. All data transfers are performed by the main processor at the request of the MC68881/MC68882; thus, memory management, bus errors, address errors, and bus arbitration function as if the MC68881/MC68882 instructions are executed by the main processor. The floating-point unit and the processor can operate at different clock speeds, and up to seven floating-point coprocessors can simultaneously reside in an MC68020/EC020 system.

Figure 9-1 illustrates the coprocessor interface connection of an MC68881/MC68882 to an MC68020/EC020 (uses entire 32-bit data bus). The MC68881/MC68882 is configured to operate with a 32-bit data bus when both its A0 and $\overline{SIZE}$ pins are connected to V_{CC}. Refer to the MC68881UM/AD, *MC68881/MC68882 Floating-Point Coprocessor User's Manual*, for configuring the MC68881/MC68882 for smaller data bus widths.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



* For the MC68EC020, A23–A20.

Figure 9-1. 32-Bit Data Bus Coprocessor Connection

The chip select ($\overline{CS}$) decode circuitry is asynchronous logic that detects when a particular floating-point coprocessor is addressed. The MC68020/EC020 signals used by the logic include FC2–FC0 and A19–A13. Refer to **Section 7 Coprocessor Interface Description** for more information concerning the encoding of these signals. All or just a subset of these lines may be decoded, depending on the number of coprocessors in the system and the degree of redundant mapping allowed in the system.

For example, if a system has only one coprocessor, the full decoding of the ten signals (FC2–FC0 and A19–A13), provided by the PAL equations in Figure 9-3, is not absolutely necessary. It may be sufficient to use only FC1–FC0 and A17–A16. FC1–FC0 indicate when a bus cycle is operating in either CPU space ($7) or user-defined space ($3), and A17–A16 encode the CPU space type as coprocessor space ($2). A15–A13 can be ignored in this case because they encode the coprocessor identification code (CpID) used to differentiate between multiple coprocessors in a system. Motorola assemblers always default to a CpID of $1 for floating-point instructions; this can be controlled with assembler directives if a different CpID is desired or if multiple coprocessors exist in the system.

For More Information On This Product,
Go to: www.freescale.com

The major concern of a system designer is to design a $\overline{CS}$ interface that meets the AC electrical specifications for both the MC68020/EC020 (MPU) and the MC68881/MC68882 (FPCP) without adding unnecessary wait states to FPCP accesses. The following maximum specifications (relative to CLK low) meet these objectives:

$$t_{CLK} \text{ low to } \overline{AS} \text{ low} \leq (\text{MPU Spec 1} - \text{MPU Spec 47A} - \text{FPCP Spec 19}) \quad (9\text{-}1)$$

$$t_{CLK} \text{ low to } \overline{CS} \text{ low} \leq (\text{MPU Spec 1} - \text{MPU Spec 47A} - \text{FPCP Spec 19}) \quad (9\text{-}2)$$

Even though requirement (9-1) is not met under worst-case conditions, if the MPU $\overline{AS}$ is loaded within specifications and the $\overline{AS}$ input to the FPCP is unbuffered, the requirement is met under typical conditions. Designing the $\overline{CS}$ generation circuit to meet requirement (9-2) provides the highest probability that accesses to the FPCP occur without unnecessary wait states. A PAL 16L8 (see Figure 9-2) with a maximum propagation delay of 10 ns, programmed according to the equations in Figure 9-3, can be used to generate $\overline{CS}$. For a 25-MHz system, t_{CLK} low to $\overline{CS}$ low is less than or equal to 10 ns when this design is used. Should worst-case conditions cause t_{CLK} low to $\overline{AS}$ low to exceed requirement (1), one wait state is inserted in the access to the FPCP; no other adverse effects occur. Figure 9-4 shows the bus cycle timing for this interface. Refer to MC68881UM/AD, *MC68881/MC68882 Floating-Point Coprocessor User's Manual*, for FPCP specifications.

The circuit that generates $\overline{CS}$ must meet another requirement. When a nonfloating-point access immediately follows a floating-point access, $\overline{CS}$ (for the floating-point access) must be negated before $\overline{AS}$ and $\overline{DS}$ (for the subsequent access) are asserted. The PAL circuit previously described also meets this requirement.

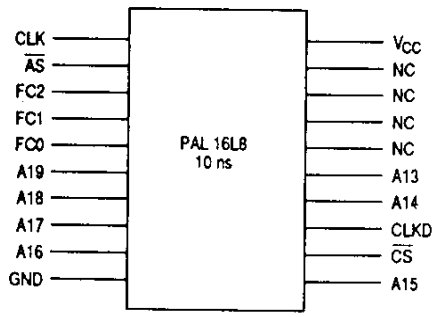


Figure 9-2. Chip Select Generation PAL

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
PAL16L8
FPCP CS GENERATION CIRCUITRY FOR 25 MHz OPERATION
MOTOROLA INC., AUSTIN, TEXAS
INPUTS:     CLK   ~AS   FC2   FC1   FC0   A19   A18   A17   A16   A15   A14   A13
OUTPUTS:    ~CS   CLKD
!~CS        = FC2        *FC1        *FC0                    ;cpu space = $7
            *!A19        *!A18       *A17        *!A16       ;coprocessor access = $2
            *!A15        *!A14       *A13                    ;coprocessor id = $1
            *!CLK                                            ;qualified by MPU clock low
            +FC2         *FC1        *FC0                    ;cpu space = $7
            *!A19        *!A18       *A17        *!A16       ;coprocessor access = $2
            *!A15        *!A14       *A13                    ;coprocessor id = $1
            *!~AS                                            ;qualified by address strobe low
            +FC2         *FC1        *FC0                    ;cpu space = $7
            *!A19        *!A18       *A17        *!A16       ;coprocessor access = $2
            *!A15        *!A14       *A13                    ;coprocessor id = $1
            *CLKD                                            ;qualified by CLKD (delayed CLK)
CLKD = CLK
```

Description: There are three terms to the $\overline{CS}$ generation. The first term denotes the earliest time $\overline{CS}$ can be asserted. The second term is used to assert $\overline{CS}$ until the end of the FPCP access. The third term is to ensure that no race condition occurs in case of a late $\overline{AS}$.

Figure 9-3. Chip Select PAL Equations

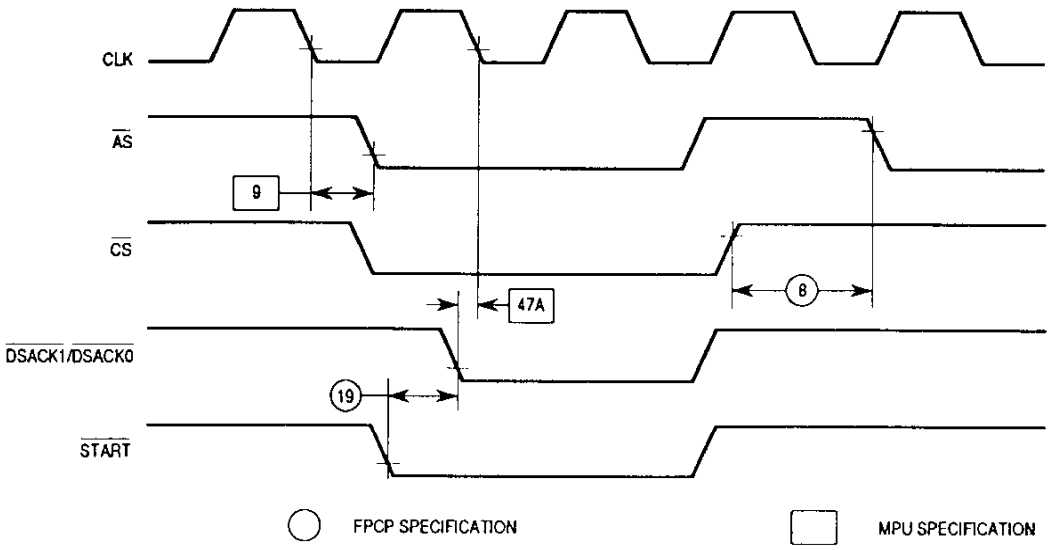


Figure 9-4. Bus Cycle Timing Diagram

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

9.2 BYTE SELECT LOGIC FOR THE MC68020/EC020

The MC68020/EC020 architecture supports byte, word, and long-word operand transfers to any 8-, 16-, or 32-bit data port, regardless of alignment. This feature allows the programmer to write code that is not bus-width specific. When accessed, the peripheral or memory subsystem reports its actual port size to the controller, and the MC68020/EC020 then dynamically sizes the data transfer accordingly, using multiple bus cycles when necessary. The following paragraphs describe the generation of byte select control signals that enable the dynamic bus sizing mechanism, the transfer of differently sized operands, and the transfer of misaligned operands to operate correctly.

The following signals control the MC68020/EC020 operand transfer mechanism:

A1, A0 — Address signals. The most significant byte of the operand to be transferred is addressed directly.

SIZ1, SIZ0 — Transfer size signals. Output of the MC68020/EC020. These indicate the number of bytes of an operand remaining to be transferred during a given bus cycle.

R/$\overline{W}$ — Read/Write signal. Output of the MC68020/EC020. For byte select generation in MC68020/EC020 systems.

$\overline{DSACK1}$, $\overline{DSACK0}$ — Data transfer and size acknowledge signals. Driven by an asynchronous port to indicate the actual bus width of the port.

The MC68020/EC020 assumes that 16-bit ports are situated on data lines D31–D16, and that 8-bit ports are situated on data lines D31–D24. This ensures that the following logic works correctly with the MC68020/EC020's on-chip internal-to-external data bus multiplexer. Refer to **Section 5 Bus Operation** for more details on the dynamic bus sizing mechanism.

The need for byte select signals is best illustrated by an example. Consider a long-word write cycle to an odd address in word-organized memory. The transfer requires three bus cycles to complete. The first bus cycle transfers the most significant byte of the long word on D23–D16. The second bus cycle transfers a word on D31–D16, and the last bus cycle transfers the least significant byte of the original long word on D31–D24. To prevent overwriting those bytes that are not used in these transfers, a unique byte data strobe must be generated for each byte when using devices with 16- and 32-bit port widths.

For noncachable read cycles and all write cycles, the required active bytes of the data bus for any given bus transfer are a function of the SIZ1, SIZ0 and A1, A0 outputs (see Table 9-1). Individual strobes or select signals can be generated by decoding these four signals for every bus cycle. Devices residing on 8-bit ports can utilize $\overline{DS}$ or $\overline{AS}$ since there is only one valid byte for any transfer.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 9-1. Data Bus Activity for Byte, Word, and Long-Word Ports

Transfer Size	SIZ1	SIZ0	A1	A0	Data Bus Active Sections Byte (B), Word (W), Long-Word (L) Ports			
					D31–D24	D23–D16	D15–D8	D7–D0
Byte	0	1	0	0	B W L	—	—	—
	0	1	0	1	B	W L		—
	0	1	1	0	B W	—	L	—
	0	1	1	1	B	W	—	L
Word	1	0	0	0	B W L	W L	—	—
	1	0	0	1	B	W L	L	—
	1	0	1	0	B W	W	L	L
	1	0	1	1	B	W	—	L
3 Bytes	1	1	0	0	B W L	W L	L	—
	1	1	0	1	B	W L	L	L
	1	1	1	0	B W	W	L	L
	1	1	1	1	B	W	—	L
Long Word	0	0	0	0	B W L	W L	L	L
	0	0	0	1	B	W L	L	L
	0	0	1	0	B W	W	L	L
	0	0	1	1	B	W	—	L

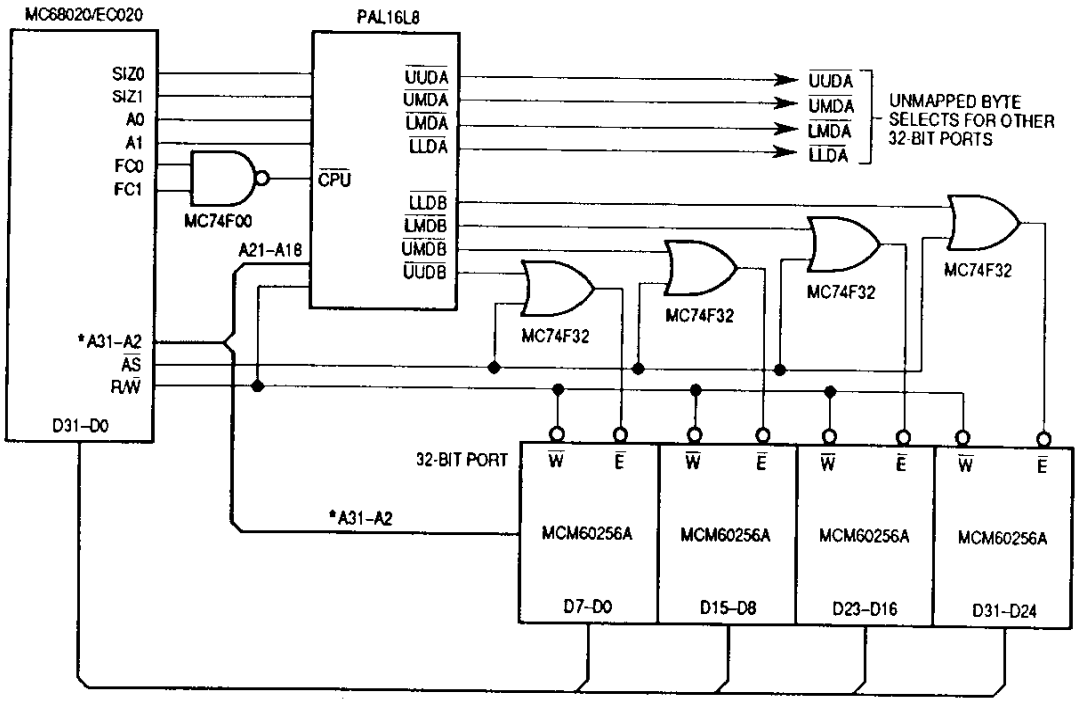
During cachable read cycles, the addressed device must provide valid data over its full bus width as indicated by $\overline{DSACK1}/\overline{DSACK0}$. While instructions are always prefetched as long-word-aligned accesses, data fetches can occur with any alignment and size. Because the MC68020/EC020 assumes that the entire data bus port size contains valid data, cachable data read bus cycles must provide as much data as signaled by the port size during a bus cycle. To satisfy this requirement, the R/$\overline{W}$ signal must be included in the byte select logic for the MC68020/EC020.

Figure 9-5 shows a block diagram of an MC68020/EC020 system with a single memory bank. The PAL provides memory-mapped byte select signals for an asynchronous 32-bit port and unmapped byte select signals for other memory banks or ports. Figure 9-6 provides sample equations for the PAL.

The PAL equations and circuits presented here cannot be the optimal implementation for every system. Depending on the CPU clock frequency, memory access times, and system architecture, different circuits may be required.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A2.

9

Figure 9-5. Example MC68020/EC020 Byte Select PAL System Configuration

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
PAL16L8
BYTE_SELECT
MC68020/EC020 BYTE DATA SELECT GENERATION FOR 32-BIT PORTS, MAPPED AND UNMAPPED
MOTOROLA INC., AUSTIN, TEXAS
INPUTS:     A0    A1    SIZ0    SIZ1    RW    A18    A19    A20    A21    ~CPU
OUTPUTS:    ~UUDA    ~UMDA    ~LMDA    ~LLDA    ~UUDA    ~UMDB    ~LMDB    ~LLDB

!~UUDA      = RW                                    ;enable upper byte on read of 32-bit port
            +!A0 *!A1                               ;directly addressed, any size
!~UMDA      = RW                                    ;enable upper middle byte on read of 32-bit port
            +A0 *!A1                                ;directly addressed, any size
            +!A1 *!SIZ0                             ;even word aligned, size word or long word
            +!A1 *SIZ1                              ;even word aligned, size is word or three byte
!~LMDA      = RW                                    ;enable lower middle byte on read of 32-bit port    +!A0 *A1
            ;directly addressed, any size
            +!A1 *!SIZ0 *!SIZ1                      ;even word aligned, size is long word
            +!A1 *SIZ0 *SIZ1                        ;even word aligned, size is three byte
            +!A1 *A0 *!SIZ0                         ;even word aligned, size is word or long word
!~LLDA      = RW                                    ;enable lower byte on read of 32-bit port
            +A0 *A1                                 ;directly addressed, any size
            +A0 *SIZ0 *SIZ1                         ;odd byte alignment, three byte size
            +!SIZ0 *!SIZ1                           ;size is long word, any address
            +A1 *SIZ1                               ;odd word aligned, word or three byte size

!~UUDB      = RW *!~CPU * (addressb)                ;enable upper byte on read of 32-bit port
            +!A0 *!A1 *!~CPU * (addressb)           ;directly addressed, any size
!~UMDB      = RW *!~CPU * (addressb)                ;enable upper middle byte on read of 32-bit port
            + A0 *!A1 *!~CPU * (addressb)           ;directly addressed, any size
            +!A1 *!SIZ0 *!~CPU * (addressb)         ;even word aligned, size word or long word
            +!A1 *SIZ1 *!~CPU * (addressb)          ;even word aligned, size is word or three byte
!~LMDB      =RW *!~CPU * (addressb)                 ;enable lower middle byte on read of 32-bit port
            +!A0 * A1 *!~CPU * (addressb)           ;directly addressed, any size
            +!A1 *!SIZ0 *!SIZ1 *!~CPU * (addressb)  ;even word aligned, size is long word
            +!A1 * SIZ0 * SIZ1 *!~CPU * (addressb)  ;even word aligned, size is three byte
            +!A1 * A0 *!SIZ0 *!~CPU * (addressb)    ;even word aligned, size is word or long word
!~LLDB      =RW *!~CPU * (addressb)                 ;enable lower byte on read of 32-bit port
            +A0 * A1 *!~CPU * (addressb)            ;directly addressed, any size
            + A0 * SIZ0 * SIZ1 *!~CPU * (addressb)  ;odd byte alignment, three byte size
            +!SIZ0 *!SIZ1 *!~CPU * (addressb)       ;size is long word, any address
            +A1 * SIZ1 *!~CPU * (addressb)          ;odd word aligned, word or three byte size
```

DESCRIPTION: Byte select signals for writing. On reads, all byte selects are asserted if the respective memory block is addressed. The input signal $\overline{CPU}$ prevents byte select assertion during CPU space cycles and is derived from NANDing FC1–FC0 or FC2–FC0. The label (addressb) is a designer-selectable combination of address lines used to generate the proper address decode for the system's memory bank. With the address lines given here, the decode block size is 256 Kbytes to 2 Mbytes. A similar address might be included in the equations for $\overline{UUDA}$, $\overline{UMDA}$, etc. if the designer wishes them to be memory mapped also.

