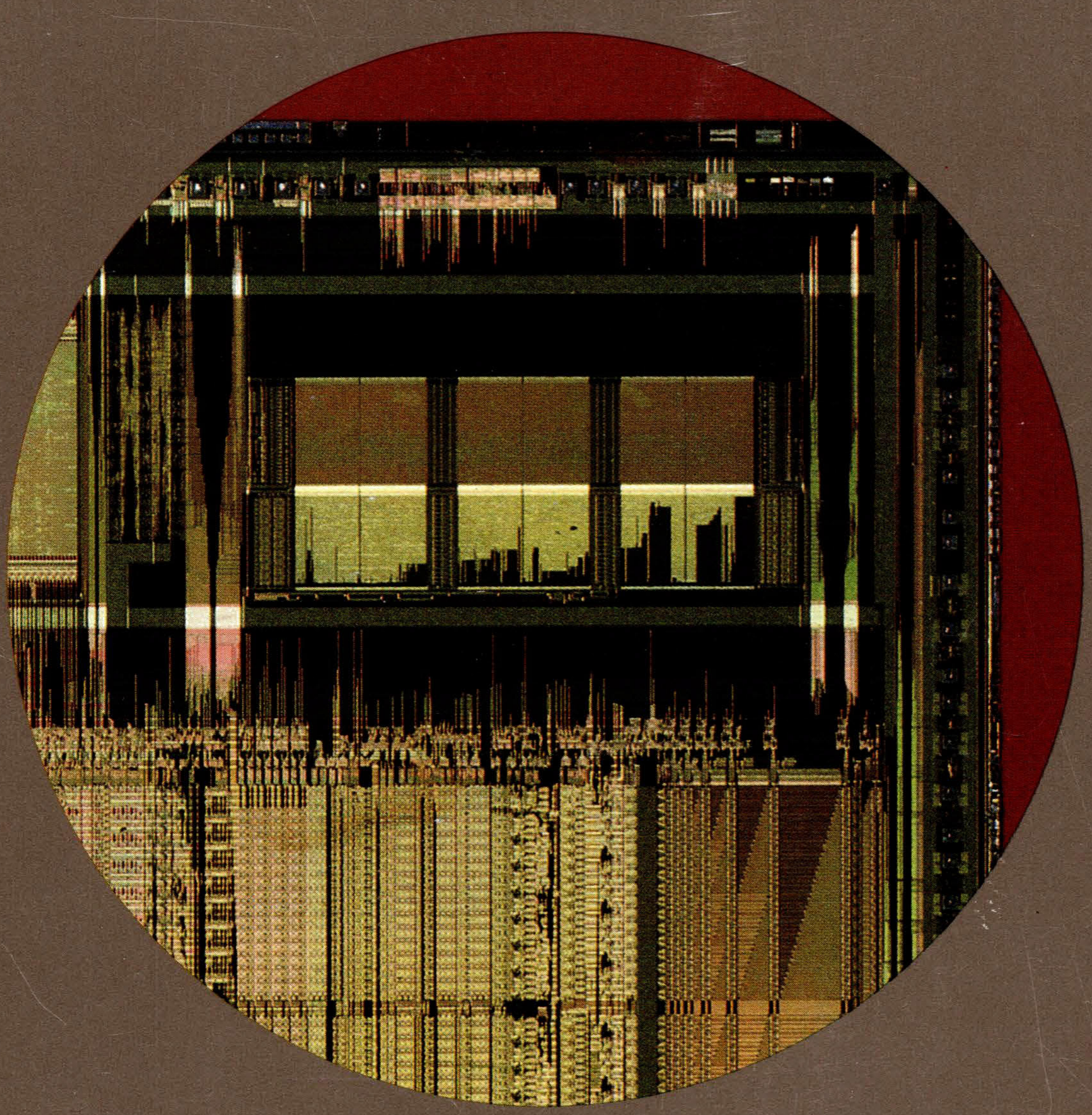
Roscoe

AT&T

WE® 32106 Math Acceleration Unit Information Manual



AT&T *UNIX*® Microsystem

Select Code 307-734

November 1986

A WORD ABOUT TRADEMARKS . . .

The following trademarks mentioned in this manual:

WE® 32100 Microprocessor, CPU

WE® 32101 Memory Management Unit

WE® 32102 Clock

WE® 32106 Math Acceleration Unit, MAU

WE® 321SG Software Generation Programs

UNIX® Operating System

are registered trademarks of AT&T.

Printed in United States of America

AT&T

November 1986

WE® 32106 Math Acceleration Unit

Information Manual

Select Code 307-734

COMCODE 105195390

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* 32106 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 documentation from the *UNIX* Operating System.

For details concerning warranty, repairs, or refurbishment 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 307-734, call:

- **1-800-432-6600**

WE 32106 MATH ACCELERATION UNIT

INFORMATION MANUAL

CONTENTS

CHAPTER 1. INTRODUCTION

CHAPTER 2. FUNCTIONAL DESCRIPTIONS

CHAPTER 3. MAU SIGNAL DESCRIPTIONS

CHAPTER 4. BUS TRANSACTIONS

CHAPTER 5. INSTRUCTION SET

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

CHAPTER 1. INTRODUCTION

CONTENTS

1. INTRODUCTION

The *WE* 32106 Math Acceleration Unit (MAU) provides floating-point coprocessing capability for the *WE* 32100 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* 32100 CPU). It provides a tightly coupled interface, which increases performance over that of peripheral mode. Since peripheral mode provides a loosely coupled interface between the MAU and CPU, it is best used with 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. 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.

In systems using the *WE* 32100 CPU, programming the MAU is simplified by using the MAU instruction set (MIS), an addition to the *WE* 32100 Microprocessor assembly language instruction set. By using the MIS, the task of providing the proper sequences of CPU coprocessor instructions and MAU operands is eliminated. These instructions, as well as a full description of floating-point support, are found in the ***WE*® 32ISG Software Generation Programs User Guide**.

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 10-, 14-, and 18-MHz 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.

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.

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* 32106 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* 32100 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's command register (CR) format (see **2.2 Registers**). 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.

