

A WORD ABOUT TRADEMARKS . . .

The following products referred to in this manual are AT&T registered trademarks:

WE® 32106 Math Acceleration Unit, MAU

WE® 32200 Microprocessor, CPU

WE® 32201 Memory Management Unit, MMU

WE® 32206 Math Acceleration Unit, MAU

UNIX® Microsystem, Operating System

AT&T reserves the right to make changes to the product(s), including any hardware, software, and/or firmware contained therein, described herein without notice. No liability is assumed as a result of the use or application of this product(s). No rights under any patent accompany the sale of any such product(s).

Copyright© 1987 AT & T. All Rights Reserved.

Printed in United States of America



February 1987

***WE*[®] 32206 Math Acceleration Unit
Information Manual**

PRELIMINARY

Select Code 307-707

COMCODE 105311823

ACKNOWLEDGEMENTS

Published by
The AT&T Documentation Management Organization

for the

Microsystems Product Management Organization
AT&T Information Systems

and the

62401343 Microsystems Systems Peripheral Development Department
AT&T Information Systems, Holmdel

FOREWORD

This manual contains information on the *WE 32206 Math Acceleration Unit* that is essential to computer designers, software architects, and system design engineers.

Additional information is available in the form of data sheets, application notes, and on-line documentation from the *UNIX Operating System*.

For details concerning warranty or replacement, or to obtain additional information, contact your AT&T Account Manager or call:

1-800-372-2447

To obtain additional copies of this manual, Select Code 307-707, call:

□ **1-800-432-6600**

WE 32206 MATH ACCELERATION UNIT
INFORMATION MANUAL

CONTENTS

CHAPTER 1. INTRODUCTION

1. INTRODUCTION	1-1
1.1 CONVENTIONS	1-1
1.2 ARCHITECTURAL SUMMARY	1-2
1.3 PROTOCOL MODES	1-2
1.3.1 Coprocessor Mode	1-2
1.3.2 Peripheral Mode	1-3
1.4 DATA TYPES AND FORMATS	1-3
1.5 IEEE COMPATIBILITY	1-4
1.6 CHAPTER DESCRIPTIONS	1-4
1.6.1 Functional Descriptions	1-4
1.6.2 MAU Signal Descriptions	1-4
1.6.3 Bus Transactions	1-4
1.6.4 Instruction Set	1-5

CHAPTER 2. FUNCTIONAL DESCRIPTIONS

2. FUNCTIONAL DESCRIPTIONS	2-1
2.1 DATA TYPES AND FORMATS	2-1
2.1.1 Single Precision	2-1
Normalized Numbers	2-1
Denormalized Numbers	2-2
Special Case Values	2-2
2.1.2 Double Precision	2-3
Normalized Numbers	2-4
Denormalized Numbers	2-4
Special Case Values	2-5
2.1.3 Double-Extended Precision	2-6
Normalized Numbers	2-6
Unnormalized and Denormalized Numbers	2-6
Special Case Values	2-7
2.1.4 Decimal	2-8
Numbers	2-9
Special Case Values	2-9
2.1.5 Integer	2-10
2.2 REGISTERS	2-10
2.2.1 Auxiliary Status Register	2-10
2.2.2 Operand Registers	2-14

2.2.3 Command Register	2-14
2.2.4 Data Register	2-17
2.3 ROUNDING	2-18
2.4 EXCEPTIONS	2-19
2.4.1 Exception Types	2-19
Invalid Operation	2-19
Divide-By-Zero	2-19
Overflow	2-20
Underflow	2-21
Inexact	2-22
2.5 CONTEXT SAVING AND RESTORING	2-22
2.5.1 Peripheral Mode	2-22
2.5.2 Coprocessor Mode	2-23
2.6 INTERRUPTS	2-23
2.6.1 Peripheral Mode Protocol	2-23
2.6.2 Coprocessor Mode Protocol	2-24
2.7 MAU RESET	2-24
2.8 QUIESCENT STATE	2-24
2.9 MAU-CPU INTERCONNECTIONS	2-25
2.9.1 Peripheral Mode	2-25
2.9.2 Coprocessor Mode	2-26

CHAPTER 3. SIGNAL DESCRIPTIONS

3. SIGNAL DESCRIPTIONS	3-1
3.1 ADDRESS AND DATA SIGNALS	3-1
Address (ADDR00—ADDR04)	3-1
Data (DATA00—DATA31)	3-1
3.2 INTERFACE AND CONTROL SIGNALS	3-2
Chip Select (\overline{CS})	3-2
Cycle Initiate (\overline{CYCLEI})	3-2
Data Bus Shadow (\overline{DSHAD})	3-2
Data Ready (\overline{DRDY})	3-2
Data Strobe (\overline{DS})	3-2
Data Transfer Acknowledge (\overline{DTACK})	3-2
Done (\overline{DONE})	3-2
Peripheral Interrupt Acknowledge (\overline{PIACK})	3-2
Peripheral Interrupt Request (\overline{PIREQ})	3-2
Synchronous Ready (\overline{SRDY})	3-3
3.3 STATUS SIGNALS	3-3
Access Status Codes (SAS0—SAS3)	3-3
Data Size (DSIZE0—DSIZE2)	3-3
Read/Write (R/ \overline{W})	3-3
3.4 BUS EXCEPTION SIGNALS	3-4
Fault (\overline{FAULT})	3-4
Reset (\overline{RESET})	3-4

3.5 CLOCKS	3-4
Input Clock (CLK23)	3-4
Input Clock (CLK34)	3-4
3.6 TEST SIGNAL ($\overline{\text{HIGHZ}}$)	3-4
3.7 PIN ASSIGNMENTS	3-4

CHAPTER 4. BUS TRANSACTIONS

4. BUS TRANSACTIONS	4-1
4.1 PERIPHERAL MODE TRANSACTIONS	4-1
4.1.1 Peripheral Mode Accesses	4-1
4.1.2 Bus Transaction Sequence	4-2
Transaction Sequence	4-3
4.1.3 Peripheral Mode Read and Write	4-4
4.1.4 Configuration Restrictions	4-4
4.1.5 Peripheral Mode Interrupts	4-9
4.1.6 Peripheral Mode Faults	4-9
4.1.7 Peripheral Mode Restartability	4-10
4.1.8 Peripheral Mode Operating System Implications	4-10
4.2 COPROCESSOR MODE TRANSACTIONS	4-11
4.2.1 Bus Transaction Sequence	4-11
Transaction Sequence	4-12
4.2.2 Coprocessor Broadcast Transaction	4-13
4.2.3 Coprocessor Data Fetch Transaction	4-17
4.2.4 Coprocessor Status Read Transaction	4-21
4.2.5 Coprocessor Data Write Transaction	4-24
4.2.6 Coprocessor Faults	4-30
Status Fetch Faults	4-30
Non-Existent MAU Faults	4-31
4.2.7 Coprocessor Restartability	4-31
4.2.8 Restrictions on System Configuration	4-32
4.2.9 Coprocessor Operating System Implications	4-33

CHAPTER 5. INSTRUCTION SET

5. INSTRUCTION SET	5-1
5.1 FUNCTIONAL GROUPS	5-1
5.1.1 Arithmetic Instructions	5-2
5.1.2 Compare Instructions	5-2
5.1.3 Data Transfer Instructions	5-2
5.1.4 Conversion Instructions	5-3
5.1.5 Transcendental Instructions	5-3
5.1.6 Miscellaneous Instructions	5-3
5.2 INSTRUCTION SET LISTINGS	5-3
5.2.1 Notation	5-4
5.2.2 Instruction Set Descriptions	5-4
Absolute Value (ABS)	5-7
Add (ADD)	5-8

Arctangent (ATAN)	5-9
Compare (CMP)	5-11
Compare With Exceptions (CMPE)	5-12
Compare With Exceptions and Flags Swapped (CMPES)	5-13
Compare With Flags Swapped (CMPS)	5-14
Cosine (COS)	5-15
Divide (DIV)	5-17
Convert Decimal to Float (DFOB)	5-18
Extract Result on Fault (EROF)	5-19
Convert Float to Decimal (FTOD)	5-20
Convert Float to Integer (FTOI)	5-22
Convert Integer to Float (ITOF)	5-24
Load Data Register (LDR)	5-25
Move (MOVE)	5-26
Multiply (MUL)	5-28
Negate (NEG)	5-29
No Operation (NOP)	5-30
Pi (PI)	5-31
Move From ASR (RDASR)	5-32
Remainder (REM)	5-33
Round to Integral Value (RTOI)	5-35
Sine (SIN)	5-37
Square Root (SQRT)	5-39
Subtract (SUB)	5-40
Move to ASR (WRASR)	5-42
5.2.3 Instruction Set Summary by Mnemonic	5-43
5.2.4 Instruction Set Summary by Opcode	5-46

GLOSSARY AND ACRONYMS

INDEX

LIST OF FIGURES

Figure 1-1. WE 32206 Math Acceleration Unit Block Diagram	1-2
Figure 2-1. Integer Format	2-10
Figure 2-2. MAU-CPU Peripheral Mode Protocol Interconnections	2-25
Figure 2-3. MAU-CPU Coprocessor Mode Protocol Interconnections	2-26
Figure 3-1. Input and Output Signals	3-1
Figure 3-2. 125-Pin Square Ceramic Pin Grid Array - Bottom View	3-5
Figure 4-1. Example of Peripheral Mode Transaction Sequence	4-2
Figure 4-2. Peripheral Mode Read	4-5
Figure 4-3. Peripheral Mode Write	4-6
Figure 4-4. Peripheral Mode Read and Write	4-7

Figure 4-5. $\overline{\text{PIREQ}}$ and $\overline{\text{PIACK}}$ Timing	4-8
Figure 4-6. Example of Peripheral Mode Exception Sequence	4-9
Figure 4-7. Example of Coprocessor Mode Transaction Sequence	4-12
Figure 4-8. Coprocessor Broadcast (1 Wait State)	4-14
Figure 4-9. Coprocessor Broadcast (2 Wait States)	4-15
Figure 4-10. Coprocessor Broadcast with Bus Exceptions (1 Wait State)	4-16
Figure 4-11. Coprocessor Data Fetch (1 Wait State)	4-17
Figure 4-12. Coprocessor Data Fetch (0 Wait States)	4-18
Figure 4-13. Coprocessor Data Fetch ($R/\overline{W} = 0$)	4-19
Figure 4-14. Coprocessor Data Fetch with Bus Exceptions (0 Wait States)	4-20
Figure 4-15. Coprocessor Data Fetch with Nonword Data (0 Wait States)	4-21
Figure 4-16. $\overline{\text{DONE}}$ Assertion and Coprocessor Status Read (No Exceptions)	4-22
Figure 4-17. Coprocessor Status Read (With Exceptions)	4-23
Figure 4-18. Coprocessor Data Write (1 Wait State)	4-24
Figure 4-19. Coprocessor Data Write (0 Wait States)	4-25
Figure 4-20. Coprocessor Data Write ($R/\overline{W} = 1$)	4-26
Figure 4-21. Coprocessor Data Write with Bus Exceptions (0 Wait States)	4-27
Figure 4-22. Coprocessor Data Write with Nonword Data	4-28
Figure 4-23. Coprocessor Data Write ($\overline{\text{DSHAD}}$ Assertion)	4-29
Figure 4-24. Faulted Coprocessor Status Fetch Transaction	4-31

LIST OF TABLES

Table 2-1. Single Precision Format	2-1
Table 2-2. Single Precision Special Case Values	2-3
Table 2-3. Double Precision Format	2-4
Table 2-4. Double Precision Special Case Values	2-5
Table 2-5. Double-Extended Precision Format	2-6
Table 2-6. Double-Extended Precision Format Special Case Values	2-7
Table 2-7. Decimal Format	2-8
Table 2-8. Decimal Format Special Case Values	2-9
Table 2-9. Auxiliary Status Register	2-11
Table 2-10. Operand Registers	2-14
Table 2-11. Command Register	2-15
Table 2-12. Data Register Exception Handler Information	2-17
Table 3-1. Access Codes	3-3
Table 3-2. Data Size Codes	3-3
Table 3-3. <i>WE</i> 32206 MAU Pin Descriptions	3-6
Table 4-1. Peripheral Mode Address Fields	4-1
Table 5-1. Arithmetic Instruction Group	5-2
Table 5-2. Compare Instruction Group	5-2
Table 5-3. Data Transfer Instruction Group	5-2
Table 5-4. Conversion Instruction Group	5-3

Table 5-5. Transcendental Instruction Group	5-3
Table 5-6. Miscellaneous Instruction Group	5-3
Table 5-7. Notation For Special Cases	5-5
Table 5-8. Instruction Set Summary by Mnemonic	5-43
Table 5-9. Instruction Set Summary by Opcode	5-46

Chapter 1
Introduction

CHAPTER 1. INTRODUCTION

CONTENTS

1. INTRODUCTION	1-1
1.1 CONVENTIONS	1-1
1.2 ARCHITECTURAL SUMMARY	1-2
1.3 PROTOCOL MODES	1-2
1.3.1 Coprocessor Mode	1-2
1.3.2 Peripheral Mode	1-3
1.4 DATA TYPES AND FORMATS	1-3
1.5 IEEE COMPATIBILITY	1-4
1.6 CHAPTER DESCRIPTIONS	1-4
1.6.1 Functional Descriptions	1-4
1.6.2 MAU Signal Descriptions	1-4
1.6.3 Bus Transactions	1-4
1.6.4 Instruction Set	1-5

1. INTRODUCTION

The *WE* 32206 Math Acceleration Unit (MAU) provides floating-point coprocessing capability for the *WE* 32200 Microprocessor (CPU) and is fully compatible with the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985). The MAU's coprocessing capability is available in two modes: coprocessor and peripheral. Coprocessor mode is used when the MAU is integrated with the AT&T *UNIX* Microsystem products (e.g., *WE* 32200 CPU). Peripheral mode provides a more general interface between the MAU and other commercial microprocessors.

The MAU supports single (32-bit), double (64-bit), and double-extended (80-bit) precision floating-point values. It can perform add, subtract, multiply, divide, remainder, square root, and compare operations on any of these data formats. The MAU can perform sine, cosine, and arctangent on single- and double-precision data formats. It also supports conversions to and from integer and decimal data formats. The operand, result, and status and command information transfers take place over a 32-bit bidirectional data bus that provides the interface to the host microprocessor.

The MAU is implemented in CMOS technology and is available in a 125-pin square, ceramic pin grid array (PGA) package. It is available in 24 MHz and higher frequency versions and requires a single 5 V supply.

1.1 CONVENTIONS

Where necessary in this document, formulas are written in C-language notation, with the addition of the construct **(**)** to represent exponentiation.

A pseudo-C-language code is used to define some algorithms. Thus, the construct **(==)** means "equal to" and the construct **(=)** means "assign to." The *X* field of element *Y* is referenced as *Y<X>*.

Fields named *Unused* appear in various places. The behavior and uses of these fields are specified for testing purposes only. They are reserved for future use and should not be used by current software or hardware. When written, they should be written with zeroes.

The word *set* is used when "given the value 1" is meant. The word *cleared* is used when "given the value 0" is meant.

The word *asserted* is used when a signal is "active" (e.g., low for an active low signal, high for an active high signal). The word *negated* is used when a signal is "inactive."

The Glossary should be consulted for the definitions of any unfamiliar terms used in this manual.

INTRODUCTION

Architectural Summary

1.2 ARCHITECTURAL SUMMARY

The MAU has two major subsystems: the fraction data path and the exponent data path. The data width of each data path was chosen to facilitate calculations in the widest data format. All operands are converted, upon entry to the MAU from memory, to the double-extended-precision format. This causes no loss of precision and saves internal hardware that would be necessary to explicitly support the narrower formats in all internal operations. Figure 1-1 contains a block diagram of the *WE 32206* Math Acceleration Unit.

1.3 PROTOCOL MODES

The MAU interacts with a system in either coprocessor or peripheral mode protocol. The coprocessor mode protocol allows high system throughput when the MAU is used with the *WE 32200* Microprocessor. The peripheral mode protocol allows use of the MAU with any CPU that can perform data read and write bus transactions.

1.3.1 Coprocessor Mode

In coprocessor mode protocol, the CPU initiates an MAU transaction by performing a coprocessor broadcast access. This access includes a word in the MAU command register (CR) format (see section 2.2). The MAU checks the identification field of this word against the MAU's identification value. If they match, the word is stored in one of the MAU's internal registers.

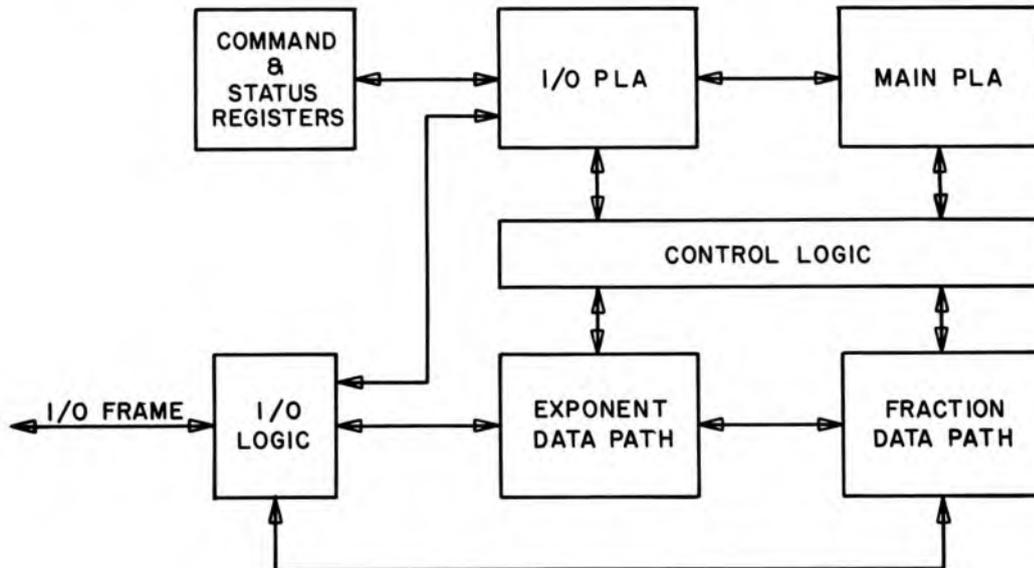


Figure 1-1. *WE 32206* Math Acceleration Unit Block Diagram

If any of the arguments passed along with the command word indicate that an operand is to be obtained from memory, the MAU waits until the proper number of coprocessor data fetch bus transactions occur. The words are stored by the MAU in internal registers.

The MAU performs the operation, generating a result, new condition codes, and possibly an exception. The MAU signals the CPU that it is done with its operation and waits until a processor status fetch access is seen. If an exception is present, the MAU indicates an exception and returns to the quiescent state. If there is no exception, the MAU returns the status of the operation to the CPU.

If the result is to be placed in memory, the MAU waits until the proper number of coprocessor data write bus transactions occur, putting a word of the result on the bus as each transaction occurs. The MAU then returns to the quiescent state.

1.3.2 Peripheral Mode

In peripheral mode protocol, the MAU registers appear as memory-mapped locations and are accessed via normal read and write operations. The CPU (or any other bus master) starts an MAU transaction by writing a word into the MAU's command register. When the write access is completed, the MAU clears the result available (RA) bit in the auxiliary status register (ASR). The command is then executed by using the operands available in registers F0 through F7 or by reading values via the data register (DR).

If any of the operands are to come from memory, the bus master must write the operands into the DR one word at a time after the CR has been written. These operands are latched into internal registers. If the result is to be placed in memory, the bus master must read the result from the DR one word at a time and transfer the words to memory.

After the operation has been completed, the MAU updates the ASR contents to reflect the new condition codes, exception bits, and any other bits resulting from the operation. The RA bit in the ASR is set, peripheral interrupt request (PIREQ) is asserted, and the MAU enters the quiescent state.

1.4 DATA TYPES AND FORMATS

The WE 32206 MAU supports IEEE single, double, double-extended, decimal, and integer data formats. These are:

- Single-precision format with an 8-bit exponent and a 23-bit fraction.
- Double-precision format with an 11-bit exponent and a 52-bit fraction.
- Double-extended-precision format with a 15-bit exponent and a 64-bit fraction.
- Decimal format with 18 binary coded decimal (BCD) digits and a sign nibble.
- Integer format, which provides a binary representation of signed integers in 32-bit 2's complement format.

1.5 IEEE COMPATIBILITY

The *WE 32206* MAU supports all requirements of the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985), as well as some optional and additional features.

The MAU supports the required single-precision and the optional double-precision and double-extended-precision data formats. The optional single-extended-precision format is not supported since the double-precision format fulfills all of the single-extended requirements.

All required operations, rounding modes, and exception types are also supported. The MAU's rounding scheme provides accuracy of the computed result to within half of one unit in the least significant bit (LSB) position. Masking and sticky bits are provided for the invalid-operation, underflow, overflow, divide-by-zero, and inexact exceptions.

The MAU operates on all operands internally in double-extended-precision format. The internal operations are performed as if the intermediate results are computed to infinite precision. The last intermediate result is converted to fit the destination format and then rounded before being stored in the destination.

1.6 CHAPTER DESCRIPTIONS

This section provides a summary of the remaining chapters in this manual. For detailed electrical, timing, and physical characteristics, refer to the data sheet for the *WE 32206* MAU.

1.6.1 Functional Descriptions

Chapter 2 provides a description of the hardware associated with the *WE 32206* MAU, beginning with a thorough description of data types and formats. Also included are information about the MAU's internal registers and a discussion of rounding, exceptions, context saving and restoring, and interrupts. The chapter concludes with a discussion of the MAU's quiescent state and its reset function.

1.6.2 MAU Signal Descriptions

Chapter 3 describes the MAU's input and output signals, arranged in the functional groups of address and data, interface and control, status, bus exception, clocks, and test signals.

1.6.3 Bus Transactions

Chapter 4 describes the MAU-related bus transactions. Both peripheral mode and coprocessor mode transactions are discussed. Protocol diagrams, showing the operation of the MAU, are also included in this chapter.

1.6.4 Instruction Set

Chapter 5 describes, in detail, the instructions that the MAU is capable of performing. The instructions are divided functionally and include arithmetic, logical, data transfer, conversion, transcendental, and miscellaneous groups. Information on special case handling, condition flags, and exceptions is also provided.

Chapter 2
Functional Descriptions

CHAPTER 2. FUNCTIONAL DESCRIPTIONS

CONTENTS

2. FUNCTIONAL DESCRIPTIONS	2-1
2.1 DATA TYPES AND FORMATS	2-1
2.1.1 Single Precision	2-1
Normalized Numbers	2-1
Denormalized Numbers	2-2
Special Case Values	2-2
2.1.2 Double Precision	2-3
Normalized Numbers	2-4
Denormalized Numbers	2-4
Special Case Values	2-5
2.1.3 Double-Extended Precision	2-6
Normalized Numbers	2-6
Unnormalized and Denormalized Numbers	2-6
Special Case Values	2-7
2.1.4 Decimal	2-8
Numbers	2-9
Special Case Values	2-9
2.1.5 Integer	2-10
2.2 REGISTERS	2-10
2.2.1 Auxiliary Status Register	2-10
2.2.2 Operand Registers	2-14
2.2.3 Command Register	2-14
2.2.4 Data Register	2-17
2.3 ROUNDING	2-18
2.4 EXCEPTIONS	2-19
2.4.1 Exception Types	2-19
Invalid Operation	2-19
Divide-By-Zero	2-19
Overflow	2-20
Underflow	2-21
Inexact	2-22
2.5 CONTEXT SAVING AND RESTORING	2-22
2.5.1 Peripheral Mode	2-22
2.5.2 Coprocessor Mode	2-23
2.6 INTERRUPTS	2-23
2.6.1 Peripheral Mode Protocol	2-23
2.6.2 Coprocessor Mode Protocol	2-24
2.7 MAU RESET	2-24
2.8 QUIESCENT STATE	2-24
2.9 MAU-CPU INTERCONNECTIONS	2-25
2.9.1 Peripheral Mode	2-25
2.9.2 Coprocessor Mode	2-26

2. FUNCTIONAL DESCRIPTIONS

This chapter provides functional descriptions of the MAU, including data types, registers, exceptions, and context saving and restoring. Also included are discussions on interrupts, reset, the MAU's quiescent state, and CPU-MAU interconnections.

2.1 DATA TYPES AND FORMATS

The *WE 32206* Math Acceleration Unit supports single, double, double-extended, decimal, and integer data formats. The following sections describe the data types and formats.

2.1.1 Single Precision

Single-precision operands are 32 bits long. The single-precision format is shown and described in Table 2-1.

Table 2-1. Single-Precision Format					
Bit	31	30	23	22	0
Field	Sign		Exponent		Fraction
Bit	Field	Contents	Description		
0–22	Fraction	Fraction	A 23-bit string that encodes the significant bits of the number. For normalized numbers, an implicit bit that has a value of 1 resides immediately to the left of the binary point. The implicit bit and fraction can represent values in the range 1 to $(2-2^{-23})$, inclusive.		
23–30	Exponent	Exponent	These 8 bits represent an exponent biased by 127.		
31	Sign	Sign Bit	A 1 represents a negative value; 0 represents a positive value.		

Normalized Numbers

A number is normalized if the exponent field contains some value other than all 1s or all 0s.

The exponent field contains an exponent biased by 127. Thus the exponent of a normalized single-precision number is in the range -126 through 127.

FUNCTIONAL DESCRIPTIONS

Denormalized Numbers

There is an implicit bit associated with this format. The implicit bit is not explicitly stored anywhere (hence its name). Logically, for normalized operands, the implicit bit has a value of 1 and resides immediately to the left of the binary point, which lies immediately to the left of fraction bit 22. Thus, the implicit bit and fraction field together can represent values in the range 1 through $(2-2^{-23})$ and normalized single-precision numbers can be in the range $\pm 2^{-126}$ through $(2-2^{-23}) * 2^{127}$.

Denormalized Numbers

A number is denormalized if the exponent field contains all 0s and the fraction field does not contain all 0s. In this case, the implicit bit is 0 and denormalized single-precision numbers can be in the range $\pm 2^{-126} * 2^{-23}$ through $2^{-126} * (1-2^{-23})$.

Special Case Values

The algorithm for defining a special case value for the single-precision data format is as follows:

```
If (exponent == max)
    If (fraction == min)
        Then the number is infinity (positive or negative as determined by
        the sign bit).
    Else the number is NaN (trapping if the fraction MSB == 0; nontrapping
    if the fraction MSB == 1).
Else if (exponent == min)
    If (fraction == min)
        Then the number is zero (positive or negative as determined by the
        sign bit).
    Else the number is denormalized.
Else the number is normalized.
```

Table 2-2 gives the names of special cases and how each is represented.

Table 2-2. Single-Precision Special Case Values				
Value Name	Sign Value	Exponent Value	Fraction Value	
			Bit 22	Bits 21–0
Nontrapping NaN	x	max	1	x
Trapping NaN	x	max	0	nonzero
Positive Infinity	0	max	min	
Negative Infinity	1	max	min	
Positive Zero	0	min	min	
Negative Zero	1	min	min	
Denormalized Number	x	min	nonzero	
Normalized Number	x	notmm	x	

Legend:

x = don't care

max = the maximum value that can be stored in the field (all 1s)

min = the minimum value that can be stored in the field (all 0s)

NaN = not a number

notmm = the field is not equal to either the min or max values
(is not all 0s or all 1s)

nonzero = the field contains at least one "1" bit

When the MAU generates a nontrapping NaN, the fraction contains all 1s. The MAU never generates a trapping NaN.

2.1.2 Double Precision

The double-precision operands are 64 bits long. The double-precision format is presented in Table 2-3.

FUNCTIONAL DESCRIPTIONS
Normalized Numbers

Table 2-3. Double-Precision Format						
	Bit	63	62	52	51	0
	Field	Sign	Exponent		Fraction	
Bit	Field	Contents	Description			
0–51	Fraction	Fraction	A 52-bit string that encodes the significant bits of the number. For normalized numbers, an implicit bit that has a value of 1 resides immediately to the left of the binary point, which lies immediately to the left of fraction bit 51.			
52–62	Exponent	Exponent	These 11 bits represent an exponent biased by 1023.			
63	Sign	Sign Bit	A 1 represents a negative value; a 0 represents a positive value.			

Normalized Numbers

A number is normalized if the exponent field contains some value other than all 1s or all 0s.

The exponent field contains an exponent biased to 1023. Thus the exponent of a normalized double-precision number is in the range -1022 through 1023 .

There is an implicit bit associated with this format. The implicit bit and fraction field together can represent values in the range 1 through $(2-2^{-52})$. Thus, normalized double-precision numbers can be in the range $\pm 2^{-1022}$ through $(2-2^{-52}) * 2^{1023}$.

Denormalized Numbers

A number is denormalized if the exponent field contains all 0s and the fraction field does not contain all 0s. In this case, the implicit bit is 0 and denormalized double-precision numbers can be in the range $2^{-1022} * 2^{-52}$ through $2^{-1022} * (1-2^{-52})$.

Special Case Values

The algorithm for defining a special case value for the double-precision data format is as follows:

```

If (exponent == max)
  If (fraction == min)
    Then the number is infinity (positive or negative as determined by
    the sign bit).
  Else the number is NaN (trapping if the fraction MSB == 0; nontrapping
  if the fraction MSB == 1).
Else if (exponent == min)
  If (fraction == min)
    Then the number is zero (positive or negative as determined by the
    sign bit).
  Else the number is denormalized.
Else the number is normalized.
  
```

Table 2-4 gives the names of special cases and how each is represented.

Table 2-4. Double-Precision Special Case Values				
Value Name	Sign Value	Exponent Value	Fraction Value	
			Bit 51	Bits 50–0
Nontrapping NaN	x	max	1	x
Trapping NaN	x	max	0	nonzero
Positive Infinity	0	max	min	
Negative Infinity	1	max	min	
Positive Zero	0	min	min	
Negative Zero	1	min	min	
Denormalized Number	x	min	nonzero	
Normalized Number	x	notmm	x	

Legend:

- x = don't care
- max = the maximum value that can be stored in the field (all 1s)
- min = the minimum value that can be stored in the field (all 0s)
- NaN = not a number
- notmm = the field is not equal to either the min or max values
(is not all 0s or all 1s)
- nonzero = the field contains at least one "1" bit

When the MAU generates a nontrapping NaN, the fraction contains all 1s. The MAU never generates a trapping NaN.

FUNCTIONAL DESCRIPTIONS

Double-Extended Precision

2.1.3 Double-Extended Precision

Double-extended-precision operands are 80 bits long. Table 2-5 describes the double-extended-precision format.

Table 2-5. Double-Extended Precision Format								
Bit	95	80	79	78	64	63	62	0
Field	—		Sign	Exponent		J	Fraction	
Bit	Field	Contents	Description					
0—62	Fraction	Fraction	A 63-bit string that encodes the significant bits of the number.					
63	J	Explicit Bit	This bit resides immediately to the left of the binary point.					
64—78	Exponent	Exponent	These 15 bits represent an exponent biased by 16383.					
79	Sign	Sign Bit	A 1 represents a negative value; a 0 represents a positive value.					
80—95	—	Unused	These bits are ignored and overwritten with 0s.					

Normalized Numbers

A number is normalized if the exponent field contains some value other than all 1s and the J (explicit) bit is equal to 1.

The exponent field contains an exponent biased by 16383. Thus the exponent of a normalized double-extended-precision number is in the range -16382 through 16383 .

There is an explicit (J) bit associated with this format. The J bit resides to the left of the binary point (in the 2^0 position). Therefore, the J bit and the fraction field together can represent values in the range 0 through $(2-2^{-63})$ and normalized double-extended-precision numbers can be in the range $\pm 2^{-16382}$ through $(2-2^{-63}) * 2^{16383}$.

Unnormalized and Denormalized Numbers

A number is unnormalized if the exponent field contains some value other than all 1s or all 0s and the J bit is equal to 0. A number is denormalized if the exponent is 0, $J = 0$, and the fraction is nonzero.

Denormalized, double-extended-precision numbers can be in the range $\pm 2^{-63} * 2^{-16382}$ through $(1-2^{-63}) * 2^{-16383}$.

Special Case Values

The algorithm for defining a special case value for the double-extended-precision data format is as follows:

```

If (exponent == max)
  If (J+fraction == min)
    Then the number is infinity (positive or negative as determined by
    the sign bit).
  Else the number is NaN (trapping if the digit to the right of the binary
  point in the significand is zero, and nontrapping if this bit is one).
Else if (exponent == min)
  If (J+fraction == min)
    Then the number is zero (positive or negative as determined by the
    sign bit).
  Else if (J == 0) and fraction == 0 then the number is denormalized,
  else the number is normalized.
Else if (J == 0)
  Then the number is unnormalized.
Else the number is normalized.
  
```

Table 2-6 shows the names of special cases and how each is represented.

Value Name	Sign Value	Exponent Value	J Value	Fraction Value	
				Bit 62	Bits 0—61
Nontrapping NaN	x	max	x	1	x
Trapping NaN	x	max	x	0	nonzero
Positive Infinity	0	max	0	min	
Negative Infinity	1	max	0	min	
Positive Zero	0	min	0	min	
Negative Zero	1	min	0	min	
Denormalized Number	x	min	0	nonzero	
Unnormalized Number	x	notmm	0	x	
Normalized Number	x	notmax	1	x	

Legend:

- x = don't care
- max = the maximum value that can be stored in the field (all 1s)
- min = the minimum value that can be stored in the field (all 0s)
- NaN = not a number
- notmm = the field is not equal to either the min or max values
(is not all 0s or all 1s)
- nonzero = the field contains at least one "1" bit
- notmax = the field is not equal to the max value (is not all 1s)

FUNCTIONAL DESCRIPTIONS

Decimal

When the MAU generates a nontrapping NaN, J and the fraction contain all 1s. The MAU never generates a trapping NaN.