Figure 9-6. MC68020/EC020 Byte Select PAL Equations

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

9.3 POWER AND GROUND CONSIDERATIONS

The MC68020/EC020 is fabricated in Motorola's advanced HCMOS process and is capable of operating at clock frequencies of up to 25 MHz. While the use of CMOS for a device containing such a large number of transistors allows significantly reduced power consumption compared to an equivalent NMOS circuit, the high clock speed makes the characteristics of power supplied to the device very important. The power supply must be able to furnish large amounts of instantaneous current when the MC68020/EC020 performs certain operations, and it must remain within the rated specification at all times. To meet these requirements, more detailed attention must be given to the power supply connection to the MC68020/EC020 than is required for NMOS devices operating at slower clock rates.

To reduce the amount of noise in the power supply connected to the MC68020/EC020 and to provide for the instantaneous current requirements, common capacitive decoupling techniques should be observed. While there is no recommended layout for this capacitive decoupling, it is essential that the inductance and distance between these devices and the MC68020/EC020 be minimized to provide sufficiently fast response time to satisfy momentary current demands and to maintain a constant supply voltage. It is suggested that high-frequency, high-quality capacitors be placed as close to the MC68020/EC020 as possible. Table 9-2 lists the V_{CC} and GND pin assignments for the MC68EC020 PPGA (RP suffix) package. Table 9-3 lists the V_{CC} and GND pin assignments for the MC68EC020 PQFP (FG suffix) package. Refer to **Section 11 Ordering Information and Mechanical Data** for the V_{CC} and GND pin assignments for the MC68020 packages. When assigning capacitors to the V_{CC} and GND pins, the noisier pins (address and data buses) should be heavily decoupled from the internal logic pins. Typical decoupling practices include a high-frequency, high-quality capacitor to decouple every device on the printed circuit board; however, due to the power requirements and drive capability of the MC68020/EC020, each V_{CC} pin should be decoupled with an individual capacitor. Motorola recommends using a capacitor in the range of 0.01 μF to 0.1 μF on each V_{CC} pin on each device to provide filtering for most frequencies prevalent in a digital system. In addition to the individual decoupling, several bulk decoupling capacitors should be placed onto the printed circuit board with typical values in the range of 33 μF to 330 μF. When power and ground planes are used with an adequate number of high-frequency, high-quality capacitors, the system noise will be reduced to the required levels, and the MC68020/EC020 will function properly. Similar decoupling techniques should also be observed for other VLSI devices in the system.

In addition to the capacitive decoupling of the power supply, care must be taken to ensure a low-impedance connection between all MC68020/EC020 V_{CC} and GND pins and the system power supply. A solid power supply connection from the power and ground planes to the MC68020/EC020 V_{CC} and GND pins, respectively, will meet this requirement. Failure to provide connections of sufficient quality between the MC68020/EC020 power pins and the system power supplies will result in increased assertion delays for external signals, decreased voltage noise margins, increased system noise, and possible errors in MC68020/EC020 internal logic.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 9-2. V_{CC} and GND Pin Assignments—
MC68EC020 PPGA (RP Suffix)

Pin Group	V_{CC}	GND
Address Bus	B7, C7	A1, A7, C8, D13
Data Bus	K12, M9, N9	J13, L8, M1, M8
Internal Logic	D1, D2, E12, E13	F11, F12, J1, J2
Clock	—	B1

Table 9-3. V_{CC} and GND Pin Assignments—
MC68EC020 PQFP (FG Suffix)

Pin Group	V_{CC}	GND
Address Bus	90	72, 89, 100
Data Bus	44, 57	26, 43, 58, 59
Internal Logic	7, 8, 70, 71	3, 20, 21, 68, 69
Clock	—	4

9.4 CLOCK DRIVER

The MC68020/EC020 is designed to sustain high performance while using low-cost memory subsystems. The MC68020/EC020 requires a stable clock source that is free of ringing and ground bounce, has sufficient rise and fall times, and meets the minimum and maximum high and low cycle times. The individual system may require additional clocks for peripherals with a minimum amount of clock skew. Two possible clock solutions are provided with the MC88916 and MC74F803. Many other clock solutions can be used. Some crystal clock drivers are capable of driving the MC68020/EC020 directly. For slower speed designs, a simple 74F74 flip-flop meets the clocking needs of the MC68020/EC020. Coupled with the MC88916 or MC74F803 clock generation and distribution circuit, the MC68020/EC020 provides simple interface to lower speed memory subsystems. The MC88916 (see Figure 9-7) and MC74F803 (see Figure 9-8) generate the clock signals required to minimize the skew between different clocks to multiple devices such as coprocessors, synchronous state machines, DRAM controllers, and memory subsystems. The MC88916 clock driver can be used in doubling and synchronizing a low-frequency clock source. The MC74F803 will provide a controlled skew output for clocking other peripherals.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

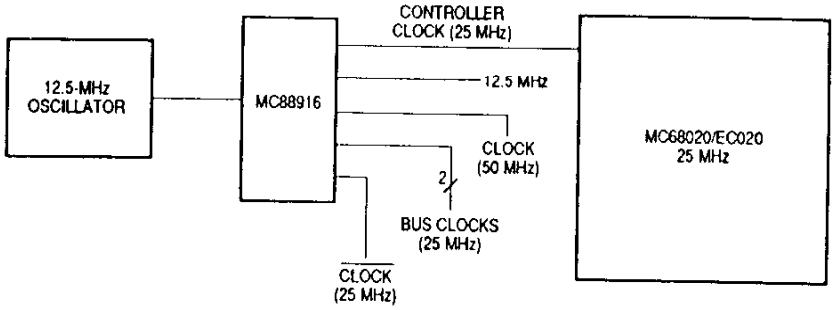


Figure 9-7. High-Resolution Clock Controller

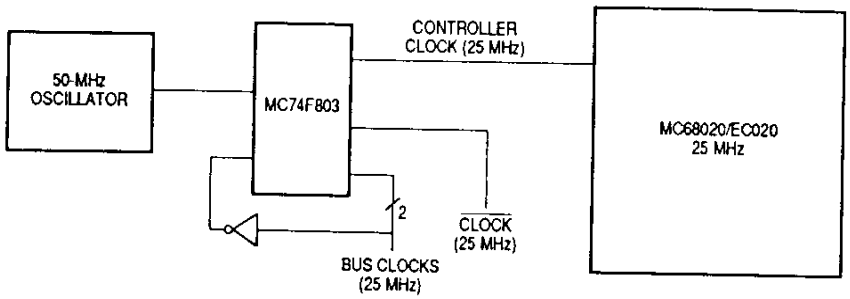


Figure 9-8. Alternate Clock Solution

9.5 MEMORY INTERFACE

The MC68020/EC020 is capable of running an external bus cycle in a minimum of three clocks (refer to **Section 5 Bus Operation**). The MC68020/EC020 runs an asynchronous bus cycle, terminated by the $\overline{\text{DSACK1}}/\overline{\text{DSACK0}}$ signals, and has a minimum duration of three controller clock periods in which up to four bytes (32 bits) are transferred.

During read operations, the MC68020/EC020 latches data on the last falling clock edge of the bus cycle, one-half clock before the bus cycle ends. Latching data here, instead of the next rising clock edge, helps to avoid data bus contention with the next bus cycle and allows the MC68020/EC020 to receive the data into its execution unit sooner for a net performance increase.

Write operations also use this data bus timing to allow data hold times from the negating strobes and to avoid any bus contention with the following bus cycle. This MC68020/EC020 characteristic allows the system to be designed with a minimum of bus buffers and latches.

One benefit of the MC68020/EC020 on-chip instruction cache is that the effect of external wait states on performance is lessened because the caches are always accessed in fewer than "no wait states," regardless of the external memory configuration.

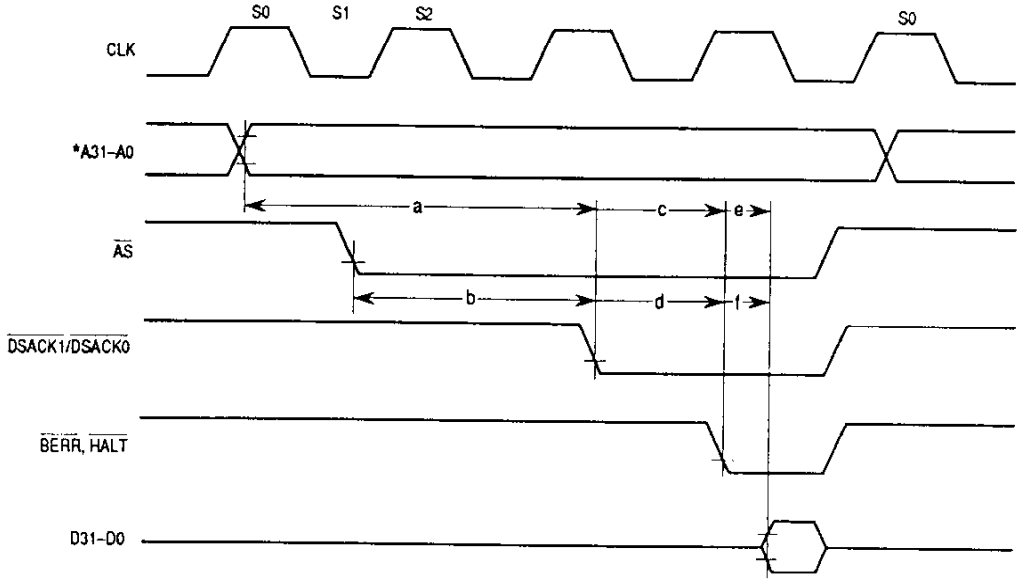
For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

9.6 ACCESS TIME CALCULATIONS

The timing paths that are critical in any memory interface are illustrated and defined in Figure 9-9.

The type of device that is interfaced to the MC68020/EC020 determines exactly which of the paths is most critical. The address-to-data paths are typically the critical paths for static devices since there is no penalty for initiating a cycle to these devices and later validating that access with the appropriate bus control signal. Conversely, the address-strobe-to-data-valid path is often most critical for dynamic devices since the cycle must be validated before an access can be initiated. For devices that signal termination of a bus cycle before data is validated (e.g., error detection and correction hardware and some external caches), to improve performance, the critical path may be from the address or strobes to the assertion of $\overline{BERR}$ (or $\overline{BERR}$ and $\overline{HALT}$). Finally, the address-valid-to-$\overline{DSACK1}$/$\overline{DSACK0}$-asserted path is most critical for very fast devices and external caches, since the time available between the address becoming valid and the $\overline{DSACK1}$/$\overline{DSACK0}$ assertion to terminate the bus cycle is minimal. Table 9-4 provides the equations required to calculate the various memory access times assuming a 50-percent duty cycle clock.



*For the MC68EC020, A23–A0.

NOTE: This diagram illustrates access time calculations only.

Parameter	Description	System	Equation
a	Address Valid to $\overline{DSACK1}$/$\overline{DSACK0}$ Asserted	t_{AVDL}	9-3
b	$\overline{AS}$ Asserted to $\overline{DSACK1}$/$\overline{DSACK0}$ Asserted	t_{SADL}	9-4
c	Address Valid to $\overline{BERR}$/$\overline{HALT}$ Asserted	t_{AVBHL}	9-5
d	$\overline{AS}$ Asserted to $\overline{BERR}$/$\overline{HALT}$ Asserted	t_{SABHL}	9-6
e	Address Valid to Data Valid	t_{AVDV}	9-7
f	$\overline{AS}$ Asserted to Data Valid	t_{SADV}	9-8

Figure 9-9. Access Time Computation Diagram

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table 9-4. Memory Access Time Equations at 16.67 and 25 MHz

Equation	16.667 MHz	N = 3	N = 4	N = 5	N = 6	N = 7	Unit
9-3	$t_{AVDL} = (N - 1) \cdot t1 - t2 - t6 - t47A$	61	121	181	241	301	ns
9-4	$t_{SADL} = (N - 1) \cdot t1 - t9 - t60$	25	85	145	205	265	ns
9-5	$t_{AVBHL} = N \cdot t1 - t2 - t6 - t27A$	22	46	70	94	118	ns
9-6	$t_{SABHL} = (N - 1) \cdot t1 - t9 - t27A$	40	70	100	130	160	ns
9-7	$t_{AVDV} = N \cdot t1 - t2 - t6 - t27$	121	181	241	301	361	ns
9-8	$t_{SADV} = (N - 1) \cdot t1 - t9 - t27$	85	145	205	265	325	ns

Equation	25 MHz	N = 3	N = 4	N = 5	N = 6	N = 7	Unit
9-3	$t_{AVDL} = (N - 1) \cdot t1 - t2 - t6 - t47A$	31	71	111	151	191	ns
9-4	$t_{SADL} = (N - 1) \cdot t1 - t9 - t60$	17	57	97	137	177	ns
9-5	$t_{AVBHL} = N \cdot t1 - t2 - t6 - t27A$	22	41	60	79	98	ns
9-6	$t_{SABHL} = (N - 1) \cdot t1 - t9 - t27A$	26	44	62	80	98	ns
9-7	$t_{AVDV} = N \cdot t1 - t2 - t6 - t27$	71	111	151	191	231	ns
9-8	$t_{SADV} = (N - 1) \cdot t1 - t9 - t27$	57	97	137	177	217	ns

Where:

- tX = Refers to AC Electrical Specification X
- t1 = The Clock Period
- t2 = The Clock Low Time
- t3 = The Clock High Time
- t6 = The Clock High to Address Valid Time
- t9 = The Clock Low to $\overline{AS}$ Low Delay
- t27 = The Data-In to Clock Low Setup Time
- t27A = The $\overline{BERR}/\overline{HALT}$ to Clock Low Setup Time
- t47A = The Asynchronous Input Setup Time
- N = The Total Number of Clock Periods in the Bus Cycle (N ≥ 3 Cycles)

During asynchronous bus cycles, $\overline{DSACK1}/\overline{DSACK0}$ are used to terminate the current bus cycle. In true asynchronous operations, such as accesses to peripherals operating at a different clock frequency, either or both signals may be asserted without regard to the clock, and then data must be valid a certain amount of time later as defined by specification 31. With a 25-MHz controller, this time is 32 ns after $\overline{DSACK1}/\overline{DSACK0}$ asserts; with a 16.67-MHz controller, this time is 50 ns after $\overline{DSACK1}/\overline{DSACK0}$ asserts (both numbers vary with the actual clock frequency).

However, many local memory systems do not operate in a truly asynchronous manner because either the memory control logic can be related to the MC68020/EC020 clock or worst-case propagation delays are known; thus, asynchronous setup times for the $\overline{DSACK1}/\overline{DSACK0}$ signals can be guaranteed. The timing requirements for this pseudo-synchronous $\overline{DSACK1}/\overline{DSACK0}$ generation is governed by the equation for t_{AVDL}.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Another way to optimize the CPU-to-memory access times in a system is to use a clock frequency less than the rated maximum of the specific MC68020/EC020 device. Table 9-5 provides calculated t_{AVDV} (see Equation 9-7 of Table 9-4) results for a 16 MHz MC68020/EC020 and a 25 MHz MC68020/EC020 operating at various clock frequencies. If the system uses other clock frequencies, the above equations can be used to calculate the exact access times.

Table 9-5. Calculated t_{AVDV} Values for Operation at Frequencies Less Than or Equal to the CPU Maximum Frequency Rating

Equation 9-7 t_{AVDV}		16-MHz MC68020/EC020		25-MHz MC68020/EC020		
Clocks Per (N) and Type Bus Cycle	Wait States	Clock at 12.5 MHz	Clock at 16.67 MHz	Clock at 16.67 MHz	Clock at 20 MHz	Clock at 25 MHz
3 Clock Asynchronous	0	181	121	131	101	71
4 Clock Asynchronous	1	261	181	191	151	111
5 Clock Asynchronous	2	341	241	251	201	151
6 Clock Asynchronous	3	421	301	311	251	191

9.7 MODULE SUPPORT

The MC68020/EC020 includes support for modules with the CALLM and RTM instructions. The CALLM instruction references a module descriptor. This descriptor contains control information for entry into the called module. The CALLM instruction creates a module stack frame and stores the current module state in that frame and loads a new module state from the referenced descriptor. The RTM instruction recovers the previous module state from the stack frame and returns to the calling module.

The module interface facilitates finer resolution of access control by external hardware. Although the MC68020/EC020 does not interpret the access control information, it communicates with external hardware when the access control is to be changed and relies on the external hardware to verify that the changes are legal.