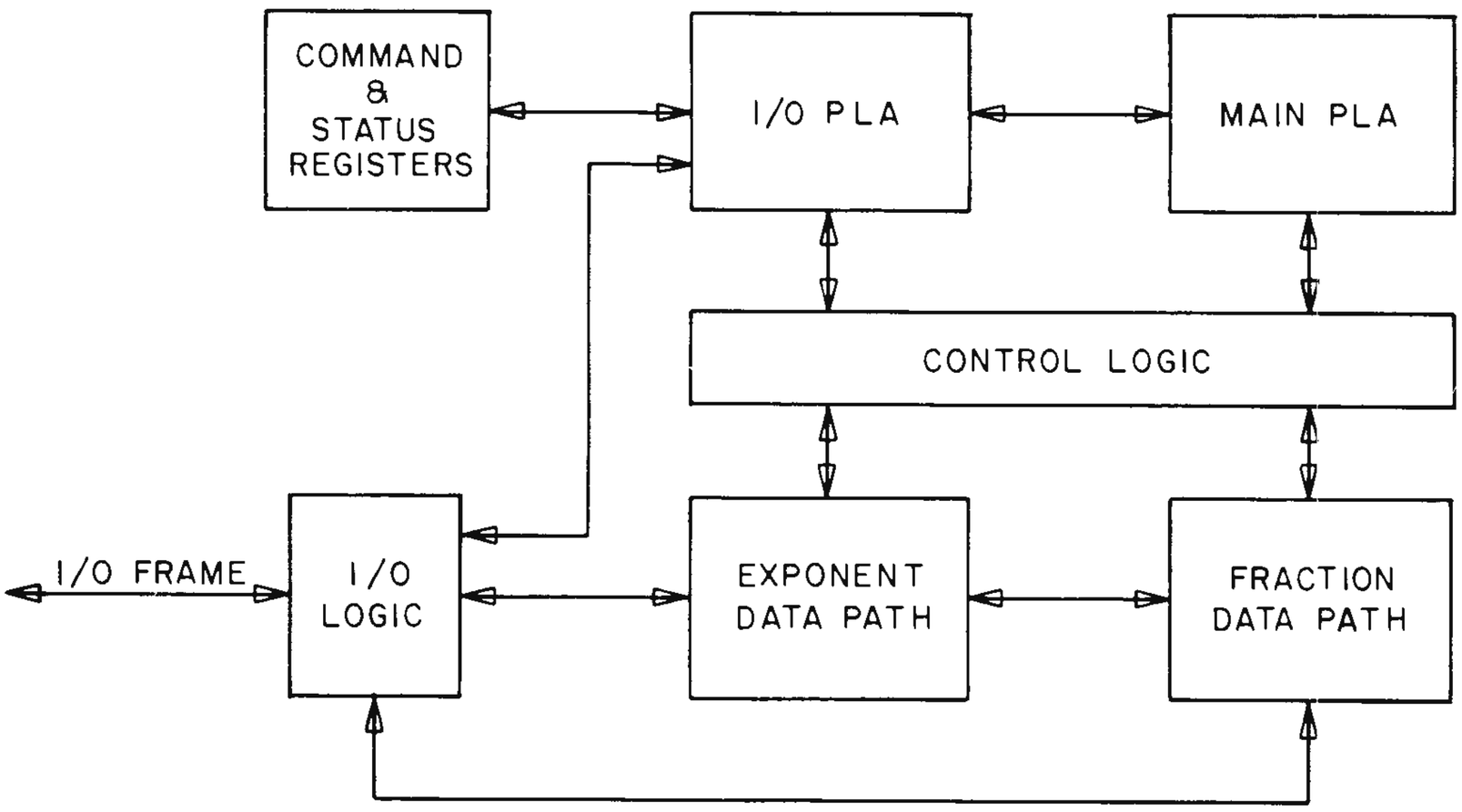


Figure 1-1. *WE* 32106 Math Acceleration Unit Block Diagram

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 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 and negates the $\overline{\text{DONE}}$ signal. The command is then executed by using the operands available in registers F0 through F3 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 (1) and the MAU enters the quiescent state.

1.4 DATA TYPES AND FORMATS

The *WE* 32106 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.

The integer format provides a binary representation of signed integers in 32-bit 2's complement format.

1.5 IEEE COMPATIBILITY

The *WE* 32106 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 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 ***WE*® 32106 Math Acceleration Unit Data Sheet.**

1.6.1 Functional Descriptions

Chapter 2 provides a description of the hardware associated with the *WE* 32106 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 bus transactions of which the MAU is a participant. 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, 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

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* 32106 Math Acceleration Unit (MAU) 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	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, which lies immediately to the left of fraction bit 22.
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.

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 $2-2^{-23}$.

Thus 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.

Thus 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 sored 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.

Table 2-3. Double Precision Format

Bit	63	62 52	51 0
Field	Sign	Exponent	Fraction

Bit	Field	Name/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.

Therefore, 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.

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	Unused	Sign	Exponent	J	Fraction

Bit	Field	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	**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 $2-2^{-63}$.

Thus 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 (explicit) 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.

Table 2-6. Double-Extended Precision Special Case Values

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 sored 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
J+fraction refers to the value of the J and fraction bits, considered as one large field

When the MAU generates a nontrapping NaN, J+fraction contains 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	95 76	75 72	71 68	67 64	63 60	59 56	55 52	51 48	47 44	43 40
Field	UNUSED	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. Each digit is 4-bits wide.

Bit	Field	Description
0—3	Sign	**Sign Bits.** 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:
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	**Unused.** These bits are ignored and overwritten with 0s.

Bits 3,2,1,0	Meaning	Bits 3,2,1,0	Meaning
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

Numbers

Each digit is in the range 0 to 9. Thus, the decimal format can represent numbers in the range ± 0 through $10^{18}-1$.

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.

Table 2-8. Decimal Format Special Case Values

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

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.

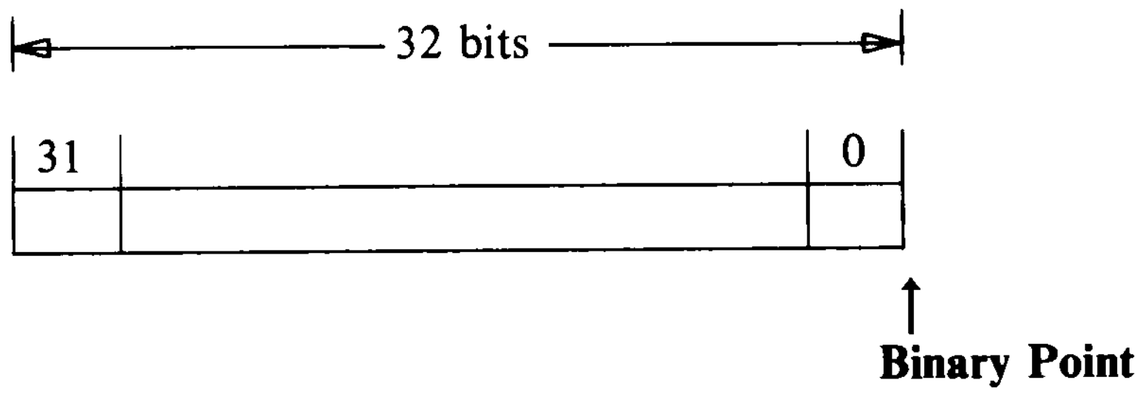


Figure 2-1. Integer Format

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.
- Four operand registers (F0—F3), used to hold floating-point operands.
- A command register (CR), used to specify and initiate operations via peripheral mode accesses. 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 of an exception occurs, the information supplied to the exception handler is stored in 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* 32100 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.

Table 2-9. Auxiliary Status Register

Bit	15	14	13	12	11	10	9	8	7	6	5	4 0
Field	UNUSED	IM	OM	UM	QM	PM	IS	OS	US	QS	PR	UNUSED

Bit(s)	Field	Contents	Description
0—4	Unused	Unused	These bits appear as 0 when read.
5	PR	Partial Remainder	Set (1) when result of a remainder operation is a partial remainder; cleared (0) when result is a full remainder.
6	QS	Divide by Zero Sticky	Set (1) if the divisor is normalized zero and dividend is a finite nonzero number.
7	US	Underflow Sticky	Set (1) 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 (1) 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 (1) 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 (1) by the user, an exception occurs when bit 18 of the ASR is set; if this bit is cleared (0), there are no inexact exceptions.
11	QM	Divide by Zero Mask	If this bit is set (1) by the user, an exception occurs when bit 6 of the ASR is set; if this bit is cleared (0), there are no divide by zero exceptions.