2.1.4 Decimal

Decimal operands are 76 bits long. Table 2-7 shows the format for a decimal operand and describes each bit position. The decimal format supported by the MAU is a fixed-point decimal format. When performing conversions to and from floating-point decimal format, software support is needed to provide the range required by the IEEE standard for these operations.

Table 2-7. Decimal Format

Bit	Field	Contents	Description							
95 76	75 72	71 68	67 64	63 60	59 56	55 52	51 48	47 44	43 40	
Field	—	D17	D16	D15	D14	D13	D12	D11	D10	D9
Bit	39 36	35 32	31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
Field	D8	D7	D6	D5	D4	D3	D2	D1	D0	Sign

*D stands for digit

Bit	Field	Contents	Description																																								
0—3	Sign	Sign Bits	<p>These 4 bits represent the sign of the number. They are also used to represent NaNs and infinity. The sign values have the following meanings:</p> <table border="1"> <thead> <tr> <th>Bits</th> <th>Meaning</th> <th>Bits</th> <th>Meaning</th> </tr> <tr> <th>3,2,1,0</th> <th></th> <th>3,2,1,0</th> <th></th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>Positive infinity</td> <td>1000</td> <td>Trapping NaN</td> </tr> <tr> <td>0001</td> <td>Negative infinity</td> <td>1001</td> <td>Trapping NaN</td> </tr> <tr> <td>0010</td> <td>Positive NaN</td> <td>1010</td> <td>Positive number</td> </tr> <tr> <td>0011</td> <td>Negative NaN</td> <td>1011</td> <td>Negative number</td> </tr> <tr> <td>0100</td> <td>Trapping NaN</td> <td>1100</td> <td>Positive number</td> </tr> <tr> <td>0101</td> <td>Trapping NaN</td> <td>1101</td> <td>Negative number</td> </tr> <tr> <td>0110</td> <td>Trapping NaN</td> <td>1110</td> <td>Positive number</td> </tr> <tr> <td>0111</td> <td>Trapping NaN</td> <td>1111</td> <td>Positive number</td> </tr> </tbody> </table>	Bits	Meaning	Bits	Meaning	3,2,1,0		3,2,1,0		0000	Positive infinity	1000	Trapping NaN	0001	Negative infinity	1001	Trapping NaN	0010	Positive NaN	1010	Positive number	0011	Negative NaN	1011	Negative number	0100	Trapping NaN	1100	Positive number	0101	Trapping NaN	1101	Negative number	0110	Trapping NaN	1110	Positive number	0111	Trapping NaN	1111	Positive number
Bits	Meaning	Bits	Meaning																																								
3,2,1,0		3,2,1,0																																									
0000	Positive infinity	1000	Trapping NaN																																								
0001	Negative infinity	1001	Trapping NaN																																								
0010	Positive NaN	1010	Positive number																																								
0011	Negative NaN	1011	Negative number																																								
0100	Trapping NaN	1100	Positive number																																								
0101	Trapping NaN	1101	Negative number																																								
0110	Trapping NaN	1110	Positive number																																								
0111	Trapping NaN	1111	Positive number																																								
4—75	D0—D17	Digits	These 72 bits make up 18 decimal digits (each digit is 4 bits wide). The leftmost digit (D17) is the most significant.																																								
76—95	—	Unused	These bits are ignored and overwritten with 0s.																																								

Numbers

Each digit is in the range 0 to 9 and the decimal format can represent numbers in the range ± 0 through $10^{18}-1$, where \pm is the sign bit.

Special Case Values

The algorithm for defining a special case value in the decimal format is as follows:

```

If (sign == 0)
    Then the number is positive infinity.
Else if (sign == 1)
    Then the number is negative infinity.
Else if (sign == 2)
    Then the number is a positive nontrapping NaN.
Else if (sign == 3)
    Then the number is a negative nontrapping NaN.
Else if ((sign >= 4) && (sign <= 9))
    Then the number is a trapping NaN.
Else if (sign >= 10)
    Then if (any digit >= 10)
        Then the number is a nontrapping NaN.
    Else if ((sign == 10) || (sign == 12) || (sign == 14) || (sign == 15))
        Then the number is a positive number.
    Else the number is a negative number.
    
```

Table 2-8 shows the names of special cases and how each is represented.

Value Name	Digit Values	Sign Value
Nontrapping NaN	x	0010 or 0011
Nontrapping NaN	1010,1011,1100,1101, 1110 or 1111	1010, 1011, 1100, 1101, 1110 or 1111
Trapping NaN	x	0100, 0101, 0110, 0111, 1000 or 1001
Positive Infinity	x	0000
Negative Infinity	x	0001
Number	0000 through 1001	1010, 1011, 1100, 1101 1110 or 1111

x = don't care

FUNCTIONAL DESCRIPTIONS

Integer

2.1.5 Integer

An integer number is represented as a 2's complement 32-bit operand. This representation is shown on Figure 2-1.

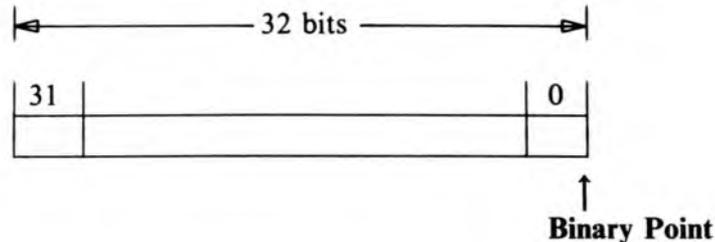


Figure 2-1. Integer Format

2.2 REGISTERS

This section describes the functions and formats of the MAU registers. The MAU has the following registers:

- An auxiliary status register (ASR) used to control various features of the MAU and to record various events.
- Eight operand registers (F0—F7) used to hold floating-point operands.
- A command register (CR) used to specify and initiate MAU transactions. This register is not readable.
- A data register (DR) used to transfer operands in and out of the MAU during operations via peripheral mode accesses. Also, when an exception occurs, the information supplied to the exception handler is stored in the DR.

The ASR, CR, and DR are accessible in peripheral mode.

2.2.1 Auxiliary Status Register

The auxiliary status register (ASR) performs such functions as signaling the state of an operation (result available bit), disabling and recording exceptions (mask and sticky bits), controlling rounding of results (round control bits), and recording condition codes (negative and zero bits).

The negative, zero, integer overflow, and inexact bits in the ASR match the positions of the negative, zero, overflow, and carry bits (the condition codes) in the PSW register of the WE 32200 Microprocessor. This allows the bits to be copied into the PSW as part of the coprocessor status access and to be easily tested by CPU software.

The bits of the ASR are described in Table 2-9.

FUNCTIONAL DESCRIPTIONS
Auxiliary Status Register

Table 2-9. Auxiliary Status Register

Bit	31	30	26	25	24	23	22	21	20	19	18	17	16		
Field	RA	—		ECP	NTNC	RC	N	Z	IO	PS	CSC	UO			
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	1	0
Field	WF	IM	OM	UM	QM	PM	IS	OS	US	QS	PR	UW	VER	FE	

Bit	Field	Contents	Description
0	FE	Feature Enable	Enables the WF and UW bits of the auxiliary status register when set.
1–3	VER	Version Number	Value=001 for <i>WE</i> 32206 Math Acceleration Unit, 000 for <i>WE</i> 32106 Math Acceleration Unit. These bits are only readable and cannot be written.
4	UW	Unaligned Word Bit	Set when a fault condition occurs due to an unaligned word being received by the MAU, if bit 0 (feature enable) is set.
5	PR	Partial Remainder	Set when result of a remainder operation is a partial remainder; cleared when result is a full remainder.
6	QS	Divide by Zero Sticky	Set if the divisor is normalized zero and dividend is a finite nonzero number.
7	US	Underflow Sticky	Set if exponent of a rounded result of an arithmetic operation is too small to be represented in the exponent field of the destination format.
8	OS	Overflow Sticky	Set if exponent of a rounded result of an arithmetic operation is too large for the exponent field of the destination format.
9	IS	Invalid Operation Sticky	Set if a result cannot be stored in a destination legally or if illegal operands are given to some operation.
10	PM	Inexact Mask	If this bit is set by the user, an exception occurs when bit 18 (inexact sticky) is set. If this bit is cleared, there are no inexact exceptions.
11	QM	Divide by Zero Mask	If this bit is set by the user, an exception occurs when bit 6 is set. If this bit is cleared, there are no divide by zero exceptions.

FUNCTIONAL DESCRIPTIONS

Auxiliary Status Register

Bit	Field	Contents	Description
12	UM	Underflow Mask	If this bit is set by the user, an exception occurs when bit 7 is set. If this bit is cleared, there are no underflow exceptions.
13	OM	Overflow Mask	If this bit is set by the user, an exception occurs when bit 8 is set. If this bit is cleared, there are no overflow exceptions.
14	IM	Invalid Operation Mask	If this bit is set by the user, an exception occurs when bit 9 is set. If this bit is cleared, there are no invalid operation exceptions.
15	WF	Write Fault Indicator	Set when a fault condition occurs during the writing of any result to memory if bit 0 (feature enable) is set. Remains set until the instruction completes without fault. If a RDASR, WRASR, LDR, EROF, or NOP occurs while this bit is set, it will not be interpreted as a restart of the operation that originally caused the fault. When this bit is set the MAU, upon restart of the operation by the CPU, will not reexecute the operation. It will, instead, return a DONE signal and use the result already residing in the DR from the previously faulted operation and store it in memory. This bit may be cleared with a WRASR.
16	UO	Unordered	Set when a compare operation results in an unordered indication; otherwise it is cleared.
17	CSC	Context Switch Control	Set on every MAU instruction execution except for RDASR, WRASR, LDR, EROF, or NOP.
18	PS	Inexact Sticky	Set if the result cannot be specified in the destination format. For any COMPARE operation, the MAU will output the status of the unordered (UO) bit in this position when the ASR is copied back to the CPU PSW.
19	IO	Integer Overflow	Set when a convert float to integer operation causes an overflow. Note that the integer overflow condition is non-maskable in the MAU.

FUNCTIONAL DESCRIPTIONS
Auxiliary Status Register

Table 2-9. Auxiliary Status Register (Continued)													
Bit	Field	Contents	Description										
20	Z	Zero	Set if the result of the last operation is zero. If the result is nonzero, the bit is cleared.										
21	N	Negative	Set if the result of the last operation is negative. If the result is positive, the bit is cleared. Note that it is possible for an operation to result in both bits 20 and 21 being set since negative zero is a representable number.										
22, 23	RC	Round Control	Represent the round control mode. The code is interpreted as: <table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding-right: 10px;">Bit Code</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">00</td> <td>Round to nearest</td> </tr> <tr> <td style="text-align: center;">01</td> <td>Round towards plus infinity</td> </tr> <tr> <td style="text-align: center;">10</td> <td>Round towards minus infinity</td> </tr> <tr> <td style="text-align: center;">11</td> <td>Round towards zero (truncation)</td> </tr> </tbody> </table>	Bit Code	Description	00	Round to nearest	01	Round towards plus infinity	10	Round towards minus infinity	11	Round towards zero (truncation)
Bit Code	Description												
00	Round to nearest												
01	Round towards plus infinity												
10	Round towards minus infinity												
11	Round towards zero (truncation)												
24	NTNC	Nontrapping NaN Control	This bit is tested when an invalid operation exception occurs and bit 14 is cleared. If this bit is set, an exception occurs and bit 9 is set. It is expected that software will write a virtual program counter value into the fraction portion of the nontrapping NaN generated.										
25	ECP	Exception Condition Present	Set if any one of the floating-point exception conditions is present.										
26–30	—	Reserved	These bits appear as 0 when read.										
31	RA	Result Available	Cleared at beginning of an operation and set when the result of an operation is available. During the quiescent state, this bit has a value of 1 (except when it has been written with a WRASR).										

FUNCTIONAL DESCRIPTIONS

Command Register

2.2.2 Operand Registers

Each of the eight operand registers (F0–F7) is 80 bits wide and contains one operand in double-extended format (see Table 2-10). These registers, used to hold operands for MAU operations, are shown as 96 bits wide because they are read and written as three 32-bit words through the DR. In peripheral mode, bits 80 through 95 are ignored during writes and returned as 0s during read operations. Registers F0 through F7 are unchanged on reset; they are indeterminate on power-up.

Table 2-10. Operand Registers

Bit	95	80	79	78	64	63	62	0
Field	—		Sign	Exponent		J	Fraction	
Bit(s)	Field	Contents	Description					
0–62	Fraction	Fraction	These bits represent the fractional part of the number.					
63	J	Explicit Bit	The J bit resides to the left of the binary point in the 2^0 position. Together, the J bit and fraction field can represent values in the range 0 through $(2-2^{-63})$.					
64–78	Exponent	Exponent	The exponent field contains an exponent that is biased by 16383.					
79	Sign	Sign bit	A 1 represents a negative value; a 0 represents a positive value.					
80–95	—	Unused	These bits are returned as 0 when read.					

2.2.3 Command Register

The command register (CR) accepts command words that are used to initiate an MAU transaction. The command register format is shown in Table 2-11.

Table 2-11. Command Register

Bit	31	24	23	21	20	19	18	17	16	15	14	10	9	7	6	4	3	0																																																							
Field	ID		—		RB1	RB2	RB3	RCS	RC		Opcode		Op1	Op2	Op3																																																										
Bit	Field	Contents	Description																																																																						
0—3	Op3	Operand Specifier	<p>Specifies whether destination operand is an MAU register, a memory-based operand of a given size, or nonexistent (no operand). Even though the register destinations are specified as single, double, or double extended, the result is stored in the registers in double-extended precision, but rounded to the requested precision. The precision designations are used for rounding and checking for underflow and overflow. The operand register designation also depends on the RB3 specifier. The value of this field is interpreted as:</p> <table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Bits</th> <th style="text-align: center;">Operand Register</th> <th style="text-align: center;">Destination Precision</th> </tr> <tr> <th style="text-align: center;">3,2,1,0</th> <th></th> <th></th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td style="text-align: center;">F0 or F4</td><td style="text-align: center;">Single</td></tr> <tr><td style="text-align: center;">0001</td><td style="text-align: center;">F1 or F5</td><td style="text-align: center;">Single</td></tr> <tr><td style="text-align: center;">0010</td><td style="text-align: center;">F2 or F6</td><td style="text-align: center;">Single</td></tr> <tr><td style="text-align: center;">0011</td><td style="text-align: center;">F3 or F7</td><td style="text-align: center;">Single</td></tr> <tr><td style="text-align: center;">0100</td><td style="text-align: center;">F0 or F4</td><td style="text-align: center;">Double</td></tr> <tr><td style="text-align: center;">0101</td><td style="text-align: center;">F1 or F5</td><td style="text-align: center;">Double</td></tr> <tr><td style="text-align: center;">0110</td><td style="text-align: center;">F2 or F6</td><td style="text-align: center;">Double</td></tr> <tr><td style="text-align: center;">0111</td><td style="text-align: center;">F3 or F7</td><td style="text-align: center;">Double</td></tr> <tr><td style="text-align: center;">1000</td><td style="text-align: center;">F0 or F4</td><td style="text-align: center;">Double-extended</td></tr> <tr><td style="text-align: center;">1001</td><td style="text-align: center;">F1 or F5</td><td style="text-align: center;">Double-extended</td></tr> <tr><td style="text-align: center;">1010</td><td style="text-align: center;">F2 or F6</td><td style="text-align: center;">Double-extended</td></tr> <tr><td style="text-align: center;">1011</td><td style="text-align: center;">F3 or F7</td><td style="text-align: center;">Double-extended</td></tr> <tr><td style="text-align: center;">1100</td><td style="text-align: center;">—</td><td style="text-align: center;">Memory-based single word</td></tr> <tr><td style="text-align: center;">1101</td><td style="text-align: center;">—</td><td style="text-align: center;">Memory-based double word</td></tr> <tr><td style="text-align: center;">1110</td><td style="text-align: center;">—</td><td style="text-align: center;">Memory-based triple word</td></tr> <tr><td style="text-align: center;">1111</td><td style="text-align: center;">—</td><td style="text-align: center;">No operand</td></tr> </tbody> </table>																	Bits	Operand Register	Destination Precision	3,2,1,0			0000	F0 or F4	Single	0001	F1 or F5	Single	0010	F2 or F6	Single	0011	F3 or F7	Single	0100	F0 or F4	Double	0101	F1 or F5	Double	0110	F2 or F6	Double	0111	F3 or F7	Double	1000	F0 or F4	Double-extended	1001	F1 or F5	Double-extended	1010	F2 or F6	Double-extended	1011	F3 or F7	Double-extended	1100	—	Memory-based single word	1101	—	Memory-based double word	1110	—	Memory-based triple word	1111	—	No operand
Bits	Operand Register	Destination Precision																																																																							
3,2,1,0																																																																									
0000	F0 or F4	Single																																																																							
0001	F1 or F5	Single																																																																							
0010	F2 or F6	Single																																																																							
0011	F3 or F7	Single																																																																							
0100	F0 or F4	Double																																																																							
0101	F1 or F5	Double																																																																							
0110	F2 or F6	Double																																																																							
0111	F3 or F7	Double																																																																							
1000	F0 or F4	Double-extended																																																																							
1001	F1 or F5	Double-extended																																																																							
1010	F2 or F6	Double-extended																																																																							
1011	F3 or F7	Double-extended																																																																							
1100	—	Memory-based single word																																																																							
1101	—	Memory-based double word																																																																							
1110	—	Memory-based triple word																																																																							
1111	—	No operand																																																																							
4—6	Op2	Operand Specifier	<p>Specifies whether second-source operand is an MAU register, a memory-based operand of a given size, or nonexistent (no operand). The value of this field is interpreted as:</p> <table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Value</th> <th style="text-align: center;">Bits 6,5,4</th> <th style="text-align: center;">Operand Location</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">000</td><td></td><td style="text-align: center;">Register F0 or F4 (depending on RB2)</td></tr> <tr><td style="text-align: center;">001</td><td></td><td style="text-align: center;">Register F1 or F5 (depending on RB2)</td></tr> <tr><td style="text-align: center;">010</td><td></td><td style="text-align: center;">Register F2 or F6 (depending on RB2)</td></tr> <tr><td style="text-align: center;">011</td><td></td><td style="text-align: center;">Register F3 or F7 (depending on RB2)</td></tr> <tr><td style="text-align: center;">100</td><td></td><td style="text-align: center;">Memory-based single word</td></tr> <tr><td style="text-align: center;">101</td><td></td><td style="text-align: center;">Memory-based double word</td></tr> <tr><td style="text-align: center;">110</td><td></td><td style="text-align: center;">Memory-based triple word</td></tr> <tr><td style="text-align: center;">111</td><td></td><td style="text-align: center;">No operand</td></tr> </tbody> </table>																	Value	Bits 6,5,4	Operand Location	000		Register F0 or F4 (depending on RB2)	001		Register F1 or F5 (depending on RB2)	010		Register F2 or F6 (depending on RB2)	011		Register F3 or F7 (depending on RB2)	100		Memory-based single word	101		Memory-based double word	110		Memory-based triple word	111		No operand																											
Value	Bits 6,5,4	Operand Location																																																																							
000		Register F0 or F4 (depending on RB2)																																																																							
001		Register F1 or F5 (depending on RB2)																																																																							
010		Register F2 or F6 (depending on RB2)																																																																							
011		Register F3 or F7 (depending on RB2)																																																																							
100		Memory-based single word																																																																							
101		Memory-based double word																																																																							
110		Memory-based triple word																																																																							
111		No operand																																																																							

FUNCTIONAL DESCRIPTIONS

Command Register

Bit	Field	Contents	Description																		
7–9	Op1	Operand Specifier	<p>Specifies whether first source operand is an MAU register, a memory-based operand of a given size, or nonexistent (no operand). The value of this field is interpreted as:</p> <table border="0"> <thead> <tr> <th>Value Bits 9,8,7</th> <th>Operand Location</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>Register F0 or F4 (depending on RB1)</td> </tr> <tr> <td>001</td> <td>Register F1 or F5 (depending on RB1)</td> </tr> <tr> <td>010</td> <td>Register F2 or F6 (depending on RB1)</td> </tr> <tr> <td>011</td> <td>Register F3 or F7 (depending on RB1)</td> </tr> <tr> <td>100</td> <td>Memory-based single word</td> </tr> <tr> <td>101</td> <td>Memory-based double word</td> </tr> <tr> <td>110</td> <td>Memory-based triple word</td> </tr> <tr> <td>111</td> <td>No operand</td> </tr> </tbody> </table>	Value Bits 9,8,7	Operand Location	000	Register F0 or F4 (depending on RB1)	001	Register F1 or F5 (depending on RB1)	010	Register F2 or F6 (depending on RB1)	011	Register F3 or F7 (depending on RB1)	100	Memory-based single word	101	Memory-based double word	110	Memory-based triple word	111	No operand
Value Bits 9,8,7	Operand Location																				
000	Register F0 or F4 (depending on RB1)																				
001	Register F1 or F5 (depending on RB1)																				
010	Register F2 or F6 (depending on RB1)																				
011	Register F3 or F7 (depending on RB1)																				
100	Memory-based single word																				
101	Memory-based double word																				
110	Memory-based triple word																				
111	No operand																				
10–14	Opcode	Operation Code	Specifies operation to be performed. See Chapter 5 for opcode designations.																		
15–16	RC	Round Control	<p>Determines the method of rounding used if the RCS bit = 1. When the RC bits in the CR are in control, they have the following meaning:</p> <table border="0"> <thead> <tr> <th>Value Bits 15,16</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Round to nearest</td> </tr> <tr> <td>01</td> <td>Round towards positive infinity</td> </tr> <tr> <td>10</td> <td>Round towards minus infinity</td> </tr> <tr> <td>11</td> <td>Round towards zero (truncation)</td> </tr> </tbody> </table>	Value Bits 15,16	Description	00	Round to nearest	01	Round towards positive infinity	10	Round towards minus infinity	11	Round towards zero (truncation)								
Value Bits 15,16	Description																				
00	Round to nearest																				
01	Round towards positive infinity																				
10	Round towards minus infinity																				
11	Round towards zero (truncation)																				
17	RCS	Round Control Selection	<p>Determines where the control of the rounding is located. It has the following meaning:</p> <table border="0"> <thead> <tr> <th>Value Bit 17</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Round control is determined by the RC bits in the ASR</td> </tr> <tr> <td>1</td> <td>Round control is determined by the RC bits in the CR (the RC bits in the ASR are left unchanged)</td> </tr> </tbody> </table>	Value Bit 17	Description	0	Round control is determined by the RC bits in the ASR	1	Round control is determined by the RC bits in the CR (the RC bits in the ASR are left unchanged)												
Value Bit 17	Description																				
0	Round control is determined by the RC bits in the ASR																				
1	Round control is determined by the RC bits in the CR (the RC bits in the ASR are left unchanged)																				
18–20	RB3–RB1	Register Bank	<p>Specifies which bank of registers will be used as an operand:</p> <table border="0"> <thead> <tr> <th>Value Bit 18, 19, or 20</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Use register F0, F1, F2, or F3</td> </tr> <tr> <td>1</td> <td>Use register F4, F5, F6, or F7</td> </tr> </tbody> </table>	Value Bit 18, 19, or 20	Meaning	0	Use register F0, F1, F2, or F3	1	Use register F4, F5, F6, or F7												
Value Bit 18, 19, or 20	Meaning																				
0	Use register F0, F1, F2, or F3																				
1	Use register F4, F5, F6, or F7																				
21–23	–	Reserved	These bits are returned as 0 when read.																		
24–31	ID	Processor ID Number	Specifies identification number of the processor that should react to the command word. The MAU ID is 0.																		

2.2.4 Data Register

The data register (DR) is used to read and write operands via peripheral mode accesses. The DR appears as three 32-bit words in the peripheral mode address space. Each access is a word access, and the source or destination (the format of the word) is determined by the context of the access (e.g., if it is the third read during an ADD double-extended operation where the first operand is in memory, the destination is bits 31–0 of Op1). Refer to section 4-1.

When an exception occurs, the information supplied to the exception handler is stored in the DR. This information is summarized in Table 2-12. An extract result on a fault (EROF) instruction can be executed to retrieve this information from the DR.

Table 2-12. Data Register Exception Handler Information

Exception Type	Information Supplied																				
Invalid Operation	If either of the source operands is a trapping NaN, the DR contains the NaN converted to double-extended precision (80 bits) if necessary. If both source operands are trapping NaNs or infinities of different signs, the second operand (Op2) in double-extended precision (80 bits) is stored in the DR. The L, G, R, and S bits are undefined.																				
Overflow or Underflow	<p>The significand along with the 17-bit internal result exponent and result sign is stored in the DR. The MSB of the 17-bit exponent behaves like a sign bit in 2's complement notation. An additional bit (bit 79) in the exponent ensures that no significant exponent bits are lost from an internal representation on occurrence of an overflow or underflow condition. The exponent value is biased by 16383. The LSB (bit L) of the unrounded result, as well as the guard (G), round (R), and sticky (S) bits of the unrounded result, will be stored in bits 95 through 92, respectively, of operand 3. The format is:</p> <table border="1" data-bbox="474 1444 1268 1549"> <tr> <td data-bbox="482 1451 535 1478">Bit</td> <td data-bbox="543 1451 597 1478">95</td> <td data-bbox="605 1451 659 1478">94</td> <td data-bbox="666 1451 720 1478">93</td> <td data-bbox="728 1451 782 1478">92</td> <td data-bbox="790 1451 843 1478">91 82</td> <td data-bbox="851 1451 905 1478">81</td> <td data-bbox="913 1451 967 1478">80 64</td> <td data-bbox="974 1451 1028 1478">63</td> <td data-bbox="1036 1451 1090 1478">62 0</td> </tr> <tr> <td data-bbox="482 1486 535 1514">Field</td> <td data-bbox="543 1486 597 1514">L</td> <td data-bbox="605 1486 659 1514">G</td> <td data-bbox="666 1486 720 1514">R</td> <td data-bbox="728 1486 782 1514">S</td> <td data-bbox="790 1486 843 1514">—</td> <td data-bbox="851 1486 905 1514">Sign</td> <td data-bbox="913 1486 967 1514">Exponent</td> <td data-bbox="974 1486 1028 1514">J</td> <td data-bbox="1036 1486 1090 1514">Fraction</td> </tr> </table> <p style="text-align: right; margin-right: 100px;">←Significand→</p>	Bit	95	94	93	92	91 82	81	80 64	63	62 0	Field	L	G	R	S	—	Sign	Exponent	J	Fraction
Bit	95	94	93	92	91 82	81	80 64	63	62 0												
Field	L	G	R	S	—	Sign	Exponent	J	Fraction												
Divide by Zero	The dividend (Op2) converted to double-extended precision, if necessary, is stored in the DR. The L, G, R, and S bits are stored in bits 95 through 92, respectively.																				
Inexact	The rounded result converted to double-extended precision, if necessary, is stored in the DR. The L, G, R, and S bits are stored in bits 95 through 92, respectively.																				

FUNCTIONAL DESCRIPTIONS

Data Register

2.3 ROUNDING

Rounding of the result is performed at the end of every MAU arithmetic operation: it is not a separate operation triggered via an opcode in a command word.

The type of rounding performed is controlled by the round control bits: either bits 22 and 23 of the ASR, or bits 15 and 16 of the CR. The four types of rounding are:

Mnemonic	Description
RN	Round to nearest
RP	Round towards plus infinity
RM	Round towards minus infinity
RZ	Round towards zero (truncation)

Rounding can be specified in terms of three numbers: Z , $Z1$, and $Z2$. The value Z is the preliminary result of the operation before rounding and is considered to be represented by an infinitely long sequence of bits. $Z1$ and $Z2$ are the two numbers representable in the destination format which most closely bracket the value Z . Therefore, Z , $Z1$, and $Z2$ satisfy the relation:

$$Z1 \leq Z < Z2.$$

If $Z1 == Z$, then there is no rounding error and $RN(Z) == RP(Z) == RM(Z) == RZ(Z)$. Otherwise, an inexact condition exists.

Using this model, the action associated with each of the rounding modes is specified as follows:

RN: The result is the nearer of $Z1$ and $Z2$ to Z . If $Z1$ and $Z2$ are equidistant from Z , then the quantity whose least significant fraction bit is zero will be the result.

RP: The result is $Z2$.

RM: The result is $Z1$.

RZ: The smaller of $Z1$ and $Z2$ (in absolute value) is the result.

Note that upon exponent underflow, the result is not rounded twice; that is, denormalization is performed before rounding.

2.4 EXCEPTIONS

For each exception type, there is a mask bit and a sticky bit in the ASR. If the condition associated with the exception type is satisfied, the MAU sets the sticky bit for that exception type. If the mask bit is 1, an exception occurs; if the mask bit is 0, no exception occurs. For the format of the data stored in DR on occurrence of invalid-operation, divide-by-zero, or inexact-exception, refer to the format descriptions in Overflow and Underflow in section 2.4.1.

The MAU never changes the value of the mask bits and never clears the sticky bits. These actions, if necessary, must be done with software writes to the ASR.

2.4.1 Exception Types

This section specifies the exception types and the conditions that they represent. The actions performed upon detection of the exceptional condition are also specified.

The exceptions follow an order of precedence, so that the highest priority exception is taken if more than one occurs. The ordering is as follows:

1. invalid operation (highest)
2. divide-by-zero, overflow and underflow
3. inexact (lowest)

Invalid Operation

There are two classes of the invalid-operation exception. The first results if the operand is illegal for the operation (e.g., square root of a negative number). The other arises if the result is illegal for the destination (e.g., the destination is an MAU register on a float-to-decimal conversion operation).

The invalid-operation exception's sticky flag (ASR<IS>) is set and the result is a nontrapping NaN. An invalid-operation exception is raised if the invalid-operation exception is enabled (ASR<IM>==1).

Divide-By-Zero

The divide-by-zero condition exists in a division operation when the divisor is zero and the dividend is a finite nonzero number.

The sticky flag for this condition (ASR<QS>) is set and the result is infinity with the sign according to convention. The divide-by-zero exception is raised if this exception is enabled (ASR<QM>==1).

FUNCTIONAL DESCRIPTIONS

Overflow

Overflow

A floating overflow condition exists when an arithmetic operation's rounded exponent result overflows the range of the destination (note that the range does not include the destination format's maximum exponent value).

On all operations, if the overflow exception condition is enabled ($ASR<OM>==1$) and the destination for the operation is single or double, the rounded intermediate result significand is first converted to double-extended precision. If the destination for the operation is double-extended, then no conversion is performed.

This significand, along with the 17-bit internal result exponent and result sign, is stored in the data register. The destination operand for the operation is left unaltered. Also, the sticky flag for this condition ($ASR<OS>$) is set. The format of the result stored in the DR is as shown below:

81	80	64	63	62	0
Sign	Exponent		J	Fraction	
			←	Significand	→

The MSB of the 17-bit exponent behaves like a sign bit in 2's complement notation. An additional bit (bit 79) in the exponent ensures that no significant exponent bits are lost from an internal representation when an overflow condition occurs. The exponent value stored in the DR has the bias of the double-extended-precision exponent (that is, 16383) riding on it.

If the overflow exception is disabled ($ASR<OM>==0$), the result of the operation is based on the rounding mode selected and the sign of the overflowed intermediate result.

RN mode carries all overflows to infinity with the sign of the overflowed intermediate result.

RZ mode carries all overflows to the format's largest finite number with the sign of the overflowed intermediate result.

RM mode carries positive overflows to the format's largest finite number and carries negative overflows to negative infinity.

RP mode carries negative overflows to the format's most negative finite number and carries positive overflows to positive infinity.

Underflow

A floating underflow condition exists if the exponent of an intermediate result lies below the exponent range of the destination field (note that this range does not include the destination format's minimum exponent value).

The test for underflow is performed before rounding the intermediate result, however, certain underflow conditions may disappear after rounding. In such situations, no underflow condition exists and the result is a correctly rounded number.

On all operations, if the underflow exception condition is enabled ($ASR<UM>==1$) and the destination for the operation is single or double, the rounded intermediate result significand is first converted to double-extended precision. If the destination for the operation is double-extended, no conversion is performed.

This significand, along with the 17-bit internal result exponent and result sign, is stored in the DR. The destination operand for the operation is left unaltered. Also, the sticky flag for this condition ($ASR<US>$) is set. The format of the result stored in the DR is as shown below:

81	80	64	63	62	0
Sign	Exponent		J	Fraction	
			←	Significand	→

The MSB of the 17-bit exponent behaves like a sign bit in 2's complement notation. An additional bit ($bit<79>$) in the exponent ensures that no significant exponent bits are lost from an internal representation when an underflow condition occurs. The exponent value stored in the DR has the bias of the double-extended-precision exponent (that is, 16383) riding on it.