9.7.1 Module Descriptor

Figure 9-10 illustrates the format of the module descriptor. The first long word contains control information used during execution of the CALLM instruction. The remaining locations contain data that can be loaded into processor registers by the CALLM instruction.

For More Information On This Product,
Go to: www.freescale.com

	31 29	28 24	23 16	15 0
BASE →	OPT	TYPE	ACCESS LEVEL	(RESERVED, MUST BE ZERO)
+$04	MODULE ENTRY WORD POINTER			
+$08	MODULE DATA AREA POINTER			
+$0C				
+$10	ADDITIONAL USER-DEFINED INFORMATION			

Figure 9-10. Module Descriptor Format

The opt field specifies how arguments are to be passed to the called module; the MC68020/EC020 recognizes only the options of 000 and 100; all others cause a format exception. The 000 option indicates that the called module expects to find arguments from the calling module on the stack just below the module stack frame. In cases where there is a change of stack pointer during the call, the MC68020/EC020 will copy the arguments from the old stack to the new stack. The 100 option indicates that the called module will access the arguments from the calling module through an indirect pointer in the stack of the calling module. Hence, the arguments are not copied, but the MC68020/EC020 puts the value of the stack pointer from the calling module in the module stack frame.

The type field specifies the type of the descriptor; the MC68020/EC020 only recognizes descriptors of type $00 and $01; all others cause a format exception. The $00 type descriptor defines a module for which there is no change in access rights, and the called module builds its stack frame on top of the stack used by the calling module. The $01 type descriptor defines a module for which there may be a change in access rights; such a called module may have a separate stack area from that of the calling module.

The access level field is used only with the type $01 descriptor and is passed to external hardware to change the access control.

The module entry word pointer specifies the entry address of the called module. The first word at the entry address (see Figure 9-11) specifies the register to be saved in the module stack frame and then loaded with the module descriptor data area pointer; the first instruction of the module starts with the next word. The module descriptor data area pointer field contains the address of the called module data area.

If the access change requires a change of stack pointer, the old value is saved in the module stack frame, and the new value is taken from the module descriptor stack pointer field. Any further information in the module descriptor is user defined.

15	14 12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER	0	0	0	0	0	0	0	0	0	0	0	0
OPERATION WORD OF FIRST INSTRUCTION													

Figure 9-11. Module Entry Word

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

All module descriptor types $10–$1F are reserved for user definition and cause a format error exception. This provides the user with a means of disabling any module by setting a single bit in its descriptor without loss of any descriptor information.

If the called module does not wish the module data area pointer to be loaded into a register, the module entry word can select register A7, and the loaded value will be overwritten with the correction stack pointer value after the module stack frame is created and filled.

9.7.2 Module Stack Frame

Figure 9-12 illustrates the format of the module stack frame. This frame is constructed by the CALLM instruction and is removed by the RTM instruction. The first and second long words contain control information passed by the CALLM instruction to the RTM instruction. The module descriptor pointer contains the address of the descriptor used during the module call. All other locations contain information to be restored on return to the calling module.

The PC is the saved address of the instruction following the CALLM instruction. The opt and type fields, which specify the argument options and type of module stack frame, are copied to the frame from the module descriptor by the CALLM instruction; the RTM instruction will cause a format error if the opt and type fields do not have recognizable values. The access level is the saved access control information, which is saved from external hardware by the CALLM instruction and restored by the RTM instruction. The argument count field is set by the CALLM instruction and is used by the RTM instruction to remove arguments from the stack of the calling module. The contents of the CCR are saved by the CALLM instruction and restored by the RTM instruction. The saved stack pointer field contains the value of the stack pointer when the CALLM instruction started execution, and that value is restored by RTM. The saved module data area pointer field contains the saved value of the module data area pointer register from the calling module.

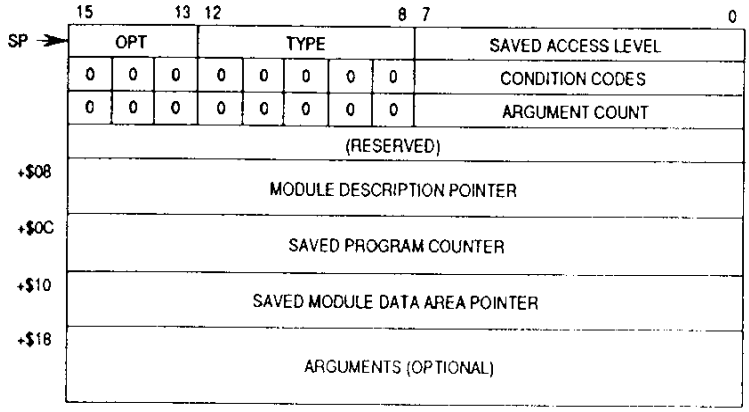


Figure 9-12. Module Call Stack Frame

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

9.8 ACCESS LEVELS

The MC68020/EC020 module mechanism supports a finer level of access control beyond the distinction between user and supervisor privilege levels. The module mechanism allows a module with limited access rights to call a module with greater access rights. With the help of external hardware, the processor can verify that an increase in access rights is allowable or can detect attempts by a module to gain access rights to which it is not entitled.

Type $01 module descriptors and module stack frames indicate a request to change access levels. While processing a type $01 descriptor or frame, the CALLM and RTM instructions communicate with external access control hardware via accesses in the CPU space. For these accesses, A19–A16 equal 0001. Figure 9-13 shows the address map for these CPU space accesses. If the processor receives a bus error on any of these CPU space accesses during the execution of a CALLM or RTM instruction, the processor will take a format error exception.

	31 – 24	23 – 0
$00	CAL	(UNUSED, RESERVED)
$04	ACCESS STATUS REGISTER	(UNUSED, RESERVED)
$08	IAL	(UNUSED, RESERVED)
$0C	DAL	(UNUSED, RESERVED)

$40	FUNCTION CODE 0 DESCRIPTOR ADDRESS
$44	FUNCTION CODE 1 DESCRIPTOR ADDRESS (USER DATA)
$48	FUNCTION CODE 2 DESCRIPTOR ADDRESS (USER PROGRAM)
$4C	FUNCTION CODE 3 DESCRIPTOR ADDRESS
$50	FUNCTION CODE 4 DESCRIPTOR ADDRESS (SUPERVISOR DATA)
$54	FUNCTION CODE 5 DESCRIPTOR ADDRESS (SUPERVISOR PROGRAM)
$58	FUNCTION CODE 6 DESCRIPTOR ADDRESS
$5C	FUNCTION CODE 7 DESCRIPTOR ADDRESS (CPU SPACE)

Figure 9-13. Access Level Control Bus Registers

The current access level register (CAL) contains the access level rights of the currently executing module. The increase access level register (IAL) is the register through which the processor requests increased access rights. The decrease access level register (DAL) is the register through which the processor requests decreased access rights. The formats of these three registers are undefined to the main processor, but the main processor assumes that information read from the module descriptor stack frame or the CAL can be meaningfully written to the IAL or the DAL. The access status register allows the processor to query the external hardware as to the legality of intended access level transitions. Table 9-6 lists the valid values of the access status register.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 9-6. Access Status Register Codes

Value	Validity	Processor Action
$00	Invalid	Format Error
$01	Valid	No Change in Access Rights
$02–$03	Valid	Change Access Rights with No Change of Stack Pointer
$04–$07	Valid	Change Access Rights and Change Stack Pointer
Other	Undefined	Undefined (Take Format Error Exception)

The processor uses the descriptor address registers during the CALLM instruction to communicate the address of the type $01 descriptor, allowing external hardware to verify that the address is a valid address for a type $01 descriptor. This validation prevents a module from creating a type $01 descriptor to increase its access rights.

9.8.1 Module Call

The CALLM instruction is used to make the module call. For the type $00 module descriptor, the processor creates and fills the module stack frame at the top of the active system stack. The condition codes of the calling module are saved in the CCR field of the frame. If opt is equal to 000 (arguments passed on the stack) in the module descriptor, the MC68020/EC020 does not save the stack pointer or load a new stack pointer value. The processor uses the module entry word to save and load the module data area pointer register and then begins execution of the called module.

For the type $01 module descriptor, the processor must first obtain the current access level from external hardware. It also verifies that the calling module has the right to read from the area pointed to by the current value of the stack pointer by reading from that address. It passes the descriptor address and increase access level to external hardware for validation and then reads the access status. If external hardware determines that the change in access rights should not be granted, the access status is zero, and the processor takes a format error exception. No visible processor registers are changed, nor should the current access level enforced by external hardware be changed. If external hardware determines that a change should be granted, the external hardware changes its access level, and the processor proceeds. If the access status register indicates that a change in the stack pointer is required, the stack pointer is saved internally, a new value is loaded from the module descriptor, and arguments are copied from the calling stack to the new stack. Finally, the module stack frame is created and filled on the top of the current stack. The condition codes of the calling module are saved in the CCR field of the frame. Execution of the called module then begins as with a type $00 descriptor.

For More Information On This Product,
Go to: www.freescale.com

9.8.2 Module Return

The RTM instruction is used to return from a module. For the type $00 module stack frame, the processor reloads the condition codes, the PC, and the module data area pointer register from the frame. The frame is removed from the top of the stack, the argument count is added to the stack pointer, and execution returns to the calling module.

For the type $01 module stack frame, the processor reads the access level, condition codes, PC, saved module data area pointer, and saved stack pointer from the module stack frame. The access level is written to the DAL for validation by external hardware; the processor then reads the access status to check the validation. If the external hardware determines that the change in access right should not be granted, the access status is zero, and the processor takes a format error exception. No visible processor registers are changed, nor should the current access level enforced by external hardware be changed. If the external hardware determines that the change in access rights should be granted, the external hardware changes its access level, the values read from the module stack frame are loaded into the corresponding processor registers, the argument count is added to the new stack pointer value, and execution returns to the calling module.

If the called module does not wish the saved module data pointer to be loaded into a register, the RTM instruction word can select register A7, and the loaded value will be overwritten with the correct stack pointer value after the module stack frame is deallocated.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 10
ELECTRICAL CHARACTERISTICS

This section provides the thermal characteristics and electrical specifications for the MC68020/EC020. Note that the thermal and DC electrical characteristics are listed separately for the MC68020 and the MC68EC020. All other data applies to both the MC68020 and the MC68EC020 unless otherwise noted.

10.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	V
Input Voltage	V_{in}	–0.5 to +7.0	V
Operating Temperature Range Minimum Ambient Temperature	T_A	0	°C
Maximum Ambient Temperature PGA, PPGA, PQFP	T_A	70	°C
Maximum Junction Temperature CQFP	T_J	110	°C
Storage Temperature Range	T_{stg}	–55 to 150	°C

The device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, normal precautions should be taken to avoid application of voltages higher than maximum-rated voltages to these high-impedance circuits. Tying unused inputs to the appropriate logic voltage level (e.g., either GND or V_{CC}) enhances reliability of operation.

10.2 THERMAL CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \qquad (10\text{-}1)$$

where:

T_A = Ambient Temperature, °C
θ_{JA} = Package Thermal Resistance, Junction-to-Ambient, °C/W
P_D = $P_{INT} + P_{I/O}$
P_{INT} = $I_{CC} \times V_{CC}$, Watts—Chip Internal Power
$P_{I/O}$ = Power Dissipation on Input and Output Pins—User Determined

For most applications, $P_{I/O} < P_{INT}$ and can be neglected.

An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

$$P_D = K + (T_J + 273°C) \qquad (10\text{-}2)$$

10

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Solving Equations (10-1) and (10-2) for K gives:

$$K = P_D \cdot (T_A + 273°C) + \theta_{JA} \cdot P_D^2 \qquad (10\text{-}3)$$

where K is a constant pertaining to the particular part. K can be determined from equation (10-3) by measuring P_D (at thermal equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J can be obtained by solving equations (10-1) and (10-2) iteratively for any value of T_A.

The total thermal resistance of a package (θ_{JA}) can be separated into two components, θ_{JC} and θ_{CA}. θ_{JC} represents the barrier to heat flow from the semiconductor junction to the package (case) surface, and θ_{CA} represents the barrier to heat flow from the case to the ambient air. These terms are related by the equation:

$$\theta_{JA} = \theta_{JC} + \theta_{CA} \qquad (10\text{-}4)$$

θ_{JC} is device related and cannot be influenced by the user. However, θ_{CA} is user dependent and can be minimized by such thermal management techniques as heat sinks, forced air cooling, and use of thermal convection to increase air flow over the device. Thus, good thermal design on the part of the user can significantly reduce θ_{CA} so that θ_{JA} approximately equals θ_{JC}. Substitution of θ_{JC} for θ_{JA} in equation (10-1) results in a lower semiconductor junction temperature.

10.2.1 MC68020 Thermal Characteristics and DC Electrical Characteristics

MC68020 Thermal Resistance (°C/W)

The following table provides thermal resistance characteristics for junction to ambient and junction to case for the MC68020 packages with natural convection and no heatsink.

Characteristic—Natural Convection and No Heatsink	θ_{JA}	θ_{JC}
Thermal Resistance		
PGA Package (RC Suffix)	26	3
PPGA Package (RP Suffix)	32	10
CQFP Package (FE Suffix)	46	15
PQFP Package (FC Suffix)	42	20

Resistance is to bottom center (pin side) of case for PGA and PPGA packages, top center of case for CQFP and PQFP packages.

MC68020 CQFP Package

Table 10-1 provides typical and worst case thermal characteristics for the MC68020 CQFP package both with and without a heatsink. The heatsink used is black anodized aluminum alloy, 0.72"x0.75"x0.6" high with an omnidirectional 5x6 array of fins. Attachment was made using Epolite 6400 one part epoxy.

For More Information On This Product,
Go to: www.freescale.com

Table 10-1. θ_{JA} vs. Airflow—MC68020 CQFP Package

θ_{JA}	Airflow in Linear Feet/Minute		
	0*	200	500
Maximum			
No Heatsink	46	28	24
With Heatsink	35	20	18
Typical			
No Heatsink	43	25	21
With Heatsink	32	17	15

*Natural convection

Table 10-2 shows the relationship between clock speed and power dissipation for any package type. The worst case operating conditions are used for thermal management design, while typical values are used for reliability analysis.

Table 10-2. Power vs. Rated Frequency (at T_J Maximum = 110°C)

Rated Frequency (MHz)	P_D Maximum (Watts)	P_D Typical (Watts)
33	1.4	0.84
25	1.2	0.72
20	1.0	0.60
16	0.9	0.54

Table 10-3 shows the relationship between board temperature rise and power dissipation in the test environment for the CQFP package. Derate θ_{JA} based on measurements made in the application by adding $(0.8/P_D) * [T_{ba(application)} - T_{ba(table)}]$ to the θ_{JA} values in the table. Board temperature was measured on the top surface of the board directly under the device.

Table 10-3. Temperature Rise of Board vs. P_D —MC68020 CQFP Package

Natural Convection	P_D		
	0.6W	1.0W	1.75W
T_{ba} (°C)—No Heatsink	18	27	53

Values for thermal resistance presented in this document were derived using the procedure described in Motorola Reliability Report 7843, "Thermal Resistance Measurement Method for MC68XX Microcomponent Devices," and are provided for design purposes only. Thermal measurements are complex and dependent on procedure and setup. User-derived values for thermal resistance may differ.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

MC68020 DC Electrical Characteristics

(V_{CC} = 5.0 V_{dc} ± 5%; GND = 0 V_{dc}; Temperature within defined ranges)

Characteristics		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	GND −0.5	0.8	V
Input Leakage Current GND ≤ V_{in} ≤ V_{CC}	$\overline{BERR}$, $\overline{BR}$, $\overline{BGACK}$, CLK, $\overline{IPL2}$–$\overline{IPL0}$, $\overline{AVEC}$, $\overline{DSACK1}$, $\overline{DSACK0}$, $\overline{CDIS}$ $\overline{HALT}$, $\overline{RESET}$	I_{in}	−1.0 −20	1.0 20	μA
Hi-Z (Off-State) Leakage Current @ 2.4 V/0.5 V	A31–A0, $\overline{AS}$, $\overline{DBEN}$, $\overline{DS}$, D31–D0, FC2–FC0, R/$\overline{W}$, $\overline{RMC}$, SIZ1–SIZ0	I_{TSI}	−20	20	μA
Output High Voltage I_{OH} = 400 μA	A31–A0, $\overline{AS}$, $\overline{BG}$, D31–D0, $\overline{DBEN}$, $\overline{DS}$, R/$\overline{W}$, $\overline{ECS}$, $\overline{IPEND}$, $\overline{RMC}$, SIZ1–SIZ0, FC2–FC0	V_{OH}	2.4	—	V
Output Low Voltage I_{OL} = 3.2 mA I_{OL} = 5.3 mA I_{OL} = 2.0 mA I_{OL} = 10.7 mA	A31–A0, FC2–FC0, SIZ1–SIZ0, $\overline{BG}$, D31–0 $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, $\overline{RMC}$, $\overline{DBEN}$, $\overline{IPEND}$, $\overline{ECS}$, $\overline{OCS}$ $\overline{HALT}$, $\overline{RESET}$	V_{OL}	— — — —	0.5 0.5 0.5 0.5	V
Power Dissipation (T_A = 0°C)		P_D	—	2.0	W
Capacitance (see Note) V_{in} = 0 V, T_A = 25°C, f = 1 MHz		C_{in}	—	20	pF
Load Capacitance	$\overline{ECS}$, $\overline{OCS}$ All Other	C_L	— —	50 130	pF

NOTE: Capacitance is periodically sampled rather than 100% tested.