Table 2-9. Auxiliary Status Register (Continued)

Bit	31	30 26	25	24	23 22	21	20	19	18	17	16
Field	RA	UNUSED	ECP	NTNC	RC	N	Z	IO	PS	CSC	UO

Bit(s)	Field	Contents	Description
12	UM	Underflow Mask	If this bit is set (1) by the user, an exception occurs when bit 7 of the ASR is set; if this bit is cleared (0), there are no underflow exceptions.
13	OM	Overflow Mask	If this bit is set (1) by the user, an exception occurs when bit 8 of the ASR is set; if this bit is cleared (0), there are no overflow exceptions.
14	IM	Invalid Operation Mask	If this bit is set (1) by the user, an exception occurs when bit 9 of the ASR is set. If this bit is cleared (0), there are no invalid operation exceptions.
15	Unused	Unused	This bit appears as 0 when read.
16	UO	Unordered	Set (1) when a compare operation results in an unordered indication; otherwise, it is cleared (0).
17	CSC	Context Switch Control	Set (1) on every MAU instruction execution, except move from ASR (RDASR) and load DR (LDR) instructions. **Note:** Reading the ASR leaves the contents of the DR unchanged.
18	PS	Inexact Sticky	Set (1) if the result cannot be specified in the destination format.
19	IO	Integer Overflow	Set (1) when a convert float-to-integer operation causes an overflow. **Note**: The integer overflow condition is nonmaskable in the MAU.
20	Z	Zero*	Set (1) if result of last operation is zero; cleared (0) if result is nonzero.
21	N	Negative*	Set (1) if result of the last operation is negative; cleared (0) if result is positive. **Note:** It is possible for an operation to result in both bits 20 and 21 being set since negative zero is a representable number.

*Bit is undefined when the result for the destination is NaN.

Table 2-9. Auxiliary Status Register (Continued)

Bit(s)	Field	Contents	Description
22,23	RC	Round Control	Represents the round control mode. The code is interpreted as: **Bits 23, 22** — **Mnemonic** — **Description** 00 — RN — Round to nearest 01 — RP — Round towards plus infinity 10 — RM — Round towards minus infinity 11 — RZ — 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 (bit 24) is set (1), 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. If bit 24 is cleared (0), no exception occurs and a nontrapping NaN is generated (generated nontrapping NaNs have the least significant bit of the fraction portion set).
25	ECP	Exception Condition Present	Set (1) if any one of the floating-point exceptions is present.
26—30	Unused	Unused	These bits appear as 0 when read.
31	RA	Result Available	Cleared (0) at beginning of an operation and set (1) when result of an operation is available. During the quiescent state, this bit has a value of 1.

2.2.2 Operand Registers

Each of the four operand registers (F0—F3) 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 data register. In peripheral mode, bits 80 through 95 are ignored during writes and returned as 0s during read operations. Registers F0 through F3 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	UNUSED	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	A 1 represents a negative value; a 0 represents a positive value.
80—95	Unused	—	—

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 — 15	14 — 10	9 — 7	6 — 4	3 — 0
Field	ID	UNUSED	OPCODE	OP1	OP2	OP3

Bit(s)	Field	Contents	Description
0—3	Op3	Operand Specifier	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. The precision designations are used for rounding and checking for underflow and overflow. The value of this field is interpreted as: **Bits 3,2,1,0 / Operand Register / Destination Precision** 0000 F0 Single 0001 F1 Single 0010 F2 Single 0011 F3 Single 0100 F4 Double 0101 F1 Double 0110 F2 Double 0111 F3 Double 1000 F0 Double-extended 1001 F1 Double-extended 1010 F2 Double-extended 1011 F3 Double-extended 1100 none Memory-based single word 1101 none Memory-based double word 1110 none Memory-based triple word 1111 none No operand
4—6	Op2	Operand Specifier	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: **Value Bits 6,5,4 / Operand Location** 000 Register F0 001 Register F1 010 Register F2 011 Register F3 100 Memory-based single word 101 Memory-based double word 110 Memory-based triple word 111 No operand

Table 2-11. Command Register (Continued)

Bit(s)	Field	Contents	Description
7—9	Op1	Operand Specifier	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: **Value Bits 9,8,7 — Operand Location** 000 — Register F0 001 — Register F1 010 — Register F2 011 — Register F3 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.
15—23	—	Unused	These bits are returned as 0 when read.
24—31	ID	Processor ID Number	Specifies identification number of the coprocessor that should react to the command word. The MAU's 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 **4.1 Peripheral Mode Transactions**.

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. If this exception occurs during conversion or compare operations, refer to the corresponding instruction description for information regarding the contents of the DR.
Overflow or Underflow*	The significand, along with the 17-bit internal result exponent and result sign, is stored in the DR. The most significant bit of the 17-bit exponent behaves like a sign bit in 2's complement notation. An addition bit (bit 79) in the exponent ensures that no significant exponent bits are lost from an internal representation when an overflow or underflow condition occurs. The exponent value is biased by 16383. The format is: **Bit** 81 | 80 64 | 63 | 62 0 **Field** SIGN | EXPONENT | J | FRACTION SIGNIFICAND
Divide by Zero	The dividend (Op2) converted to double-extended precision, if necessary, is stored in the DR.
Inexact	The rounded result converted to double-extended precision, if necessary, is stored in the DR.

* This format is not IEEE standard. It contains more information than the standard requires and can be converted to the standard with software.

2.3 ROUNDING

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

The type of rounding performed is controlled by the round control bits, 22 and 23, of the ASR. 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:

$$Z1 <= Z < Z2.$$

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 (Round to nearest): 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 (Round towards plus infinity): The result is Z2.

RM (Round towards minus infinity): The result is Z1.

RZ (Round towards zero): 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 **2.4.1 Exception Types**.

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 to 1 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 to 1 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).

Overflow

A floating overflow condition exists if the exponent of the rounded result of an arithmetic operation 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 (DR). The destination operand for the operation is left unaltered. Also, the sticky flag for this condition (ASR<OS>) is set to 1. The format of the result stored in DR is as shown below:

81	80 64	63	62 0
SIGN	EXPONENT	J	FRACTION
			← SIGNIFICAND →

The most significant bit 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 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.

Round-to-nearest (RN) mode of rounding carries all overflows to infinity, with the sign of the overflown intermediate result.

Round-towards-zero (RZ) mode of rounding carries all overflows to the format's largest finite number with the sign of the overflown intermediate result.

Round-towards-minus-infinity (RM) mode of rounding carries positive overflows to the format's largest finite number and carries negative overflows to negative infinity.

Round-towards-plus-infinity (RP) mode of rounding 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 the intermediate result. 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. Note that if the destination for the operation is double-extended, no conversion is performed and the significand is rounded to double-extended-precision format.

This significand, along with the 17-bit internal result exponent and result sign, is stored in the data register (DR). The destination operand for the operation is left unaltered. Also, the sticky flag for this condition (ASR<US>) is set to 1. The format of the result stored in DR is as shown below:

81	80 64	63	62 0
SIGN	EXPONENT	J	FRACTION
			← SIGNIFICAND →

The most significant bit 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 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 to 1 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 DR is unnormalized, with its exponent one less than the correct value.