If the underflow exception is disabled ($ASR<UM>==0$), the unrounded intermediate (underflowed) result is denormalized by shifting the significand to the right while incrementing the exponent until the exponent reaches its minimum allowable value (note that this is 1 plus the format's minimum). The denormalized number is then rounded and stored in the destination. Also, $ASR<US>$ is set if the denormalization results in the loss of accuracy and, therefore, requires rounding.

Note: If a gradual underflow occurs and yields a nonzero result and if inexact exception is not masked, the internal result stored in the DR is unnormalized with its exponent one less than the correct value.

FUNCTIONAL DESCRIPTIONS

Inexact

Inexact

If the result cannot be specified exactly in the destination format, a floating inexact condition exists.

The sticky flag for this condition (ASR<PS>) is set and the result is the correctly rounded number. The floating inexact condition is raised if this exception is enabled (ASR<PM>==1). The data register contains the inexact result intended for the exception handler. However, if the operation is float-to-decimal or float-to-integer when this exception occurs, the contents of the DR are unspecified.

Note: If, in peripheral mode, a floating-point operation yields an inexact result and the inexact fault is enabled and no other higher priority enabled fault occurs, the DR will contain the result in the original operation's destination precision instead of double-extended precision. The result's exponent will be sign extended to 17 bits; the mantissa will be right justified and will include the implicit bit. The result will be exactly as expected if the original operation had double-extended-precision destination specified.

2.5 CONTEXT SAVING AND RESTORING

If it is necessary to save and restore the context of the MAU, it can be done in either protocol mode (peripheral or coprocessor) using MOVE operations. One operand must be in memory and the other in the MAU.

2.5.1 Peripheral Mode

The MAU context can be saved by:

- Reading the ASR via peripheral mode and saving its value.
- Clearing the ASR.
- Reading the three DR words via peripheral mode and saving their value.
- Reading registers F0–F7 via MAU move instructions with memory destinations. The values appear in the DR and must then be moved to memory.

The MAU context can be restored by:

- Performing eight MAU move instructions to restore F0–F7. The values must be written to the DR via peripheral mode accesses.
- Performing an MAU move operation, with the source being a memory location (the value must be written into the DR via a peripheral mode access) and the destination being a memory location (the value will appear in the DR). This restores the DR.
- Writing the saved ASR value into the ASR via peripheral mode.

2.5.2 Coprocessor Mode

The MAU context can be saved by:

- Reading the DR with an extract result on fault (EROF) instruction and saving its value.
- Reading the ASR mode with a move from ASR (RDASR) instruction (or peripheral mode) and saving its value.
- Clearing the ASR.
- Performing eight MAU move operations to save registers F0–F7.

The MAU context can be restored by:

- Clearing the ASR.
- Performing eight MAU move operations to restore registers F0–F7.
- Writing the saved ASR value into the ASR via peripheral mode or a move to ASR (WRASR) instruction.
- Performing a load DR operation using the LDR instruction to restore the DR.

2.6 INTERRUPTS

The following sections describe what occurs when system interrupts are encountered and how the MAU reacts to them in both peripheral and coprocessor modes.

2.6.1 Peripheral Mode Protocol

Using peripheral mode protocol, the MAU does not abort its operation when an interrupt acknowledge cycle occurs. The peripheral mode protocol assumes that the CPU is executing a stream of discrete instructions to implement the protocol and thus would not be able to back up the first instruction in the event of an interrupt. The CPU is free to allow or disallow interrupts during MAU operations; however, MAU context saving is simplified if the interrupts can be disabled.

The state of the MAU must be saved and restored later if an interrupt causes control of the MAU to be given to another process. If the MAU is in the middle of an operation and the RA bit of the ASR is cleared or if peripheral interrupt request ($\overline{\text{PIREQ}}$) is cleared, the MAU's state cannot be saved and restored without explicit software provisions. If the first CPU instruction of the sequence can be found, the context of the partially executed transaction can be saved so that the execution can resume there. The context to be saved and restored consists of the ASR, DR, and F0–F7. It may be necessary to run the context save and restore sequences with interrupts disabled.

If an interrupt occurs and the RA bit of the ASR has been set, the MAU is placed in a quiescent state. The transaction may or may not have been completed (some reads from the DR may not be complete). The context to be saved and restored consists of the ASR, DR, and F0–F7 (see section 2.5).

FUNCTIONAL DESCRIPTIONS

Coprocessor Mode Protocol

Once a transaction has been initiated via a peripheral mode write to the CR, the MAU ignores any interrupt acknowledge or autovector interrupt acknowledge bus cycles. This behavior continues after completion of the transaction until the next MAU transaction is started.

2.6.2 Coprocessor Mode Protocol

If an interrupt occurs during an MAU transaction but before the $\overline{\text{DONE}}$ signal goes to 0, the MAU aborts the operation. It is possible that the ASR has been changed to reflect exceptions or new condition codes; however, when the CPU returns from the interrupt, it can restart the transaction from the beginning and the spurious ASR value will not affect the repeated operation.

The context of the MAU must be saved and restored later if the interrupt causes control of the MAU to be given to another process. The context to be saved and restored consists of ASR, DR, and F0–F7.

Once an operation has been initiated via a coprocessor broadcast access, the MAU responds to any interrupt acknowledge or autovector interrupt acknowledge bus cycles by aborting the transaction and returning to quiescent state ($\text{ASR} \langle \text{RA} \rangle = 1$).

2.7 MAU RESET

The MAU responds to the reset signal immediately and identically from any state, taking the following actions:

- Any access or operation in progress is immediately terminated.
- The MAU goes into its quiescent state (see section 2.8).

All bits of the CR, ASR, DR, and F0–F7 are unaffected by a reset. In the case of a reset upon power-up, these bits may have either 0 or 1 values. In the case of a reset of a running MAU, these bits contain the same values they had before the reset occurred.

2.8 QUIESCENT STATE

The MAU returns to its quiescent state after the completion of an operation and after a reset.

In quiescent state, the MAU satisfies the following conditions:

- It is ready to accept and react to the following access types only:
 - Data Read (if chip-select is asserted)
 - Data Write (if chip-select is asserted)
 - Coprocessor Broadcast

- It is ready to accept a reset.
- It is not operating on any of its internal objects in any way (either reading or writing them).
- It is not affecting the rest of the system in any way (driving any bus or other signal to the system).

2.9 MAU-CPU INTERCONNECTIONS

The following sections provide examples of the connections between the MAU and the *WE* 32200 Microprocessor (CPU). Refer to the *WE* 32200 Microprocessor Information Manual for a complete description of that device.

2.9.1 Peripheral Mode

Figure 2-2 demonstrates how the MAU is directly interfaced to the *WE* 32200 Microprocessor for peripheral mode operations.

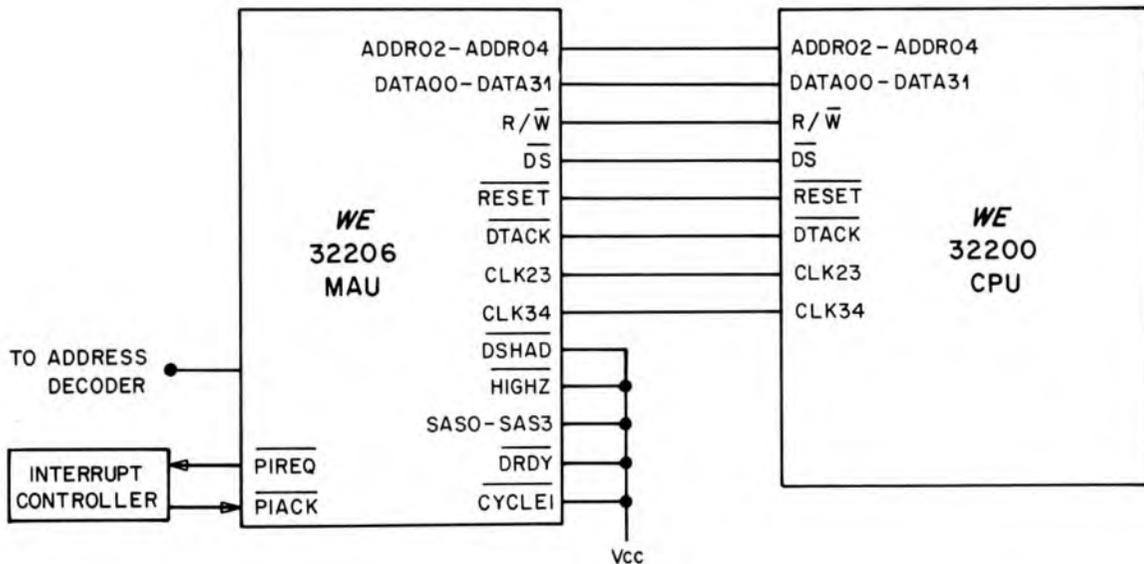


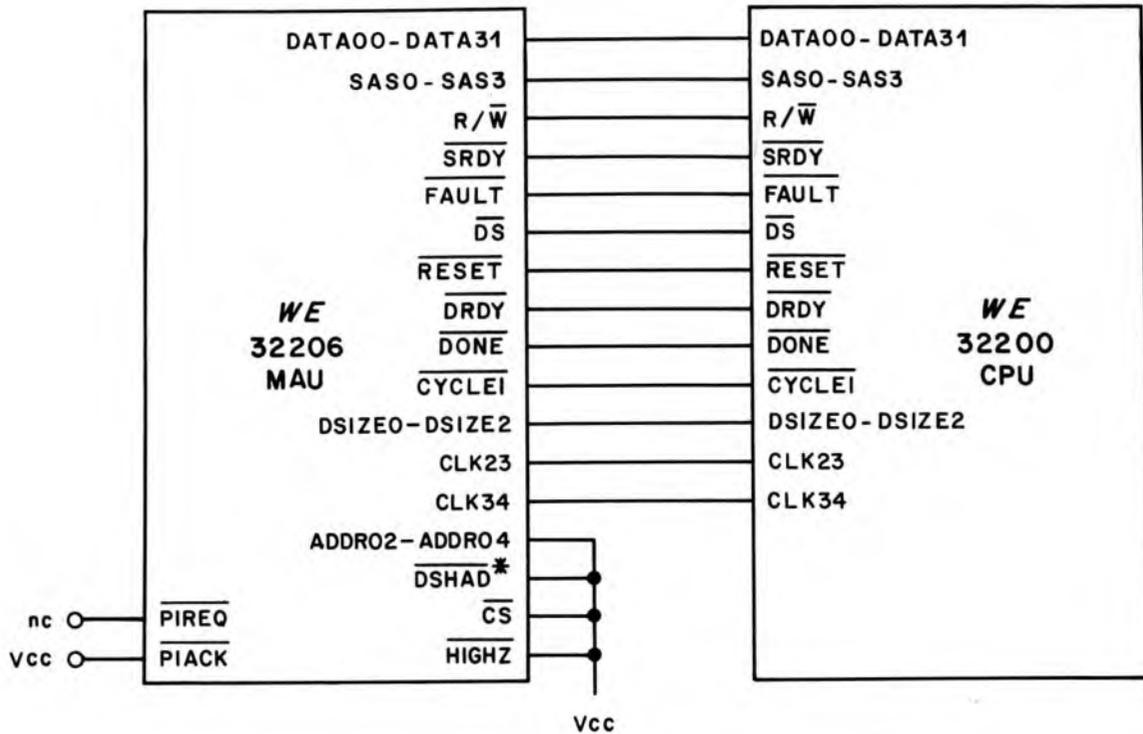
Figure 2-2. MAU-CPU Peripheral Mode Protocol Interconnections

FUNCTIONAL DESCRIPTIONS

Coprocessor Mode

2.9.2 Coprocessor Mode

Figure 2-3 demonstrates how the MAU is directly interfaced to the WE 32200 CPU for coprocessor mode operations.



*If a WE 32201 Memory Management Unit (MMU) is included in the system, this signal must be connected to the MMU's DSHAD output.

Figure 2-3. MAU-CPU Coprocessor Mode Protocol Interconnections

Chapter 3
Signal Descriptions

CHAPTER 3. SIGNAL DESCRIPTIONS

CONTENTS

3. SIGNAL DESCRIPTIONS	3-1
3.1 ADDRESS AND DATA SIGNALS	3-1
Address (ADDR00—ADDR04)	3-1
Data (DATA00—DATA31)	3-1
3.2 INTERFACE AND CONTROL SIGNALS	3-2
Chip Select (\overline{CS})	3-2
Cycle Initiate (\overline{CYCLEI})	3-2
Data Bus Shadow (\overline{DSHAD})	3-2
Data Ready (\overline{DRDY})	3-2
Data Strobe (\overline{DS})	3-2
Data Transfer Acknowledge (\overline{DTACK})	3-2
Done (\overline{DONE})	3-2
Peripheral Interrupt Acknowledge (\overline{PIACK})	3-2
Peripheral Interrupt Request (\overline{PIREQ})	3-2
Synchronous Ready (\overline{SRDY})	3-3
3.3 STATUS SIGNALS	3-3
Access Status Codes (SAS0—SAS3)	3-3
Data Size (DSIZE0—DSIZE2)	3-3
Read/Write (R/ \overline{W})	3-3
3.4 BUS EXCEPTION SIGNALS	3-4
Fault (\overline{FAULT})	3-4
Reset (\overline{RESET})	3-4
3.5 CLOCKS	3-4
Input Clock (CLK23)	3-4
Input Clock (CLK34)	3-4
3.6 TEST SIGNALS (HIGHZ)	3-4
3.7 PIN ASSIGNMENTS	3-4

3. SIGNAL DESCRIPTIONS

The input and output signals can be organized into the functional groups shown on Figure 3-1. These functional groupings are described in the remainder of this chapter.

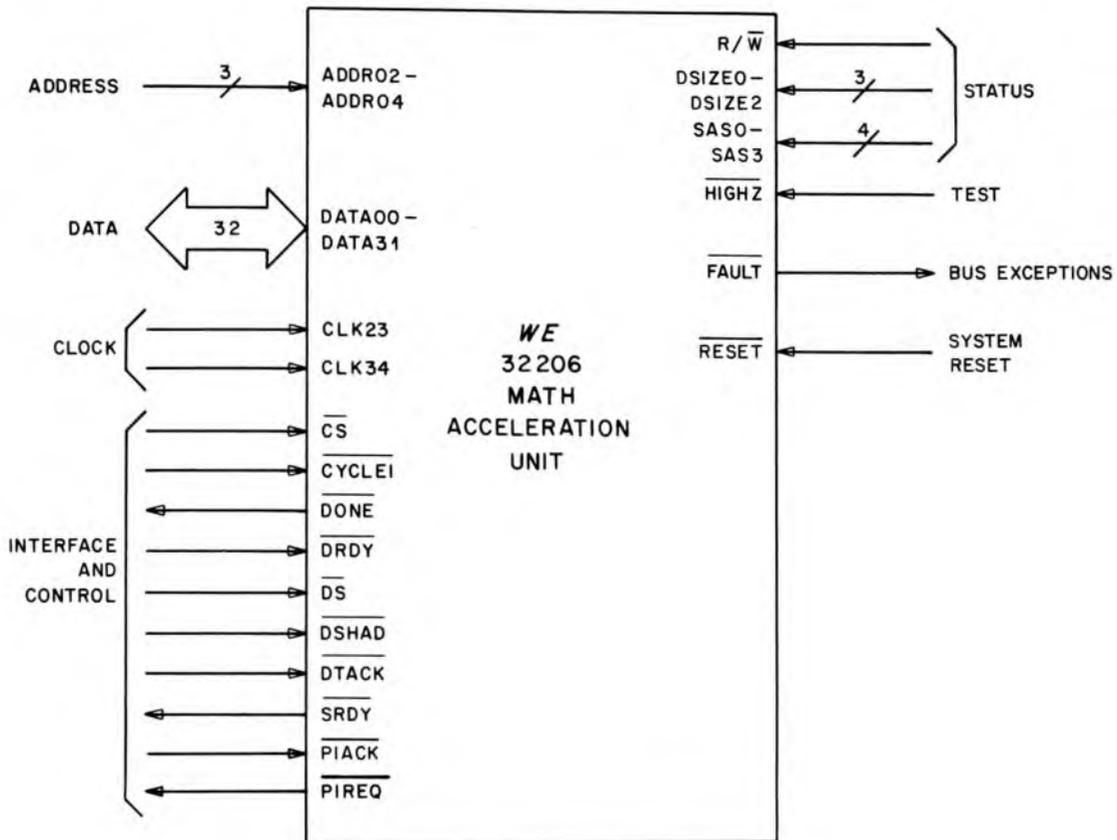


Figure 3-1. Input and Output Signals

3.1 ADDRESS AND DATA SIGNALS

Address (ADDR02-ADDR04)

Bits 2 to 4 of the system address bus are used as inputs to select the peripheral mode registers when the MAU is in peripheral mode.

Data (DATA00-DATA31)

These signals provide a bidirectional bus to transmit data to and from the MAU.

SIGNAL DESCRIPTIONS

Interface and Control Signals

3.2 INTERFACE AND CONTROL SIGNALS

Chip Select ($\overline{\text{CS}}$)

Assertion of this input signal puts the MAU in peripheral mode.

Cycle Initiate ($\overline{\text{CYCLEI}}$)

Assertion of this input indicates that the CPU has started a bus cycle.

Data Bus Shadow ($\overline{\text{DSHAD}}$)

Assertion of this input causes the data bus to 3-state if the MAU is driving it and causes $\overline{\text{DRDY}}$ to be ignored.

Data Ready ($\overline{\text{DRDY}}$)

Assertion of this input signal indicates to the MAU that no bus exceptions have been detected during the current bus cycle. It is ignored if $\overline{\text{DSHAD}}$ is asserted.

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

When transferring data to the MAU, assertion of this input indicates valid data on the data bus. When transferring data from the MAU, $\overline{\text{DS}}$ indicates that the MAU may drive the data bus.

Data Transfer Acknowledge ($\overline{\text{DTACK}}$)

Assertion of this output signal causes the CPU to end the current access. This signal is used for asynchronous handshaking between the CPU and MAU in peripheral mode.

Done ($\overline{\text{DONE}}$)

Assertion of this output signal indicates to the CPU that the MAU has completed execution of the current instruction.

Peripheral Interrupt Acknowledge ($\overline{\text{PIACK}}$)

Assertion of this input negates the $\overline{\text{PIREQ}}$ signal.

Peripheral Interrupt Request ($\overline{\text{PIREQ}}$)

When asserted, this output signals that the MAU has completed an operation in peripheral mode.

Synchronous Ready ($\overline{\text{SRDY}}$)

Assertion of this output causes the CPU to begin the termination of a coprocessor broadcast or coprocessor status fetch access.

3.3 STATUS SIGNALS

Access Status Codes (SAS0—SAS3)

These input signals describe the type of bus cycle being executed. SAS0 is the LSB of the access status code.

Table 3-1 lists and describes the access codes recognized by the MAU.

Table 3-1. Access Codes	
Code SAS3—0	Description
0001	Coprocessor data write
0010	Autovector interrupt acknowledge
0011	Coprocessor data fetch
0101	Coprocessor broadcast
0110	Coprocessor status fetch
1011	Interrupt acknowledge

Data Size (DSIZE0—DSIZE2)

These inputs indicate the size of a bus transaction from a *WE* 32200 CPU. DSIZE0 is the LSB of the data size code. Table 3-2 lists and describes the data size codes recognized by the MAU.

Table 3-2. Data Size Codes	
Code DSIZE2—0	Description
001	0 bytes
100	word
101	double word
110	halfword
111	byte

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

In peripheral mode, a 1 on this input indicates a read from the MAU; a 0 indicates a write to the MAU.

SIGNAL DESCRIPTIONS

Read/Write Signals

3.4 BUS EXCEPTION SIGNALS

Fault ($\overline{\text{FAULT}}$)

Assertion of this output signal indicates that an exception occurred during execution of an instruction in the MAU.

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

Assertion of this input for at least two consecutive cycles causes the MAU to reset.

3.5 CLOCKS

Input Clock (CLK34)

The falling edge of this clock input signifies the beginning of a machine cycle. This clock input has the same frequency as CLK23 and lags it by 90° .

Input Clock (CLK23)

This clock input has the same frequency as CLK34 and leads it by 90° .

3.6 TEST SIGNAL ($\overline{\text{HIGHZ}}$)

Assertion of this input signal 3-states all output signals for testing.

3.7 PIN ASSIGNMENTS

The WE 32206 Math Acceleration Unit is available in a 125-pin square, ceramic pin grid array (PGA) package. It has 53 active pins, 16 power pins, and 18 ground pins. Figure 3-2 illustrates the square pin-array package; Table 3-3 lists the pin assignments. All pin locations which are not explicitly listed are "no connection" and are reserved for future use.

SIGNAL DESCRIPTIONS
Pin Assignments

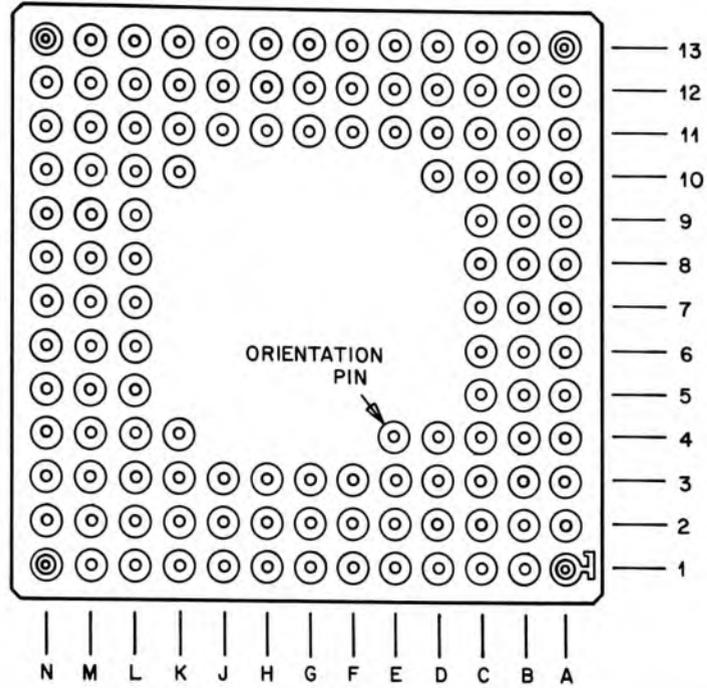


Figure 3-2. 125-Pin Square Ceramic Pin Grid Array – Bottom View

SIGNAL DESCRIPTIONS

Pin Assignments

Table 3-3. WE 32206 MAU Pin Descriptions			
Pin	Name	Type	Description
A1	DATA10	I/O	Data 10
A2	DATA13	I/O	Data 13
A4	DATA16	I/O	Data 16
A6	DATA18	I/O	Data 18
A8	DATA21	I/O	Data 21
A10	DATA24	I/O	Data 24
A11	DATA26	I/O	Data 26
A13	$\overline{\text{SRDY}}$	O	Synchronous ready
B2	DATA11	I/O	Data 11
B3	DATA15	I/O	Data 15
B5	DATA17	I/O	Data 17
B6	GND	—	Ground
B7	DATA19	I/O	Data 19
B8	DATA20	I/O	Data 20
B9	DATA22	I/O	Data 22
B10	DATA23	I/O	Data 23
B11	DATA25	I/O	Data 25
B12	DATA30	I/O	Data 30
C1	DATA09	I/O	Data 09
C2	GND	—	Ground
C3	DATA12	I/O	Data 12
C4	+5V	—	Power
C5	GND	—	Ground
C6	+5V	—	Power
C7	GND	—	Ground
C8	+5V	—	Power
C9	GND	—	Ground
C10	+5V	—	Power
C11	DATA27	I/O	Data 27
C12	GND	—	Ground
C13	DATA28	I/O	Data 28
D2	DATA08	I/O	Data 08
D3	+5V	—	Power
D4	DATA14	I/O	Data 14
D11	+5V	—	Power
D12	DATA29	I/O	Data 29
E1	DATA07	I/O	Data 07
E3	GND	—	Ground
E4	—	—	Device socket orientation pin
E11	GND	—	Ground
E12	GND	—	Ground
E13	DATA31	I/O	Data 31

SIGNAL DESCRIPTIONS
Pin Assignments

Table 3-3. WE 32206 MAU Pin Descriptions (Continued)			
Pin	Name	Type	Description
F2	DATA05	I/O	Data 05
F3	+5V	—	Power
F11	+5V	—	Power
F12	CLK23	I	Input clock 23
G1	DATA06	I/O	Data 06
G3	GND	—	Ground
G11	GND	—	Ground
G12	CLK34	I	Input clock 34
G13	PIACK	I	Peripheral interrupt acknowledge
H2	DATA02	I/O	Data 02
H3	+5V	—	Power
H11	+5V	—	Power
H12	DONE	O	Coprocessor done
H13	PIREQ	O	Peripheral interrupt request
J1	DATA04	I/O	Data 04
J2	GND	—	Ground
J3	GND	—	Ground
J11	GND	—	Ground
J12	DSIZE0	I	Data size 0
J13	FAULT	O	Fault
K2	DATA01	I/O	Data 01
K3	+5V	—	Power
K11	DSIZE1	I	Data size 1
K12	ADDR03	I	Address 03
L1	DATA03	I/O	Data 03
L3	DATA00	I/O	Data 00
L4	+5V	—	Power
L5	GND	—	Ground
L6	+5V	—	Power
L7	GND	—	Ground
L8	+5V	—	Power
L9	GND	—	Ground
L10	+5V	—	Power
L11	ADDR04	I	Address 04
L12	DSIZE2	I	Data size 2
M2	DSHAD	I	Data bus shadow
M3	GND	—	Ground
M4	SAS2	I	Access status code 2
M5	GND	—	Ground
M6	SAS1	I	Access status code 1
M7	DS	I	Data strobe

SIGNAL DESCRIPTIONS
Pin Assignments

Table 3-3. WE 32206 MAU Pin Descriptions (Continued)

Pin	Name	Type	Description
M8	$\overline{\text{RESET}}$	I	Reset
M9	GND	—	Ground
M10	$\overline{\text{DTACK}}$	O	Data transfer acknowledge
M11	$\overline{\text{DRDY}}$	I	Data ready
M12	$\text{R}/\overline{\text{W}}$	I	Read/write
N1	$\overline{\text{HIGHZ}}$	I	High impedance
N3	$\overline{\text{CYCLEI}}$	I	Cycle initiate
N5	SAS3	I	Access status code 3
N6	SAS0	I	Access status code 0
N8	$\overline{\text{CS}}$	I	Chip select
N13	ADDR02	I	Address 02

Chapter 4
Bus Transactions

CHAPTER 4. BUS TRANSACTIONS

CONTENTS

4. BUS TRANSACTIONS	4-1
4.1 PERIPHERAL MODE TRANSACTIONS	4-1
4.1.1 Peripheral Mode Accesses	4-1
4.1.2 Bus Transaction Sequence	4-2
Transaction Sequence	4-3
4.1.3 Peripheral Mode Read and Write	4-4
4.1.4 Configuration Restrictions	4-4
4.1.5 Peripheral Mode Interrupts	4-9
4.1.6 Peripheral Mode Faults	4-9
4.1.7 Peripheral Mode Restartability	4-10
4.1.8 Peripheral Mode Operating System Implications	4-10
4.2 COPROCESSOR MODE TRANSACTIONS	4-11
4.2.1 Bus Transaction Sequence	4-11
Transaction Sequence	4-12
4.2.2 Coprocessor Broadcast Transaction	4-13
4.2.3 Coprocessor Data Fetch Transaction	4-17
4.2.4 Coprocessor Status Read Transaction	4-21
4.2.5 Coprocessor Data Write Transaction	4-24
4.2.6 Coprocessor Faults	4-30
Status Fetch Faults	4-30
Non-Existent MAU Faults	4-31
4.2.7 Coprocessor Restartability	4-31
4.2.8 Restrictions on System Configuration	4-32
4.2.9 Coprocessor Operating System Implications	4-33

4. BUS TRANSACTIONS

The MAU may operate in either peripheral mode or coprocessor mode. The following sections describe the bus operations in which the MAU is a participant.

4.1 PERIPHERAL MODE TRANSACTIONS

Configured in the peripheral mode, the MAU is capable of the following bus transactions:

- Peripheral mode read
- Peripheral mode write.

4.1.1 Peripheral Mode Accesses

In peripheral mode, objects in the MAU can be accessed as memory-mapped addresses. A peripheral mode access occurs when there is a data read or data write access and the MAU's chip select input is being asserted. The MAU latches the address associated with the access and interprets it as shown in Table 4-1. If the access is a read, the contents of the selected MAU object are returned as data. If the access is a write, the data associated with it is stored in the MAU object selected.

Table 4-1. Peripheral Mode Address Fields																						
	Bit	31 5 4 2 1 0																				
	Field	Unused Object Unused																				
Bit	Field	Description																				
0–1	Unused	Ignored by the MAU.																				
2–4	Object	Select which object is to be accessed according to the object value specified. (Bit 2 is the least significant bit.) <table style="margin-left: 20px; border: none;"> <thead> <tr> <th style="text-align: left;">Object Bit</th> <th style="text-align: left;">Object Addressed</th> </tr> </thead> <tbody> <tr> <td>4 3 2</td> <td></td> </tr> <tr> <td>0 0 0</td> <td>ASR</td> </tr> <tr> <td>0 0 1</td> <td>ASR</td> </tr> <tr> <td>0 1 0</td> <td>ASR</td> </tr> <tr> <td>0 1 1</td> <td>ASR</td> </tr> <tr> <td>1 0 0</td> <td>DR bits 64–95</td> </tr> <tr> <td>1 0 1</td> <td>DR bits 32–63</td> </tr> <tr> <td>1 1 0</td> <td>DR bits 0–31</td> </tr> <tr> <td>1 1 1</td> <td>CR</td> </tr> </tbody> </table>	Object Bit	Object Addressed	4 3 2		0 0 0	ASR	0 0 1	ASR	0 1 0	ASR	0 1 1	ASR	1 0 0	DR bits 64–95	1 0 1	DR bits 32–63	1 1 0	DR bits 0–31	1 1 1	CR
Object Bit	Object Addressed																					
4 3 2																						
0 0 0	ASR																					
0 0 1	ASR																					
0 1 0	ASR																					
0 1 1	ASR																					
1 0 0	DR bits 64–95																					
1 0 1	DR bits 32–63																					
1 1 0	DR bits 0–31																					
1 1 1	CR																					
5–31	Unused	Ignored by the MAU.																				

BUS TRANSACTIONS

Bus Transaction Sequence

The ASR is readable and writable in peripheral mode. The CR can be written but not read in peripheral mode. The DR can be read directly in peripheral mode and can be written by executing an LDR instruction in peripheral mode. The operand registers are not directly accessible in peripheral mode.

All peripheral mode accesses are treated as word accesses by the MAU. If the WE 32200 CPU is used to perform byte or halfword reads from the MAU in peripheral mode, these accesses are completed correctly because the CPU ignores the extra bytes returned by the MAU. If the CPU is used to perform byte or halfword writes to the MAU, these accesses terminate correctly but do not yield useful results.

4.1.2 Bus Transaction Sequence

An example of a peripheral mode transaction is shown on Figure 4-1.