10.2.2 MC68EC020 Thermal Characteristics and DC Electrical Characteristics

MC68EC020 Thermal Resistance (°C/W)

The following table provides thermal resistance characteristics for junction to ambient and junction to case for the MC68EC020 packages with natural convection and no heatsink.

Characteristic – Natural Convection and No Heatsink	θ_{JA}	θ_{JC}
Thermal Resistance		
PPGA Package (RP Suffix)	32	10
PQFP Package (FG Suffix)	53	18

MC68EC020 PQFP Package

Table 10-4 provides typical and worst case thermal characteristics for the MC68EC020 PQFP package without a heatsink.

Table 10-4. θ_{JA} vs. Airflow—MC68EC020 PQFP Package

θ_{JA}	Airflow in Linear Feet/Minute					
	0*	50	100	200	300	400
Maximum—No Heatsink	53	49	45	41	38	36
Typical—No Heatsink	51	47	43	39	37	35

*Natural convection

For More Information On This Product,
Go to: www.freescale.com

MC68EC020 DC Electrical Characteristics

(V_{CC} = 5.0 V_{dc} ± 5%; GND = 0 V_{dc}; Temperature within defined ranges)

Characteristics		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	GND -0.5	0.8	V
Input Leakage Current GND ≤ V_{in} ≤ V_{CC}	$\overline{BERR}$, $\overline{BR}$, CLK, $\overline{IPL2}$–$\overline{IPL0}$, $\overline{AVEC}$, $\overline{DSACK1}$, $\overline{DSACK0}$, $\overline{CDIS}$, $\overline{HALT}$, $\overline{RESET}$	I_{in}	-1.0 -20	1.0 20	μA
Hi-Z (Off-State) Leakage Current @ 2.4 V/0.5 V	A23–A0, $\overline{AS}$, $\overline{DS}$, D31–D0, FC2–FC0, R/$\overline{W}$, $\overline{RMC}$, SIZ1–SIZ0	I_{TSI}	-20	20	μA
Output High Voltage I_{OH} = 400 μA	A23–A0, $\overline{AS}$, $\overline{BG}$, D31–D0, $\overline{DS}$, R/$\overline{W}$, $\overline{RMC}$, SIZ1–SIZ0, FC2–FC0	V_{OH}	2.4	—	V
Output Low Voltage I_{OL} = 3.2 mA I_{OL} = 5.3 mA I_{OL} = 10.7 mA	A23–A0, FC2–FC0, SIZ1–SIZ0, $\overline{BG}$, D31–D0 $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, $\overline{RMC}$, $\overline{HALT}$, $\overline{RESET}$	V_{OL}	— — —	0.5 0.5 0.5	V
Power Dissipation (T_A = 0°C)	f = 25 MHz f = 16 MHz	P_{INT}	— —	1.5 1.2	W
Capacitance (see Note) V_{in} = 0 V, T_A = 25°C, f = 1 MHz		C_{in}	—	20	pF
Load Capacitance		C_L	—	130	pF

NOTE: Capacitance is periodically sampled rather than 100% tested.

10.3 AC ELECTRICAL CHARACTERISTICS

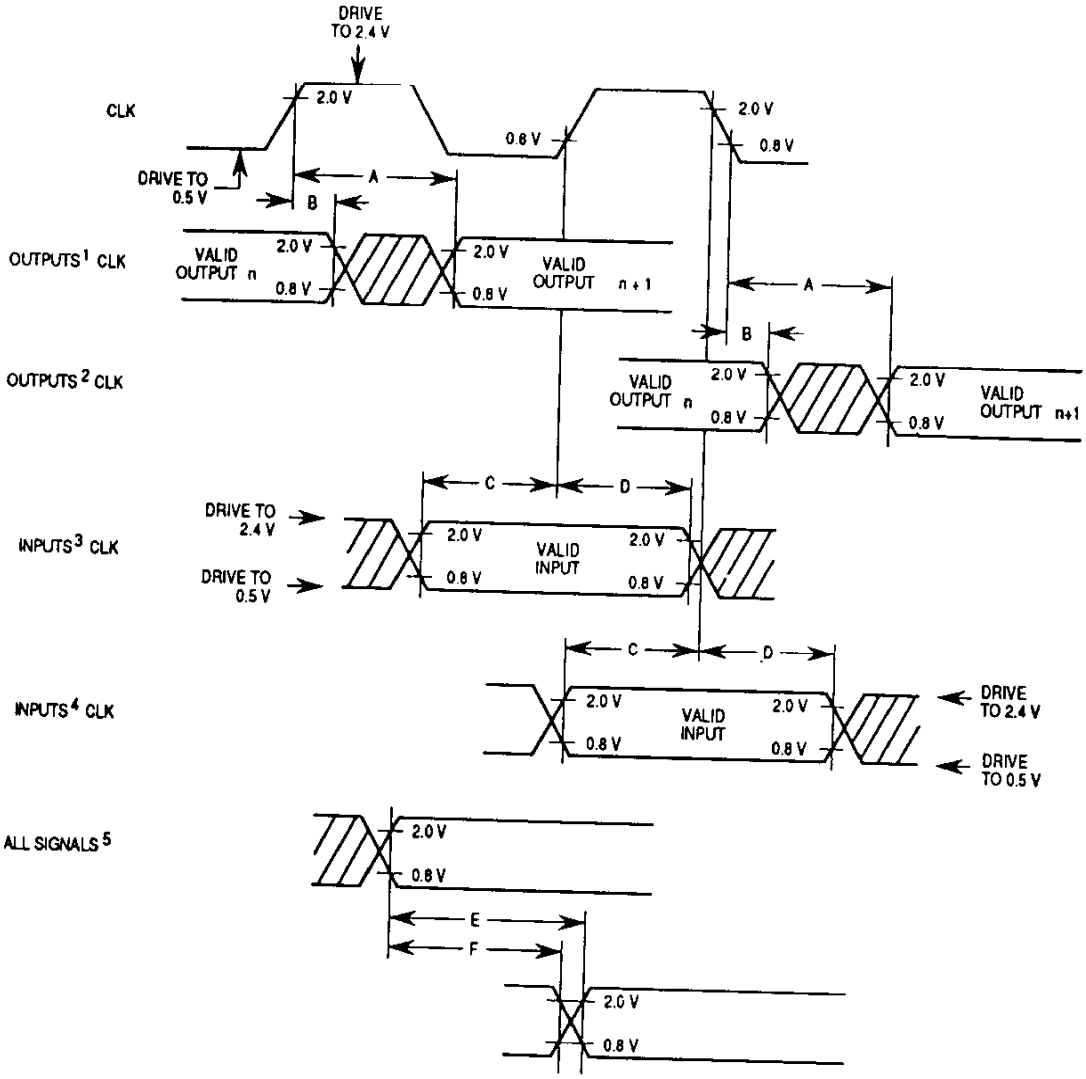
The AC specifications presented consist of output delays, input setup and hold times, and signal skew times. All signals are specified relative to an appropriate edge of the clock and possibly to one or more other signals.

The measurement of the AC specifications is defined by the waveforms shown in Figure 10-1. To test the parameters guaranteed by Motorola, inputs must be driven to the voltage levels specified in Figure 10-1. Outputs are specified with minimum and/or maximum limits, as appropriate, and are measured as shown in Figure 10-1. Inputs are specified with minimum setup and hold times, and are measured as shown. Finally, the measurement for signal-to-signal specifications is also shown.

Note that the testing levels used to verify conformance to the AC specifications do not affect the guaranteed DC operation of the device as specified in the DC electrical specifications. The 20 MHz and 33.33 MHz specifications do not apply to the MC68EC020.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



NOTES:
1. This output timing is applicable to all parameters specified relative to the rising edge of the clock.
2. This output timing is applicable to all parameters specified relative to the falling edge of the clock.
3. This input timing is applicable to all parameters specified relative to the rising edge of the clock.
4. This input timing is applicable to all parameters specified relative to the falling edge of the clock.
5. This timing is applicable to all parameters specified relative to the assertion/negation of another 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. Signal valid to signal valid specification (maximum or minimum).
F. Signal valid to signal invalid specification (maximum or minimum).

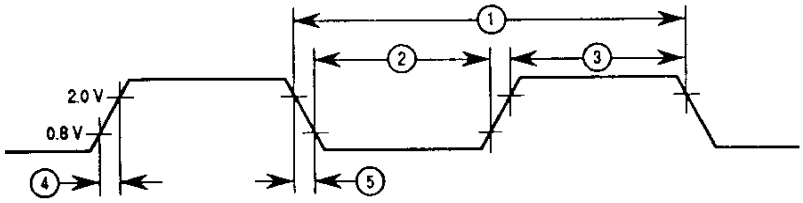
Figure 10-1. Drive Levels and Test Points for AC Specifications

Freescale Semiconductor, Inc.
10
For More Information On This Product,
Go to: www.freescale.com

AC ELECTRICAL CHARACTERISTICS—CLOCK INPUT (see Figure 10-2)

Num.	Characteristic	16.67 MHz		20 MHz		25 MHz*		33.33 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
	Frequency of Operation	8	16.67	12.5	20	12.5	25	12.5	33.33	MHz
1	Cycle Time	60	125	50	80	40	80	30	80	ns
2,3	Clock Pulse Width (Measured from 1.5 V to 1.5 V)	24	95	20	54	19	61	14	66	ns
4,5	Clock Rise and Fall Times	—	5	—	5	—	4	—	3	ns

*These specifications represent an improvement over previously published specifications for the 25-MHz MC68020 and are valid only for products bearing date codes of 8827 and later.



NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 2.0 V.

Figure 10-2. Clock Input Timing Diagram

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

AC ELECTRICAL CHARACTERISTICS—READ AND WRITE CYCLES

(V_{CC} = 5.0 V_{dc} ± 5%; GND = 0 V_{dc}; Temperature within defined ranges; see Figures 10-3–10-5)

Num.	Characteristics	16.67 MHz		20 MHz		25 MHz**		33.33 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
6	Clock High to FC, Size, $\overline{RMC}$, Address Valid	0	30	0	25	0	25	0	21	ns
*6A	Clock High to $\overline{ECS}$, $\overline{OCS}$ Asserted	0	20	0	15	0	12	0	10	ns
7	Clock High to Address, Data, FC, Size, $\overline{RMC}$ High Impedance	0	60	0	50	0	40	0	30	ns
8	Clock High to Address, FC, Size, $\overline{RMC}$ Invalid	0	—	0	—	0	—	0	—	ns
9	Clock Low to $\overline{AS}$, $\overline{DS}$ Asserted	3	30	3	25	3	18	3	15	ns
9A[1]	$\overline{AS}$ to $\overline{DS}$ Assertion Skew (Read)	15	15	–10	10	–10	10	–10	10	ns
9B[11]	$\overline{AS}$ Asserted to $\overline{DS}$ Asserted (Write)	37	—	32	—	27		22	—	ns
*10	$\overline{ECS}$ Width Asserted	20	—	15	—	15	—	10	—	ns
*10A	$\overline{OCS}$ Width Asserted	20	—	15	—	15	—	10	—	ns
*10B[7]	$\overline{ECS}$, $\overline{OCS}$ Width Negated	15	—	10	—	5	—	5	—	ns
11	Address, FC, Size, $\overline{RMC}$ Valid to $\overline{AS}$ (and $\overline{DS}$ Asserted, Read)	15	—	10	—	6	—	5	—	ns
12	Clock Low to $\overline{AS}$, $\overline{DS}$ Negated	0	30	0	25	0	15	0	15	ns
*12A	Clock Low to $\overline{ECS}$, $\overline{OCS}$ Negated	0	30	0	25	0	15	0	15	ns
13	$\overline{AS}$, $\overline{DS}$ Negated to Address, FC, Size, $\overline{RMC}$ Invalid	15	—	10	—	10	—	5	—	ns
14	$\overline{AS}$ (and $\overline{DS}$ Read) Width Asserted	100	—	85	—	70	—	50	—	ns
14A	$\overline{DS}$ Width Asserted (Write)	40	—	38	—	30	—	25	—	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	40	—	38	—	30	—	23	—	ns
15A[8]	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	35	—	30	—	25	—	18	—	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, $\overline{DBEN}$ High Impedance	—	60	—	50	—	40	—	30	ns
17	$\overline{AS}$, $\overline{DS}$ Negated to R/$\overline{W}$ Invalid	15	—	10	—	10	—	5	—	ns
18	Clock High to R/$\overline{W}$ High	0	30	0	25	0	20	0	15	ns
20	Clock High to R/$\overline{W}$ Low	0	30	0	25	0	20	0	15	ns
21	R/$\overline{W}$ High to $\overline{AS}$ Asserted (Read)	15	—	10	—	5	—	5	—	ns
22	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	75	—	60	—	50	—	35	—	ns
23	Clock High to Data Out Valid	—	30	—	25	—	25	—	18	ns
25	$\overline{AS}$, $\overline{DS}$ Negated to Data-Out Invalid	15	—	10	—	5	—	5	—	ns
*25A[9]	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	15	—	10	—	5	—	5	—	ns
26	Data-Out Valid to $\overline{DS}$ Asserted (Write)	15	—	10	—	5	—	5	—	ns
27	Data-In Valid to Clock Low (Setup) (Read)	5	—	5	—	5	—	5	—	ns
27A	Late $\overline{BERR}$/$\overline{HALT}$ Asserted to Clock Low (Setup)	20		15		10	—	5	—	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$, $\overline{BERR}$, $\overline{HALT}$, $\overline{AVEC}$ Negated	0	80	0	65	0	50	0	40	ns

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

AC ELECTRICAL CHARACTERISTICS—READ AND WRITE CYCLES

(Continued)

Num.	Characteristics	16.67 MHz Min	16.67 MHz Max	20 MHz Min	20 MHz Max	25 MHz** Min	25 MHz** Max	33.33 MHz Min	33.33 MHz Max	Unit
29	$\overline{AS}$, $\overline{DS}$ Negated to Data-In Invalid (Data-In Hold Time)	0	—	0		0	—	0	—	ns
29A	$\overline{AS}$, $\overline{DS}$ Negated to Data-In (High Impedance)	—	60	—	50	—	40	—	30	ns
30	Clock Low to Data-In Invalid (Data-In Hold Time)	15	—	15	—	10	—	10	—	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid	—	50	—	43	—	32	—	17	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid ($\overline{DSACKx}$ Asserted Skew)	—	15	—	10	—	10	—	10	ns
32	$\overline{RESET}$ Input Transition Time	—	1.5	—	1.5	—	1.5	—	1.5	Clks
33	Clock Low to $\overline{BG}$ Asserted	0	30	0	25	0	20	0	20	ns
34	Clock Low to $\overline{BG}$ Negated	0	30	0	25	0	20	0	20	ns
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clks
*37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clks
*37A[6]	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	0	1.5	0	1.5	0	1.5	0	1.5	Clks
39	$\overline{BG}$ Width Negated	90	—	75	—	60	—	50	—	ns
39A	$\overline{BG}$ Width Asserted	90	—	75	—	60	—	50	—	ns
*40	Clock High to $\overline{DBEN}$ Asserted (Read)	0	30	0	25	0	20	0	15	ns
*41	Clock Low to $\overline{DBEN}$ Negated (Read)	0	30	0	25	0	20	0	15	ns
*42	Clock Low to $\overline{DBEN}$ Asserted (Write)	0	30	0	25	0	20	0	15	ns
*43	Clock High to $\overline{DBEN}$ Negated (Write)	0	30	0	25	0	20	0	15	ns
*44	R/$\overline{W}$ Low to $\overline{DBEN}$ Asserted (Write)	15	—	10	—	10	—	5	—	ns
*45[5]	$\overline{DBEN}$ Width Asserted Read Write	60 120	— —	50 100	— —	40 80	— —	30 60	— —	ns
46	R/$\overline{W}$ Width Valid (Write or Read)	150	—	125	—	100	—	75	—	ns
47A	Asynchronous Input Setup Time	5	—	5	—	5	—	5	—	ns
47B	Asynchronous Input Hold Time	15	—	15		10	—	10	—	ns
48[4]	$\overline{DSACKx}$ Asserted to $\overline{BERR}$, $\overline{HALT}$ Asserted	—	30	—	20	—	18	—	15	ns
53	Data-Out Hold from Clock High	0	—	0	—	0	—	0		ns
55	R/$\overline{W}$ Valid to Data Bus Impedance Change	30	—	25	—	20	—	20	—	ns
56	$\overline{RESET}$ Pulse Width (Reset Instruction)	512		512	—	512	—	512	—	Clks
57	$\overline{BERR}$ Negated to $\overline{HALT}$ Negated (Rerun)	0	—	0	—	0	—	0	—	ns
*58[10]	$\overline{BGACK}$ Negated to Bus Driven	1	—	1	—	1	—	1	—	Clks
59[10]	$\overline{BG}$ Negated to Bus Driven	1	—	1	—	1	—	1	—	Clks

*This specification does not apply to the MC68EC020.

**These specifications represent an improvement over previously published specifications for the 25-MHz MC68020 and are valid only for product bearing date codes of 8827 and later.