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 to 1 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 is 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.
- Reading the three DR words via peripheral mode and saving their value.
- Reading registers F0—F3 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 four MAU **MOVE** instructions to restore F0—F3. 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 DR with an extract result on fault (EROF) instruction and saving its value.
- Reading the ASR mode or a move from ASR (RDASR) instruction (or peripheral mode) and saving its value.
- Clearing the ASR.
- Performing four MAU **MOVE** operations to save registers F0—F3.

The MAU context can be restored by:

- Clearing the ASR.
- Performing four MAU **MOVE** operations to restore registers F0—F3.
- Writing the saved ASR value into the ASR via peripheral mode or a move to ASR (WRASR) instruction (or peripheral mode).
- Performing a load DR operation using the LDR instruction (or via peripheral mode move) to restore 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, 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—F3. 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 in the 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—F3 (see **2.5 Context Saving and Restoring**).

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—F3.

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 (ASR<RA> = 1).

2.7 MAU RESET

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

1. Any access or operation in progress is immediately terminated.

2. The MAU goes into its quiescent state (see below).

All bits of the command register (CR) and ASR 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 that 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:
  - ☐ Data Read (if chip-select is asserted)
  - ☐ Data Write (if chip-select is asserted)
  - ☐ Coprocessor Broadcast

The MAU will not accept or react to any other access types.

- 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* 32100 Microprocessor (CPU). Refer to the ***WE*® 32100 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* 32100 Microprocessor for peripheral mode operations.

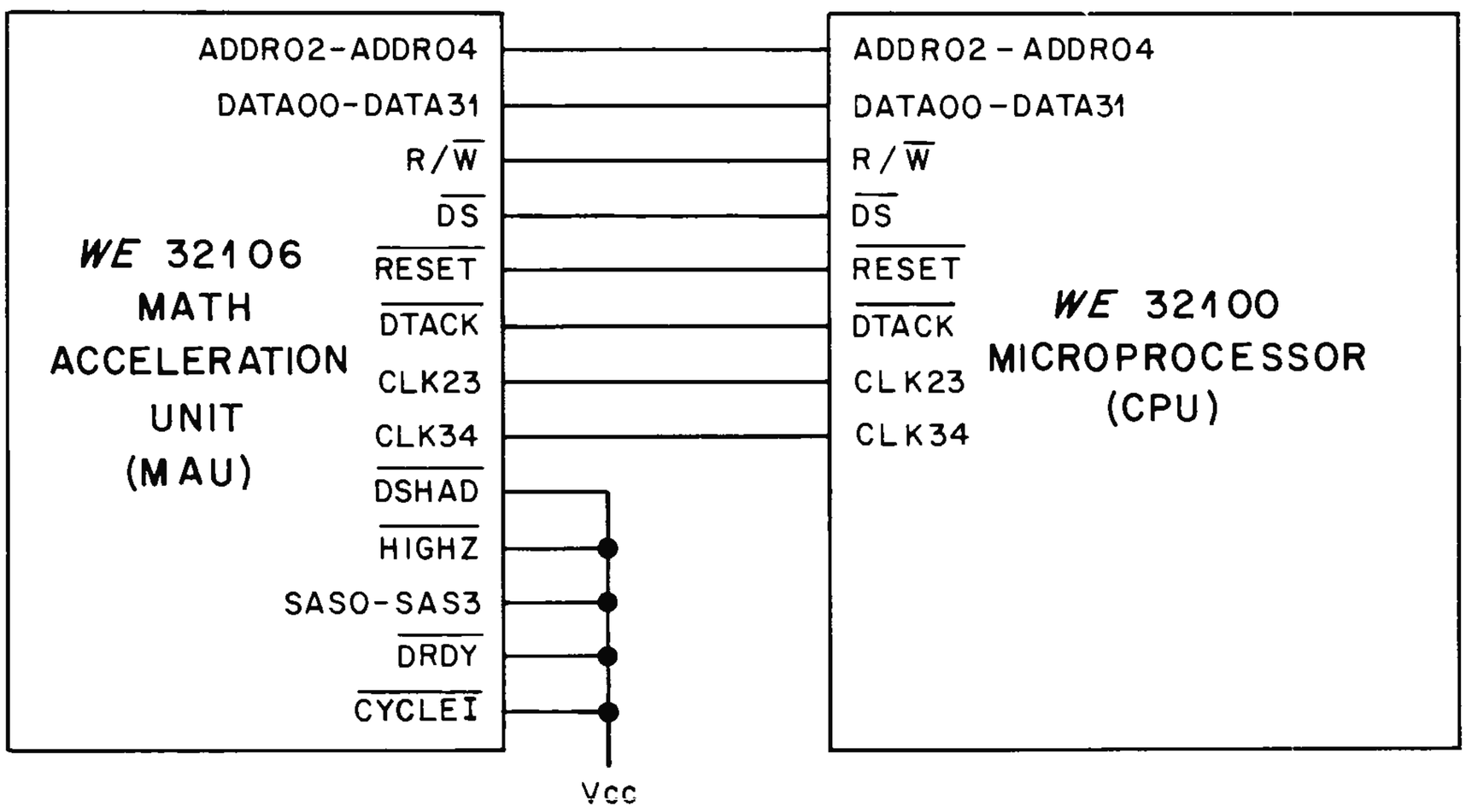
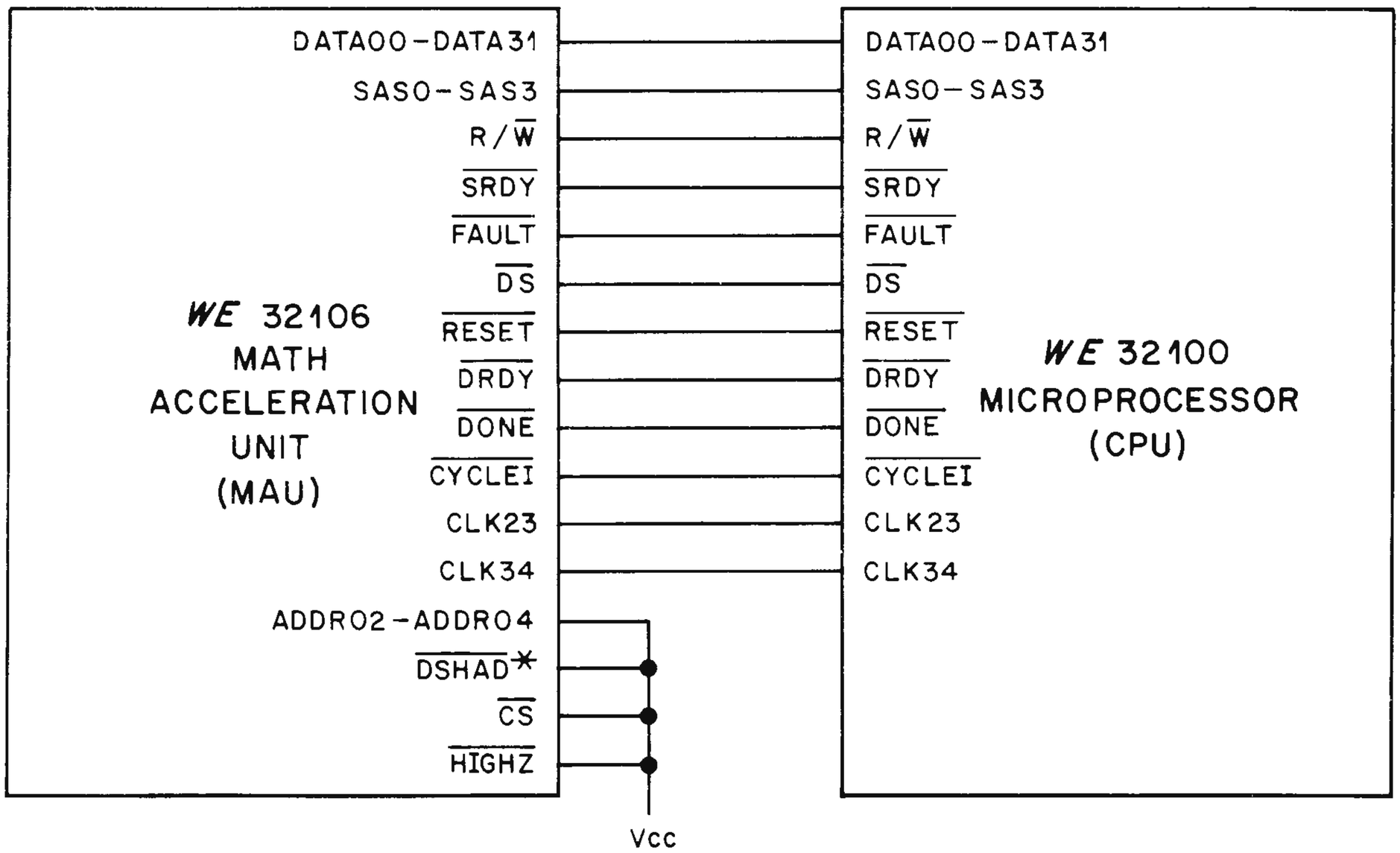