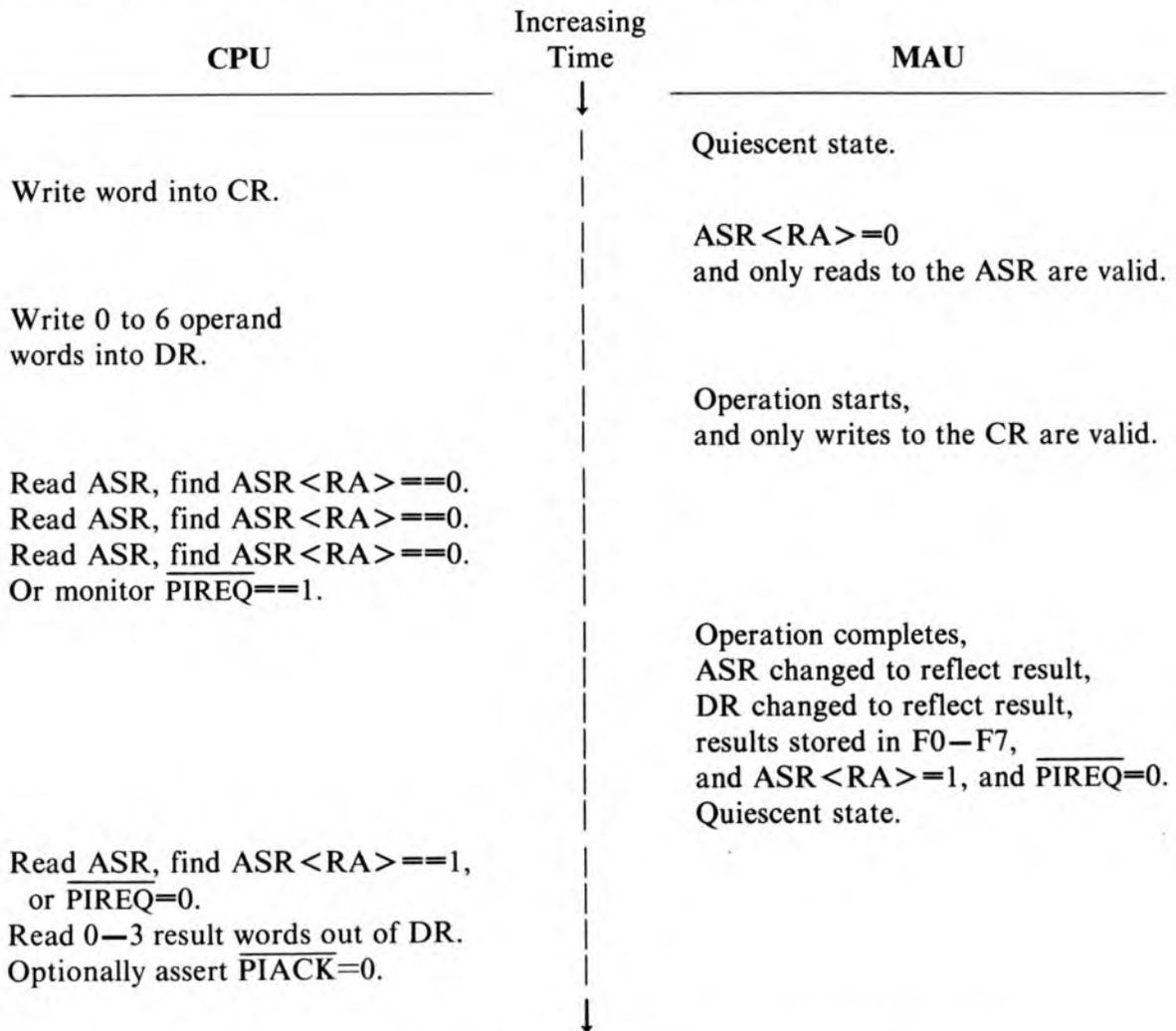


Figure 4-1. Example of Peripheral Mode Transaction Sequence

Transaction Sequence

1. The CPU initiates an MAU transaction by writing a word into the command register via a data write access with the MAU chip-select asserted.
2. The MAU terminates any peripheral mode initiated transaction currently in progress, clears the ASR<RA> bit, and enters a state in which any peripheral mode read operations addressing the ASR act normally. Other peripheral mode read operations return indeterminate values. $\overline{\text{PIREQ}}$ is not affected.
3. If the command word operand specifiers specify any source operands as being in memory, the CPU must perform the requisite number of data write accesses to the data register. The MAU latches each word into internal (hidden) registers. For single-word operands, one word must be written. For double- and triple-word operands, the words must be written in order from most to least significant. The data words can be written to any MAU address other than the CR. Words written to the CR will write the CR and start the new instruction.
4. When the required number of data write accesses have been performed, the MAU begins the specified operation. It enters a state in which peripheral mode write accesses to the CR write the CR. In addition, peripheral mode reads to the ASR get the ASR. All other peripheral mode reads and writes in this state terminate normally but do not yield useful results.
5. When the operation has been completed, the MAU changes the ASR contents to reflect the new condition codes, exception bits, and any other bits resulting from the operation just completed. The ASR<RA> bit is set, $\overline{\text{PIREQ}}$ is asserted, and the MAU enters the quiescent state. As a result, peripheral mode accesses now behave normally (i.e., they read or write the addressed object). If one of the internal operand registers (F0—F7) is the destination, the result is written to it at this time. If the result operand specifier value is in memory, the result appears in the data register.
6. The CPU may then perform data fetch accesses to the data register, reading the addressed word of the result operand value.
7. If $\overline{\text{PIREQ}}$ is monitored, then $\overline{\text{PIACK}}$ should be toggled; this negates $\overline{\text{PIREQ}}$.

If the operand size is single-word, the result appears in DR bits 95—64 and the contents of DR bits 63—0 are indeterminate. If the operand size is double-word, the result appears in DR bits 95—32 and the contents of DR bits 31—0 are indeterminate. If the operand size is triple-word, the result appears in DR bits 95—0.

Note: The RDASR and WRASR opcodes should not be used in peripheral mode to access the ASR. Instead, access to the ASR should always be performed with direct memory mapped reads/writes to MAU location 000. This is the fastest method of access.

BUS TRANSACTIONS

Peripheral Mode Read and Write

4.1.3 Peripheral Mode Read and Write

On a peripheral mode read from the MAU (see Figure 4-2), the MAU starts driving the data bus when the chip select (\overline{CS}) and data strobe (\overline{DS}) are asserted and the read/write (R/\overline{W}) signal is a logic 1. The MAU then drives valid data onto the data bus and signals the CPU by asserting data transfer acknowledge, \overline{DTACK} . When \overline{DS} is negated \overline{DTACK} is also negated and data is removed from the bus. Note that \overline{DS} must be detected to be negated by the MAU between accesses.

On a peripheral mode write to the MAU (see Figure 4-3) the MAU latches data when \overline{CS} and \overline{DS} are asserted and the R/\overline{W} signal is a logic 0. The data is latched on a rising edge of $CLK34$. \overline{DTACK} is then asserted on a falling edge of $CLK23$. When \overline{DS} is negated, \overline{DTACK} is also negated. Again, \overline{DS} must be detected to be negated between accesses.

In peripheral mode operation, R/\overline{W} and the address, $ADDR$, must be valid before \overline{DS} and/or \overline{CS} is asserted. In addition, $ADDR$, \overline{CS} and R/\overline{W} must remain valid until \overline{DS} is negated. The MAU samples \overline{DS} , \overline{CS} , R/\overline{W} and $ADDR02-ADDR04$ on the rising edge of $CLK23$. \overline{CS} , \overline{DS} , and R/\overline{W} are double latched to avoid metastability.

The MAU responds to coprocessor accesses between any peripheral mode accesses. In addition, it responds to a peripheral access to its memory mapped registers in any order; e.g., there is a response for a write to the data register even if there was no write to the command register yet. All peripheral mode accesses are ignored (i.e., no response) if the MAU is performing a coprocessor mode initiated operation.

Figure 4-4 shows the protocol used for both peripheral mode reads and writes. It is intended to show the cause and effect relationships between the two operations. No timing relationships are implied.

4.1.4 Configuration Restrictions

In peripheral mode, if nonaddressed devices assert bus exceptions, incorrect data may be written into the MAU during writes.

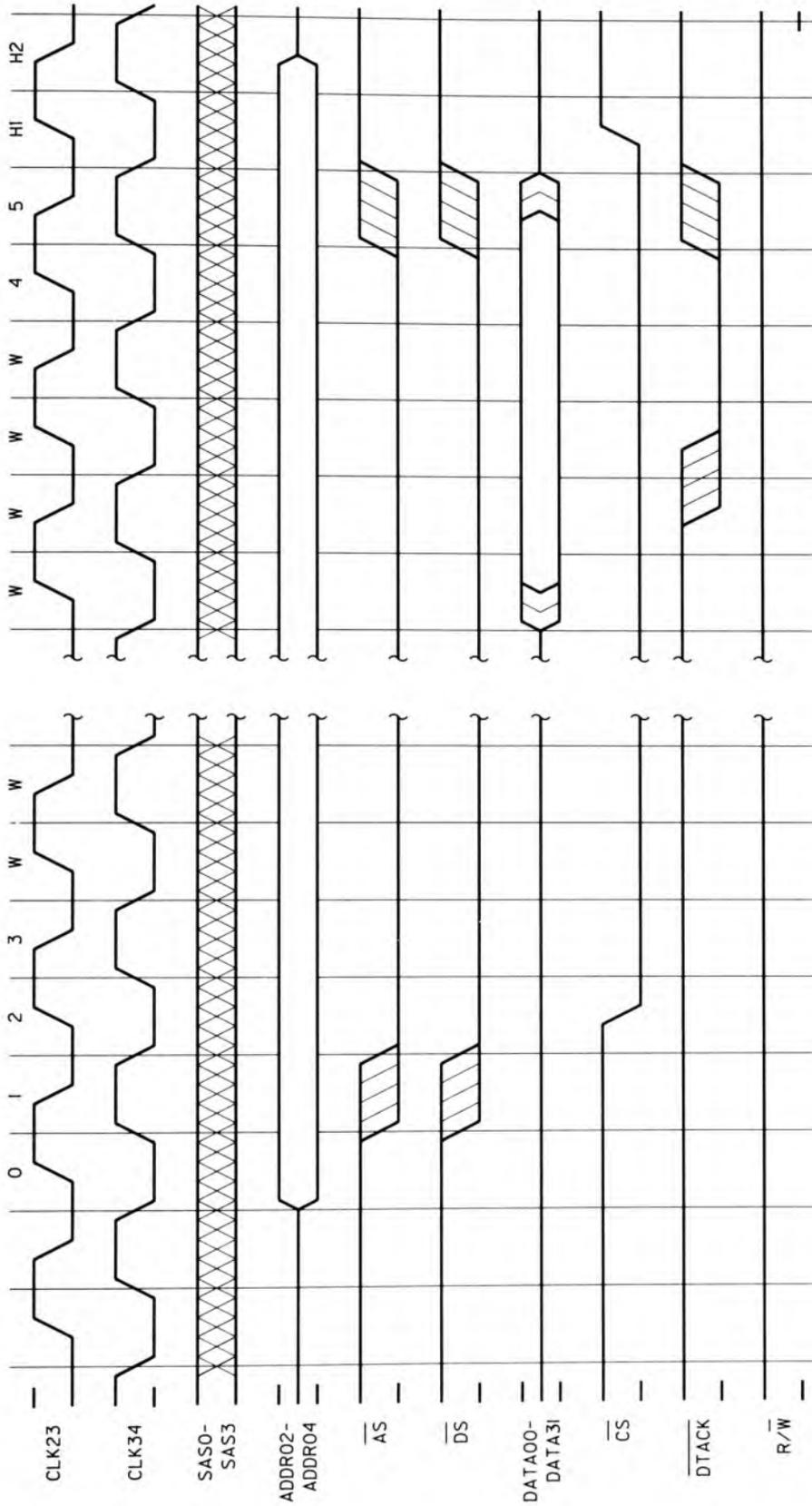


Figure 4-2. Peripheral Mode Read

BUS TRANSACTIONS
Configuration Restrictions

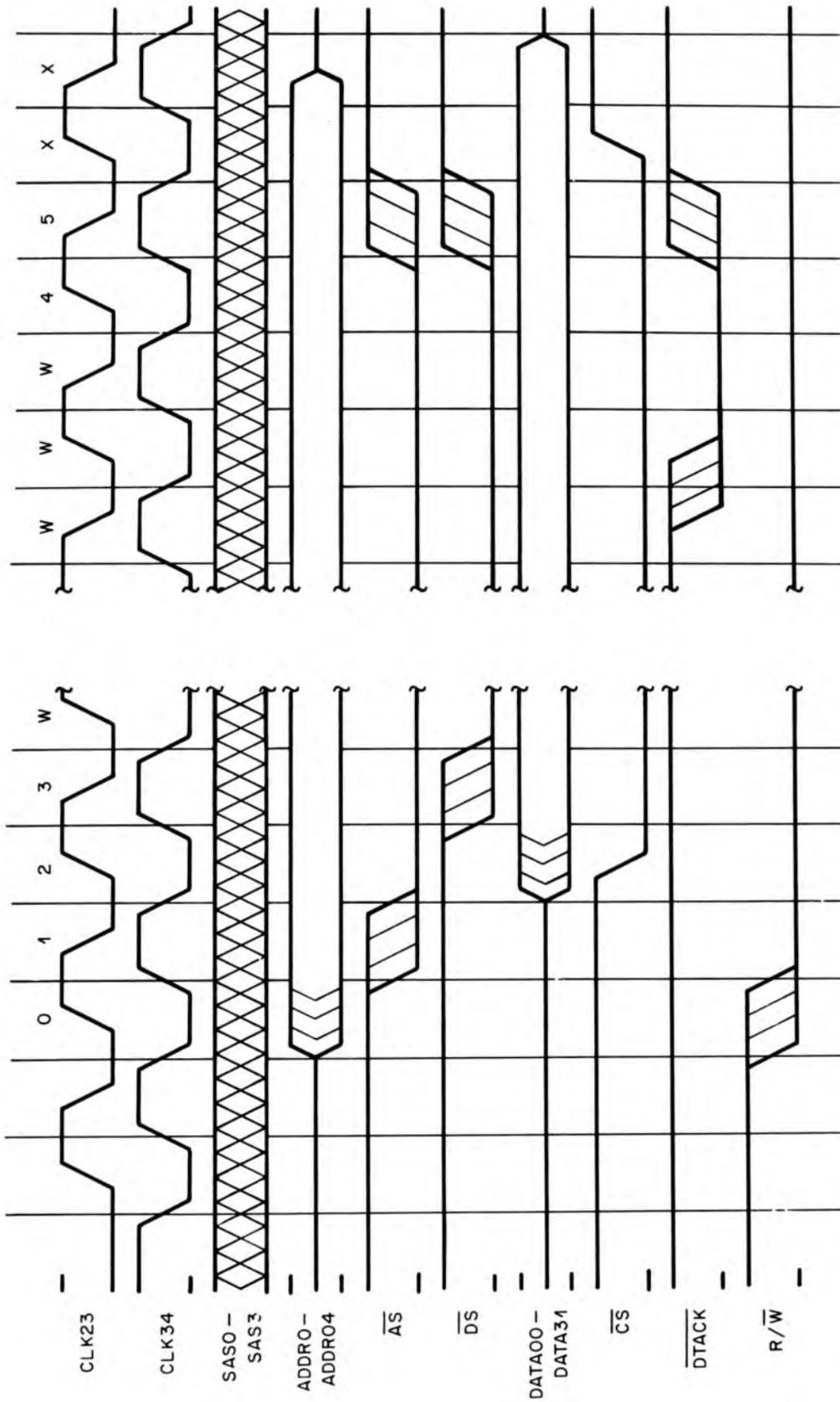


Figure 4-3. Peripheral Mode Write

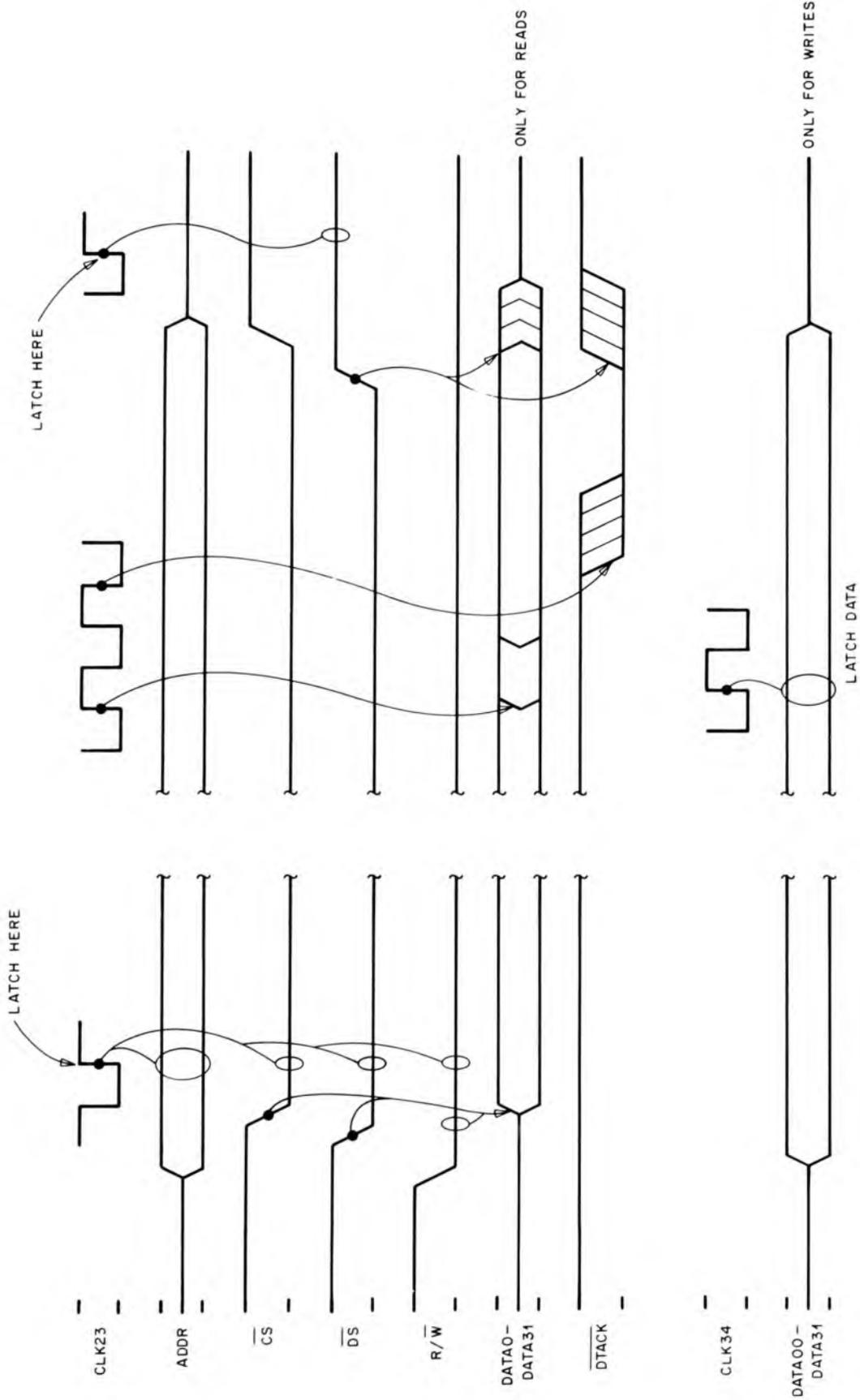


Figure 4-4. Peripheral Mode Read and Write

BUS TRANSACTIONS
Configuration Restrictions

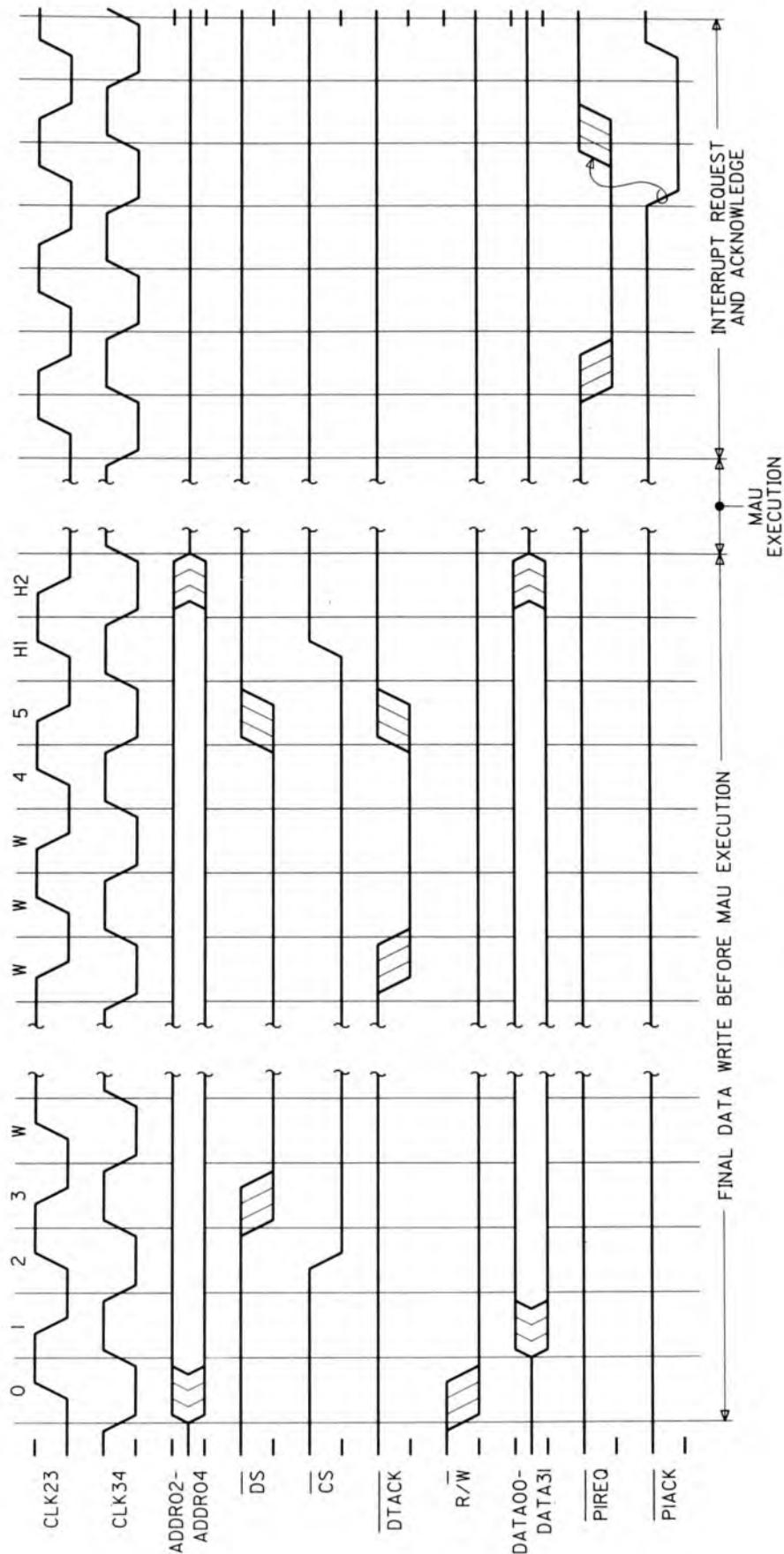


Figure 4-5. $\overline{\text{PIREQ}}$ and $\overline{\text{PIACK}}$ Timing

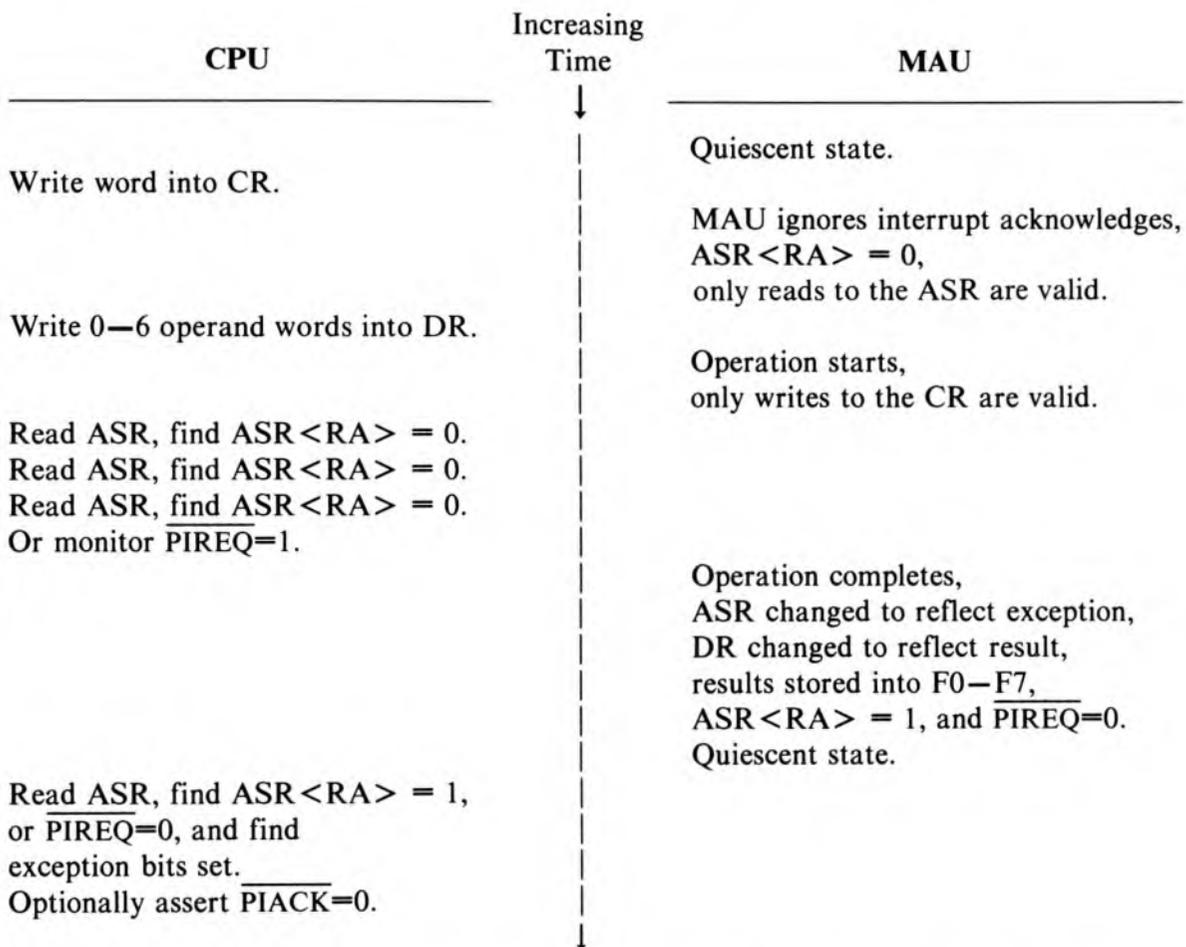
4.1.5 Peripheral Mode Interrupts

Once a transaction has been initiated via a peripheral mode write to the command register, the MAU ignores any interrupt acknowledge or autovector interrupt acknowledge bus cycles. This behavior continues after completion of the transaction, until the next MAU transaction is started.

4.1.6 Peripheral Mode Faults

The MAU does not signal any faults under the peripheral mode protocol, and does not react to any faults on the bus.

In the case of an operation that results in an exception, the exception is signaled to the CPU by setting ASR bits. Such a transaction is shown on Figure 4-6.



Note: If the exception is enabled, the result to be supplied to the trap handler is stored in the data register. If the exception is disabled, the normal result is stored in the DR.

Figure 4-6. Example of Peripheral Mode Exception Sequence

BUS TRANSACTIONS

Peripheral Mode Restartability

4.1.7 Peripheral Mode Restartability

There is never an MAU-initiated fault on a peripheral mode access. That is, reads and writes in peripheral mode always terminate normally with no fault indication.

It is possible that the CPU will read the DR successfully and then encounter a fault when it tries to write the value to memory. In this case, the CPU may restart the instruction, causing another read to the DR. The MAU will allow any number of these reads; in that sense, it supports restartability.

4.1.8 Peripheral Mode Operating System Implications

Under peripheral mode protocol, the MAU does not abort its operation when an interrupt acknowledge cycle occurs. This is because the peripheral mode protocol assumes that the CPU will be executing a stream of discrete instructions to implement the protocol, and thus would not be able to back up to the first instruction in the event of an interrupt. The CPU is free to allow or disallow interrupts during MAU operations; however, MAU context saving is simplified if the system can arrange to have interrupts disabled during MAU transactions. It may be possible to use the $\overline{\text{PIREQ}}$ signal to do this.

The state of the MAU must be saved and restored later if an interrupt causes control of the MAU to be given to another process. The MAU may be in the middle of an operation when the CPU is interrupted. If $\text{ASR}\langle\text{RA}\rangle = 0$ or if $\overline{\text{PIREQ}} = 1$ (the MAU is waiting for source operands or is in the middle of an operation), the MAU's state cannot be saved and restored later without explicit software provisions. That is, the context includes the CR value and knowledge of how many operands have been loaded, and these cannot be determined by examining the MAU alone. If the first CPU instruction of the sequence can be found, the partially executed transaction can be backed out to that point so that execution will resume there. If this is done, the context to be saved and restored consists of the ASR, DR, and F0–F7. It may be necessary to run context save and restore sequences that may not be interrupted. If possible, the easiest approach is to wait for the MAU to finish execution ($\text{ASR}\langle\text{RA}\rangle = 1$ or $\overline{\text{PIREQ}} = 0$).

If an interrupt occurs and $\text{ASR}\langle\text{RA}\rangle = 1$, the MAU is in its quiescent state. The transaction may have completed (all reads from the DR are finished) or some reads from the DR may not have occurred yet. The context to be saved and restored consists of the ASR, DR, and F0–F7.

The MAU state (ASR, DR, and F0–F7) may be saved by:

- Reading the ASR via peripheral mode and saving it.
- Clearing the ASR.
- Reading the three DR words via peripheral mode and saving them.
- Performing eight MAU move operations, with sources being F0–F7 and destinations being memory locations (the values appear in the DR, which should then be moved to the memory location via peripheral mode accesses). This saves registers F0–F7.

The MAU state (ASR, DR, and F0—F7) may be restored by:

- Perform eight MAU move operations, with sources being memory locations (the value must be written into the DR via peripheral mode accesses) and destinations being F0—F7. This restores registers F0—F7.
- Perform an MAU move operation, with the source being a memory location (the value must be written into the DR via peripheral mode accesses) and the destination being a memory location (the value appears in the DR). This restores the DR contents.
- Write the saved ASR value to the ASR via peripheral mode.

4.2 COPROCESSOR MODE TRANSACTIONS

The MAU, configured in the coprocessor mode, is capable of four bus transactions:

- Coprocessor broadcast
- Coprocessor data fetch
- Coprocessor status read
- Coprocessor data write.

4.2.1 Bus Transaction Sequence

An example of a coprocessor mode transaction is shown on Figure 4-7.

BUS TRANSACTIONS

Transaction Sequence

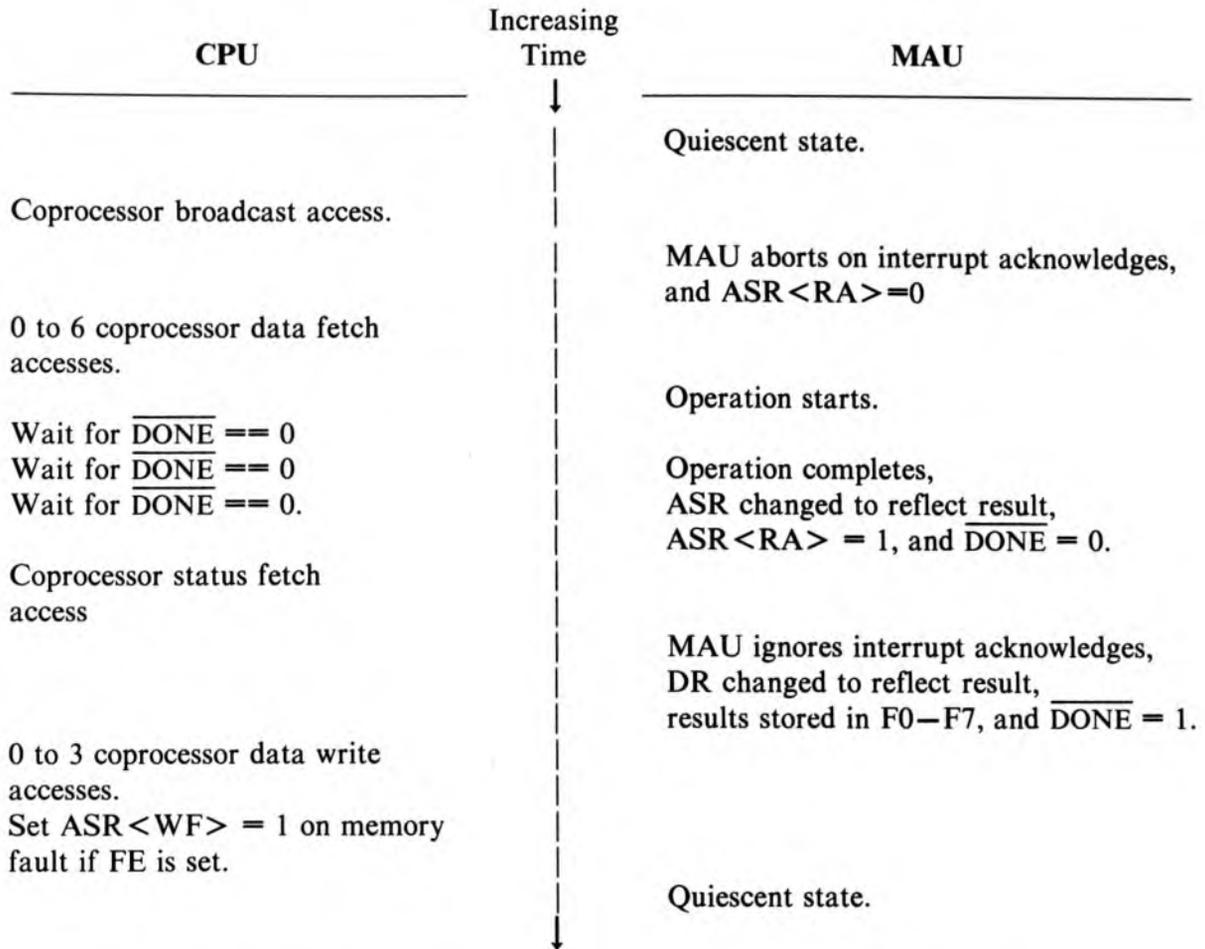


Figure 4-7. Example of Coprocessor Mode Transaction Sequence

Transaction Sequence

1. The CPU performs a coprocessor broadcast access with a command word on the data bus.
2. If the ID field of the command word is not equal to the MAU ID, the MAU enters its quiescent state (the rest of this sequence is not performed).
3. If the ID field of the command word is equal to the MAU ID, the MAU stores the command word in the command register and clears $ASR \langle RA \rangle$. It also enters a state in which peripheral mode writes are ignored (terminate normally, but without writing), and peripheral mode reads obtain indeterminate results (terminate normally).
4. If the command word operand specifiers specify any source operands as being in memory, the CPU must perform the requisite number of coprocessor data fetch accesses. The MAU latches each word on the data bus into internal (hidden) registers. The MAU expects all memory operands to be word aligned and, therefore, will not accept any byte or halfword operands. If the DSIZE bits indicate any transaction size other than a word and if $ASR \langle FE \rangle = 1$, the MAU will signal a fault for the data fetch access and set the UW bit.