10

For More Information On This Product,
Go to: www.freescale.com

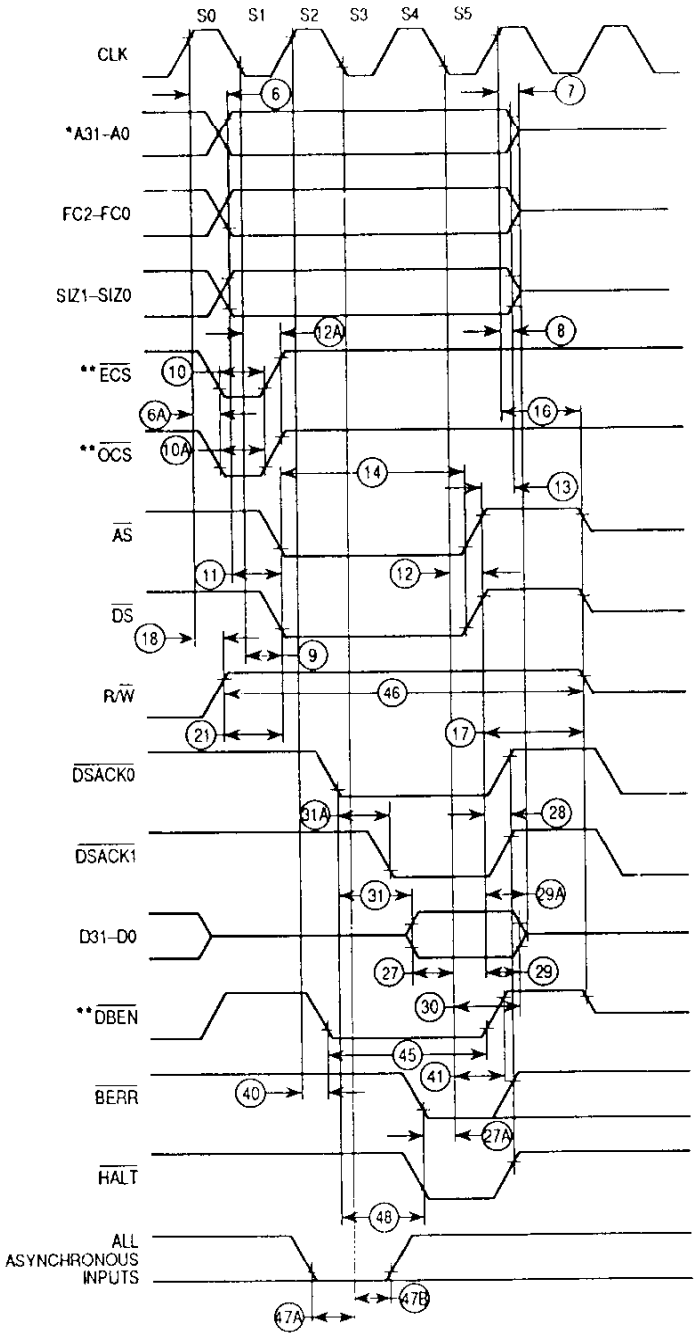
Freescale Semiconductor, Inc.

AC ELECTRICAL CHARACTERISTICS—READ AND WRITE CYCLES
(Concluded)

NOTES:

1. This number can be reduced to 5 ns if strobes have equal loads.
2. If the asynchronous setup time (#47A) requirements are satisfied, the $\overline{DSACKx}$ low to data setup time (#31) and $\overline{DSACKx}$ low to $\overline{BERR}$ low setup time (#48) can be ignored. The data must only satisfy the data-in clock low setup time (#27) for the following clock cycle, and $\overline{BERR}$ must only satisfy the late $\overline{BERR}$ low to clock low setup time (#27A) for the following clock cycle.
3. This parameter specifies the maximum allowable skew between $\overline{DSACK0}$ to $\overline{DSACK1}$ asserted or $\overline{DSACK1}$ to $\overline{DSACK0}$ asserted; specification #47A must be met by $\overline{DSACK0}$ or $\overline{DSACK1}$.
4. This specification applies to the first ($\overline{DSACK0}$ or $\overline{DSACK1}$) $\overline{DSACKx}$ signal asserted. In the absence of $\overline{DSACKx}$, $\overline{BERR}$ is an asynchronous input using the asynchronous input setup time (#47A).
5. $\overline{DBEN}$ may stay asserted on consecutive write cycles.
6. The minimum values must be met to guarantee proper operation. If this maximum value is exceeded, $\overline{BG}$ may be reasserted.
7. This specification indicates the minimum high time for $\overline{ECS}$ and $\overline{OCS}$ in the event of an internal cache hit followed immediately by a cache miss or operand cycle.
8. This specification guarantees operation with the MC68881/MC68882, which specifies a minimum time for $\overline{DS}$ negated to $\overline{AS}$ asserted (specification #13A in MC68881UM/AD, *MC68881/MC68882 Floating-Point Coprocessor User's Manual)*. Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68020/EC020 does not meet the MC68881/MC68882 requirements.
9. This specification allows a system designer to guarantee data hold times on the output side of data buffers that have output enable signals generated with $\overline{DBEN}$.
10. These specifications allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68020/EC020 regains control of the bus after an arbitration sequence.
11. This specification allows system designers to qualify the $\overline{CS}$ signal of an MC68881/MC68882 with $\overline{AS}$ (allowing 7 ns for a gate delay) and still meet the $\overline{CS}$ to $\overline{DS}$ setup time requirement (specification 8B of MC68881UM/AD, *MC68881/MC68882 Floating-Point Coprocessor User's Manual)*.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A0.

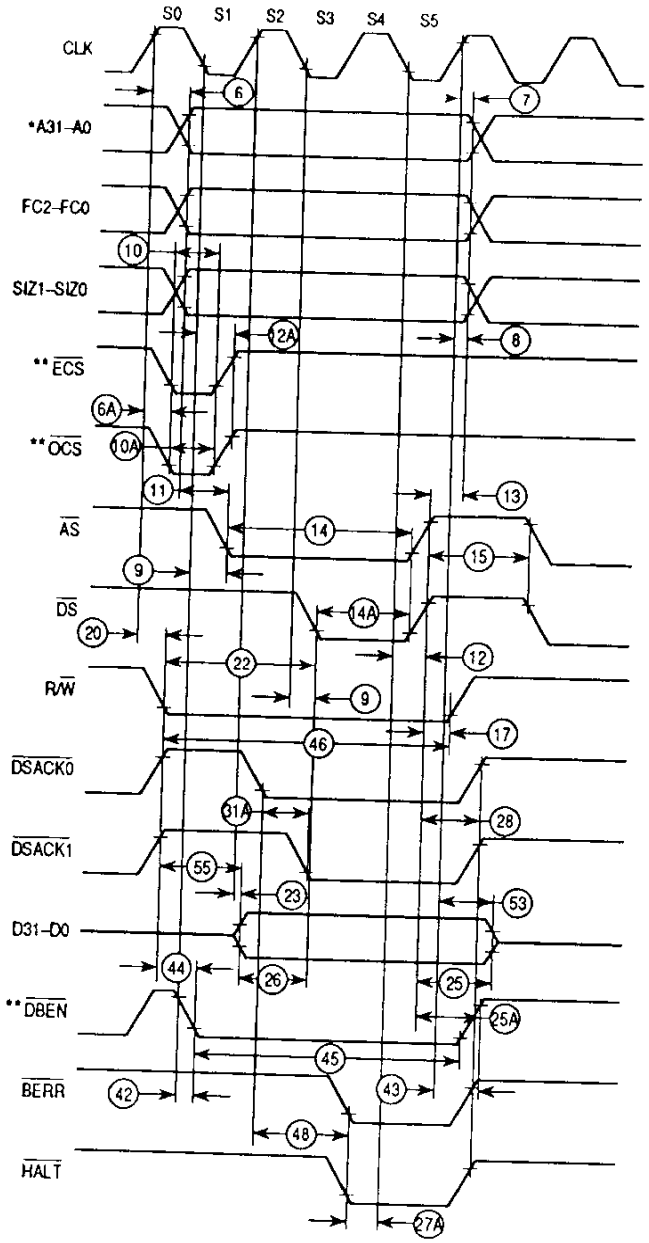
** This signal does not apply to the MC68EC020.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 2.0 V.

Figure 10-3. Read Cycle Timing Diagram

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.



* For the MC68EC020, A23–A0.

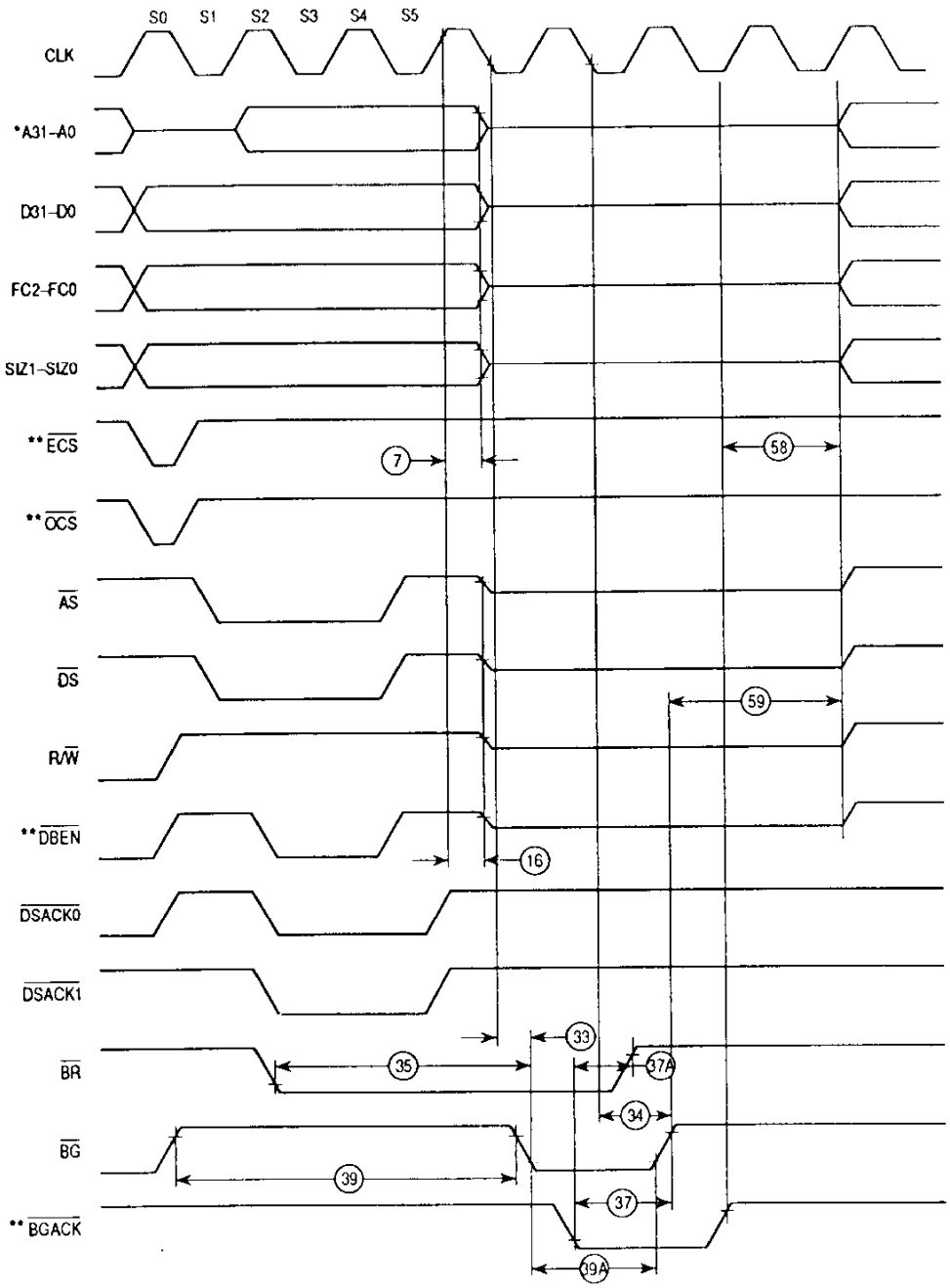
** This signal does not apply to the MC68EC020.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 2.0 V.

Figure 10-4. Write Cycle Timing Diagram

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



* For the MC68EC020, A23–A0.
** This signal does not apply to the MC68EC020.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 2.0 V.

Figure 10-5. Bus Arbitration Timing Diagram

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SECTION 11
ORDERING INFORMATION AND MECHANICAL DATA

This section contains the pin assignments and package dimensions of the MC68020 and the MC68EC020. In addition, detailed information is provided to be used as a guide when ordering.

11.1 STANDARD ORDERING INFORMATION

11.1.1 Standard MC68020 Ordering Information

Package Type	Frequency (MHz)	Temperature (°C)	Order Number
Ceramic Pin Grid Array RC Suffix	16.67	0 to 70	MC68020RC16
	20.0	0 to 70	MC68020RC20
	25.0	0 to 70	MC68020RC25
	33.33	0 to 70	MC68020RC33
Plastic Quad Flat Pack FC Suffix	16.67	0 to 70	MC68020FC16
	20.0	0 to 70	MC68020FC20
	25.0	0 to 70	MC68020FC25
Plastic Pin Grid Array RP Suffix	16.67	0 to 70	MC68020RP16
	20.0	0 to 70	MC68020RP20
	25.0	0 to 70	MC68020RP25
Ceramic Quad Flat Pack FE Suffix	16.67	0 to 70	MC68020FE16
	20.0	0 to 70	MC68020FE20
	25.0	0 to 70	MC68020FE25
	33.33	0 to 70	MC68020FE33

11.1.2 Standard MC68EC020 Ordering Information

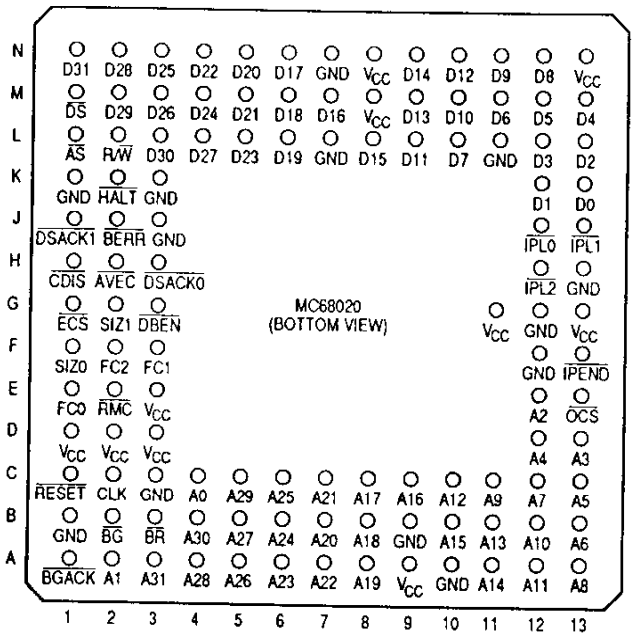
Package Type	Frequency (MHz)	Temperature (°C)	Order Number
Plastic Pin Grid Array RP Suffix	16.67	0 to 70	MC68EC020RP16
	25.0	0 to 70	MC68EC020RP25
Plastic Quad Flat Pack FG Suffix	16.67	0 to 70	MC68EC020FG16
	25.0	0 to 70	MC68EC020FG25

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

11.2 PIN ASSIGNMENTS AND PACKAGE DIMENSIONS

11.2.1 MC68020 RC and RP Suffix—Pin Assignment



The V_{CC} and GND pins are separated into four groups to provide individual power supply connections for the address bus buffers, data bus buffers, and all other output buffers and internal logic. It is recommended that all pins be connected to power and ground as indicated.

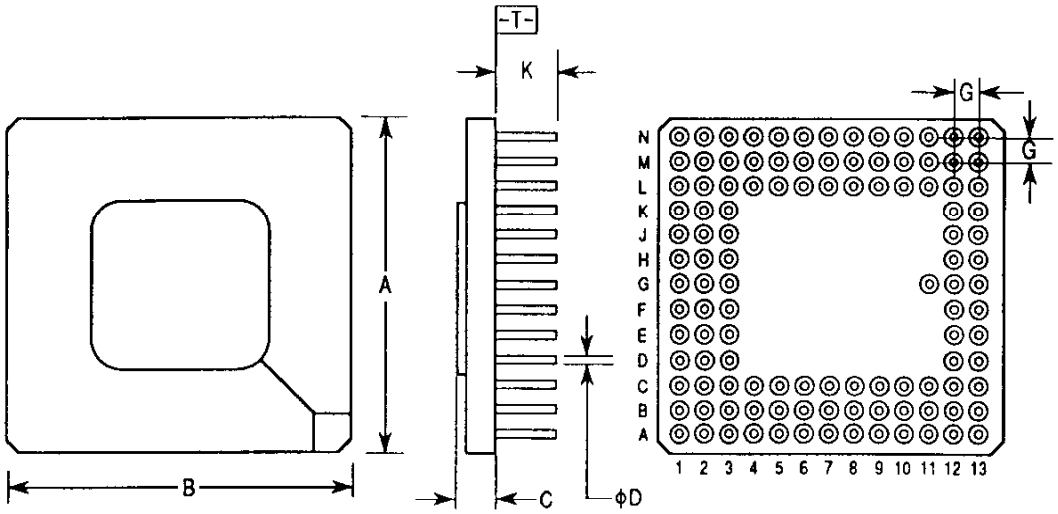
Group	V_{CC}	GND
Address Bus	A9, D3	A10, B9, C3, F12
Data Bus	M8, N8, N13	L7, L11, N7, K3
Logic	D1, D2, E3, G11, G13	G12, H13, J3, K1
Clock	--	B1

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.2.2 MC68020 RC Suffix—Package Dimensions

CASE 791-01



DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	34.04	35.05	1.340	1.380
B	34.04	35.05	1.340	1.380
C	2.54	3.81	0.100	0.150
D	0.43	0.55	0.017	0.022
G	2.54 BSC		0.100 BSC	
K	4.32	4.95	0.170	0.195

NOTES:
1. A AND B ARE DATUMS AND T IS A DATUM SURFACE.
2. POSITIONAL TOLERANCE FOR LEADS (114 PLACES).