Figure 2-2. MAU-CPU Peripheral Mode Protocol Interconnection

2.9.2 Coprocessor Mode

Figure 2-3 demonstrates how the MAU is directly interfaced to the *WE* 32100 CPU for coprocessor mode operations.



*If a *WE* 32101 Memory Management Unit (MMU) is included in the sytem, this signal must be connected to the MMU's DSHAD output.

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

Chapter 3

MAU Signal Descriptions

CHAPTER 3. MAU SIGNAL DESCRIPTIONS

CONTENTS

3. MAU 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.

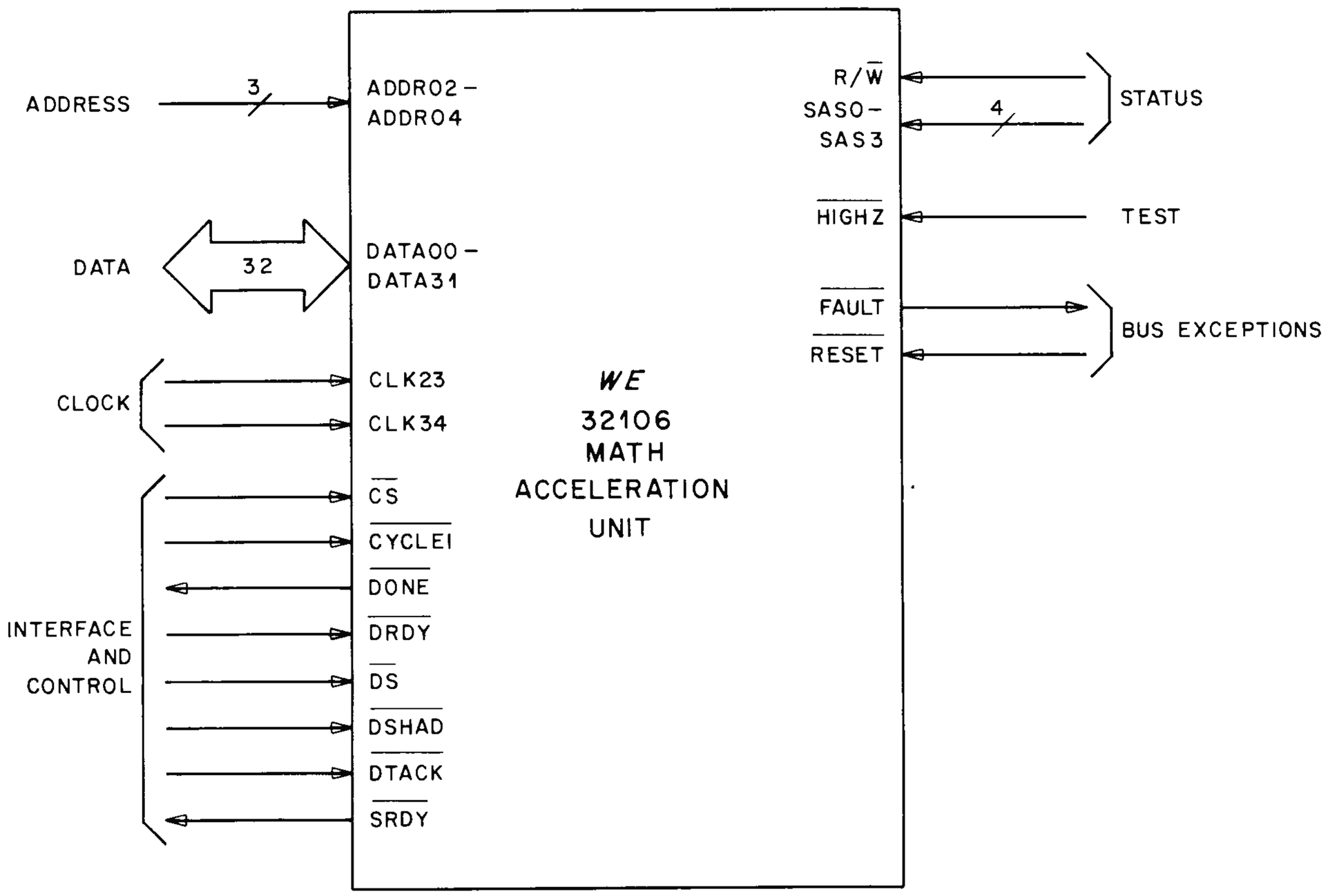


Figure 3-1. Input and Output Signals

3.1 ADDRESS AND DATA SIGNALS

Data (DATA00—DATA31). These signals provide a bidirectional bus to transmit data to and from the MAU.

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.

3.2 INTERFACE AND CONTROL SIGNALS

Chip Select ($\overline{CS}$). Assertion of this input signal puts the MAU in peripheral mode.

Cycle Initiate ($\overline{CYCLEI}$). Assertion of this input indicates that the CPU has started a bus cycle.

Done ($\overline{DONE}$). Assertion of this output signal indicates to the CPU that the MAU has completed execution of the current instruction.

Data Ready ($\overline{DRDY}$). Assertion of this input signal indicates to the MAU that no bus exceptions have been detected during the current bus cycle.

Data Strobe ($\overline{DS}$). When this input is asserted during a read operation, this signal indicates that the MAU may place data on the data bus. When asserted during a write operation, this signal indicates that the MAU may remove data from the data bus.

Data Bus Shadow ($\overline{DSHAD}$). Assertion of this input causes the data bus to 3-state if the MAU is driving it.

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

Synchronous Ready ($\overline{SRDY}$). Assertion of this output causes the CPU to begin the termination of a coprocessor broadcast or coprocessor status fetch access.