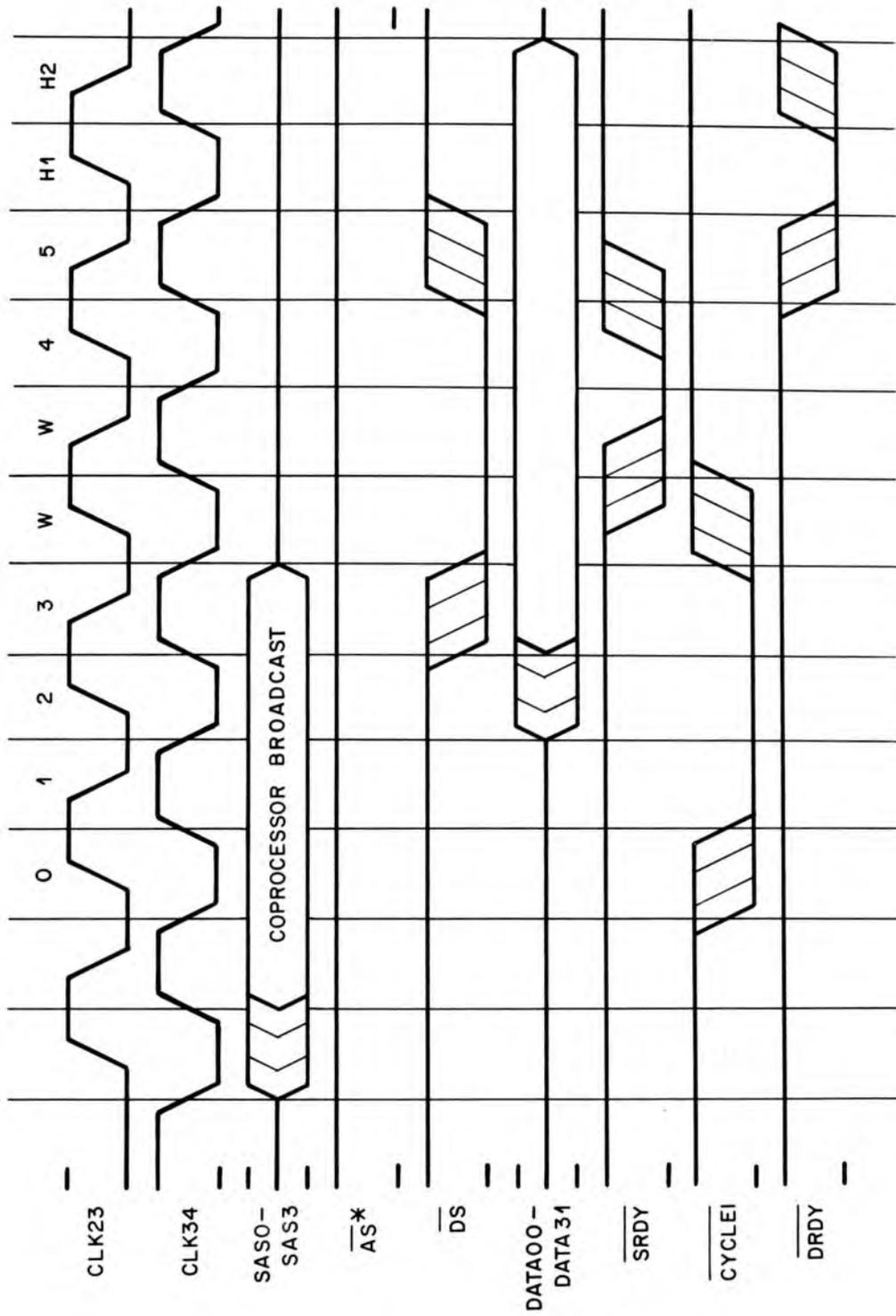
5. When the required number of coprocessor data fetch accesses have been performed, the MAU begins the specified operation. During that time, the CPU monitors the $\overline{\text{DONE}}$ signal.
6. When the operation has been completed, the MAU changes the ASR contents to reflect the new condition codes, exception bits, and any other bits resulting from the operation just completed. Then the MAU sets ASR<RA> and asserts $\overline{\text{DONE}}$.
7. The CPU must then perform a coprocessor status fetch access. The MAU reacts to this access by returning the ASR contents on the data bus and completing the access. Once the status fetch access begins, the CPU must block the interrupts until the execution of the instruction in the MAU is complete. $\overline{\text{DONE}}$ is also negated at this time.
8. If the destination is an internal MAU register (F0–F7), the result is stored in the register at this point.
9. If the command word operand specifiers specify the destination as being in memory, the CPU must perform the requisite number of coprocessor data write accesses. The MAU drives each word of the result onto the data bus at the proper time. The MAU expects all memory operands to be word aligned, and therefore will not accept any byte or halfword operands. If the DSIZE bits indicate any transaction size other than a word and if ASR<FE> = 1, the MAU will signal a fault for the data write access and set the UW bit. If a fault occurs on a data write due to either the MAU or the system, and ASR<FE> = 1, then ASR<WF> is set.
10. After the result is stored (or after completion of the coprocessor status fetch access, if there was no result), the MAU enters the quiescent state. As a side effect of returning to the quiescent state, peripheral mode accesses now behave normally (i.e., they read or write the addressed object).

4.2.2 Coprocessor Broadcast Transaction

The MAU initially remains in an idle state and monitors the $\overline{\text{CYCLEI}}$ signal on every falling edge of CLK23. If $\overline{\text{CYCLEI}}$ is asserted and the access status code (SAS0–SAS3) signals indicate that the current transaction is a coprocessor broadcast transaction, the MAU starts up. On the rising edge of CLK23, the data bus is tested 1.5 cycles later and, if the ID field on the data bus matches the ID of the MAU, the MAU responds to the CPU with $\overline{\text{SRDY}}$ (asserted on a rising edge of CLK23). At the trailing edge of state 4, the data bus is then latched into the command register. If $\overline{\text{DRDY}}$, sampled on the rising edge of CLK23, is not asserted at the end of the transaction, the data latched is ignored. If the IDs do not match, the MAU returns to the idle state monitoring the $\overline{\text{CYCLEI}}$ signal. In addition, the MAU is restarted from this point whenever a coprocessor broadcast status is issued. If the MAU was in the middle of an instruction, that instruction is aborted. Also note that the MAU instruction is aborted if an autovector interrupt acknowledge transaction (i.e., SAS3=0, SAS2=0, SAS1=1, and SAS0=0) or an interrupt acknowledge transaction (i.e., SAS3=1, SAS2=0, SAS1=1, and SAS0=1) is performed.

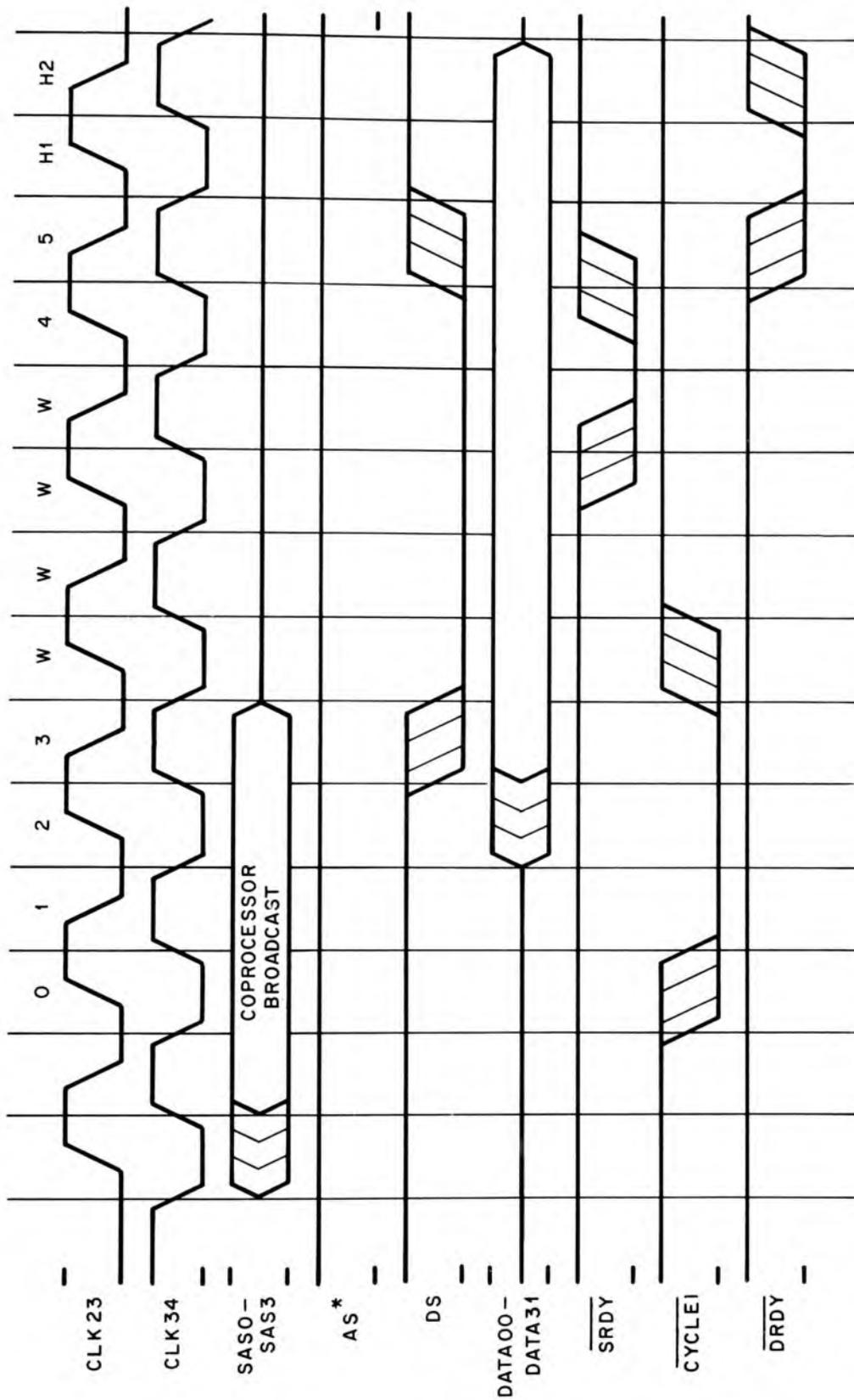
The protocol for coprocessor broadcast transactions with 1 wait state, 2 wait states, and bus exceptions are shown in Figures 4-8 through 4-10.

BUS TRANSACTIONS
Coprocessor Broadcast Transaction



*Shown only for additional clarity.

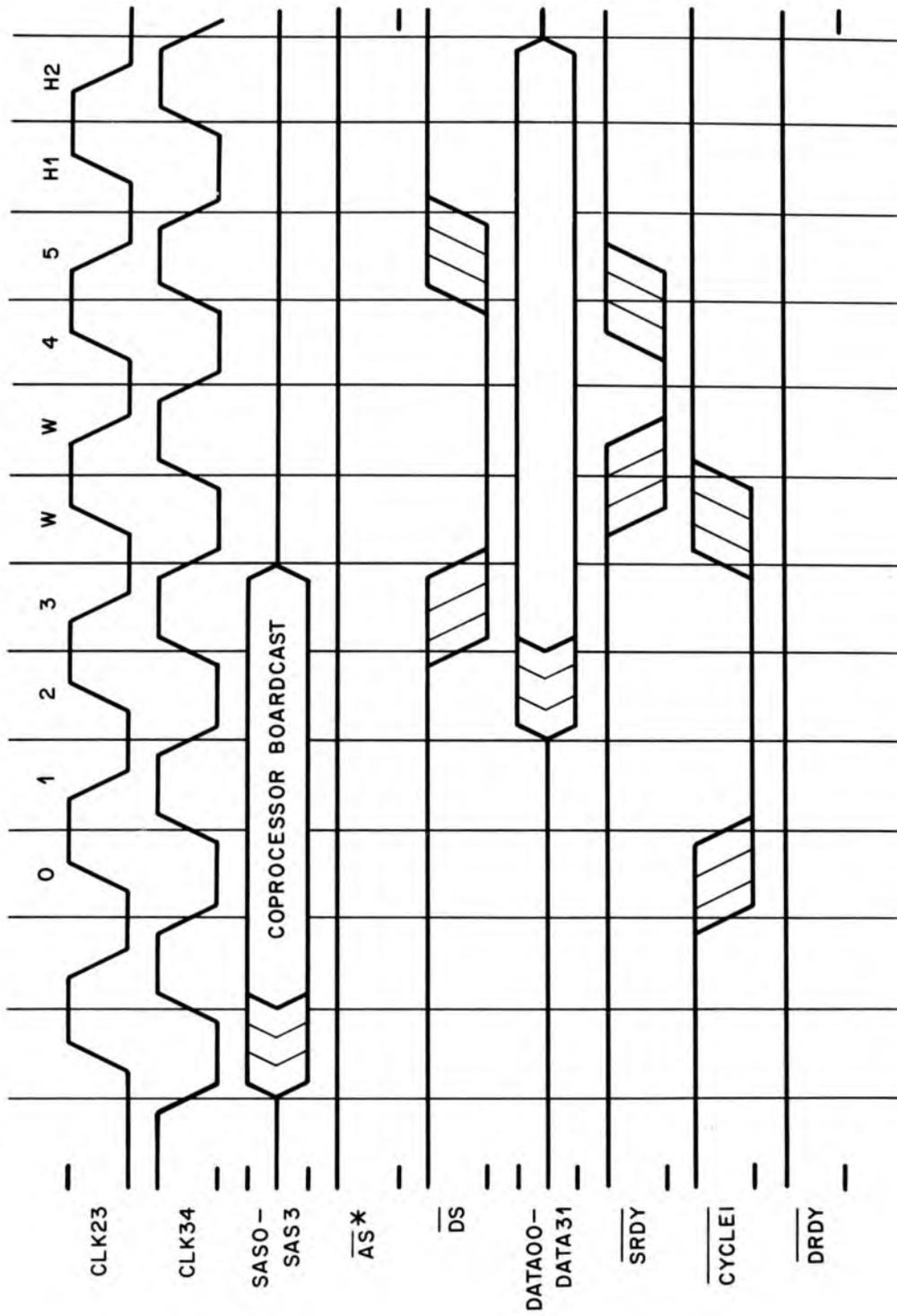
Figure 4-8. Coprocessor Broadcast (1 Wait State)



*Shown only for additional clarity.

Figure 4-9. Coprocessor Broadcast (2 Wait States)

BUS TRANSACTIONS
Coprocessor Broadcast Transaction



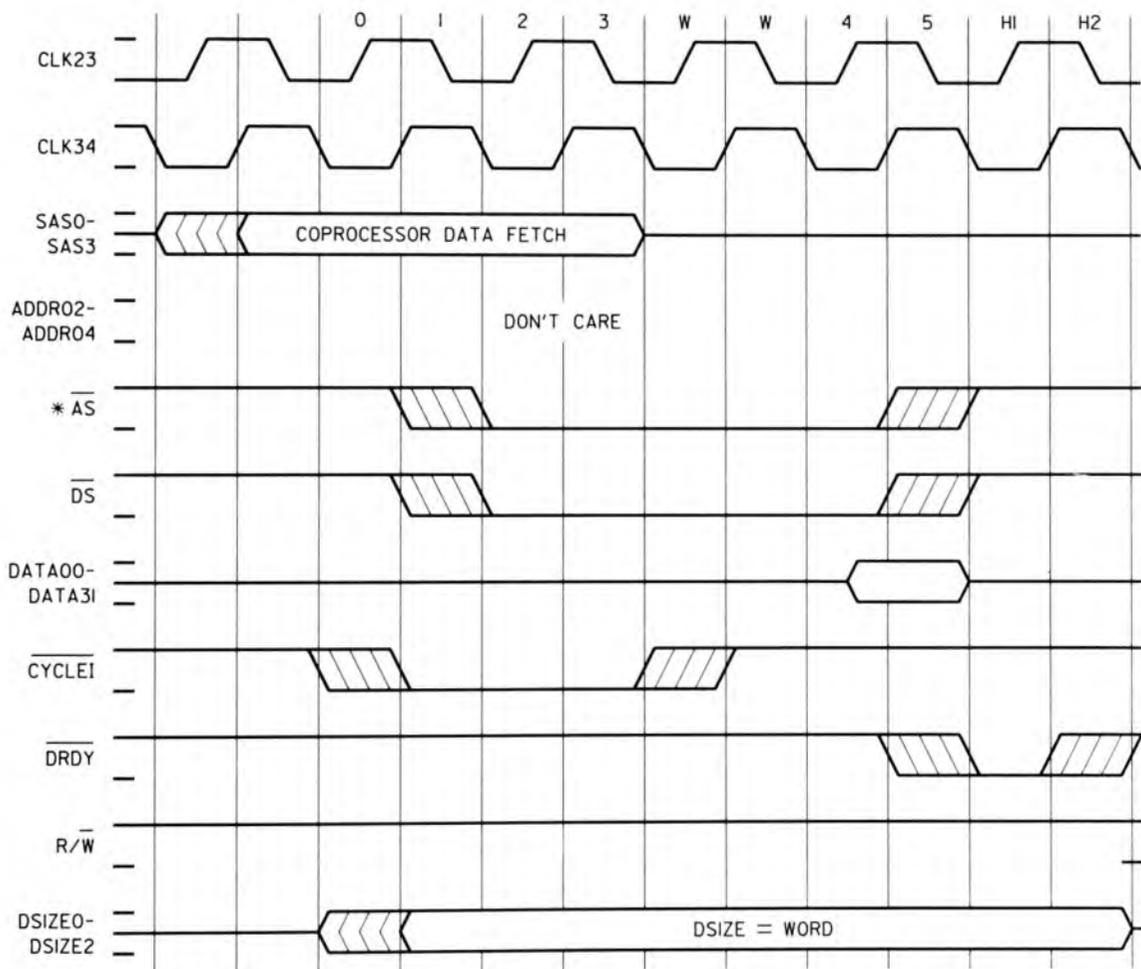
*Shown only for additional clarity.

Figure 4-10. Coprocessor Broadcast with Bus Exceptions (1 Wait State)

4.2.3 Coprocessor Data Fetch Transaction

Once the MAU has started, if it needs to load any operands, it monitors $\overline{\text{CYCLEI}}$ and the access status code (SAS0—SAS3) signals. When $\overline{\text{CYCLEI}}$ is asserted and the SAS0—SAS3 signals indicate coprocessor data fetch, 0.5 cycles later, the MAU begins to continuously latch the $\overline{\text{DS}}$ and $\overline{\text{DRDY}}$ signal on every rising edge of CLK23. If $\overline{\text{DS}}$ is asserted, the contents of the data bus are latched on every rising edge of CLK34. When $\overline{\text{DS}}$ is negated and $\overline{\text{DRDY}}$ is asserted, the last sample of the data bus contains valid data. If $\overline{\text{DRDY}}$ is not detected to be asserted and $\overline{\text{DS}}$ has been negated, the data is invalid and the next transaction of coprocessor data fetch (if there is one) repeats this access.

The protocol for coprocessor data fetch transactions with 0 wait states, 1 wait state, $\text{R}/\overline{\text{W}} = 1$, bus exceptions, and nonword data is shown on Figures 4-11 through 4-15.



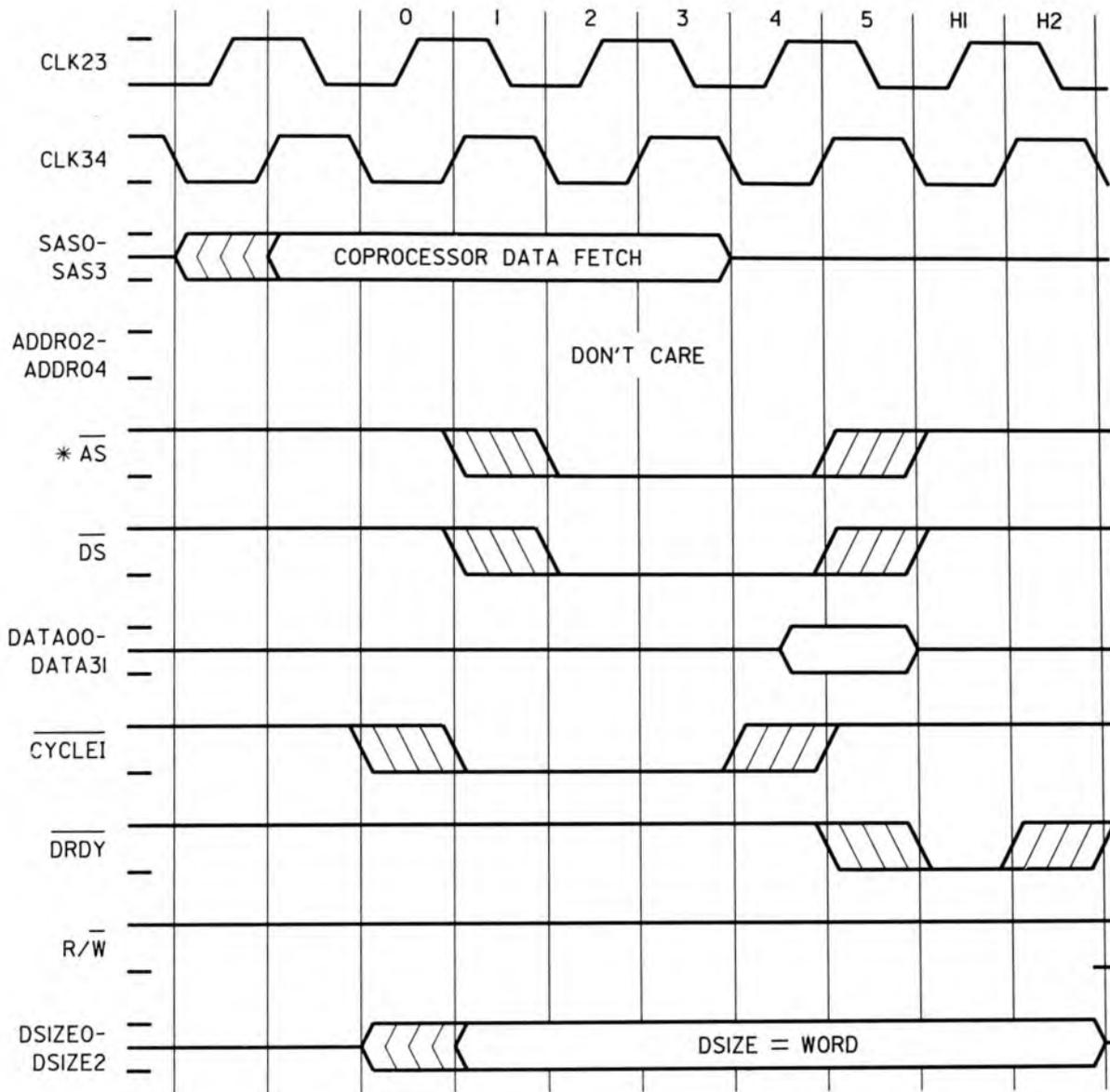
Memory responds with $\overline{\text{SRDY}}$ or $\overline{\text{DTACK}}$.

*Shown only for additional clarity.

Figure 4-11. Coprocessor Data Fetch (1 Wait State)

BUS TRANSACTIONS

Coprocessor Data Fetch Transaction



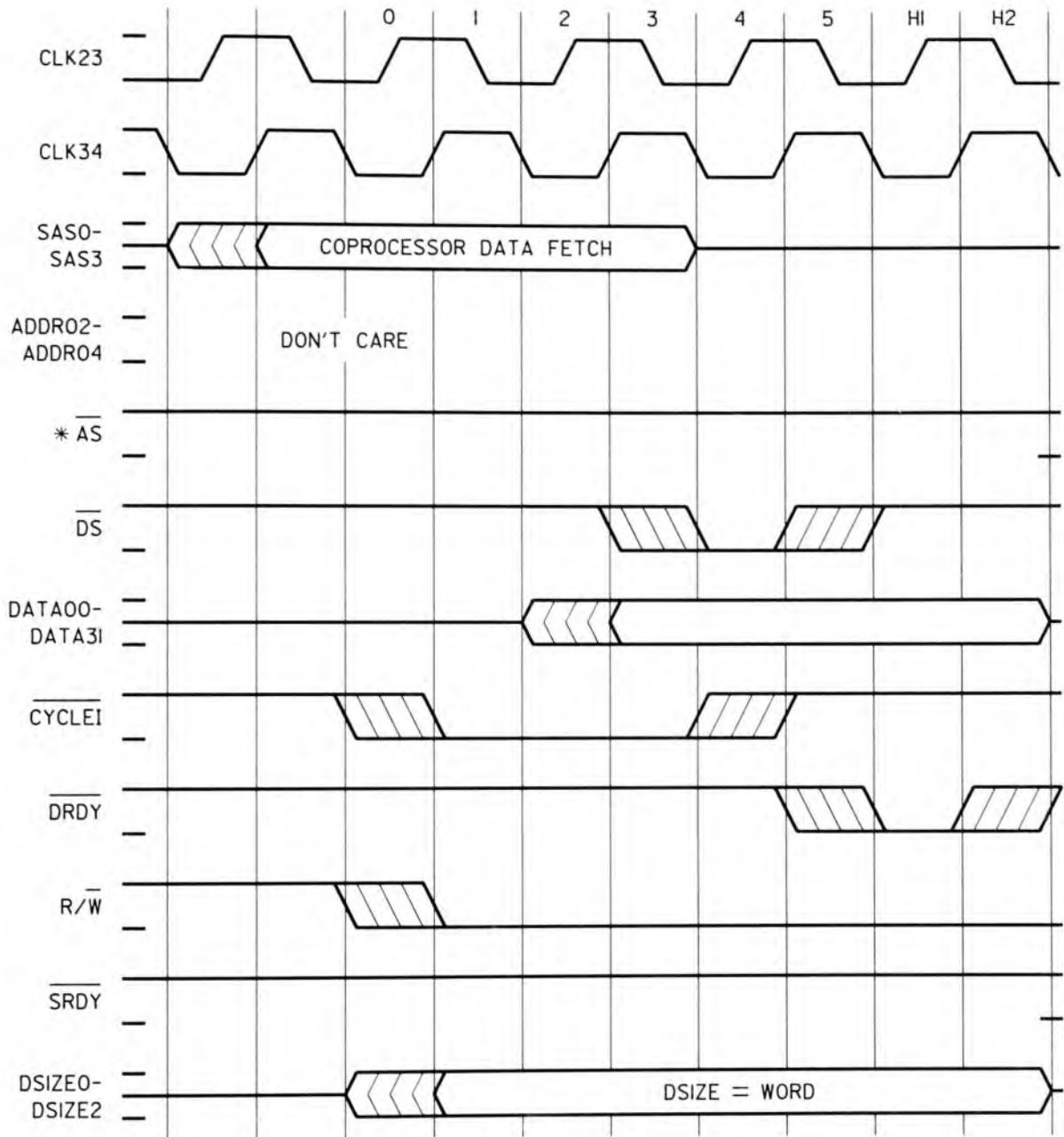
Memory responds with \overline{SRDY} or \overline{DTACK} .

*Shown only for additional clarity.

Figure 4-12. Coprocessor Data Fetch (0 Wait States)

BUS TRANSACTIONS

Coprocessor Data Fetch Transaction

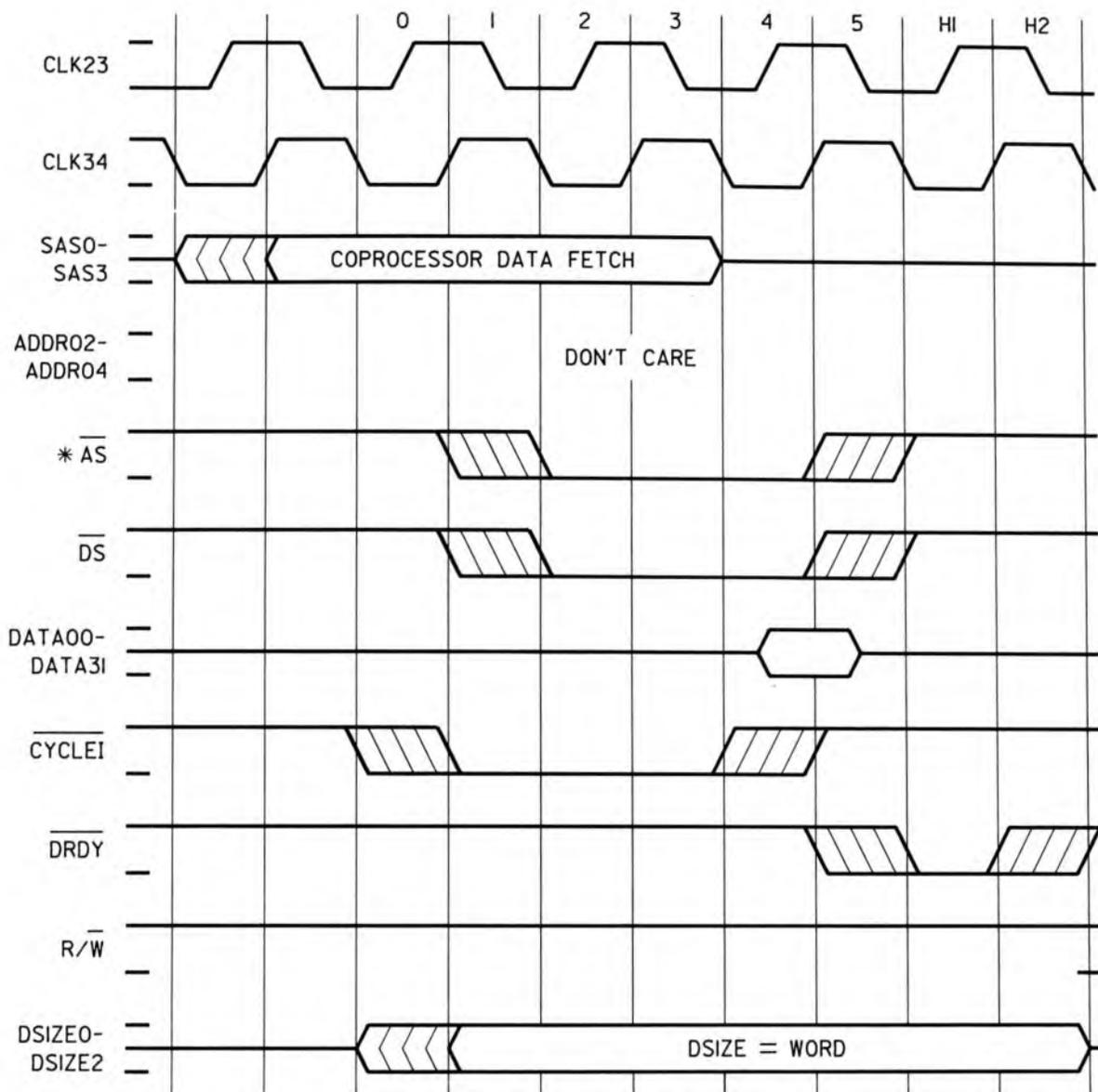


Note: This transaction never occurs with the *WE* 32200 CPU.

*Shown only for additional clarity.

Figure 4-13. Coprocessor Data Fetch ($R/\bar{W} = 0$)

BUS TRANSACTIONS
Coprocessor Data Fetch Transaction



Memory responds with $\overline{\text{FAULT}}$, $\overline{\text{RETRY}}$, or $\overline{\text{RRREQ}}$ terminating the access.

*Shown only for additional clarity.