⌖	0.13 (0.005) Ⓜ	T	A Ⓢ	B Ⓢ

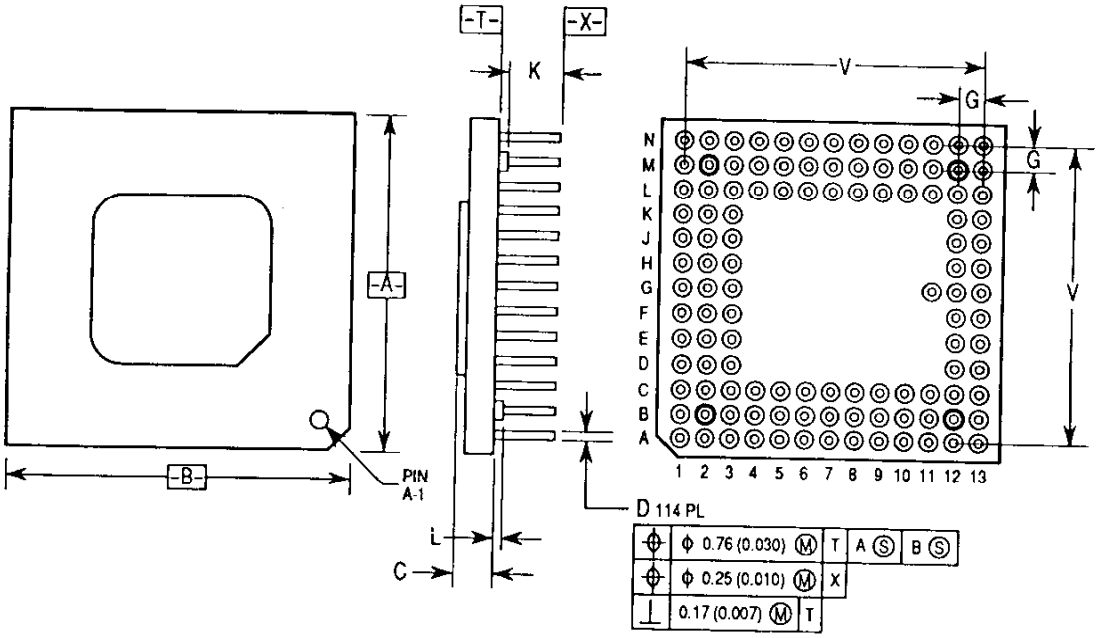
3. DIMENSIONING AND TOLERANCING PER Y14.5M,1982.
4. CONTROLLING DIMENSION: INCH.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.2.3 MC68020 RP Suffix—Package Dimensions

CASE 789E-02



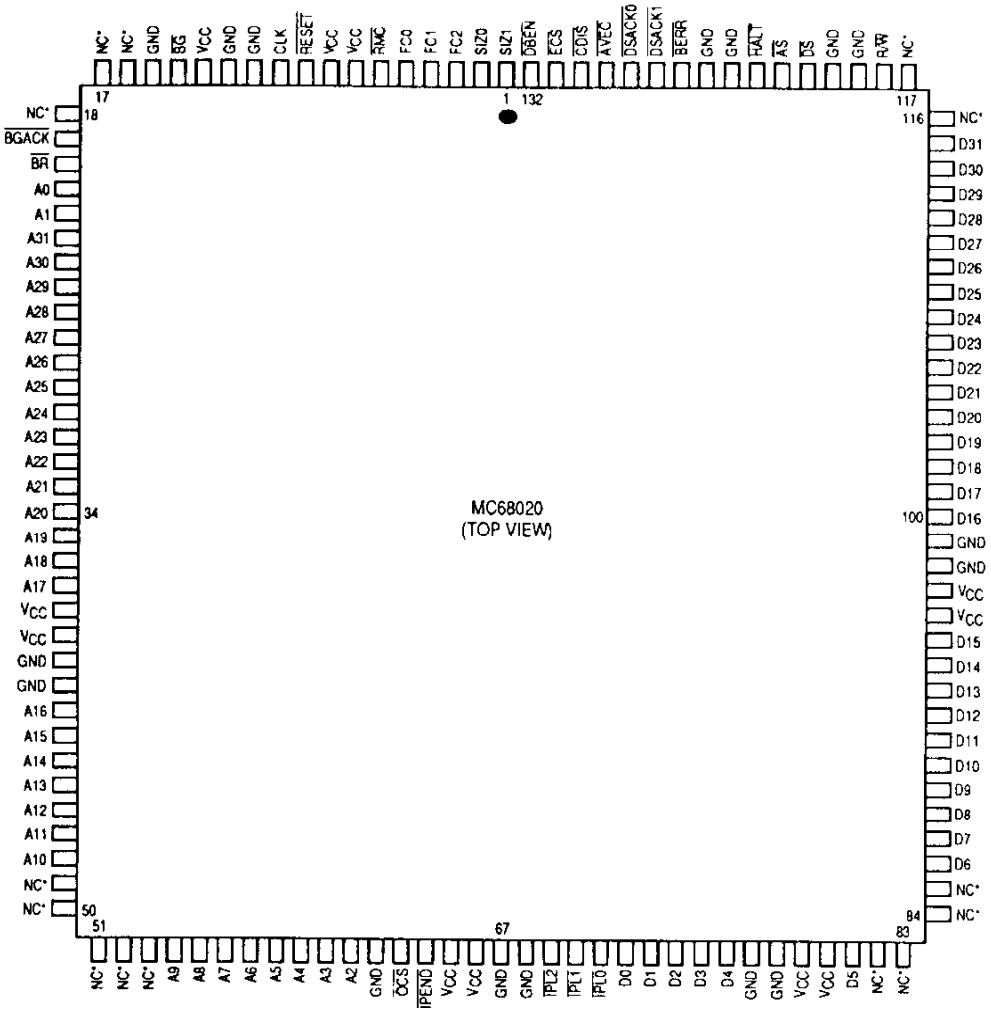
DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	34.04	35.05	1.340	1.380
B	34.04	35.05	1.340	1.380
C	2.92	3.18	0.115	0.135
D	0.44	0.55	0.017	0.022
G	2.54 BSC		0.100 BSC	
K	2.79	3.81	0.110	0.150
L	1.02	1.52	0.040	0.060
V	30.48 BSC		1.200 BSC	

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION D INCLUDES LEAD FINISH.
4. 789E-01 OBSOLETE. NEW STANDARD 789E-02.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.2.4 MC68020 FC and FE Suffix—Pin Assignment



*NC—Do not connect to this pin.

The V_{CC} and GND pins are separated into four groups to provide individual power supply connections for the address bus buffers, data bus buffers, and all other output buffers and internal logic. It is recommended that all pins be connected to power and ground as indicated. NC pins are reserved by Motorola for future use and should have no external connection.

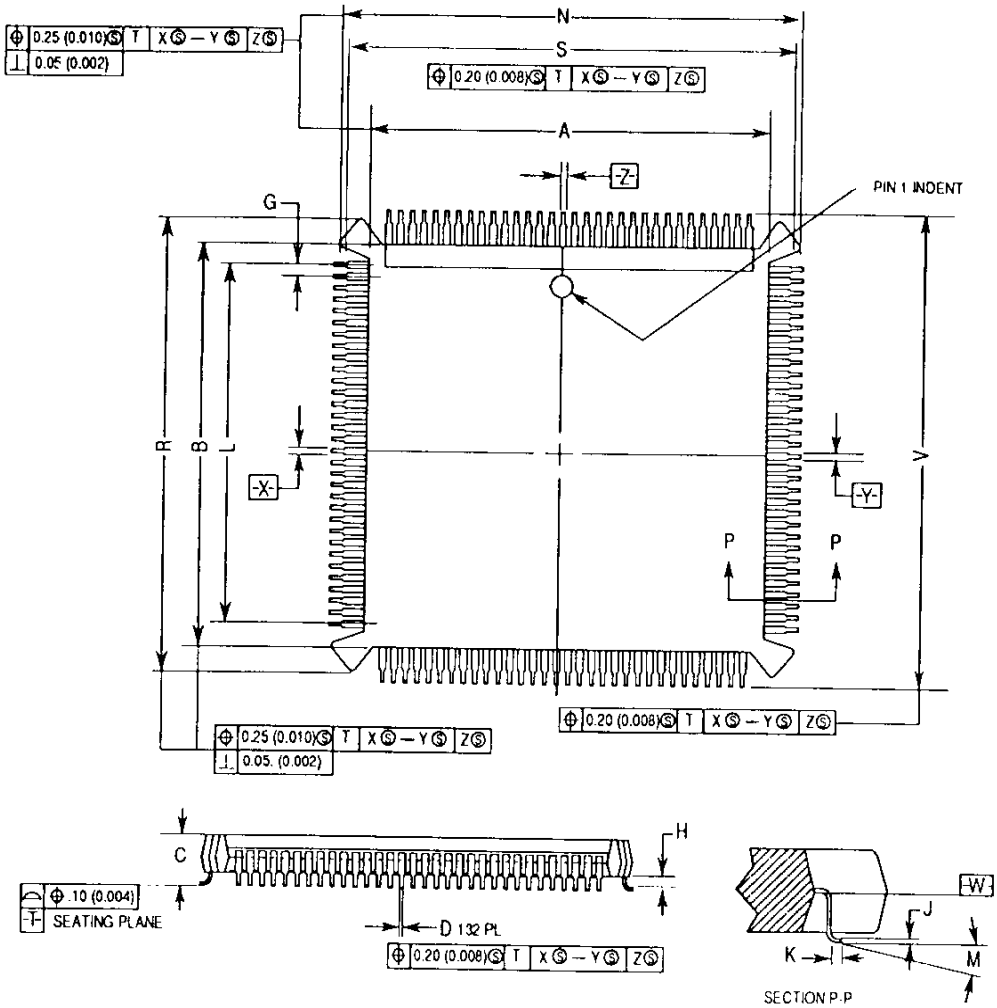
Group	V_{CC}	GND
Address Bus	13, 38, 39	15, 40, 41, 62
Data Bus	79, 80, 96, 97	77, 78, 98, 99, 119, 120
Logic	7, 8, 65, 66	67, 68, 124, 125
Clock		11, 12

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.2.5 MC68020 FC Suffix—Package Dimensions

CASE 831A-01



DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	24.06	24.20	0.947	0.953
B	24.06	24.20	0.947	0.953
C	4.07	4.57	0.160	0.180
D	0.21	0.30	0.008	0.012
G	0.64 BSC		0.025 BSC	
H	0.51	1.01	0.020	0.040
J	0.16	0.20	0.006	0.008
K	0.51	0.76	0.020	0.030
L	20.32 REF		0.800 REF	
M	0°	8°	0°	8°
N	27.88	28.01	1.097	1.103
R	27.88	28.01	1.097	1.103
S	27.31	27.55	1.075	1.085
V	27.31	27.55	1.075	1.085

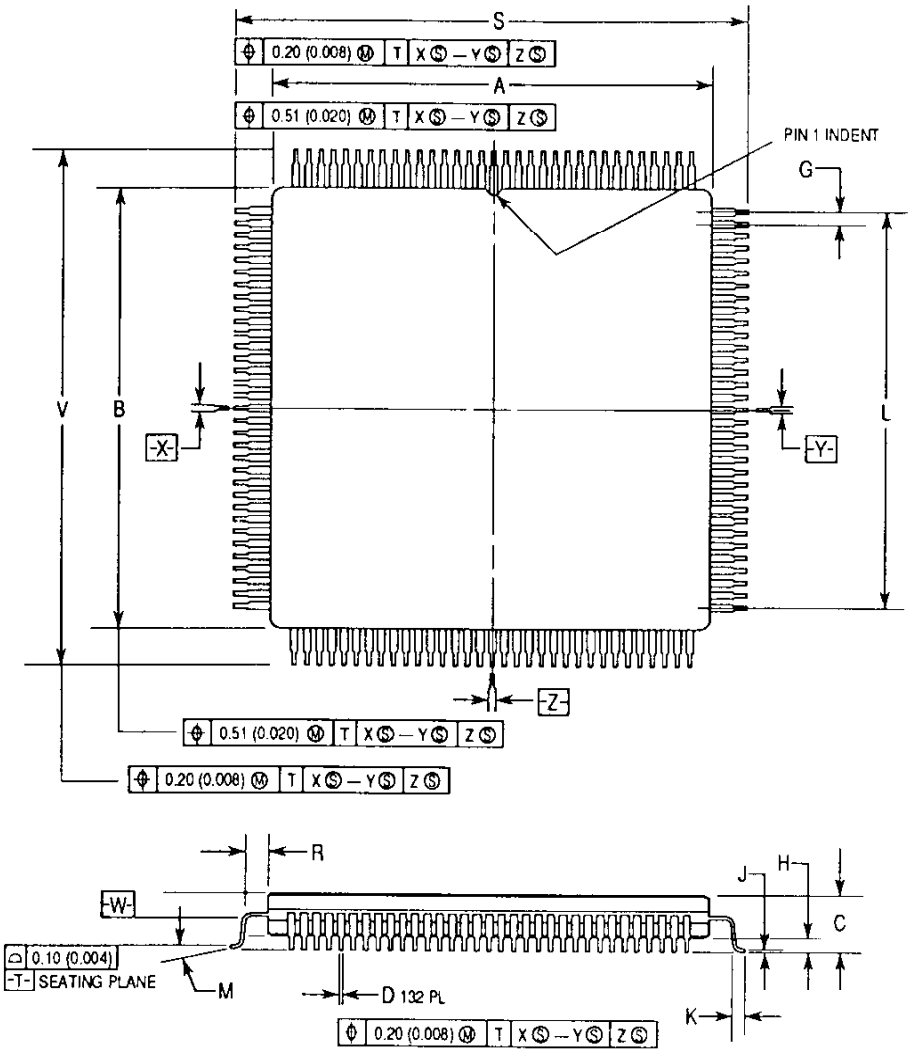
NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSIONS A, B, N, AND R DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE MOLD PROTRUSION FOR DIMENSIONS A AND B IS 0.25 (0.010). FOR DIMENSIONS N AND R IS 0.18 (0.007).
4. DATUM PLANE -W- IS LOCATED AT THE UNDERSIDE OF LEADS WHERE LEADS EXIT PACKAGE BODY.
5. DATUMS -X-, -Y-, AND -Z- TO BE DETERMINED WHERE CENTER LEADS EXIT PACKAGE BODY AT DATUM -W-.
6. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE, DATUM -T-.
7. DIMENSIONS A, B, N AND R TO BE DETERMINED AT DATUM PLANE -W-.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.2.6 MC68020 FE Suffix—Package Dimensions

CASE 831-01



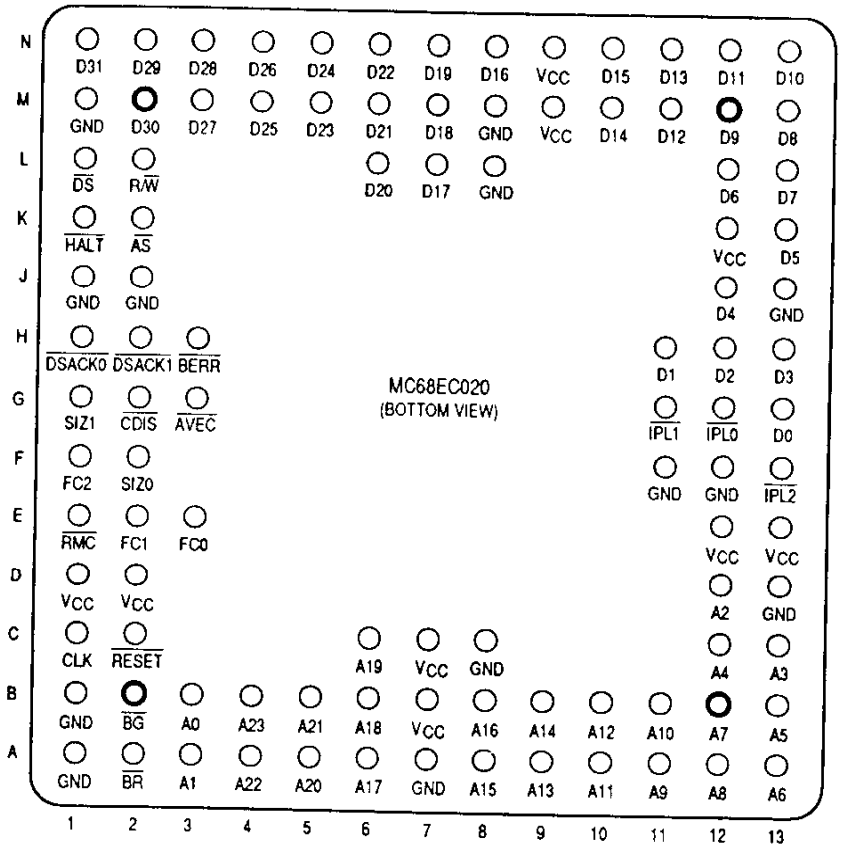
DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	21.85	22.86	0.860	0.900
B	21.85	22.86	0.860	0.900
C	3.94	4.31	0.155	0.170
D	0.204	0.292	0.0080	0.0115
G	0.64 BSC		0.025 BSC	
H	0.64	0.88	0.025	0.035
J	0.13	0.20	0.005	0.008
K	0.51	0.76	0.020	0.030
L	20.32 REF		0.800 REF	
M	0°	8°	0°	8°
R	0.64	—	0.025	—
S	27.31	27.55	1.075	1.085
V	27.31	27.55	1.075	1.085

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH
3. DIMENSIONS A AND B DEFINE MAXIMUM CERAMIC BODY DIMENSIONS INCLUDING GLASS PROTRUSION AND MISMATCH OF CERAMIC BODY TOP AND BOTTOM.
4. DATUM PLANE -W- IS LOCATED AT THE UNDERSIDE OF LEADS WHERE LEADS EXIT PACKAGE BODY.
5. DATUMS -X-, -Y-, AND -Z- TO BE DETERMINED WHERE CENTER LEADS EXIT PACKAGE BODY AT DATUM -W-.
6. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE, DATUM -T-.
7. DIMENSIONS A AND B TO BE DETERMINED AT DATUM PLANE -W-.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

11.2.7 MC68EC020 RP Suffix—Pin Assignment



The V_{CC} and GND pins are separated into four groups to provide individual power supply connections for the address bus buffers, data bus buffers, and all other output buffers and internal logic. It is recommended that all pins be connected to power and ground as indicated.