3.3 STATUS SIGNALS

Read/Write (R/$\overline{W}$). A 1 on this input indicates a read from the MAU; a 0 indicates a write to the MAU.

Access Status Codes (SAS0—SAS3). These input signals describe the type of bus cycle being executed. SAS0 is the least significant bit of the access status code.

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

Table 3-1. Access Status 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

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 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 SIGNALS

Test ($\overline{\text{HIGHZ}}$). Assertion of this input signal 3-states all output signals for testing.

3.7 PIN ASSIGNMENTS

The *WE* 32106 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 3-2 lists the pin assignments.

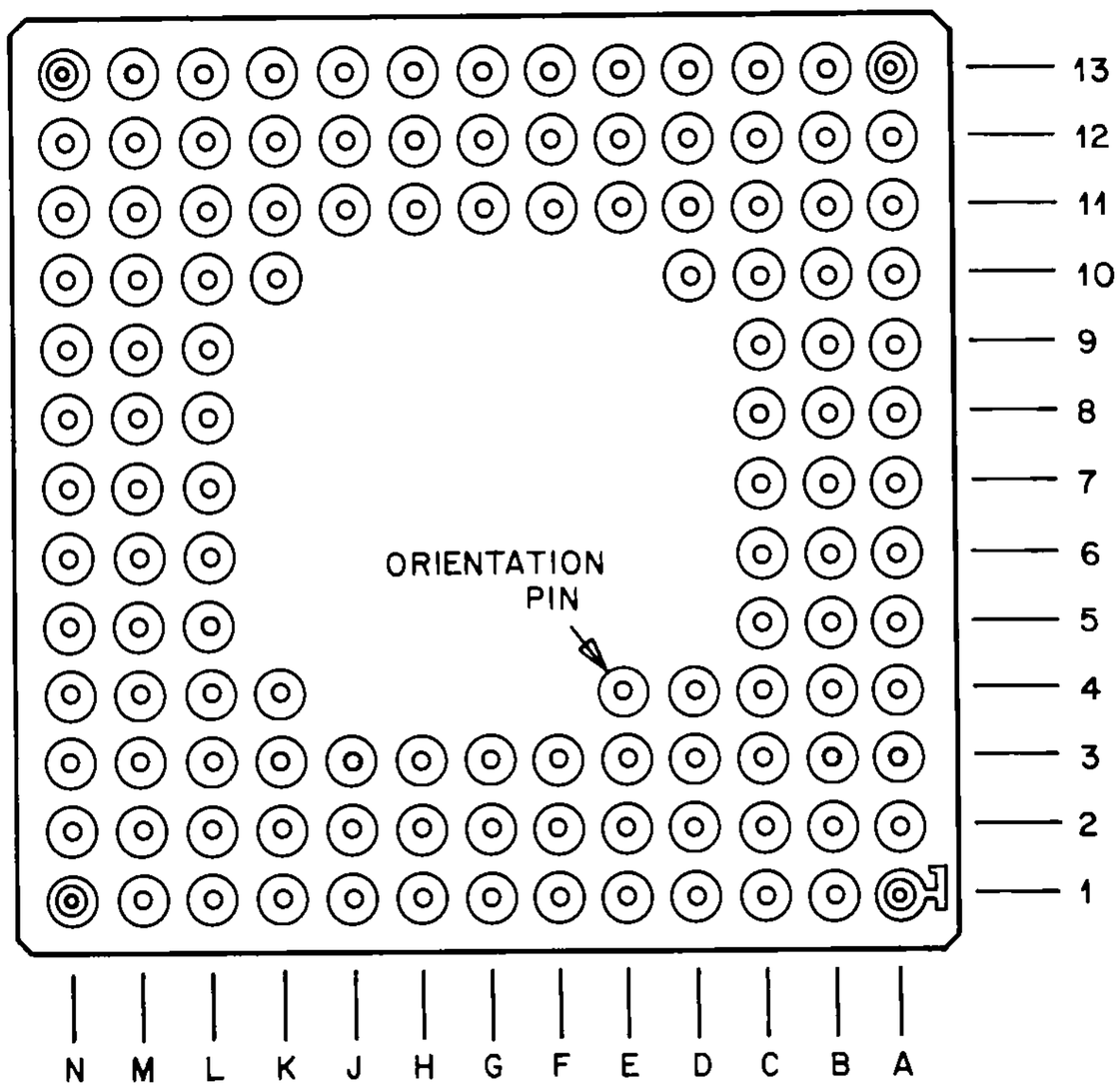


Figure 3-2. 125-Pin Square Ceramic Pin Grid Array – Bottom View

Table 3-2. Pin Descriptions, 125-Pin PGA Package*

Name	Pin	Type	Description
DATA10	A1	I/O	Data 10
DATA13	A2	I/O	Data 13
DATA16	A4	I/O	Data 16
DATA18	A6	I/O	Data 18
DATA21	A8	I/O	Data 21
DATA24	A10	I/O	Data 24
DATA26	A11	I/O	Data 26
$\overline{\text{SRDY}}$	A13	O	Synchronous Ready
DATA11	B2	I/O	Data 11
DATA15	B3	I/O	Data 15
DATA17	B5	I/O	Data 17
DATA19	B7	I/O	Data 19
DATA20	B8	I/O	Data 20
DATA22	B9	I/O	Data 22
DATA23	B10	I/O	Data 23
DATA25	B11	I/O	Data 25
DATA30	B12	I/O	Data 30
DATA09	C1	I/O	Data 09
DATA12	C3	I/O	Data 12
+5V	C4	—	Power
GRD	C5	—	Ground
+5V	C6	—	Power
GRD	C7	—	Ground
+5V	C8	—	Power
GRD	C9	—	Ground
+5V	C10	—	Power
DATA27	C11	I/O	Data 27
DATA28	C13	I/O	Data 28
DATA08	D2	I/O	Data 08
DATA14	D4	I/O	Data 14
+5V	D11	—	Power
DATA29	D12	I/O	Data 29
DATA07	E1	I/O	Data 07
GRD	E3	—	Ground
—	E4	—	Device Socket Orientation Pin
GRD	E11	—	Ground
DATA31	E13	I/O	Data 31
DATA05	F2	I/O	Data 05
+5V	F3	—	Power
+5V	F11	—	Power
CLK23	F12	I	Input Clock 23

*All pin locations not listed are reserved for future use, and should not be left unconnected.

Table 3-2. Pin Descriptions, 125-Pin PGA Package* (Continued)

Name	Pin	Type	Description
DATA06	G1	I/O	Data 06
GRD	G3	—	Ground
GRD	G11	—	Ground
CLK34	G12	I	Input Clock 34
DATA02	H2	I/O	Data 02
+5V	H3	—	Power
+5V	H11	—	Power
$\overline{\text{DONE}}$	H12	O	Coprocessor Done
DATA04	J1	I/O	Data 04
GRD	J3	—	Ground
GRD	J11	—	Ground
$\overline{\text{FAULT}}$	J13	O	Fault
DATA01	K2	I/O	Data 01
+5V	K3	—	Power
ADDR03	K12	I	Address 03
DATA03	L1	I/O	Data 03
DATA00	L3	I/O	Data 00
GRD	L5	—	Ground
+5V	L6	—	Power
GRD	L7	—	Ground
+5V	L8	—	Power
GRD	L9	—	Ground
+5V	L10	—	Power
ADDR04	L11	I	Address 04
$\overline{\text{DSHAD}}$	M2	I	Data Bus Shadow
SAS2	M4	O	Access Status Code 2
SAS1	M6	O	Access Status Code 1
$\overline{\text{DS}}$	M7	I	Data Strobe
$\overline{\text{RESET}}$	M8	I	Reset Acknowledge
$\overline{\text{DTACK}}$	M10	O	Data Transfer Acknowledge
$\overline{\text{DRDY}}$	M11	I	Data Ready
R/$\overline{\text{W}}$	M12	I	Read/Write
$\overline{\text{HIGHZ}}$	N1	I	High Impedance
$\overline{\text{CYCLEI}}$	N3	I	Cycle Initiate
SAS3	N5	O	Access Status Code 3
SAS0	N6	O	Access Status Code 0
$\overline{\text{CS}}$	N8	I	Chip Select
ADDR02	N13	I	Address 02

*All pin locations not listed are reserved for future use, and should be left unconnected.