Figure 4-14. Coprocessor Data Fetch with Bus Exceptions (0 Wait States)

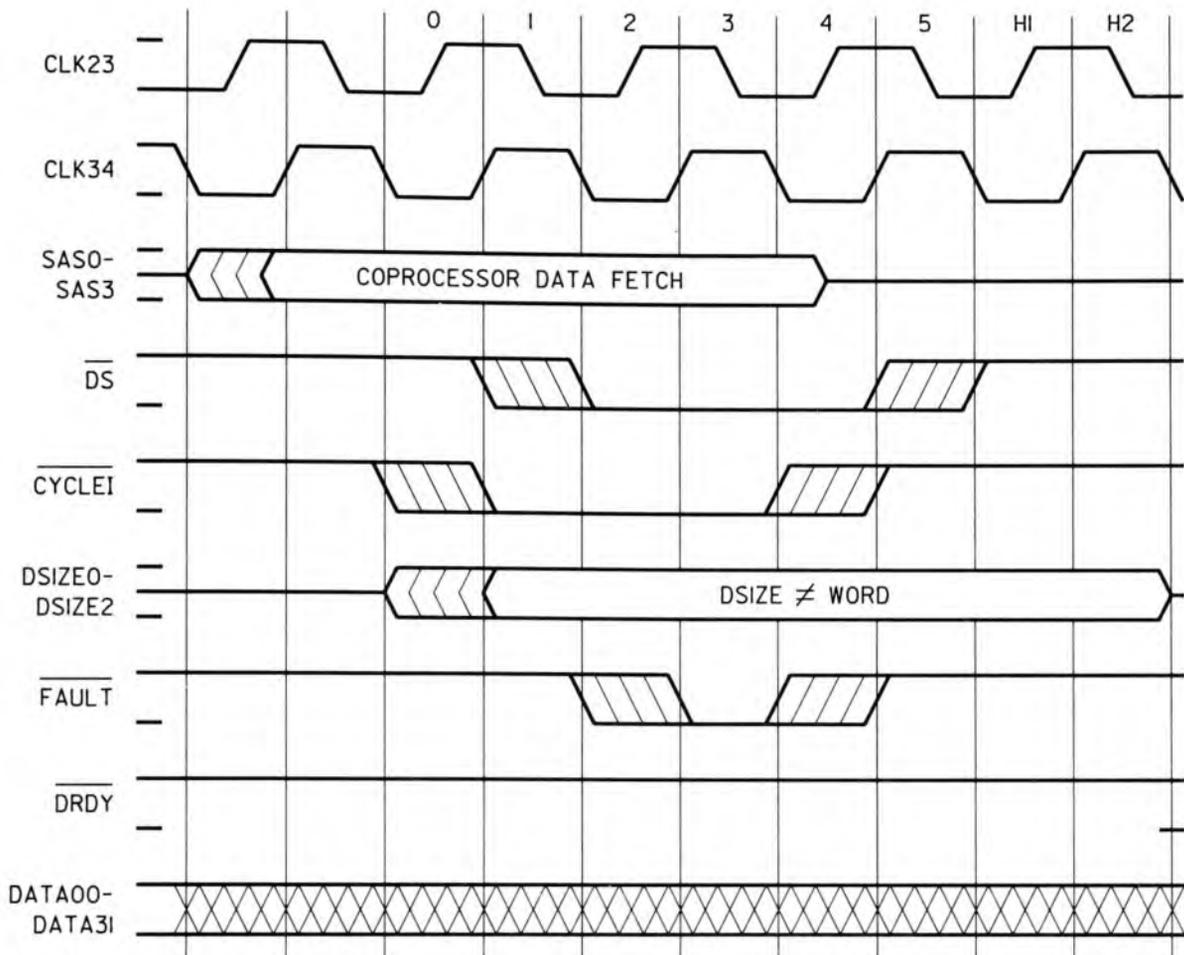


Figure 4-15. Coprocessor Data Fetch with Nonword Data (0 Wait States)

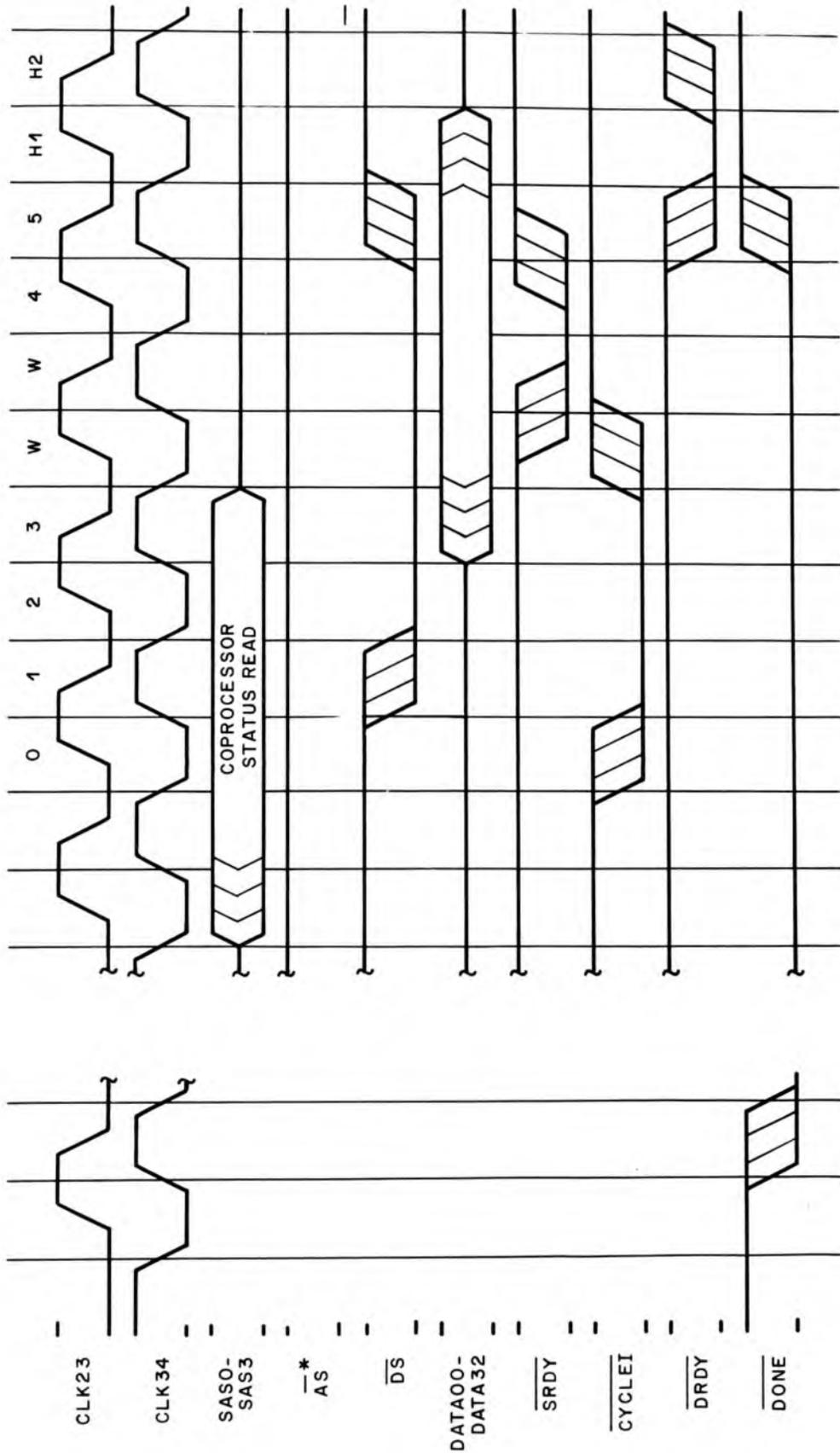
4.2.4 Coprocessor Status Read Transaction

Once the MAU has signaled to the CPU – via the assertion of the coprocessor $\overline{\text{DONE}}$ signal – that it has completed the instruction, the CPU reads the MAU status. This is done by a read transaction with the access status code SAS0–SAS3 signals indicating coprocessor status fetch. Again, by sampling $\overline{\text{CYCLEI}}$, the MAU knows the start of this access. The MAU puts its condition codes on the data bus onto the leading edge of state 3 and asserts $\overline{\text{SRDY}}$ on the next rising edge of CLK23. The MAU then negates $\overline{\text{DONE}}$ on the leading edge of state 5, i.e., the MAU 3-states and the signal is pulled to a logic 1 by a pull-up resistor. $\overline{\text{SRDY}}$ is negated after being asserted for one cycle and the data is taken off the data bus on the trailing edge of state 5, i.e., the MAU 3-states its data outputs (DATA00–DATA31). If an exception has occurred during the execution of the instruction, the MAU asserts the FAULT signal along with the $\overline{\text{SRDY}}$ signal.

Note: The MAU is able to respond correctly if the CPU retries this transaction.

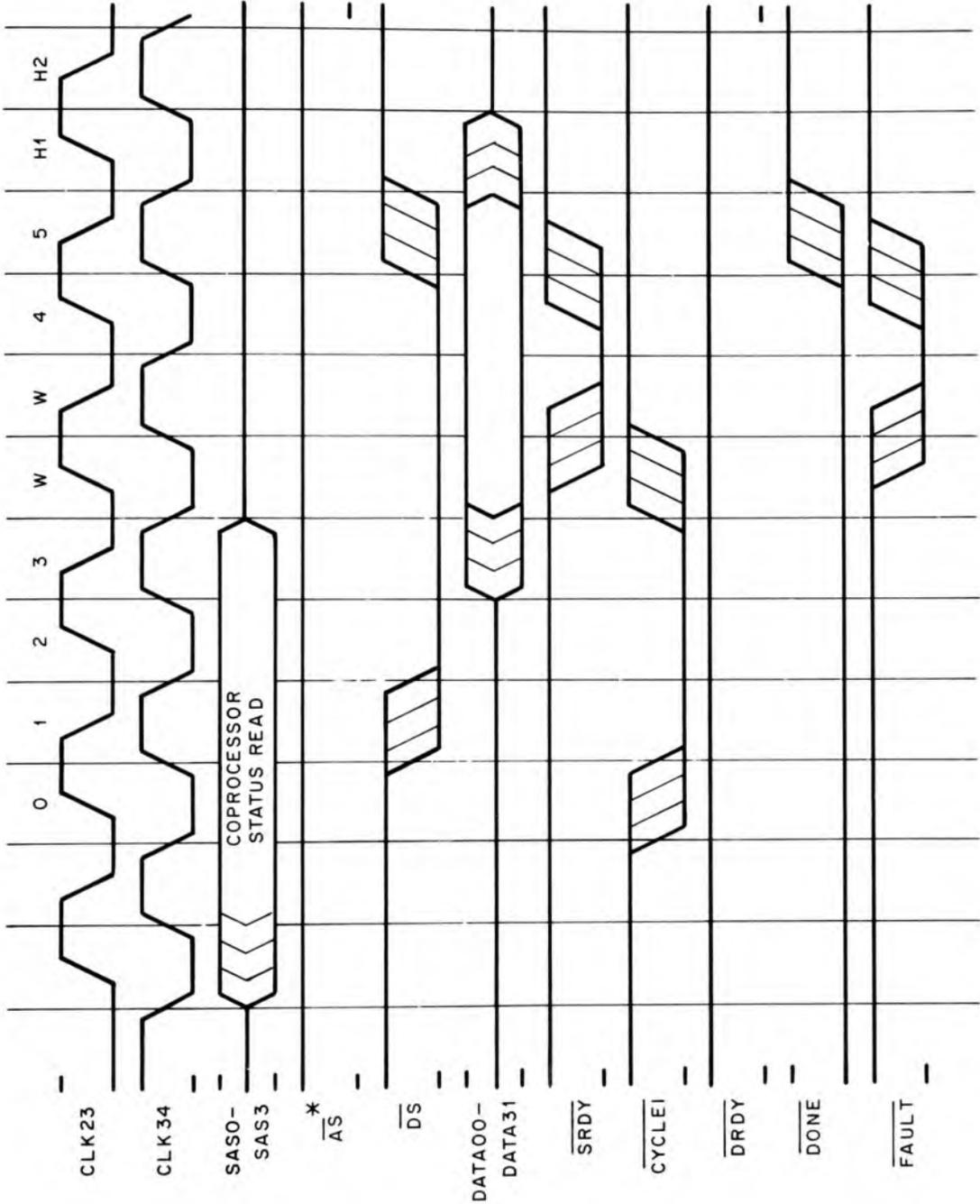
Figure 4-16 and 4-17 show the protocol for a coprocessor status read with and without exceptions.

BUS TRANSACTIONS
Coprocessor Status Read



*Shown only for additional clarity.

Figure 4-16. $\overline{\text{DONE}}$ Assertion and Coprocessor Status Read (No Exceptions)



*Shown only for additional clarity.

Figure 4-17. Coprocessor Status Read (With Exceptions)

BUS TRANSACTIONS

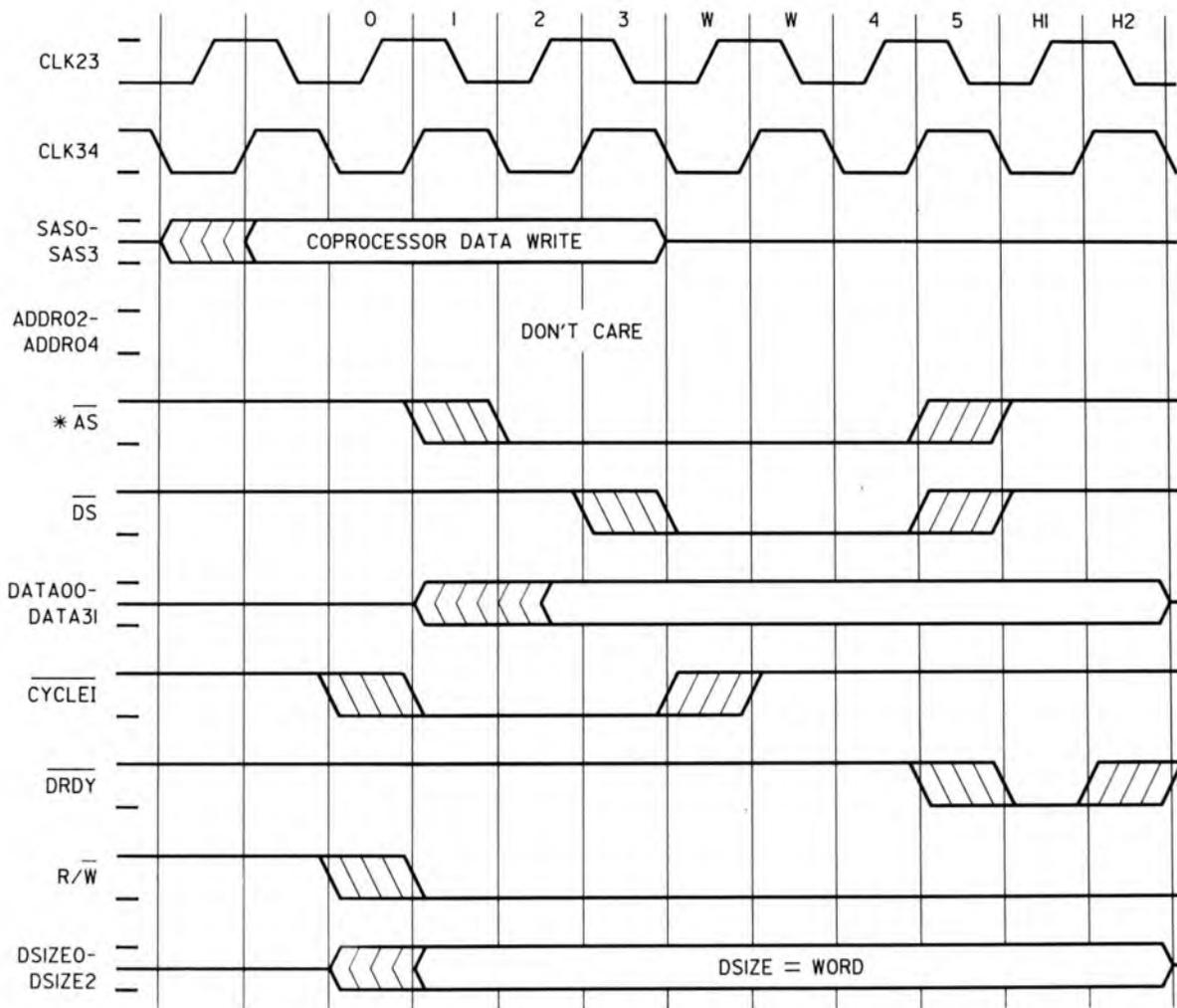
Coprocessor Data Write

4.2.5 Coprocessor Data Write Transaction

Figures 4-18 through Figure 4-23 show the protocol for coprocessor data write transactions. The protocol is described in detail below.

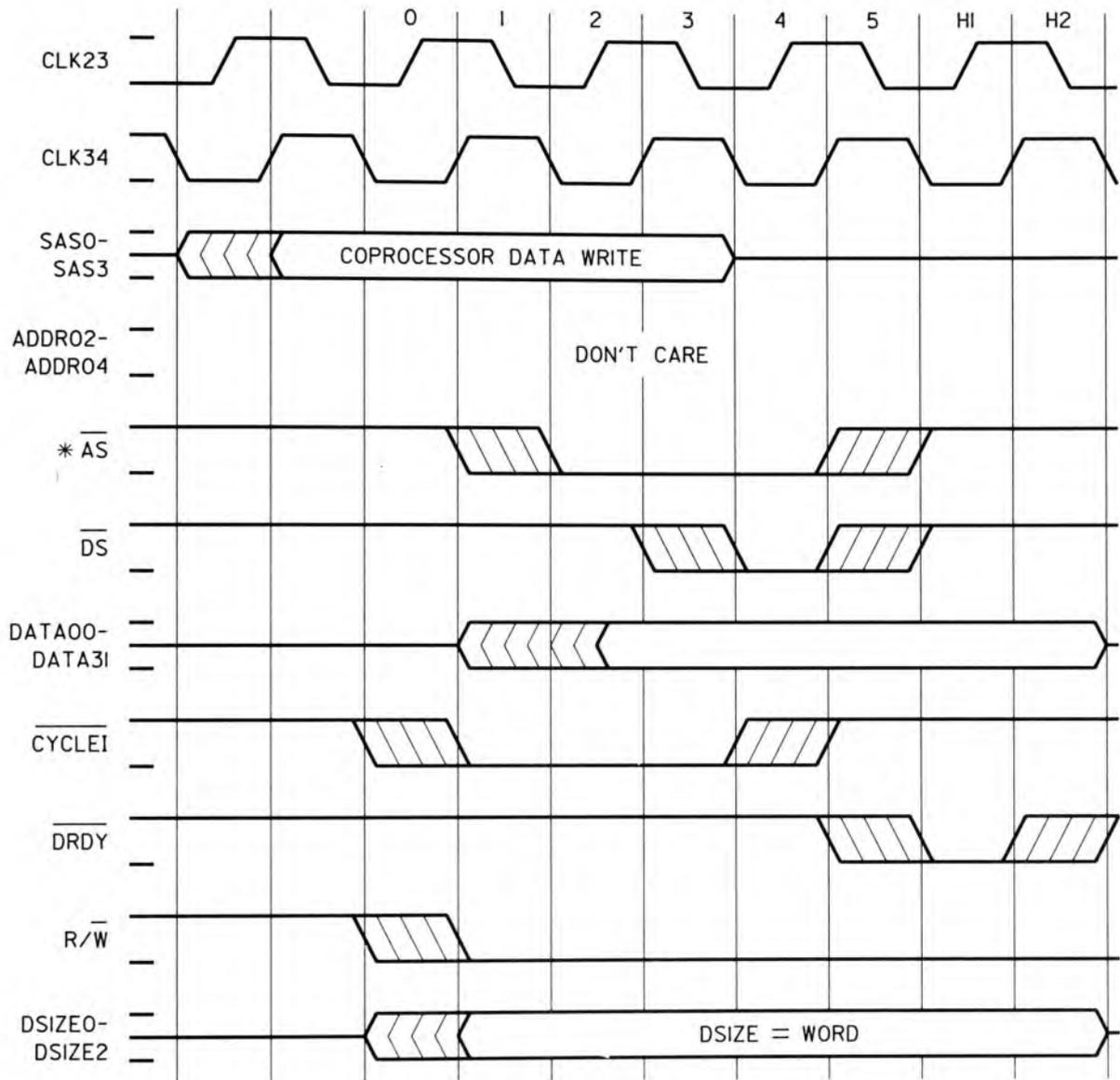
If the MAU needs to store the result, the CPU issues a coprocessor data write transaction, indicated by the SAS3—SAS0 signals. Data is driven onto the data bus at the beginning of state 1 by the MAU; the data strobe and data ready signal are sampled 1.75 cycles later on every rising edge of CLK23. When \overline{DS} is negated, data is taken off the bus (i.e., 3-stated) on the next falling edge of CLK34. If \overline{DRDY} is not detected to be asserted at the end of the transaction, the same data is put on the data bus if another coprocessor data write transaction is started.

Assertion of data bus shadow, \overline{DSHAD} , causes the MAU to immediately 3-state its data outputs until it detects that \overline{DSHAD} is negated, upon which it immediately drives its data outputs. In addition, assertion of \overline{DSHAD} causes \overline{DRDY} to be ignored.



*Shown only for additional clarity.

Figure 4-18. Coprocessor Data Write (1 Wait State)

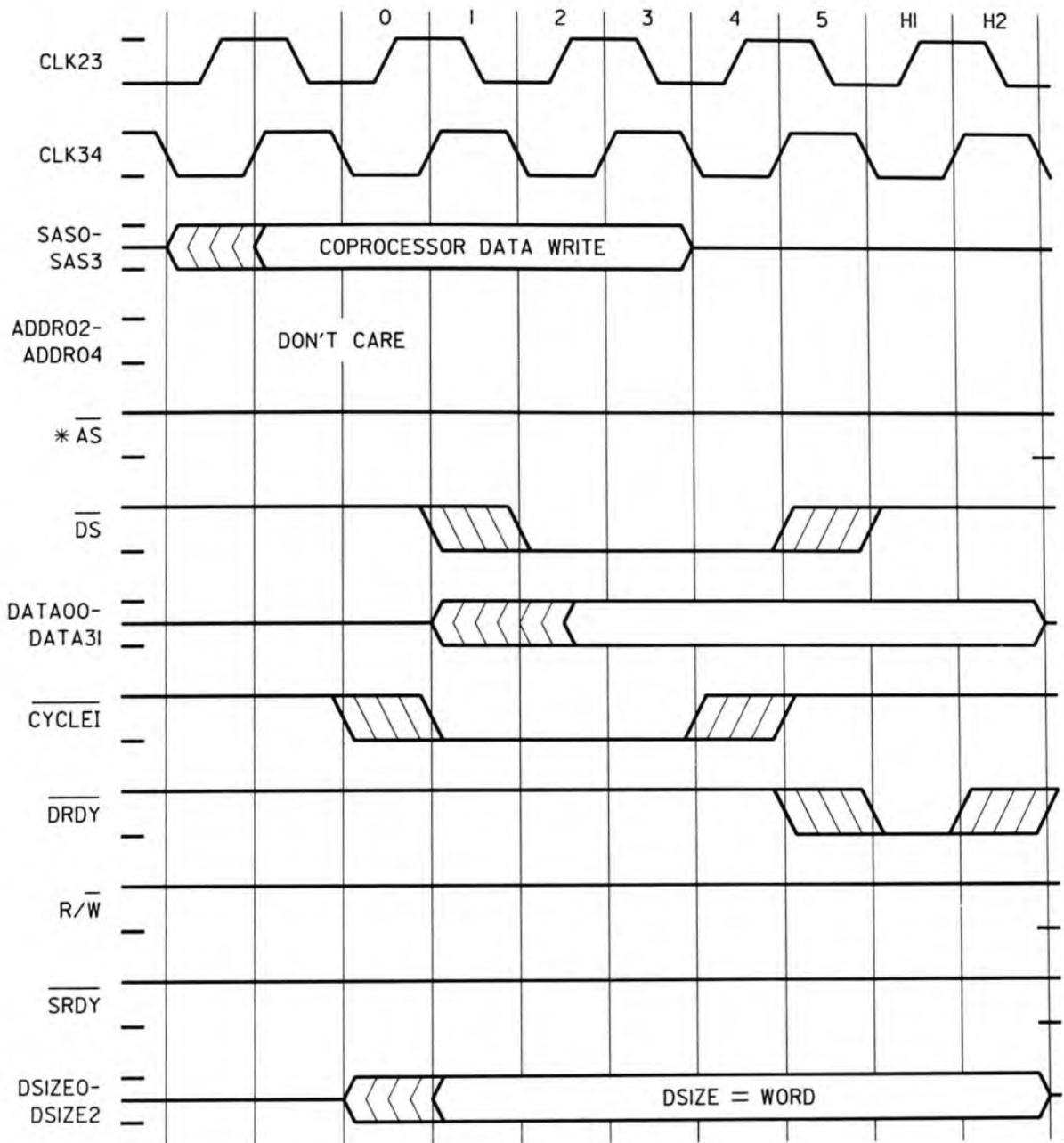


Memory responds with $\overline{\text{SRDY}}$ or $\overline{\text{DTACK}}$.

*Shown only for additional clarity.

Figure 4-19. Coprocessor Data Write (0 Wait States)

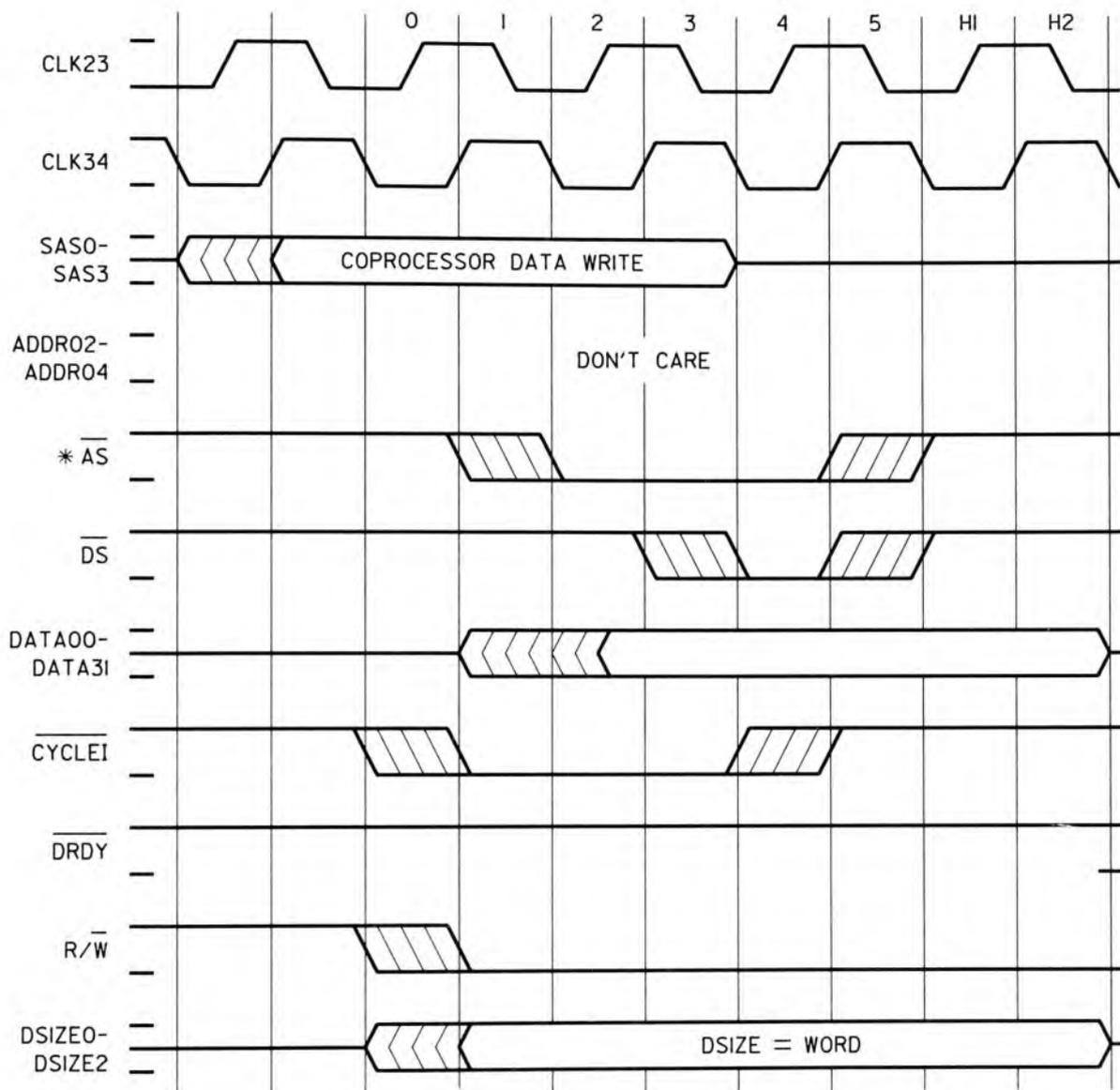
BUS TRANSACTIONS
Coprocessor Data Write



*Shown only for additional clarity.

Note: This transaction never occurs with the *WE* 32200 CPU.

Figure 4-20. Coprocessor Data Write ($R/\bar{W} = 1$)



Memory responds with $\overline{\text{FAULT}}$, $\overline{\text{RETRY}}$, or $\overline{\text{RRREQ}}$ terminating the access.

*Shown only for additional clarity.

Figure 4-21. Coprocessor Data Write With Bus Exceptions (0 Wait States)

BUS TRANSACTIONS
Coprocessor Data Write

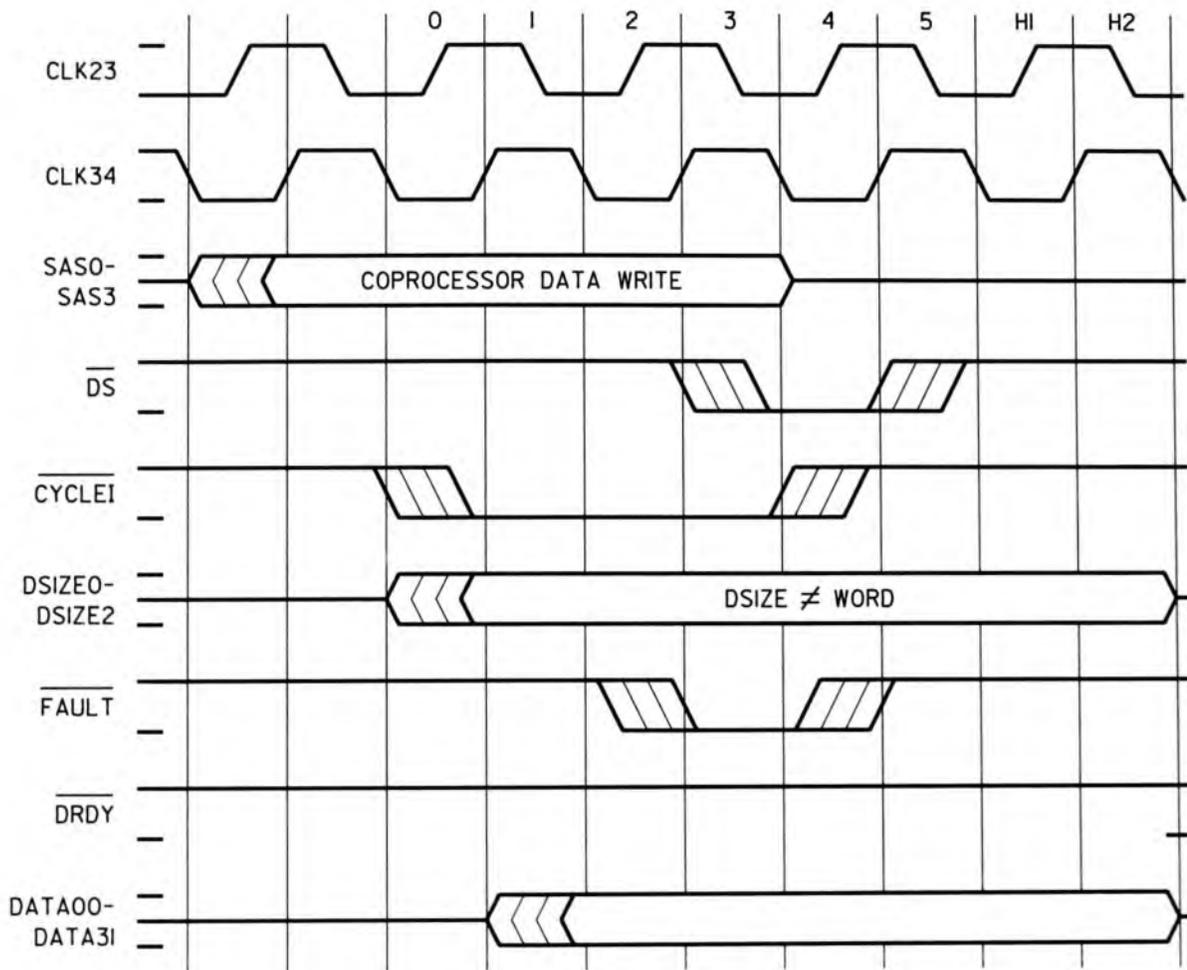
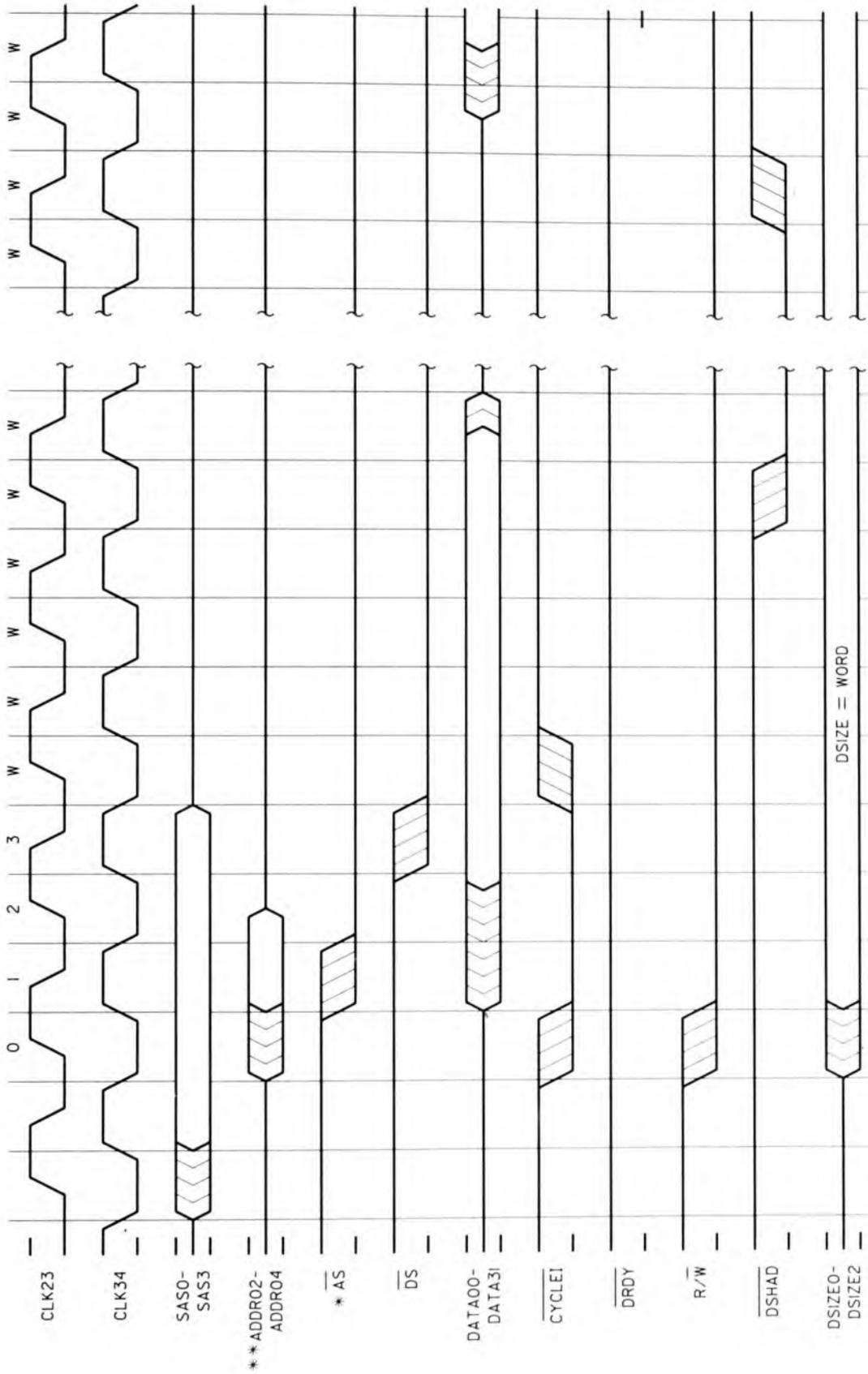


Figure 4-22. Coprocessor Data Write with Nonword Data (0 Wait States)



*Shown only for additional clarity.