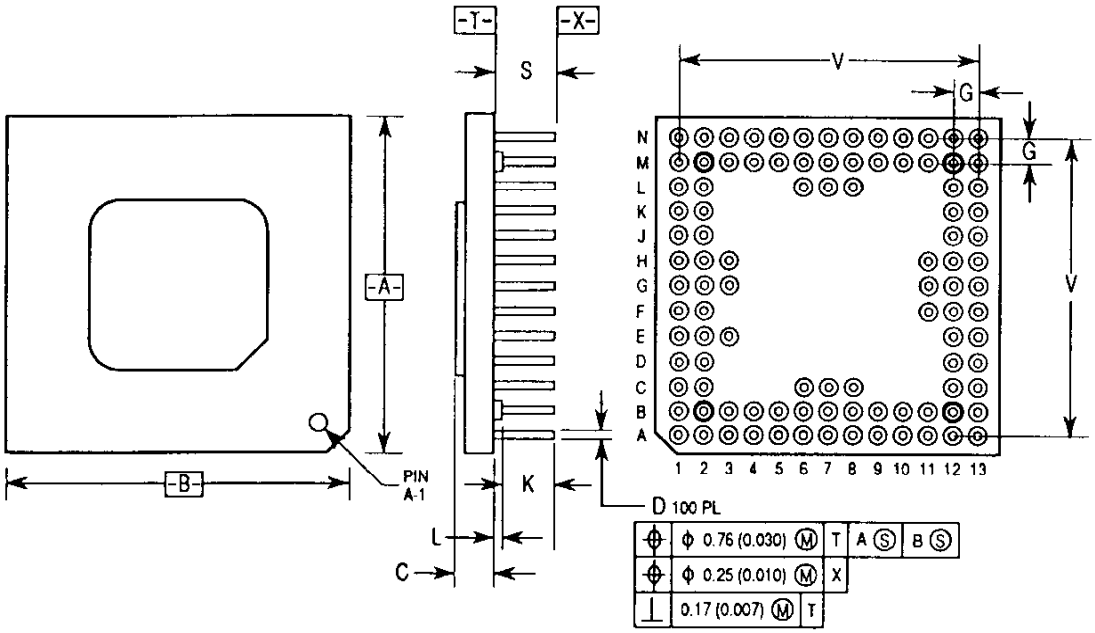
Group	V_{CC}	GND
Address Bus	B7, C7	A1, A7, C8, D13
Data Bus	K12, M9, N9	J13, L8, M1, M8
Logic	D1, D2, E12, E13	F11, F12, J1, J2
Clock	--	B1

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.2.8 MC68EC020 RP Suffix—Package Dimensions

CASE 789H-01



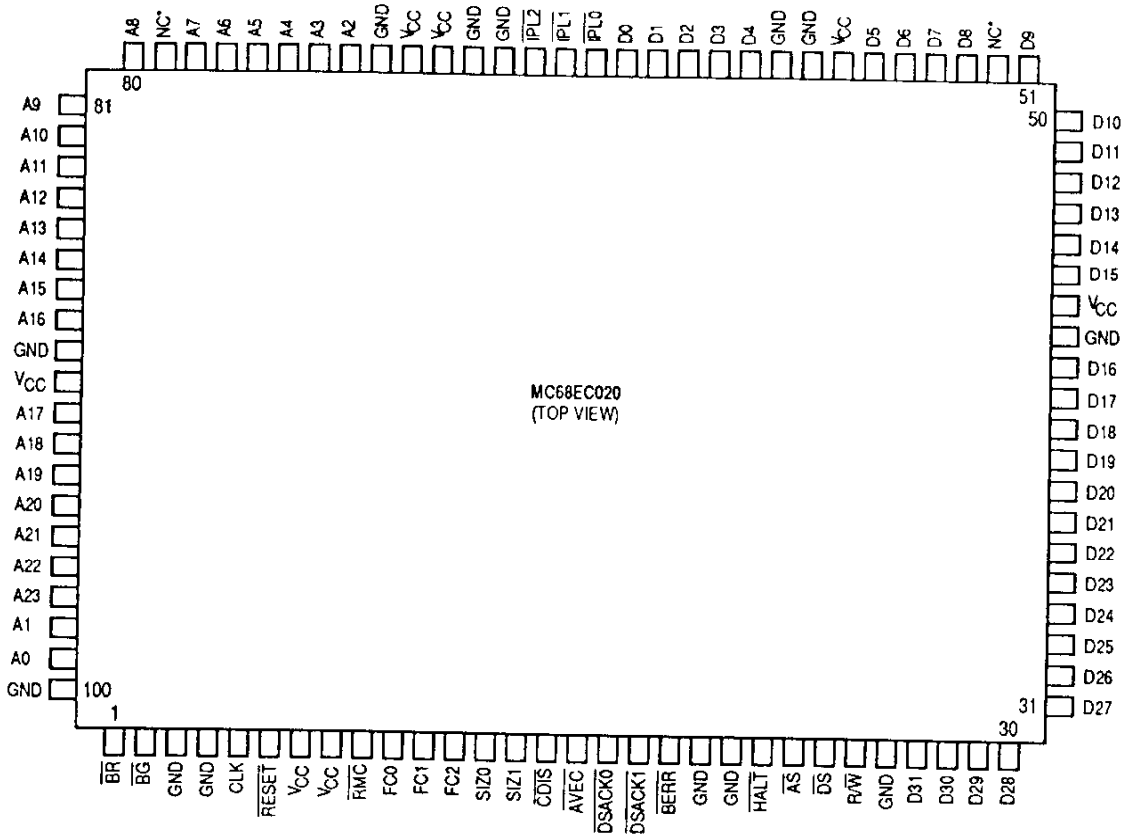
DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	34.04	35.05	1.340	1.380
B	34.04	35.05	1.340	1.380
C	2.92	3.18	0.115	0.135
D	0.44	0.55	0.017	0.022
G	2.54 BSC		0.100 BSC	
K	3.05	3.55	0.120	0.140
L	1.02	1.52	0.040	0.060
S	4.32	4.83	0.170	0.190
V	30.48 BSC		1.200 BSC	

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH
3. DIMENSION D INCLUDES LEAD FINISH.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.2.9 MC68EC020 FG Suffix—Pin Assignment



*NC—Do not connect to this pin.

The V_{CC} and GND pins are separated into four groups to provide individual power supply connections for the address bus buffers, data bus buffers, and all other output buffers and internal logic. It is recommended that all pins be connected to power and ground as indicated. NC pins are reserved by Motorola for future use and should have no external connection.

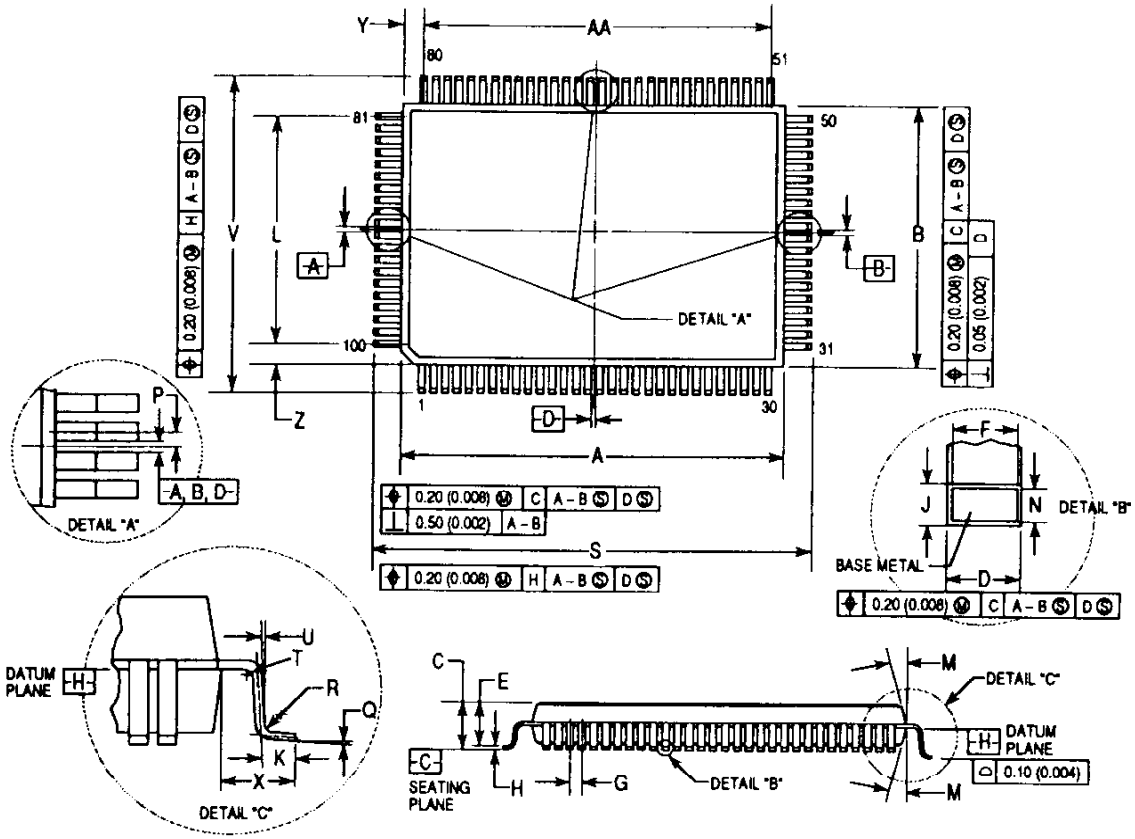
Group	V_{CC}	GND
Address Bus	90	72, 89, 100
Data Bus	44, 57	26, 43, 58, 59
Logic	7, 8, 70, 71	3, 20, 21, 68, 69
Clock	—	4

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

11.2.10 MC68EC020 FG Suffix—Package Dimensions

CASE 842D-02



DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	19.90	20.10	0.783	0.791
B	13.90	14.10	0.547	0.555
C	—	3.30	—	0.130
D	0.22	0.38	0.009	0.015
E	2.55	3.05	0.100	0.120
F	0.22	0.33	0.009	0.013
G	0.65 BSC		0.026 BSC	
H	0.10	0.36	0.004	0.014
J	0.17	0.23	0.007	0.009
K	0.65	0.95	0.026	0.037
L	12.35 REF		0.486 REF	
M	5°	16°	5°	16°
N	0.13	0.17	0.005	0.007
P	0.325 BSC		0.013 BSC	
Q	0°	7°	0°	7°
R	0.25	0.35	0.010	0.014
S	23.65	24.15	0.931	0.951
T	0.13	—	0.005	—
U	0°	—	0°	—
V	17.65	18.15	0.695	0.715
X	1.95 REF		0.077 REF	
Y	0.58 REF		0.023 REF	
Z	0.83 REF		0.033 REF	
AA	18.85 REF		0.742 REF	

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DATUM PLANE -H- IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
4. DATUMS -A-, -B-, AND -D- TO BE DETERMINED AT DATUM PLANE -H-.
5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE -C-.
6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25 (0.010) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE -H-.
7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 (0.003) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

APPENDIX A
INTERFACING AN MC68EC020 TO A DMA DEVICE THAT SUPPORTS A THREE-WIRE BUS ARBITRATION PROTOCOL

The MC68EC020 supports a two-wire bus arbitration protocol; however, it may become necessary to interface the MC68EC020 to a device that supports a three-wire arbitration protocol. Figure A-1 shows a method by which this can be achieved.

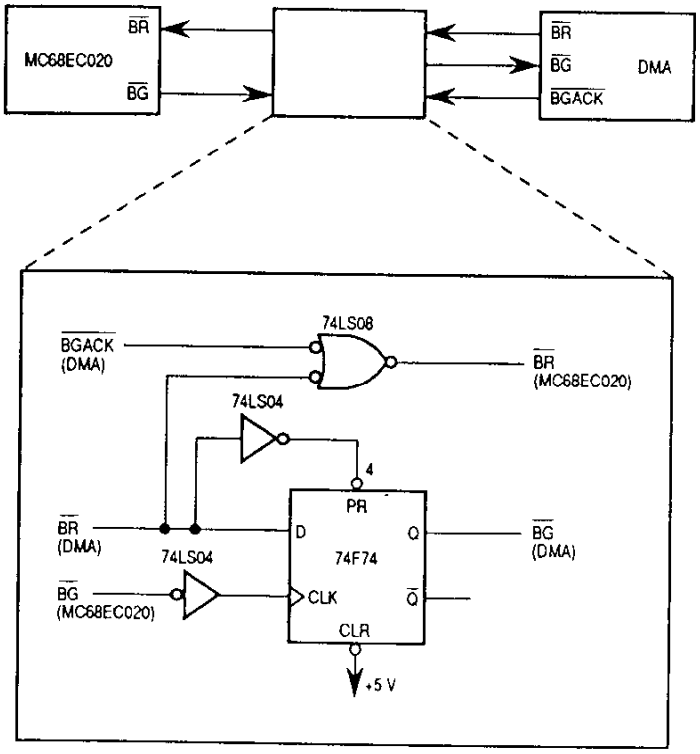


Figure A-1. Bus Arbitration Circuit—
MC68EC020 (Two-Wire) to DMA (Three-Wire)

A

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

INDEX

— A —

A1, A0 Signals, 5-2, 5-7, 5-9, 5-21, 9-5
A15–A13 Signals, 7-6
A19–A16 Signals, 7-6
A31–A24 Signals, 4-1, 5-3
AC Specifications, 10-5
Access Level, 9-17
Access Time Calculations, 9-12
Address Bus, 3-2, 5-3, 5-25
Address Error Exception, 5-14, 6-6
Address Registers, 1-4
Address Space, 2-4, 5-3
Addressing Modes, 1-8
Arithmetic/Logical Instruction, 8-30, 8-31
$\overline{AS}$ Signal, 3-4, 5-2, 5-3
Autovector, 5-48
Autovector Interrupt Acknowledge Cycle, 5-48
$\overline{AVEC}$ Signal, 3-5, 5-4, 5-48, 5-53

— B —

$\overline{BERR}$ Signal, 3-7, 5-4, 5-25, 5-53, 5-55, 6-4
$\overline{BG}$ Signal, 3-6, 5-63, 5-66, 5-70
$\overline{BGACK}$ Signal, 3-6, 5-62, 5-63, 5-66
Binary-Coded Decimal, 8-32
Bit Field Manipulation Instructions, 8-36
Bit Manipulation Instructions, 8-35
BKPT Instruction, 5-50
  Flowchart, 6-17
Block Diagram, 1-2
$\overline{BR}$ Signal, 3-6, 5-63, 5-66, 5-70, 5-71
Breakpoint Acknowledge Cycle, 5-50, 6-17
  Flowchart, 5-50
  Timing, 5-50
Breakpoint Instruction Exception, 6-17
Bus, 5-24
  Arbitration, 5-62
  Cycles, 5-1
  Master, 5-1
  Operation, 5-1, 5-24
Bus Arbitration (MC68020), 5-63
  Control Unit, 5-67
  Flowchart, 5-63
  Read-Modify-Write, 5-68
  Timing, 5-63
Bus Arbitration (MC68EC020), 5-70
  Control Unit, 5-73
  Flowchart, 5-70
  Timing, 5-70
  Two-Wire, 5-75, A-1
Bus Controller, 5-22, 8-2, 8-5
Bus Cycles, 5-1, 5-25
Bus Error Exception, 6-4, 6-21
Bus Fault, 6-21
Bus Master, 5-1, 5-25, 5-62
Bus Operation, 5-24
Byte Enable Signals, 5-21
Byte Select Control Signals, 9-5

— C —

Cache, 1-13, 4-1, 5-2, 5-22, 5-62, 8-1, 8-7, 9-11
  Control, 4-3
  Internal Cache Holding Register, 5-21
  Reset, 4-3
Cache Address Register (CAAR), 1-7, 4-3, 4-4
Cache Control Register (CACR), 1-7, 4-2, 4-3
CALLM Instruction, 9-14, 9-16, 9-18
CAS Instruction, 5-39
CAS2 Instruction, 5-39
$\overline{CDIS}$ Signal, 3-7, 4-3
CLK Signal, 3-7
Clock Drivers, 9-10
Condition Codes, 1-7
Conditional Branch Instructions, 8-37
Control Instructions, 8-38
Coprocessor, 6-25, 7-1
  Classification, 7-4
  Communication Protocol, 7-4
  Conditional Instruction Category, 7-10
  Coprocessor Context Restore Instruction Category, 7-22