Chapter 4

Bus Transactions

CHAPTER 4. BUS TRANSACTIONS

CONTENTS

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

The MAU, configured in the peripheral mode, 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.) **Object Bit 4 3 2** — **Object Addressed** 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.

The ASR and DR are readable and writable in peripheral mode. The CR can be written but not read 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* 32100 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 result in a useful behavior.

4.1.2 Bus Transaction Sequence

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

CPU	Increasing Time ↓	MAU
		Quiescent state.
Write word into CR.		
		ASR<RA>=0 and only reads to the ASR are valid.
Write 0 to 6 operand words into DR.		
		Operation starts, and only writes to the CR are valid.
Read ASR, find ASR<RA>==0. Read ASR, find ASR<RA>==0. Read ASR, find ASR<RA>==0.		
		Operation completes, ASR changed to reflect result, DR changed to reflect result, results stored in F0—F3, and ASR<RA>=1. Quiescent state.
Read ASR, find ASR<RA>==1 Read OP3 result words out of DR.	↓	

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.

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 words can be written to the CR or any MAU address. Words written to the CR 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 address and write to 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, 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—F3) 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.

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.

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.

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. $\overline{\text{DTACK}}$ is then asserted on a falling edge of CLK23. When $\overline{\text{DS}}$ is negated, $\overline{\text{DTACK}}$ is also negated. Again, $\overline{\text{DS}}$ must be detected to be negated between accesses.

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

Under peripheral mode protocol, care must be taken to ensure that no more than one drives the $\overline{\text{DONE}}$ signal at any one time.