**Transaction proceeds with translated address.

Figure 4-23. Coprocessor Data Write (\overline{DSHAD} Assertion)

BUS TRANSACTIONS

Status Fetch Faults

4.2.6 Coprocessor Faults

The MAU signals faults only during a coprocessor status fetch access and during a coprocessor read/write access when $DSIZE \neq 1$ (indicates a nonword access) and $ASR \langle FE \rangle = 1$.

If a coprocessor data write access is not completed successfully, the MAU waits until the access is repeated.

If a coprocessor broadcast access occurs during an MAU transaction, the MAU aborts the transaction. If the ID associated with the access is not the MAU's ID, the MAU enters its quiescent state, and if the ID is equal to the MAU's ID, the MAU starts the transaction specified by the access.

The MAU expects all memory operands to be word aligned and will not accept any byte or halfword operands. If the $DSIZE$ bits indicate any transaction size other than a word, and if $ASR \langle FE \rangle = 1$, the MAU will not execute the operation but will signal a fault for the offending access and sets the UW bit.

Status Fetch Faults

If the MAU wishes to cause an exception as the result of a floating-point operation, it does so by faulting the coprocessor status fetch access after the ASR contents have been updated to reflect the result of the operation. This kind of faulted transaction is shown on Figure 4-24.

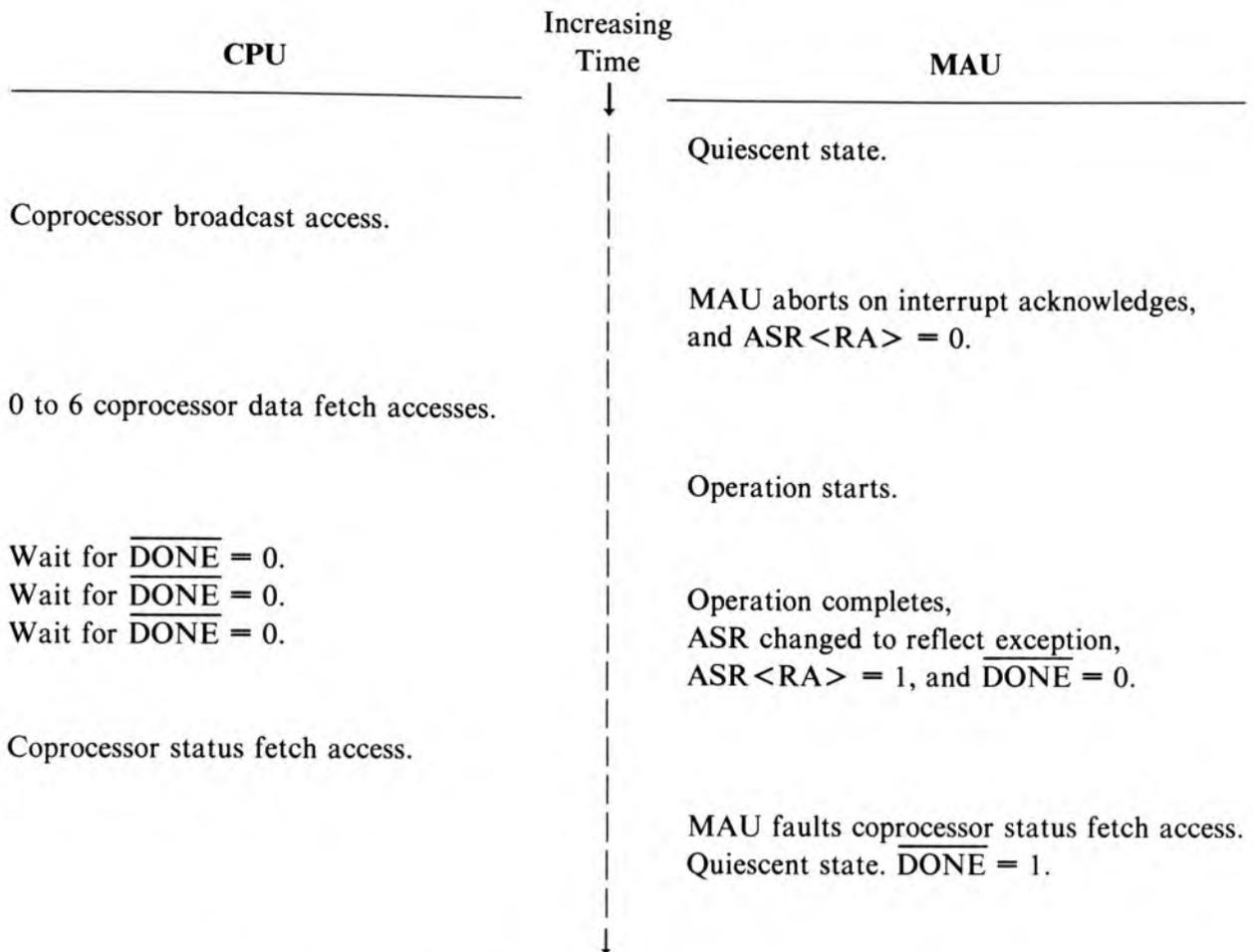


Figure 4-24. Faulted Coprocessor Status Fetch Transaction

Non-Existent MAU Faults

The bus master initiating a coprocessor broadcast access is responsible for detecting the case in which no coprocessor reacts to the access.

4.2.7 Coprocessor Restartability

The *WE* 32200 CPU instruction set is restartable and works with word aligned data. A problem exists when the following situation occurs:

- The result data for the operation is multiwords long (e.g., double or triple word size).
- Portions of the source and destination may overlap (e.g., the second word of the source overlaps in memory with the first word of the destination).
- The first word of the destination is on a different page from the second or third word of the destination.

BUS TRANSACTIONS

Restrictions on System Configuration

- In this situation, a page fault may occur after the first word of the destination has been successfully written into memory, which destroys part of the source operand.
- Restarting the operation under such circumstances will produce undefined results.

If $ASR\langle FE \rangle = 1$, the *WE* 32206 MAU eliminates the above problems as follows:

- When a memory fault occurs during the write of the result, the *WF* bit in the *ASR* is set.
- The condition codes correspond to the result of the operation.
- It is anticipated that after servicing of the page fault, the CPU restarts the operation that caused the page fault. Note that the *RDASR* and *LDR* instructions do not affect the *WF* bit and, therefore, will not be restarted.
- Under these conditions, when the *WF* bit is set, the MAU upon restart of the operation by the CPU will not reexecute the operation. Instead, it returns a *DONE* signal and uses the result already residing in the data register from the previously faulted operation and stores it in memory.
- This will not work, however, for convert float-to-decimal since the result is not stored in the *DR*. For convert float-to-decimal, the operating system should not allow memory-to-memory operations which thus prevents any overlap. If this is not the case, it must be assumed that floating-point and decimal data will not overlap in memory, given their different formats. Therefore, data in memory cannot be overwritten, and this operation can be restarted after setting $ASR\langle WF \rangle = 0$.

4.2.8 Restrictions on System Configuration

This section describes several ways in which properties of the coprocessor protocol restrict the configurations of systems containing coprocessors.

Because the MAU's ID number is fixed, it is not possible to have more than one MAU present on the same bus. This configuration is not allowed because two MAUs would react to the same coprocessor broadcast access. Various electrical problems would result from both devices attempting to manipulate the same signals and busses.

It is not possible to have more than one coprocessor protocol transaction in progress at any one time. That is, if an MAU and several other coprocessors exist on one bus, only one transaction can be in progress at any one time.

Because the MAU and future coprocessors react to various access types on the bus, it is not convenient to support systems with *M* masters and *N* coprocessors on the same bus (that is, it requires extensive handshaking and glue logic). For example, if master 1 is sending operands to an MAU shared by all masters and master 2 sends an interrupt

acknowledge access as part of some other activity, the MAU reacts to it by aborting its operation. Another example is where master 1 and master 2 each receive DONE signals from their private MAUs, MAU 1 and MAU 2. Master 1 happens to perform a coprocessor status fetch access before master 2 can, and then both MAUs react to the access from master 1. Such systems may be implemented if multiple busses are used, since that the interactions described above cannot occur. Also, no more than one coprocessor should be allowed to drive DONE at one time.

The CPU is free to allow interrupts up to the point in an MAU transaction where it sees the DONE signal go to 0. From that point until the end of the transaction (the last coprocessor data write access completes or the coprocessor status fetch access completes) the CPU must not allow interrupt acknowledge cycles.

4.2.9 Coprocessor Operating System Implications

If an interrupt occurs during an MAU transaction but before the DONE signal goes to 0, the MAU aborts the operation. It is possible that the ASR has been changed to reflect exceptions or new condition codes (N, Z, PR, etc.) at this point; however, when the CPU returns from the interrupt, it can restart the transaction from the beginning and the spurious ASR value will not affect the repeated operation.

The context of the MAU must be saved and restored later if the interrupt causes control of the MAU to be given to another process. Under the coprocessor protocol, the context of the MAU consists of the ASR, DR, and F0—F7 values. It might be necessary to run context save and restore sequences without interrupts.

The MAU context (DR, F0—F7, and ASR) can be saved by:

- Reading DR with an extract result on fault (EROF) instruction and saving its value.
- Reading the ASR mode with a move from ASR (RDASR) instruction (or peripheral mode) and saving its value.
- Clearing the ASR.
- Performing eight MAU move operations to save registers F0—F7.

The MAU context (F0—F7, ASR, and DR) can be restored by:

- Clearing the ASR.
- Performing eight MAU move operations to restore registers F0—F7.
- Writing the saved ASR value into the ASR via peripheral mode or a move to ASR (WRASR) instruction.
- Performing a load DR operation using the LDR instruction to restore DR.



Chapter 5
Instruction Set

CHAPTER 5. INSTRUCTION SET

CONTENTS

5. INSTRUCTION SET	5-1
5.1 FUNCTIONAL GROUPS.....	5-1
5.1.1 Arithmetic Instructions	5-2
5.1.2 Compare Instructions	5-2
5.1.3 Data Transfer Instructions.....	5-2
5.1.4 Conversion Instructions	5-3
5.1.5 Transcendental Instructions	5-3
5.1.6 Miscellaneous Instructions	5-3
5.2 INSTRUCTION SET LISTINGS	5-4
5.2.1 Notation	5-4
5.2.2 Instruction Set Descriptions	5-4
Absolute Value (ABS)	5-7
Add (ADD)	5-8
Arctangent (ATAN)	5-9
Compare (CMP)	5-11
Compare With Exceptions (CMPE)	5-12
Compare With Exceptions and Flags Swapped (CMPES)	5-13
Compare With Flags Swapped (CMPS).....	5-14
Cosine (COS)	5-15
Divide (DIV)	5-17
Convert Decimal to Float (DFOB)	5-18
Extract Result on Fault (EROF)	5-19
Convert Float to Decimal (FTOD)	5-20
Convert Float to Integer (FTOI).....	5-22
Convert Integer to Float (ITOF).....	5-24
Load Data Register (LDR)	5-25
Move (MOVE).....	5-26
Multiply (MUL)	5-28
Negate (MEG).....	5-29
No Operation (NOP).....	5-30
Pi (PI).....	5-31
Move From ASR (RDASR)	5-32
Remainder (REM).....	5-33
Round to Integral Value (RTOI)	5-35
Sine (SIN).....	5-37
Square Root (SQRT).....	5-39
Subtract (SUB)	5-40
Move to ASR (WRASR)	5-42
5.2.3 Instruction Set Summary by Mnemonic.....	5-43
5.2.4 Instruction Set Summary by Opcode.....	5-46

5. INSTRUCTION SET

The *WE* 32206 Math Acceleration Unit instruction set includes arithmetic, logical, data transfer, format conversion, transcendental, and miscellaneous instructions. A detailed description of these instructions is given in section 5.2.2. Tables 5-1 through 5-6 list the instructions according to mnemonics, opcodes, and functional groups.

The MAU supports operations consisting of zero to three operands. The *WE* 32200 CPU allows from zero to two operands in memory for each coprocessor protocol instruction; therefore, the MAU can be used with a CPU that has coprocessor instructions allowing all three operands to be in memory.

The command word always has three operand specifiers in it. The number of operands actually used by the MAU in the operation depends on the opcode specified in the command word. If an operation is initiated that does not use all of the operands, the operand specifiers corresponding to the unused operands should have NOP values. If they do not, they are read (for Op1 and Op2) or written (for Op3) regardless of the opcode. On a read operation, the data is ignored; on a write operation, indeterminate values are written. This holds true for operands both in memory and in MAU registers.

Any unassigned opcodes will cause behavior identical to that of the no operation (NOP) opcode, but these opcodes are reserved for future MAU use.

All operations initiated by the CPU or another bus master conform to the following sequence:

1. The operation is initiated via one of the protocols: coprocessor or peripheral.
2. All source operands from memory are converted to double-extended format.
3. The operation is performed.
4. If the destination operand specifier designates single-word or double-word format, the result is converted to single-precision or double-precision format.
5. Rounding occurs.
6. If an exception condition is satisfied, the result is stored in the DR, and the usual exception action occurs.
7. If no exception condition is satisfied, the result is stored in the destination.

5.1 FUNCTIONAL GROUPS

The *WE* 32206 Math Acceleration Unit instruction set may be separated into six functional groups: arithmetic, compare, data transfer, conversion, transcendental, and miscellaneous instructions. This section contains a description of each group, along with a listing for each group.

INSTRUCTION SET

Arithmetic Instructions

5.1.1 Arithmetic Instructions

Table 5-1 lists the instructions that perform arithmetic operations on data.

Table 5-1. Arithmetic Instruction Group		
Instruction	Mnemonic	Opcode
Absolute Value	ABS	0x0C
Add	ADD	0x02
Subtract	SUB	0x03
Multiply	MUL	0x06
Divide	DIV	0x04
Remainder	REM	0x05
Square root	SQRT	0x0D
Negate	NEG	0x17
Round to integral value	RTOI	0x0E

5.1.2 Compare Instructions

Table 5-2 lists the instructions that perform logical operations on data.

Table 5-2. Logical Instruction Group		
Instruction	Mnemonic	Opcode
Compare	CMP	0x0A
Compare with exceptions	CMPE	0x0B
Compare with flags swapped	CMPS	0x1A
Compare with exceptions and flags swapped	CMPEX	0x1B

5.1.3 Data Transfer Instructions

Table 5-3 lists the instructions that transfer data to and from registers and memory.

Table 5-3. Data Transfer Instruction Group		
Instruction	Mnemonic	Opcode
Move	MOVE	0x07
Move from ASR	RDASR	0x08
Move to ASR	WRASR	0x09
Load DR	LDR	0x18

5.1.4 Conversion Instructions

Table 5-4 lists the instructions that perform symmetric decimal, floating-point, and integer conversions.

Table 5-4. Conversion Instruction Group		
Instruction	Mnemonic	Opcode
Convert decimal to floating-point	DFOB	0x11
Convert floating-point to decimal	FTOD	0x12
Convert floating-point to integer	FTOI	0x0F
Convert integer to floating-point	ITOF	0x10

5.1.5 Transcendental Instructions

Table 5-5 lists the instructions that perform transcendental functions.

Table 5-5. Transcendental Instruction Group		
Instruction	Mnemonic	Opcode
Arc tangent	ATAN	0x1E
Cosine	COS	0x1D
Pi	PI	0x1F
Sine	SIN	0x1C

Note: The worst case error for SIN, COS, and ATAN in round nearest mode is equal to one unit in the last place for single and double precision. The worst case error for these functions in other rounding modes is equal to two units in the last place for single and double precision. For double-extended precision, the result is guaranteed to be at least as accurate as double precision, and will usually be more accurate.

5.1.6 Miscellaneous Instructions

Table 5-6 lists the instructions that do not fall into any of the previous categories.

Table 5-6. Miscellaneous Instruction Group		
Instruction	Mnemonic	Opcode
Extract result on fault	EROF	0x14
No operation	NOP	0x13

5.2 INSTRUCTION SET LISTINGS

Section 5.2.2 presents descriptions of each member of the instruction set for the *WE 32206* Math Acceleration Unit. The descriptions are listed in alphabetical order. (For quick reference to the instructions by function, mnemonic, or opcode, see sections 5.1.1 through 5.1.6, 5.2.3 and 5.2.4).

5.2.1 Notation

Each instruction description contains several parts: mnemonic, opcode, operation, description, special cases, condition flags, and exceptions.

Mnemonic. Presents the macro name for the instruction.

Opcode. Lists each instruction opcode in hexadecimal format.

Operation. Describes the operation performed.

Description. Presents a detailed explanation of the operation.

Special Cases. Specifies the action to be taken for each combination of special case input values. Table 5-7 explains the action to be taken for each special case. The number of the special case action can be extracted from the special-case tables listed under each instruction's description (see section 5.2.2).

Condition flags. Identifies the effect of the instruction on the condition flags of the auxiliary status register (ASR). Refer to section 2.2.1 for a description of the condition flags.

Exceptions. Identifies the possible exceptions that may be caused by the operation. Refer to section 2.4 for a description of possible exceptions.

5.2.2 Instruction Set Descriptions

The instruction sets are described in detail following Table 5-7.

Table 5-7. Notation for Special Cases

Number	Action
1	<p>ASR<IS> = 1; If (ASR<IM> == 1) ASR<ECP> = 1; signal invalid operation exception;*</p> <p>else if (ASR<NTNC> == 1) ASR<ECP> = 1; signal exception to generate quiet (nontrapping) NaN with virtual PC as significand via software;*</p> <p>else Op3 = quiet (non trapping) NaN, sign = 0;</p> <p>(On a fault the ASR N & Z flags are unspecified. However, as implemented, the Z flag is cleared and the N flag in the ASR remains unchanged from the previous operation.)</p> <p>*When Op1 is a trapping NaN then register DR=Op1 else DR=Op2. In addition, for single or double precision, set J=1.</p>
2	<p>The following action is taken with an invalid operation exception due to an MAU register destination for a floating-point to decimal or floating-point to integer conversion.</p> <p>ASR<IS> = 1; if (ASR<IM> == 1) data register = source operand; ASR<ECP> = 1; signal invalid operation exception;</p> <p>else if (ASR<NTNC> == 1; ASR<ECP> = 1; signal exception to generate quiet (nontrapping) NaN with virtual PC as significand via software;**</p> <p>else Op3 = quiet (nontrapping) NaN, sign = 0;</p> <p>**DR = Op1.</p>

INSTRUCTION SET

Notation

Table 5-7. Notation for Special Cases (Continued)	
Number	Action
1	<p>The following action is taken with an invalid operation exception due to an MAU register source for a decimal to floating-point or integer to floating-point conversion.</p> <pre> ASR<IS> = 1; if (ASR<IM> == 1) DR = unspecified; ASR<ECP> = 1; signal invalid operation exception; else if (ASR<NTNC> == 1) ASR<ECP> = 1; signal exception to generate quiet (nontrapping) NaN with virtual PC as significand via software;*</pre> <p>else</p> <pre> Op3 = Quiet (nontrapping) NaN, sign = 0;</pre> <p>*DR = unspecified.</p>
2	<pre> ASR<QS> = 1; if (ASR<QM> == 1) signal exception; else Op3 = infinity, normal sign calculation;</pre>
3	Op3 = Op1; (see note)
4	Op3 = 0, normal sign calculation;
5	Op3 = Infinity, normal sign calculation;
6	Op3 = Op2; (see note)
7	<pre> if (sign(Op1) == sign(Op2)) (see note) Op3 = Op1; else if (ASR<RC> == RM) Op3 = (-0); else Op3 = (+0);</pre>
8	Perform Operation;
9	Op3 = (decimal equivalent of floating Op1);
10	Op3 = (floating equivalent of decimal Op1);
11	Op3 = +1;

Note: In these actions, if the internal intermediate result is in the wider format than Op3, then the internal result is converted to conform to the Op3 format. This conversion process rounds the result if necessary. Also, this conversion could generate overflow or underflow exception conditions.

ABSOLUTE VALUE

Mnemonic	ABS
Opcode	0x0C
Operation	$Op3 = Op1 $
Description	Operand 1's sign bit is cleared and then Op1 is copied into operand 3. Operand 1 is left unaffected. All exceptions are handled as in the specification of the MOVE operation.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Action
\pm Infinity	8
NaN	8*
TNaN	1
\pm Zero	8
Unnormal Zero	4
Normal	8

* Return NaN with sign bit cleared.

Condition	N \leftarrow sign bit of value stored in Op3
Flags of the ASR	Z \leftarrow 1 if 0 stored in Op3, else 0
	IO \leftarrow 0
	PR \leftarrow 0
	UO \leftarrow 0
Exceptions	Invalid operation Overflow Underflow Inexact

ADD

ADD

ADD

Mnemonic	ADD
Opcode	0x02
Operation	$Op3 = Op1 + Op2$
Description	Operand 1 is added to operand 2 and the result is stored in operand 3.

Special Cases:

The following table specifies the action to be taken for each combination of special-case values input. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Op2					
	+Infinity	-Infinity	NaN	TNaN	Zero	Normal
+Infinity	3	1	3	1	5	5
-Infinity	1	3	3	1	5	5
NaN	6	6	3	1	6	6
TNaN	1	1	1	1	1	1
Zero	5	3	3	1	7	3
Normal	5	3	3	1	6	8

Condition	N ← sign bit of value stored in Op3
Flags of the ASR	Z ← 1 if 0 stored in Op3, else 0
	IO ← 0
	PR ← 0
	UO ← 0

Exceptions	Invalid operation
	Overflow
	Underflow
	Inexact

ARCTANGENT

Mnemonic	ATAN
Opcode	0x1E
Operation	Op3 = atan(Op1)
Description	The arctangent of operand 1 (in radians in floating-point format) is computed and stored in operand 3. The value of operand 1 must be between -1 and $+1$, inclusive. An invalid operation exception exists for all other values of operand 1.

Note: If the destination is specified to be single precision, the operation executes faster since the number of iterations performed to execute the operation corresponds to single precision instead of double precision.

Note: The ATAN operation guarantees accuracy to single and double precision only. If the destination is specified as double-extended, the result has at least the accuracy of double precision and usually more. However, it does not meet the accuracy requirements of double-extended precision. The level of accuracy beyond normal double precision of such a result is not guaranteed.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

ATAN

ATAN

Op1	Action
Infinity	1
NaN	3
TNaN	1
Zero	3
Op1 > +1	1
Op1 < -1	1
$-1 \leq \text{Op1} \leq +1$	8

Condition Flags of the ASR

N ← sign bit of the value stored in Op3
Z ← 1 if 0 stored in Op3, else 0
PR ← 0
UO ← 0
IO ← 0

Exceptions

Invalid operation
Inexact (all valid inputs except Op1 = 0)
Underflow

CMP

CMP

COMPARE

Mnemonic CMP

Opcode 0x0A

Operation Set flags according to Op1 and Op2

Description Operand 1 is compared to operand 2 and the condition codes are set as specified. Operand 1 and operand 2 are not modified.

An invalid operation exception condition exists if either or both source operands are trapping NaNs. If the exception is masked then the UO flag would be set. However, if this exception is enabled, and, if Op1 is a trapping NaN, it is converted to double-extended precision and stored in DR. Else, Op2 (converted to double-extended precision, if necessary) is stored in DR.

Special Cases:

The following table specifies the ordering relationships between the various values of Op1 and Op2. If the entry is A, an unordered condition exists. If the entry is B, the resulting N and Z values are based on the result of the subtraction: $Op1 - Op2$.

Op1	Op2						
	-Infinity	+Finite	-Finite	-Zero	+Zero	+Infinity	NaN
-Infinity	==	<	<	<	<	<	A
+Finite	>	B	>	>	>	<	A
-Finite	>	<	B	<	<	<	A
-Zero	>	<	>	==	==	<	A
+Zero	>	<	>	==	==	<	A
+Infinity	>	>	>	>	>	==	A
NaN	A	A	A	A	A	A	A

Condition N ← 1 if $Op1 < Op2$, else 0
Flags of the ASR Z ← 1 if $Op1 == Op2$, else 0
IO ← 0
PR ← 0
UO ← 1 if unordered condition exists, else 0

Exceptions Invalid operation

COMPARE WITH EXCEPTIONS

Mnemonic CMPE

Opcode 0x0B

Operation Set flags according to Op1 and Op2

Description Operand 1 is compared to operand 2 and the condition codes are set as specified. Operand 1 and operand 2 are not modified.

When two unordered values are compared, then, in addition to the response specified below, the invalid operation exception sticky flag (ASR<IS> = 1) is set and the trap invoked if the invalid operation exception is enabled.

Special Cases:

The following table specifies the ordering relationships between the various values of Op1 and Op2. If the entry is A, an unordered condition exists. If the entry is B, the resulting N and Z values are based on the result of the subtraction: Op1 - Op2.

Op1	Op2						
	-Infinity	+Finite	-Finite	-Zero	+Zero	+Infinity	NaN
-Infinity	==	<	<	<	<	<	A
+Finite	>	B	>	>	>	<	A
-Finite	>	<	B	<	<	<	A
-Zero	>	<	>	==	==	<	A
+Zero	>	<	>	==	==	<	A
+Infinity	>	>	>	>	>	==	A
NaN	A	A	A	A	A	A	A

Condition N ← 1 if Op1 < Op2, else 0
Flags of the ASR Z ← 1 if Op1 == Op2, else 0
 IO ← 0
 PR ← 0
 UO ← 1 if unordered condition exists, else 0

Exceptions Invalid operation

COMPARE WITH EXCEPTIONS AND FLAGS SWAPPED

Mnemonic	CMPES
Opcode	0x1B
Operation	Set flags according to Op1 and Op2
Description	Operand 1 is compared to operand 2 and the condition codes are set as specified. Operand 1 and operand 2 are not modified.

When two unordered values are compared, then, in addition to the response specified below, the invalid operation exception sticky flag (ASR<IS> = 1) is set and the trap invoked if the invalid operation exception is enabled.

Special Cases:

The following table specifies the ordering relationships between the various values of Op1 and Op2. If the entry is A, an unordered condition exists. If the entry is B, the resulting N and Z values are based on the result of the subtraction: $Op1 - Op2$.

Op1	Op2						
	-Infinity	+Finite	-Finite	-Zero	+Zero	+Infinity	NaN
-Infinity	==	<	<	<	<	<	A
+Finite	>	B	>	>	>	<	A
-Finite	>	<	B	<	<	<	A
-Zero	>	<	>	==	==	<	A
+Zero	>	<	>	==	==	<	A
+Infinity	>	>	>	>	>	==	A
NaN	A	A	A	A	A	A	A

Condition	N ← 1 if $Op1 == Op2$, else 0
Flags of the ASR	Z ← 1 if $Op1 < Op2$, else 0
	IO ← 0
	PR ← 0
	UO ← 1 if unordered condition exists, else 0

Exceptions Invalid operation

COMPARE WITH FLAGS SWAPPED

Mnemonic	CMPS
Opcode	0x1A
Operation	Set flags according to Op1 and Op2
Description	<p>Operand 1 is compared to operand 2 and the condition codes are set as specified. Operand 1 and operand 2 are not modified.</p> <p>An invalid operation exception condition exists if either or both source operands are trapping NaNs. If the exception is masked then the UO flag is set. However, if this exception is enabled, and, if Op1 is a trapping NaN, it is converted to double-extended precision and stored in DR. Else, Op2 (converted to double-extended precision, if necessary) is stored in DR.</p>

Special Cases:

The following table specifies the ordering relationships between the various values of Op1 and Op2. If the entry is A, an unordered condition exists. If the entry is B, the resulting N and Z values are based on the result of the subtraction: $Op1 - Op2$.

Op1	Op2						
	-Infinity	+Finite	-Finite	-Zero	+Zero	+Infinity	NaN
-Infinity	==	<	<	<	<	<	A
+Finite	>	B	>	>	>	<	A
-Finite	>	<	B	<	<	<	A
-Zero	>	<	>	==	==	<	A
+Zero	>	<	>	==	==	<	A
+Infinity	>	>	>	>	>	==	A
NaN	A	A	A	A	A	A	A

Condition	N ← 1 if $Op1 == Op2$, else 0
Flags of the ASR	Z ← 1 if $Op1 < Op2$, else 0
	IO ← 0
	PR ← 0
	UO ← 1 if unordered condition exists, else 0
Exceptions	Invalid operation

COSINE

Mnemonic	COS
Opcode	0x1D
Operation	Op3 = cosine(Op1)
Description	The cosine of operand 1 (in radians in floating-point format) is computed and stored in operand 3. The value of operand 1 must be between $-(\pi/2)$ and $+(\pi/2)$. An invalid operation exception condition exists for all other values of operand 1. For all valid inputs, $0 \leq \text{COS}(x) \leq 1$

Note: If the destination is specified to be single precision, the operation executes faster since the number of iterations performed to execute the operation corresponds to single precision instead of double precision.

Note: The COS operation guarantees accuracy to single and double precision only. If the destination is specified as double-extended, the result has at least the accuracy of double precision and usually more. However, it does not meet the accuracy requirements of double-extended precision. The level of accuracy beyond normal double precision of such a result is not guaranteed.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

COS

COS

Op1	Action
Infinity	1
NaN	3
TNaN	1
Zero	11
$Op1 > +(\pi/2)$	1
$Op1 < -(\pi/2)$	1
$-(\pi/2) < Op1 < +(\pi/2)$	8

**Condition
Flags of
the ASR**

N ← sign bit of the value stored in Op3
Z ← 1 if 0 stored in Op3, else 0
PR ← 0
UO ← 0
IO ← 0

Exceptions

Invalid operation
Inexact (all valid inputs except $Op1 = 0$)

DIV

DIV

DIVIDE

Mnemonic DIV

Opcode 0x04

Operation $Op3 = Op2 / Op1$

Description Operand 2 is divided by operand 1 and the result is stored in operand 3

Note: If the destination is specified as single precision, the operation executes faster. The number of iterations performed to execute the operation corresponds to single precision instead of double-extended.

Special Cases:

The following table specifies the action to be taken for each combination of special case values input. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Op2				
	Infinity	NaN	TNaN	Zero	Normal
Infinity	1	3	1	5	5
NaN	6	3	1	6	6
TNaN	1	1	1	1	1
Zero	4	3	1	1	4
Normal	4	3	1	2	8

Condition N ← sign bit of value stored in Op3

Flags of the ASR Z ← 1 if 0 stored in Op3, else 0

IO ← 0

PR ← 0

UO ← 0

Exceptions Invalid operation

Overflow

Underflow

Divide-by-zero

Inexact

CONVERT DECIMAL TO FLOAT

Mnemonic DTOF

Opcode 0x11

Operation Op3 = float(decimal Op1)

Description Operand 1 (in decimal format) is converted into a floating-point format and then copied into operand 3. Operand 1 is left unaffected.

 An invalid operation exception condition exists if the source operand is specified from the MAU register. If this exception is enabled, the value stored in DR (intended for the trap handler) is undefined.

 The inexact condition exists if the delivered result is not exactly equal in value to Op1 and no other exceptions have occurred.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Action
Infinity	10
NaN	10
TNaN	1
Zero	10
Normalized Number	8

Condition N ← 1 if Op3 is negative, else 0

Flags of Z ← 1 if 0 stored in Op3, else 0

the ASR IO ← 0

 PR ← 0

 UO ← 0

Exceptions Invalid operation

 Inexact

EXTRACT RESULT ON FAULT

Mnemonic	EROF
Opcode	0x14
Operation	Op3 = data register
Description	<p>The value of the data register is placed in operand 3. This is a triple word transfer.</p> <p>The purpose of this instruction is to extract the value of any result that may be generated when a fault condition occurs. It provides additional diagnostic information on the nature of the fault.</p> <p>If this instruction is executed under normal conditions, the value returned is indeterminate.</p>
Special Cases:	None
Condition Flags of the ASR	Unchanged
Exceptions	None

CONVERT FLOAT TO DECIMAL

Mnemonic FTOD

Opcode 0x12

Operation Op3 = decimal(floating Op1)

Description Operand 1 (in floating-point format) is rounded to integral value and then converted into decimal format and copied into operand 3. Operand 1 is left unaffected.