I

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Coprocessor Context Save and Context Restore Instruction Categories, 7-16
Coprocessor Context Save Instruction Category, 7-20
Coprocessor General Instruction Category, 7-8
Coprocessor Instructions, 7-1, 7-7
Format Words, 7-18
Instruction, 6-25, 7-3
Instruction Execution, 7-6
Coprocessor Detected
Data-Processing-Related Exceptions, 7-51
Exception, 7-49
Format Errors, 7-52
Illegal Coprocessor Command or Condition Words, 7-51
Protocol Violation Exceptions, 7-50
System-Related Exceptions, 7-51
Coprocessor Instruction, 6-25, 7-3
Coprocessor Interface, 5-53, 7-1, 7-2
Coprocessor Interface Register (CIR), 7-4, 7-5, 7-6, 7-24
Command CIR, 7-25
Condition CIR, 7-26
Control CIR, 7-24
Instruction Address CIR, 7-27
Memory Map, 7-24
Operand Address CIR, 7-27
Operand CIR, 7-26
Operation Word CIR, 7-25
Register Select CIR, 7-27
Response CIR, 7-24
Restore CIR, 7-25
Save CIR, 7-25
Selection, 7-6
CPU Address Space, 2-4, 5-44, 7-6
CPU Space Type, 5-44, 5-53
Cycle
Asynchronous Cycles, 5-1, 5-5
Autovector Interrupt Acknowledge Cycle, 5-45, 5-48
Breakpoint Acknowledge Cycle, 5-50
Coprocessor Interface Bus Cycles, 7-4
Interrupt Acknowledge Cycle, 5-4, 5-45
Operand Transfer Cycle, 5-5
Synchronous Cycle, 5-24

D

Data Accesses, 4-2
Data Bus (D31–D0), 3-2, 5-3, 5-5, 5-21, 5-25
Data Registers, 1-4
Data Types, 1-8
$\overline{\text{DBEN}}$ Signal, 3-5, 5-4
DC Electrical Characteristics
MC68020, 10-4
MC68EC020, 10-5
Destination Function Code Register (DFC), 1-7
Differences between MC68020 and MC68EC020, 1-1, 5-62
Double Bus Fault, 5-60
$\overline{\text{DS}}$ Signal, 3-4, 5-4, 5-21
$\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$ Signals, 3-5, 5-4, 5-5, 5-24, 5-46, 5-53, 9-5
Dynamic Bus Sizing, 5-5

— E —

$\overline{\text{ECS}}$ Signal, 3-4, 5-3
Effective Address, 8-13, 8-14, 8-16, 8-17, 8-19
Electrical Specifications, 10-1
Exception, 2-5, 7-57
Address Error Exception, 5-14, 6-6, 7-57
Breakpoint Instruction, 6-17
Bus Error Exception, 6-4, 6-21
Coprocessor-Detected Exception, 7-49
cpTRAPcc Instruction Traps, 7-55
Data-Processing-Related Exception, 7-51
F-Line Emulator Exception, 7-54
Format Error Exception, 6-10, 7-57
Illegal Instruction, 6-7
Interrupt Exception, 5-45, 6-11, 7-56
Multiple, 6-17
Privilege Violation Exception, 6-7, 6-8, 7-55
Protocol Violation, 7-50
Reset Exception, 6-4
Stack Frames, 6-25
System-Related Exception, 7-51
Trace Exception, 6-9, 7-55
Trap Exception, 6-6
Unimplemented Instruction, 6-7
Exception Handler, 6-2
Exception Processing, 2-1, 2-5, 6-1
Exception-Related Instructions, 8-39
Exception Stack Frame, 2-6, 6-25
Exception Vector Table, 2-5, 6-2

Freescale Semiconductor, Inc.

I

For More Information On This Product,
Go to: www.freescale.com

— F —
FC2–FC0 Signals, 2-2, 2-4, 3-2, 5-2, 5-3, 5-44, 7-6
Features, 1-2
F-Line Operation Words, 7-3
Floating-Point Coprocessor, 7-1, 9-1
Flowchart
  Breakpoint Acknowledge Cycle, 5-50
  Byte Read Cycle, 5-26
  Interrupt Acknowledge Cycle, 5-46
  Long-Word Read Cycle, 5-26
  MC68EC020 Bus Arbitration, 5-70
  MC68020 Bus Arbitration, 5-63
  Read-Modify-Write Cycle, 5-39
  Reset Exception, 6-4
  Write Cycle, 5-33
Format Error Exception, 6-10

— G —
Ground Connections, 3-7, 9-9

— H —
$\overline{\text{HALT}}$ Signal, 3-7, 5-4, 5-25, 5-53, 5-60

— I —
Idle Clock Cycles, 8-7
Illegal Instruction Exception, 6-7
Instruction
  Arithmetic/Logical, 8-30, 8-31
  Bcc, 8-37
  Bit Field Manipulation, 8-36
  Bit Manipulation, 8-35
  BKPT, 5-50
  CALLM, 9-14, 9-16, 9-18
  CAS, 5-39
  CAS2, 5-39
  Control, 8-38
  Coprocessor Conditional Instructions, 7-10
  Coprocessor Context Restore Instruction Category, 7-22
  Coprocessor Context Save Instruction Category, 7-20
  Coprocessor Instruction, 6-25, 7-1, 7-3, 7-7
  Coprocessor Instruction Execution, 7-6
  Coprocessor Context Save and Context Restore Instruction Categories, 7-16
  Coprocessor General Instruction Category, 7-8
  cpBcc, 7-12
  cpDBcc, 7-14
  cpRESTORE, 7-17, 7-22
  cpSAVE, 7-17, 7-20
  cpScc, 7-13
  cpTRAPcc, 7-15, 7-55
  Exception-Related, 8-39
  Illegal Instruction, 6-7
  MOVE, 8-20
  MOVE SR, 8-3
  MOVEA, 8-20
  MOVEC, 4-3
  NOP, 5-62, 8-3
  Prefetches, 4-1
  Primitive Instructions, 7-27
  RESET, 5-76, 7-58
  RTE, 6-19, 6-24
  RTM, 9-14, 9-16, 9-19
  Shift/Rotate, 8-34
  Single-Operand Instructions, 8-33
  Special-Purpose MOVE, 8-29
  STOP, 6-10
  TAS, 5-39
  Unimplemented Instruction, 6-7
Instruction Execution, 5-62, 8-1
Instruction Execution Overlap, 8-4
Instruction Pipe, 1-12, 4-1, 6-21
Instruction Prefetches, 8-1
Instruction Set, 1-10
Instruction Timing, 8-8, 8-9
Internal Cache Holding Register, 5-21
Interrupt, 6-1
  Flowchart, 6-14
  Interrupt Exception, 6-11
  Nonmaskable, 6-12
Interrupt Acknowledge Cycle, 5-4, 5-45, 6-16
  Timing, 5-46
Interrupt Exception, 5-45, 6-11
Interrupt Priority Mask, 5-45, 6-11
Interrupt Stack Pointer (ISP), 1-4, 2-2
$\overline{\text{IPEND}}$ Signal, 3-5, 6-14
$\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$ Signals 3-5, 5-45, 6-11

I

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

— L —

Long-Word Operand, 5-10, 5-14
Long-Word Read Cycle, 5-26
Long-Word Write Cycle, 5-33

— M —

M-Bit (SR), 1-7, 2-2
Main Processor Detected
  Address Error, 7-57
  Bus Faults, 7-57
  cpTRAPcc Instruction Traps, 7-55
  Exceptions, 7-52
  F-Line Emulator Exception, 7-54
  Format Error, 7-57
  Interrupts, 7-56
  Privilege Violations, 7-55
  Protocol Violation, 7-52
  Trace Exception, 7-55
Master Stack Pointer (MSP), 1-4, 2-2
Maximum Ratings, 10-1
MC68881/MC68882 Floating-Point Coprocessors, 9-1
Memory Interface, 9-11
Misaligned
  Operand, 5-6, 5-14, 8-2
  Transfer, 5-1, 5-5
Module, 9-14
Module Call, 9-18
Module Return, 9-19
Module Stack Frame, 9-16
MOVE Instruction, 8-20
MOVE SR Instruction, 8-3
MOVEA Instruction, 8-20
MOVEC Instruction, 4-3

— N —

Nonmaskable Interrupt, 6-12
NOP Instruction, 5-62, 8-3
Normal Processing State, 2-1

— O —

$\overline{\text{OCS}}$ Signal, 3-4, 5-3
Ordering Information
  MC68020, 11-1
  MC68EC020, 11-1
Overlap, 8-3

— P —

Package Dimensions
  MC68020 FC Suffix, 11-6
  MC68020 FE Suffix, 11-7
  MC68020 RC Suffix, 11-3
  MC68020 RP Suffix, 11-4
  MC68EC020 FG Suffix, 11-11
  MC68EC020 RP Suffix, 11-9
Pin Assignment
  MC68020 FC Suffix, 11-5
  MC68020 FE Suffix, 11-5
  MC68020 RC Suffix, 11-2
  MC68020 RP Suffix, 11-2
  MC68EC020 FG Suffix, 11-10
  MC68EC020 RP Suffix, 11-8
Port Size, 5-1, 5-5, 5-21, 9-5
Power Supply, 3-7, 9-9
Primitive, 7-4, 7-27
  Busy Response Primitive, 7-30
  CA Bit, 7-29
  DR Bit, 7-29
  Evaluate and Transfer Effective Address Primitive, 7-35
  Evaluate Effective Address and Transfer Data Primitive, 7-35
  Format, 7-28
  Null Coprocessor Response Primitive, 7-31
  PC Bit, 7-29
  Supervisor Check Primitive, 7-33
  Take Address and Transfer Data Primitive, 7-39
  Take Midinstruction Exception Primitive, 7-47
  Take Postinstruction Exception Primitive, 7-48
  Take Preinstruction Exception Primitive, 7-45
  Transfer from Instruction Stream Primitive, 7-34
  Transfer Main Processor Control Register Primitive, 7-41
  Transfer Multiple Coprocessor Registers Primitive, 7-42
  Transfer Multiple Main Processor Registers Primitive, 7-42
  Transfer Operation Word Primitive, 7-33
  Transfer Single Main Processor Register Primitive, 7-40

Freescale Semiconductor, Inc.

I

For More Information On This Product,
Go to: www.freescale.com

Transfer Status Register and the scanPC Primitive, 7-44
Transfer to/from Top of Stack Primitive, 7-40
Write to Previously Evaluated Effective Address Primitive, 7-37
Privilege Level, 2-2
Changing, 2-3
Supervisor Level, 1-4, 2-2
User Level, 1-4, 2-2
Privilege Violation Exception, 6-7, 6-8
Processing States, 2-1
Program Counter (PC), 1-4
Programming Model, 1-4, 7-1, 7-2

— R —

Read Cycle, 5-3, 5-4, 5-8, 5-14, 5-16, 5-18, 5-22, 5-26
Byte Read Cycle, 5-26
Long-Word Read Cycle, 5-26, 8-2
Timing, 5-26
Read-Modify-Write Cycle, 5-3, 5-39, 5-42
Timing, 5-39
Registers
Address Registers, 1-4
CAAR, 1-7, 4-3, 4-4
CACR, 1-7, 4-2, 4-3
Data Registers, 1-4
DFC, 1-7
Internal Cache Holding Register, 5-21
Program Counter (PC), 1-4
SFC, 1-7
SR, 1-7, 4-1
VBR, 1-7
Reset, 4-3
Flowchart, 6-4
Reset Exception, 6-4
RESET Instruction, 7-58
$\overline{\text{RESET}}$ Signal, 3-6, 5-76, 6-4
Reset Exception, 6-4
RESET Instruction, 5-76
$\overline{\text{RESET}}$ Signal, 3-6, 5-76, 6-4
Retry, 5-56
$\overline{\text{RMC}}$ Signal, 3-4, 5-3, 5-39
RTE Instruction, 6-19, 6-24
RTM Instruction, 9-14, 9-16, 9-19
R/$\overline{\text{W}}$ Signal, 3-4, 5-2, 5-3, 9-5

— S —

S-bit (SR), 1-7, 2-2, 2-3
Save and Restore Operations, 8-40
scanPC, 7-28
Sequencer, 8-2, 8-5
Shift/Rotate Instructions, 8-34
Signal(s), 3-8
A1,A0, 5-2, 5-7, 5-9, 5-21, 9-5
A15–A13, 7-6
A19–A16, 7-6
A31–A24, 4-1, 5-3
Address Bus, 3-2, 5-3
$\overline{\text{AS}}$, 3-4, 5-2, 5-3
$\overline{\text{AVEC}}$, 3-5, 5-4, 5-48, 5-53
$\overline{\text{BERR}}$, 3-7, 5-4, 5-25, 5-53, 5-55, 6-4
$\overline{\text{BG}}$, 3-6, 5-63, 5-66, 5-70, 5-71
$\overline{\text{BGACK}}$, 3-6, 5-62, 5-63, 5-66
$\overline{\text{BR}}$, 3-6, 5-63, 5-66, 5-70
Byte Select Control Signals, 9-5
$\overline{\text{CDIS}}$, 3-7, 4-3
CLK, 3-7
D31–D0, 3-2, 5-3
$\overline{\text{DBEN}}$, 3-5, 5-4
$\overline{\text{DS}}$, 3-4, 5-4, 5-21
$\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$, 3-5, 5-4, 5-5, 5-24, 5-46, 5-53, 9-5
$\overline{\text{ECS}}$, 3-4, 5-3
FC2–FC0, 2-4, 3-2, 5-2, 5-3, 5-44, 7-6
Functional Groups, 3-1
$\overline{\text{HALT}}$, 3-7, 5-4, 5-25, 5-53, 5-60
Input Signal, 5-2
Internal Signal, 5-2
$\overline{\text{IPEND}}$, 3-5, 6-14
$\overline{\text{IPL2}}$–$\overline{\text{IPL0}}$, 3-5, 6-11
$\overline{\text{OCS}}$, 3-4, 5-3
$\overline{\text{RESET}}$, 3-6, 5-76, 6-4
$\overline{\text{RMC}}$, 3-4, 5-3, 5-39
R/$\overline{\text{W}}$, 3-4, 5-2, 5-3, 9-5
SIZ1, SIZ0, 3-2, 5-2, 5-3, 5-7, 5-9, 5-21, 9-5
Single-Operand Instruction, 8-33
SIZ1, SIZ0 Signals, 3-2, 5-2, 5-3, 5-7, 5-9, 5-21, 9-5
Source Function Code Register (SFC), 1-7
Special-Purpose MOVE Instruction, 8-29
Special Status Word (SSW), 6-21
Spurious Interrupt, 5-48

I

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Stack Frame
Midinstruction, 7-47
Postinstruction, 7-48
Preinstruction, 7-46
Status Register (SR), 1-7, 4-1, 5-45, 6-1
STOP Instruction, 6-10
Supervisor Privilege Level, 1-4, 2-2
Supervisor Stack Pointer (SSP), 1-4, 2-2
Synchronous Cycles, 5-24

— T —

T1, T0 Bits (SR), 1-7, 6-9
TAS Instruction, 5-39
Thermal Characteristics, 10-1
MC68020, 10-2
MC68020 CQFP Package, 10-2
MC68EC020, 10-4
MC68EC020 PQFP Package, 10-4
Thermal Resistance, 10-2, 10-4
Timing, 5-26, 5-33
Trace Exception, 6-9
Trace Modes, 1-7
Tracing, 6-9
Transfer, 5-10, 5-14, 5-25
Bus Transfer, 5-1
Direction, 5-3
Misaligned, 5-1, 5-5
Operand Transfer, 5-1, 5-5
Trap Exception, 6-6

— U —

Unimplemented Instruction (F-Line Opcode) Exception, 6-7
User Privilege Level, 1-4, 2-2
User Stack Pointer (USP), 1-4, 2-2

— V —

V_{CC} Connections, 3-7, 9-9
Vector Base Register (VBR), 1-7, 2-5, 6-2
Virtual Machine, 1-12
Virtual Memory, 1-10

— W —

Write Cycle, 5-3, 5-9, 5-10, 5-12, 5-14, 5-16, 5-18, 5-22, 5-33, 5-38
Long-Word Write Cycle, 5-33
Timing, 5-33

Freescale Semiconductor, Inc.

I

For More Information On This Product,
Go to: www.freescale.com

Introduction 1

Processing States 2

Signal Description 3

On-Chip Cache Memory 4

Bus Operation 5

Exception Processing 6

Coprocessor Interface Description 7

Instruction Execution Timing 8

Applications Information 9

Electrical Characteristics 10

Ordering Information and Mechanical Data 11

Appendix A A

Index I

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

1 Introduction

2 Processing States

3 Signal Description

4 On-Chip Cache Memory

5 Bus Operation

6 Exception Processing

7 Coprocessor Interface Description

8 Instruction Execution Timing

9 Applications Information

10 Electrical Characteristics

11 Ordering Information and Mechanical Data

A Appendix A

I Index

1ATX31073.4 Printed in USA 1/98 BANTA CO. MOTO #133-4.000 LITMPUHP

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

MOTOROLA

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

Customer Focus Center: 1–800–521–6274

Mfax™: RMFAX0@email.sps.mot.com – TOUCHTONE 1–602–244–6609
Motorola Fax Back System – US & Canada ONLY 1–800–774–1848
– http://sps.motorola.com/mfax/

HOME PAGE: http://motorola.com/sps/

JAPAN: Nippon Motorola Ltd.: SPD, Strategic Planning Office, 4–32–1,
Nishi–Gotanda, Shinagawa–ku, Tokyo 141, Japan. 81–3–5487–8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,
51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852–26629298

Mfax is a trademark of Motorola, Inc.