An invalid operation exception condition exists if the destination operand is specified to be an MAU register. If this exception is enabled, then the source operand (after converting to double-extended precision if necessary) is stored in DR.

The inexact condition exists if the delivered result is not exactly equal in value to Op1 and no other exceptions have occurred.

If the intermediate result overflows the destination, then ASR<IO> is set and the destination is updated with an undefined value. By controlling the overflow enable/disable bit in the WE 32200 CPU's PSW, the user can choose to mask or unmask the decimal overflow exception.

Note: To assure restartability, the operating system should not allow memory-memory operations for this instruction. If allowed, it should be assumed that floating-point and decimal data do not overlap in memory.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Action
Infinity	9
NaN	9
TNaN	1
Zero	9
Normalized Number	8

FTOD

FTOD

Condition N ← 1 if value stored in Op3 is negative, else 0
Flags of Z ← 1 if 0 stored in Op3, else 0
the ASR IO ← 1 if decimal-overflow condition exists, else 0
 PR ← 0
 UO ← 0

Exceptions Invalid operation
 Overflow
 Underflow
 Inexact

CONVERT FLOAT TO INTEGER**Mnemonic** FTOI**Opcode** 0x0F**Operation** Op3 = integer(Op1)**Description** Operand 1 is converted from a floating-point format into 2's complement integer format and copied into operand 3. Operand 1 is left unaffected.

An invalid operation exception occurs if Op1 is a NaN or infinity. Op3 is left unaffected in this case. Also, this condition exists if the destination is specified as an MAU register. If the invalid operation exception is enabled then Op1 (after converting to double-extended precision if necessary) is stored in DR.

If the intermediate result overflows the destination, ASR<IO> is set and the destination is updated with an undefined value. By controlling the overflow enable/disable bit in the WE 32200 CPU's PSW, the user can choose to mask or unmask the integer overflow exception.

If Op3 has a size other than single-word, the resulting integer is stored into the last word of Op3 and indeterminate values are stored into the other words.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Action
Infinity	1
NaN	1
TNaN	1
± Zero	8
Normal	8

CONVERT FLOAT TO INTEGER

Condition	N ← sign bit of value stored in Op3
Flags of the ASR	Z ← 1 if 0 stored in Op3, else 0
	IO ← 0
	PR ← 0
	UO ← 0

Exceptions	Invalid operation
	Integer overflow

CONVERT INTEGER TO FLOAT

Mnemonic	ITOF
Opcode	0x10
Operation	Op3 = float(Op1)
Description	<p>Operand 1 is converted from 2's complement integer format to floating-point format and copied into operand 3. Operand 1 is left unaffected.</p> <p>An invalid operation exception condition exists if the source operand is specified from an MAU register. If this exception is enabled, the value stored in DR (intended for the trap handler) is undefined.</p> <p>The inexact condition exists if the delivered result is not exactly equal to the intermediate result and no other exceptions have occurred.</p> <p>An inexact condition can occur when Op3 is a single-word operand.</p> <p>If Op1 has a size other than single-word, all words of Op1 are fetched but only the last word is used as the source operand.</p>
Special Cases:	None
Condition Flags of the ASR	<p>N ← sign bit of value stored in Op3</p> <p>Z ← 1 if 0 stored in Op3, else 0</p> <p>IO ← 0</p> <p>PR ← 0</p> <p>UO ← 0</p>
Exceptions	<p>Inexact</p> <p>Invalid operation</p>

LDR

LDR

LOAD DATA REGISTER

Mnemonic LDR

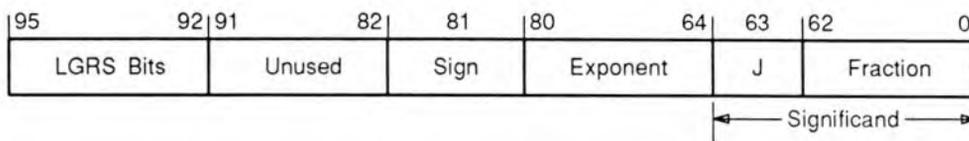
Opcode 0x18

Operation Data register = Op1

Description Operand 1 is placed in the data register. This is a triple word transfer.

The purpose of this instruction is to restore the data register. Normally, the restored value corresponds to the value generated by the original process on occurrence of a fault.

Note that the value stored in DR has the format as shown below.



If the data in the double-extended format <79:0> is stored in DR using this operation, then it appears in the least significant 80 bits of DR. Note that since the value generated on fault has two extra bits in the exponent, the sign bit could be placed in the incorrect position if the double-extended precision <79:0> operand is moved to DR.

Note: Executing this operation when WF = 1 is not interpreted by the MAU as restarting the operation which caused the fault.

Special Cases: None

**Condition
Flags of
the ASR** Unchanged

Exceptions None

MOVE

MOVE

MOVE

Mnemonic MOVE

Opcode 0x07

Operation Op3 = Op1

Description Operand 1 is copied into operand 3. Operand 1 is left unaffected.

In one MOVE operation, it is possible to convert a number from a format to any other format. However, the *WE 32200* CPU does not support this because it requires all memory-based operands to be of the same size. With the *WE 32200* CPU, this operation may be used to convert from one format to another by moving a number from memory (in format X) to an MAU register, and then moving the number from the MAU register to memory (in format Y). Thus, in two move operations, the *WE 32200* CPU can use the MAU to convert a number from one format to any other format.

An invalid operation exception occurs if the source operand is a trapping NaN. If the exception is masked, the result is a nontrapping NaN.

The inexact condition exists if Op3 is not exactly equal to Op1 and no other exceptions have occurred.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Action
Infinity	8
NaN	8
TNaN	1
± Zero	8
Normal	8

MOVE

Condition Flags of the ASR

N ← sign bit of value stored in Op3
Z ← 1 if 0 stored in Op3, else 0
IO ← 0
PR ← 0
UO ← 0

Exceptions

Invalid operation
Overflow
Underflow
Inexact

MOVE

MUL

MUL

MULTIPLY

Mnemonic MUL

Opcode 0x06

Operation $Op3 = Op1 * Op2$

Description Operand 1 is multiplied by operand 2 and the result is stored in operand 3.

Special Cases:

The following table specifies the action to be taken for each combination of special-case values input. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Op2				
	Infinity	NaN	TNaN	Zero	Normal
Infinity	5	3	1	1	5
NaN	6	3	1	6	6
TNaN	1	1	1	1	1
Zero	1	3	1	4	4
Normal	5	3	1	4	8

Note: If the destination is specified to be single precision, the operation will execute faster. Then, the number of iterations performed to execute the operation corresponds to single precision instead of double precision.

Condition N ← sign bit of value stored in Op3
Flags of the ASR Z ← 1 if 0 stored in Op3, else 0
 IO ← 0
 PR ← 0
 UO ← 0

Exceptions Invalid operation
 Overflow
 Underflow
 Inexact

NEG

NEG

NEGATE

Mnemonic NEG

Opcode 0x17

Operation $Op3 = -(Op1)$

Description Operand 1's sign bit is complemented and the Op1 is copied into operand 3. Operand 1 is left unaffected.

All exceptions and special cases are handled as in the specification of the MOVE operation.

Special Cases:

The following table specifies the action to be taken for each case where Op3 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op3	Action
Infinity	8
NaN	8*
TNaN	1
\pm Zero	8
Normal	8

* Complement sign bit of NaN

Condition N \leftarrow sign bit of value stored in Op3
Flags of Z \leftarrow 1 if 0 stored in Op3, else 0
the ASR IO \leftarrow 0
 PR \leftarrow 0
 UO \leftarrow 0

Exceptions Invalid operation
 Overflow
 Underflow
 Inexact

NOP

NOP

NO OPERATION

Mnemonic	NOP
Opcode	0x13
Operation	Stop current operation, if any, and return to quiescent state.
Description	<p>If an operation is in progress, NOP causes the operation to terminate without affecting any of the MAU internal objects. If no operation is in progress, NOP simply causes the MAU to return to the quiescent state.</p> <p>Any unassigned opcodes cause behavior identical to that of the NOP opcode, but these opcodes are reserved for future MAU use.</p> <p>Operand specifiers can cause operand fetching and storing regardless of the opcode. In order for the NOP opcode to have its intended effect, all of the operand specifiers associated with it should have NOP values.</p>
Special Cases:	None
Condition Flags of the ASR	Unchanged
Exceptions	None

PI

PI

PI

Mnemonic	PI
Opcode	0x1F
Operation	Op3 \leftarrow pi
Description	The constant pi is stored in Op3.
Special Cases:	None
Condition	N \leftarrow 0
Flags of the ASR	Z \leftarrow 0 IO \leftarrow 0 PR \leftarrow 0 UO \leftarrow 0
Exceptions	Inexact (always invoked)

Note: In rounding-to-nearest mode, the value returned in single and double-extended precision is rounded upward. Note that if this value divided by two is then given as an input to SIN or COS, an invalid operation exception will occur as the input is greater than the actual value of pi/2. Should a value of pi be desired which is always less than the actual value of pi, PI should be called in round-towards-zero mode.

RDASR

RDASR

MOVE FROM ASR

Mnemonic RDASR

Opcode 0x08

Operation Op3 = ASR

Description The ASR is copied into operand 3. The ASR is left unaffected. The DR is also left unaffected.

This operation can be used to examine the ASR value or to save its contents upon context switch.

If Op3 is of a size other than single-word, all words stored contain the ASR value.

This opcode should not be used in peripheral mode. Instead, the ASR should be read directly from MAU location 000.

Note: Executing this operation when WF = 1 is not interpreted by the MAU as restarting the operation which caused the fault.

Special Cases: None

**Condition
Flags of
the ASR** Unchanged

Exceptions None

REM

REM

REMAINDER

Mnemonic	REM
Opcode	0x05
Operation	Op3 = IEEE remainder from (Op2/Op1)
Description	Operand 2 is divided by operand 1 and the remainder from the operation is stored in operand 3.

Special Cases:

The following table specifies the action to be taken for each combination of special-case values input. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Op2				
	Infinity	NaN	TNaN	Zero	Normal
Infinity	1	3	1	1	1
NaN	6	3	1	6	6
TNaN	1	1	1	1	1
Zero	4	3	1	1	4
Normal	6	3	1	1	8

The IEEE remainder operation is defined as follows:

For Op1 not = 0, the remainder, Op3, is defined regardless of the rounding mode by the relation $Op3 = Op2 - (n)(Op1)$, where n is the integer nearest the exact value $(Op2 / Op1)$. Whenever $|n - (Op2 / Op1)| = 1/2$, then n is even.

If the PR bit is set as a result of the operation, then the result is a partial remainder, not the true remainder. To get the true remainder, the remainder operation should be repeated with the result of the previous remainder operation, Op3, replacing the dividend of the previous operation, Op2. This process should be repeated until the PR bit is zero, indicating that the result is the true remainder.

RTOI

RTOI

Exceptions

Invalid operation
Overflow
Inexact

SIN

SIN

SINE

Mnemonic SIN

Opcode 0x1C

Operation Op3 ← sine (Op1)

Description The sine of Op1 (in radians, floating-point) is computed and stored in Op3. The value of Op1 must be such that $-(\pi/2) < \text{Op1} < +(\pi/2)$. An invalid operation exception condition occurs for all other values of Op1. For all valid inputs, $|\sin(x)| \leq 1$.

Note: If the destination is specified to be single precision, then the operation will execute faster, since the number of iterations performed corresponds to single precision instead of double precision.

Note: The sine operation guarantees accuracy to single and double precision only. If the destination is specified to be double-extended precision, the result has at least the accuracy of double precision and usually has more. However, it does not meet the accuracy requirements of double-extended precision. The level of accuracy beyond normal double precision of such a result is not guaranteed.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. The meaning of each number is specified in Table 5-7.

Op1	Action
Infinity	1
NaN	3
TNaN	1
Zero	3
$\text{Op1} > +(\pi/2)$	1
$\text{Op1} < -(\pi/2)$	1
$-(\pi/2) < \text{Op1} < +(\pi/2)$	8

SIN

SIN

Condition	N ← Sign bit of value in Op3
Flags of the ASR	Z ← 1 if Op3 = 0, else 0
	PR ← 0
	UO ← 0
	IO ← 0
Exceptions	Invalid operation
	Inexact (All valid inputs except Op1 = 0)
	Underflow

SQRT

SQRT

SQUARE ROOT

Mnemonic	SQRT
Opcode	0x0D
Operation	Op3 = square root of Op1
Description	The square root of operand 1 is computed and stored in operand 3. Operand 1 is left unaffected.

Special Cases:

The following table specifies the action to be taken for each case where Op1 is a special-case value. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Action
Positive Infinity	3
Negative Infinity	1
NaN	3
TNaN	1
Positive Normalized	8
Negative Normalized	1
Positive Zero	3
Negative Zero	3

Condition	N ← sign bit of value stored in Op3
Flags of the ASR	Z ← 1 if 0 stored in Op3, else 0
	IO ← 0
	PR ← 0
	UO ← 0

Exceptions	Invalid operation
	Overflow
	Underflow
	Inexact

SUB

SUB

SUBTRACT

Mnemonic SUB

Opcode 0x03

Operation $Op3 = Op2 - Op1$

Description Operand 1 is subtracted from operand 2 and the result is stored in operand 3.

The operation of SUB is identical to that of ADD except that the sign bit of Op1 is complemented before the addition is performed.

Special Cases:

The following table specifies the action to be taken for each combination of special-case values input. Each action is specified as a number. The meaning of each number is specified in Table 5-7.

Op1	Op2					
	+Infinity	-Infinity	NaN	TNaN	Zero	Normal
+Infinity	3	1	3	1	5	5
-Infinity	1	3	3	1	5	5
NaN	6	6	3	1	6	6
TNaN	1	1	1	1	1	1
Zero	5	3	3	1	7	3
Normal	5	3	3	1	6	8

Note: The sign bit of Op1 is complemented before any special-case detection is performed.

Condition N ← sign bit of value stored in Op3
Flags of the ASR Z ← 1 if 0 stored in Op3, else 0
IO ← 0
PR ← 0
UO ← 0

SUB

SUB

Exceptions

Invalid operation
Overflow
Underflow
Inexact

WRASR

WRASR

MOVE TO ASR

Mnemonic WRASR

Opcode 0x09

Operation ASR = Op1

Description Operand 1 is copied to the ASR. Operand 1 is left unaffected. In coprocessor mode, the contents of the DR are destroyed.

This operation can be used to initialize the ASR or to restore its contents upon context switch.

If Op1 has a size other than single-word, the last word of Op1 is stored in the ASR.

This opcode should not be used in peripheral mode. Instead, the ASR should be written directly to MAU location 000.

Note: Executing this operation when WF = 1 is not interpreted by the MAU as restarting the operation which caused the fault.

Special Cases: None

Condition N ← bit in N-bit position in Op1
Flags of Z ← bit in Z-bit position in Op1
the ASR IO ← bit in IO-bit position in Op1
PR ← bit in PR-bit position in Op1
UO ← bit in UO-bit position in Op1

Exceptions None

5.2.3 Instruction Set Summary by Mnemonic

The instruction sets are listed by mnemonic in Table 5-8.

Table 5-8. Instruction Set Summary by Mnemonic				
Mnemonic	Opcode	Instruction	Operation	Condition Codes Affected
ABS	0x0C	Absolute value	Clear sign bit of operand 1 and copy operand 1 into operand 3. Operand 1 is unaffected.	N,Z,IO,PR,UO
ADD	0x02	Add	Add operand 1 to operand 2 and store result in operand 3.	N,Z,IO,PR,UO
ATAN	0x1E	Arc tangent	The arctangent of operand 1 is computed and stored in operand 3.	N,Z,PR,IO,UO
CMP	0x0A	Compare	Compare operand 1 to operand 2 and set condition codes. Operand 1 and operand 2 are not modified.	N,Z,IO,PR,UO
CMPE	0x0B	Compare with exceptions	Compare operand 1 to operand 2 and set condition codes. Operand 1 and operand 2 are not modified.	N,Z,IO,PR,UO
CMPES	0x1B	Compare with exceptions and flags swapped	Compare operand 1 to operand 2 and set condition codes. Operand 1 and operand 2 are not modified.	N,Z,IO,PR,UO
CMPS	0x1A	Compare with flags swapped	Compare operand 1 to operand 2 and set condition codes. Operand 1 and operand 2 are not modified.	N,Z,IO,PR,UO
COS	0x1D	Cosine	The cosine of operand 1 is computed and stored in operand 3.	N,Z,PR,IO,UO

INSTRUCTION SET
Instruction Set Summary by Mnemonic

Table 5-8. Instruction Set Summary by Mnemonic (Continued)				
Mnemonic	Opcode	Instruction	Operation	Condition Codes Affected
DIV	0x04	Divide	Divide operand 2 by operand 1 and store result in operand 3.	N,Z,IO,PR,UO
DTOF	0x11	Convert decimal to floating-point	Convert operand 1 (in decimal format) to floating-point format and then copy it into operand 3. Operand 1 is unaffected.	N,Z,IO,PR,UO
EROF	0x14	Extract result on fault	Place value of DR in operand 3. Only execute this instruction when a fault condition occurs. If this instruction is executed under normal conditions, the value returned is indeterminate.	None
FTOD	0x12	Convert floating-point to decimal	Round operand 1 to an integral value and then convert it into decimal format and copy it into operand 3. Operand 1 is unaffected.	N,Z,IO,PR,UO
FTOI	0x0F	Convert floating-point to integer	Convert operand 1 from floating-point to 2's complement integer and copy it into operand 3 (single word only).	N,Z,IO,PR,UO
ITOF	0x10	Convert integer to floating-point	Convert operand 1 from 2's complement integer format to floating-point and copy it into operand 3.	N,Z,IO,PR,UO
LDR	0x18	Load data register	Place operand 1 in data register.	Undefined
MOVE	0x07	Move	Copy operand 1 into operand 3. Operand 1 is unaffected.	N,Z,IO,PR,UO
MUL	0x06	Multiply	Multiply operand 1 by operand 2 and store result in operand 3.	N,Z,IO,PR,UO

INSTRUCTION SET
Instruction Set Summary by Mnemonic

Table 5-8. Instruction Set Summary by Mnemonic (Continued)

Mnemonic	Opcode	Instruction	Operation	Condition Codes Affected
NEG	0x17	Negate	Compliment sign of operand 1 and copy operand 1 into operand 3. Operand 1 is unaffected.	N,Z,IO,PR,UO
NOP	0x13	No operation	Cause any operation in progress to terminate without affecting any of the MAU internal registers.	None
PI	0x1F	Pi	The constant pi is stored in operand 3.	N,Z,IO,PR,UO
RDASR	0x08	Move from ASR	Copy ASR into operand 3. The ASR is unaffected.	None
REM	0x05	Remainder	Divide operand 2 by operand 1 and store IEEE remainder in operand 3.	N,Z,IO,PR,UO
RTOI	0x0E	Round to integral value	Round operand 1 to an integral value in floating-point format and then copy it into operand 3. Operand 1 is unaffected.	N,Z,IO,PR,UO
SIN	0x1C	Sine	The sine of operand 1 is computed and stored in operand 3.	N,Z,PR,IO,UO
SQRT	0x0D	Square root	Compute square root of operand 1 and store in operand 3. Operand 1 is unaffected.	N,Z,IO,PR,UO
SUB	0x03	Subtract	Subtract operand 1 from operand 2 and store result in operand 3.	N,Z,IO,PR,UO
WRASR	0x09	Move to ASR	Copy operand 1 into ASR. Operand 1 is unaffected.	N,Z,IO,PR,UO

INSTRUCTION SET

Instruction Set Summary by Opcode

5.2.4 Instruction Set Summary by Opcode

Table 5-9 lists the instructions by opcode.

Table 5-9. Instruction Set Summary by Opcode		
Opcode	Mnemonic	Instruction
0x02	ADD	Add
0x03	SUB	Subtract
0x04	DIV	Divide
0x05	REM	Remainder
0x06	MUL	Multiply
0x07	MOVE	Move
0x08	RDASR	Move from ASR
0x09	WRASR	Move to ASR
0x0A	CMP	Compare
0x0B	CMPE	Compare with exceptions
0x0C	ABS	Absolute value
0x0D	SQRT	Square root
0x0E	RTOI	Round to integral value
0x0F	FTOI	Convert floating-point to integer
0x10	ITOF	Convert integer to floating-point
0x11	DTOF	Convert decimal to floating-point
0x12	FTOD	Convert floating-point to decimal
0x13	NOP	No operation
0x14	EROF	Extract result on fault
0x17	NEG	Negate
0x18	LDR	Load data register
0x1A	CMPS	Compare with flags swapped
0x1B	CMPEX	Compare with exceptions and flags swapped
0x1C	SIN	Sine
0x1D	COS	Cosine
0x1E	ATAN	Arctangent
0x1F	PI	Pi

**Glossary
and
Acronyms**



Biased exponent – The sum of the exponent and a constant (bias) chosen to make the biased exponent's range nonnegative.

Binary floating-point number – A bit string characterized by three components: a sign, a signed exponent, and a significand. Its numerical value, if any, is the signed product of its significand and 2 raised to the power of its exponent.

Cleared – The condition in which a bit is given the value 0.

Coprocessor broadcast access – A transaction initiated by the CPU to bring the MAU out of its quiescent state. The MAU tests the command word on the data bus, and if the ID field of the command word matches the ID field of the MAU, the data is latched into the MAU's command register (CR).

Denormalized – A double-extended precision floating-point number whose significand has been shifted to the right, and whose exponent has been incremented to zero. The exponent is the format's minimum, the explicit or implicit bit is zero, and the number is not normal.

Explicit bit – The single bit to the left of the binary point in double-extended precision numbers. This bit is explicitly represented in this format.

Exponent – That component of a binary floating-point number which normally signifies the power to which 2 is raised in determining the value of the represented number. Occasionally the exponent is called the signed or unbiased exponent.

Format's maximum (max) – The value of a field such that all bits in the field are 1.

Format's minimum (min) – The value of a field such that all bits in the field are 0.

Fraction – The field of the significand that lies to the right of its implied binary point.

ID field – Bits 24–31 of the MAU's command register. It is included in the command word sent to the MAU by the CPU. The ID field specifies the identification number associated with the processor that should react to this command word. The MAU's identification number is 0.

Implicit bit – The single bit to the left of the binary point in single- and double-precision numbers. In these cases, the bit is assumed to be 1 and is not explicitly represented in the format. It is therefore implicit. If the exponent of single or double precision is the format's minimum, the implicit bit is 0 and the number is said to be denormalized.

Inexact – An exception that occurs if the infinite precision result cannot be represented in the destination format and no other exceptions have occurred.

Invalid operation – An exception that occurs if a result cannot be legally stored in a destination, or if illegal operands are given to some operation.

Mask bit – A bit associated with exceptions. If the bit is set, an exception may occur. If it is cleared, exceptions may not occur.

NaN – Not a number: a symbolic entity encoded in floating-point format. Operations that have no mathematical interpretation, such as zero divided by zero will produce a NaN. Such NaNs can be used to convey diagnostic information regarding their creation.

Neither max nor min (notmm) – The value of a field such that all bits are neither all 1s nor all 0s.

GLOSSARY

Non-trapping NaN – A type of NaN that does not cause a trap when it is detected by the MAU.

Normalized – A number whose significand is shifted to the left while decrementing its exponent until the leading significand bit becomes one. The exponent is regarded as if its range were unlimited. If the significand is zero, the number becomes normal zero.

Normalizing a number does not change its sign.

Operand specifiers – Operand specifiers refer to the three fields in the command word which specify where each operand is located. The two source operands are specified by bits 9–7 and bits 6–4 of the command word. The destination is specified by bits 3–0 of the command word.

Operation – The period from the time the MAU has all of the input operands until the time when a result has been computed and the MAU is ready to store the result or perform some other operation.

Overflow – An exception that occurs when the exponent of a rounded result of an arithmetic operation is too large to represent in the exponent field of the destination format.

Quiescent state – The MAU accepts and reacts to data read and write (if chip select is asserted) and coprocessor broadcast accesses only. It accepts resets, does not read or write to any internal registers, and does not affect the rest of the system in any way.

Set – The condition in which a bit is given the value 1.

Significand – The component of a binary floating-point number that consists of an explicit or implicit leading bit to the left of its binary point and a fraction field to the right of its binary point.

Status fetch access – The CPU reads the MAU's auxiliary status register (ASR) to obtain the status of the MAU. The CPU blocks interrupts until this transaction is complete.

Sticky bit – A bit associated with exceptions. If the condition associated with the exception type is satisfied, the MAU sets the sticky bit for that exception type.

Transaction – A complete set of interactions between the CPU and the MAU. This includes the transferring of a command word, transferring operands, performing an operation, and reading the MAU's status.

Trapping NaN – A type of NaN that causes a trap whenever it is detected by the MAU.

Underflow – An exception that occurs if the exponent of a rounded result of an arithmetic operation is too small to represent in the exponent field of the destination format.

Unnormalized – Occurs only in the double-extended format. The number's exponent is greater than the format's minimum and the explicit leading bit is zero. If the significand is zero this is an unnormalized zero.

x – Don't care condition.

Zero – A single, double, or double-extended precision number whose exponents and significands are the format's minimum. Normal zero may have either a positive or a negative sign. Only the extended format has unnormalized zero.

ASR – Auxiliary status register
BCD – Binary coded decimal
CMOS – Complimentary metal-oxide semiconductor
CPU – Central processing unit
CR – Command register
DR – Data register
FE – Feature enable bit
ID – Processor ID number field
IM – Invalid operation mask bit
IO – Integer overflow bit
J – Explicit bit
LSB – Least significant bit
MAU – Math acceleration unit
MIS – MAU instruction set
MSB – Most significant bit
N – Condition flag bit negative
NaN – Not a number
NOP – No operation
OM – Overflow mask bit
OS – Overflow sticky bit

PGA – Pin grid array
PM – Inexact mask bit
PR – Partial remainder bit
PS – Inexact sticky bit
PSW – Processor status word
QM – Divide-by-zero mask bit
QS – Divide-by-zero sticky bit
RA – Result available bit
RC – Round control field
RM – Round towards negative infinity
RN – Round to nearest
RP – Round towards positive infinity
RZ – Round towards zero
TNaN – Trapping not a number
UM – Underflow sticky bit
UO – Condition flag bit unordered
US – Underflow sticky bit
UW – Unaligned word bit
WF – Write fault indicator bit
Z – Condition flag bit zero



Index



A

Absolute value (ABS), 5-7
 Access status codes, 3-3
 Add (ADD), 5-8
 Address signals, 3-1
 Arc tangent (ATAN), 5-9
 Arithmetic instructions, 5-2

B**Bus**

exception signals, 3-4
 exceptions, 2-19
 transactions, chapt. 4

C

Chip select signal, 3-2
 Clock signals, 3-4
 Compare (CMP), 5-11
 Compare instructions, 5-2
 Compare with exceptions (CMPE), 5-12
 Compare with exceptions and flags swapped (CMPES), 5-13
 Compare with flags swapped (CMPS), 5-14
 Condition
 codes, 2-10
 flags, 5-4
 Configuration restrictions, 4-4, 4-33
 Context
 restoring, 2-23
 saving, 2-23
 switch control, 2-23
 Conventions, 1-1
 Conversion instructions, 5-3
 Convert
 decimal to float (DFOF), 5-18
 float to decimal (FTOD), 5-20
 float to integer (FTOI), 5-22
 integer to float (ITOF), 5-24
 Coprocessor
 broadcast
 access, 4-12
 transactions, 4-11, 4-13
 data
 fetch, 4-17
 write, 4-24

faults, 4-30

mode, 1-2, 2-24, 2-25, 4-11
 transactions, 4-11

status read, 4-21

Cosine (COS), 5-15

Cycle initiate signal, 3-2

D**Data**

bus shadow signal, 3-2

formats, 1-1, 2-1

read transactions, 4-4, 4-17

ready signal, 3-2

signals, 3-1

size signals, 3-3

strobe signal, 3-2

transfer

 acknowledge signal, 3-2

 instructions, 5-2

 types, 1-1, 2-1

 write transactions, 4-4, 4-24

Decimal format, 2-8

Denormalized numbers, 2-2, 2-4

Divide (DIV), 5-17

Divide-by-zero, 2-19

Done signal, 3-2

Double precision, 2-3

Double-extended precision, 2-6

E

Exceptions, 2-19

 bits, 2-10

 masks, 2-10

Explicit bit, 2-6, 2-14

Exponent, 2-1, 2-4, 2-6

 bias, 2-1, 2-4, 2-6

 field, 2-1, 2-4, 2-6

Extract result on fault (EROF), 5-19

F

Fault signal, 3-4

Faults

 coprocessor, 4-30

 non-existent, 4-31

 peripheral mode, 4-9

 status fetch, 4-30

INDEX

Floating-point standard, 1-4
Fraction field, 2-1, 2-4, 2-6

I

Identification field, 2-16
Implicit bit, 2-1, 2-4
Inexact exception, 2-22
Input clock signals, 3-4
Instructions, Chapt. 5
 absolute value (ABS), 5-7
 add (ADD), 5-8
 arithmetic, 5-2
 compare, 5-2
 compare (CMP), 5-11
 compare with exceptions (CMPE), 5-12
 compare with exceptions and flags swapped (CMPES), 5-13
 compare with flags swapped (CMPS), 5-14
 conversion, 5-3
 convert decimal to float (DFOF), 5-18
 convert float to decimal (FTOD), 5-20
 convert float to integer (FTOI), 5-22
 convert integer to float (ITOF), 5-24
 cosine (COS), 5-15
 data transfer, 5-2
 divide (DIV), 5-17
 extract result on fault (EROF), 5-19
 load data register (LDR), 5-25
 miscellaneous, 5-3
 move (MOVE), 5-26
 move from ASR, (RDASR), 5-32
 move to ASR (WRASR), 5-42
 multiply (MUL), 5-28
 negate (NEG), 5-29
 no operation (NOP), 5-30
 pi (PI), 5-31
 remainder (REM), 5-33
 round to integral value (RTOI), 5-35
 sine (SIN), 5-37
 square root (SQRT), 5-39
 subtract (SUB), 5-40
 summary, 5-4
 by mnemonic, 5-43
 by opcode, 5-46
 transcendental, 5-3

Integer, 2-10
 format, 2-10
Interconnections, 2-25
Interface and control signals, 3-2
Interrupt acknowledge cycle, 2-23
Interrupts, 2-23
Invalid operation, 2-19

L

Load data register (LDR), 5-25

M

Miscellaneous instructions, 5-3
Mnemonics, 5-4
Move (MOVE), 5-26
Move from ASR (RDASR), 5-32
Move to ASR (WRASR), 5-42
Multiply (MUL), 5-28

N

Negate (NEG), 5-29
No operation (NOP), 5-30
Non-existent MAU faults, 4-31
Normalized numbers, 2-1, 2-4, 2-6
Notation, 5-4

O

Opcodes, 5-4, 5-46
Operand
 registers, 2-15
 specifiers, 2-15
Overflow, 2-20

P

Peripheral mode, 1-3, 2-22, 2-23, 4-1
 address fields, 4-1
 faults, 4-9
 interrupt acknowledge signals, 3-2
 interrupt request signals, 3-2
 read, 4-4
 write, 4-4
Pi (PI), 5-31
Protocol modes, 1-2

Q

Quiescent state, 2-24

R

Read/write signal, 3-3

Registers, 2-10

 auxiliary status, 2-10

 command, 2-14

 data, 2-17

 operand, 2-14

Remainder (REM), 5-33

Reset, 2-24

 signal, 3-4

Restartability

 coprocessor, 4-31

 peripheral mode, 4-10

Round control, 2-10, 2-13

 bits, 2-13, 2-18

Round to integral value (RTOI), 5-35

Rounding, 2-18

S

Sign field, 2-1, 2-4, 2-6

Signals, Chapt. 3

 access status codes, 3-3

 address, 3-1

 bus exception, 3-4

 chip select, 3-2

 clock, 3-4

 cycle initiate, 3-2

 data, 3-1

 bus shadow, 3-2

 ready, 3-2

 strobe, 3-2

 transfer acknowledge, 3-2

 done, 3-2

 fault, 3-4

 input clock, 3-4

 interface and control, 3-2

 peripheral mode

 interrupt acknowledge, 3-2

 interrupt request, 3-2

 read/write, 3-3

 reset, 3-4

 status, 3-3

 synchronous ready, 3-3

 test, 3-4

Significand, 2-17, 2-20

Sine (SIN), 5-37

Single precision, 2-1

Special cases, 2-5, 2-7, 2-9, 5-4

Square root (SQRT), 5-39

Status

 fetch faults, 4-30

 signals, 3-3

Sticky bits, 2-10

Subtract (SUB), 5-40

Synchronous ready signal, 3-3

T

Test signal, 3-4

Transcendental instructions, 5-3

U

Underflow, 2-21

Notes