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

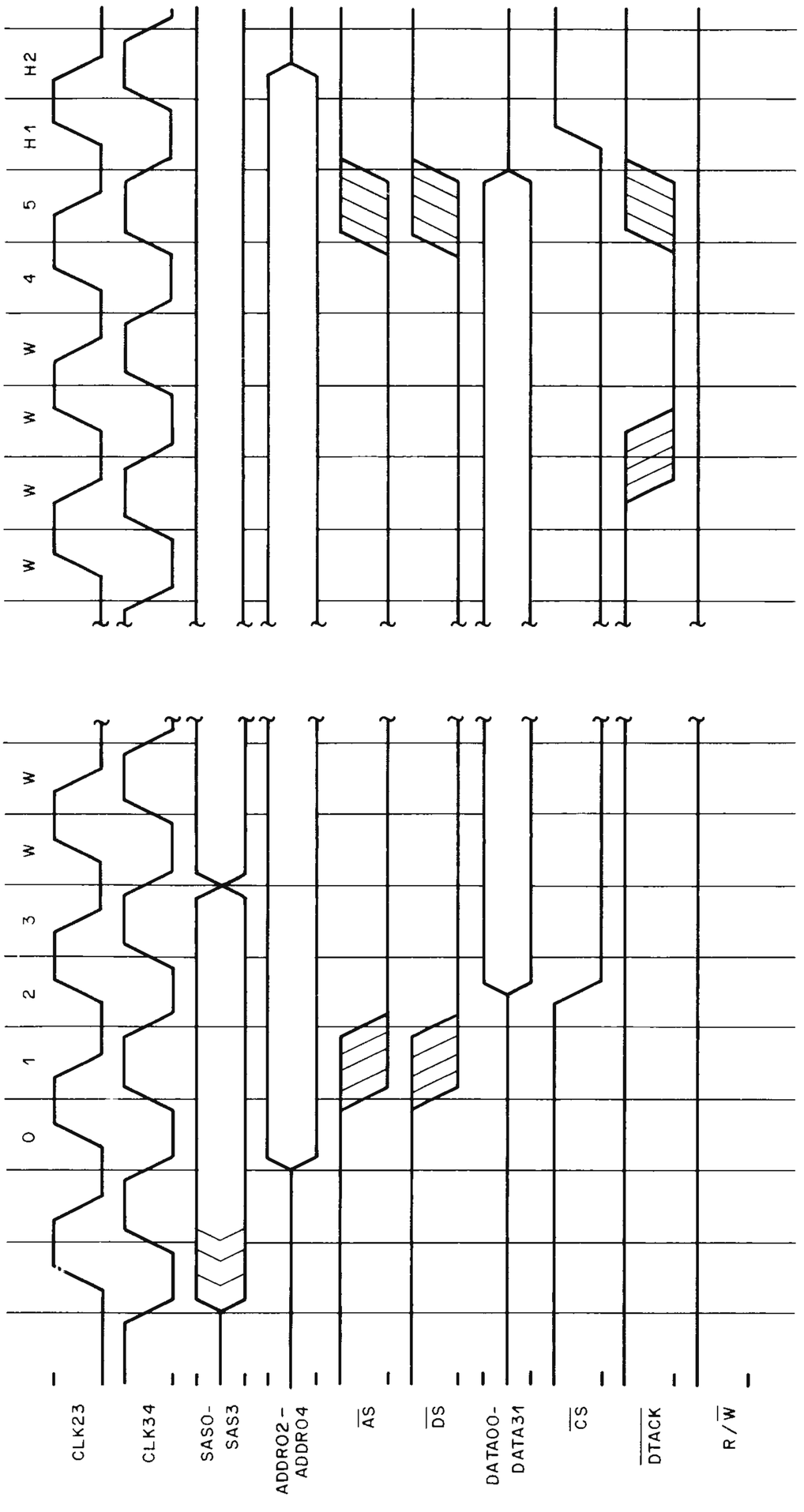


Figure 4-2. Peripheral Mode Read

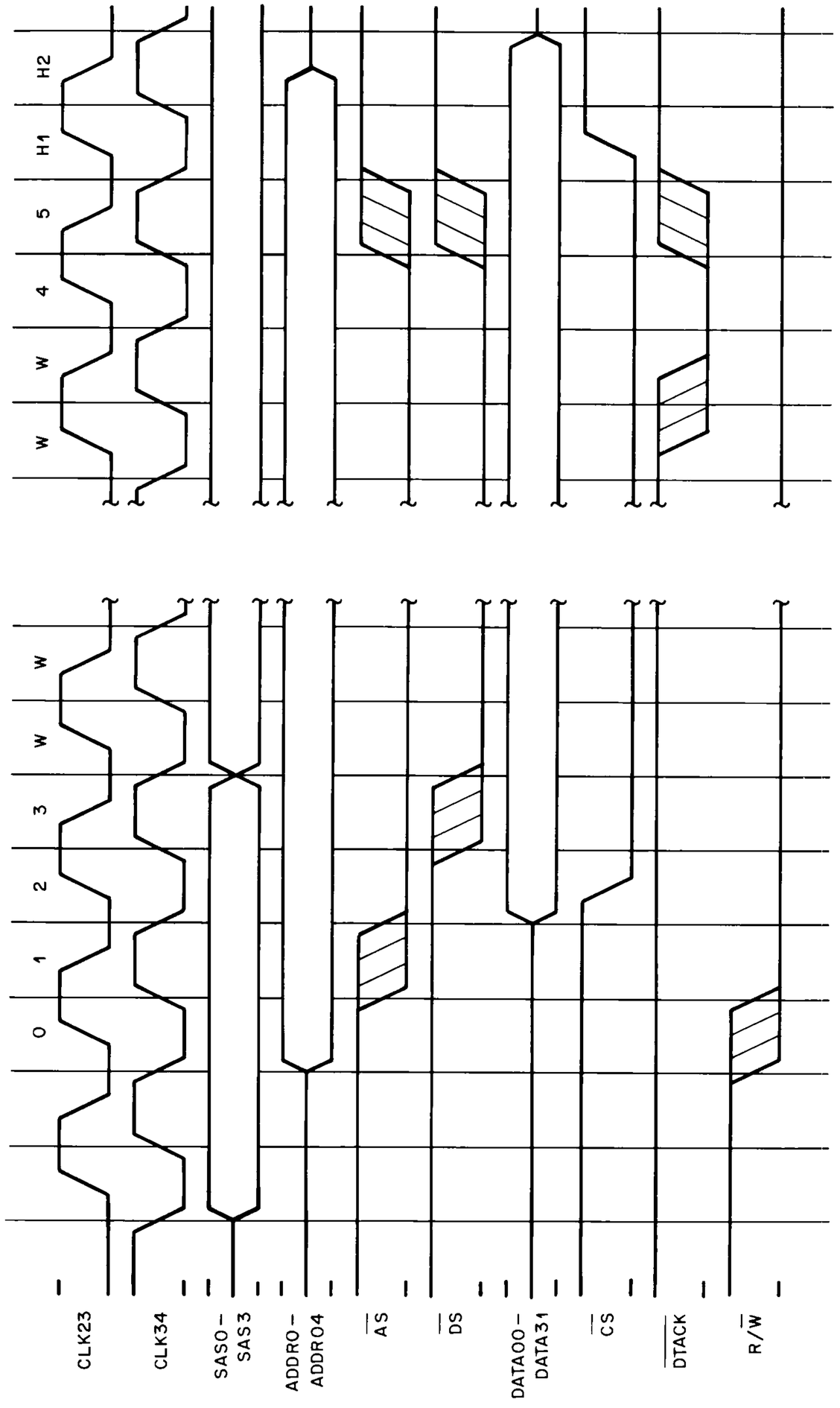


Figure 4-3. Peripheral Mode Write

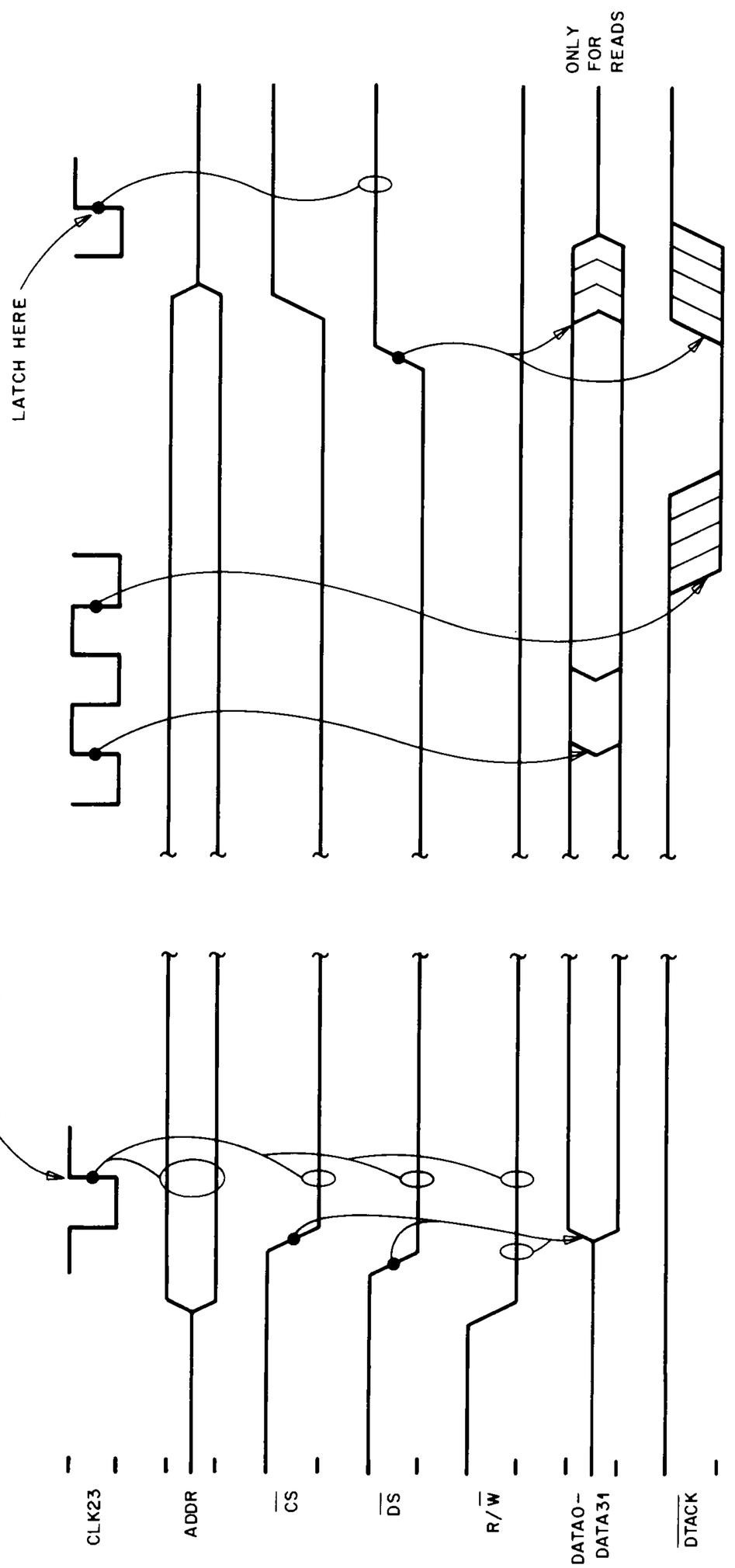


Figure 4-4. Peripheral Mode Read and Write

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 Configuration Restrictions

Under coprocessor mode protocol, care must be taken ensure that no more than one coprocessor drives the DONE signal at any one time.

The MAU's ID number is fixed, so it is not possible to have more than one MAU on the same bus in coprocessor mode.

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

In coprocessor mode, if nonaddressed devices assert $\overline{\text{FAULT}}$ during a coprocessor status fetch transaction, the CPU may incorrectly assume that an internal exception occurred in the MAU.

4.2.2 Bus Transaction Sequence

An example of a coprocessor mode transaction is shown on Figure 4-5.

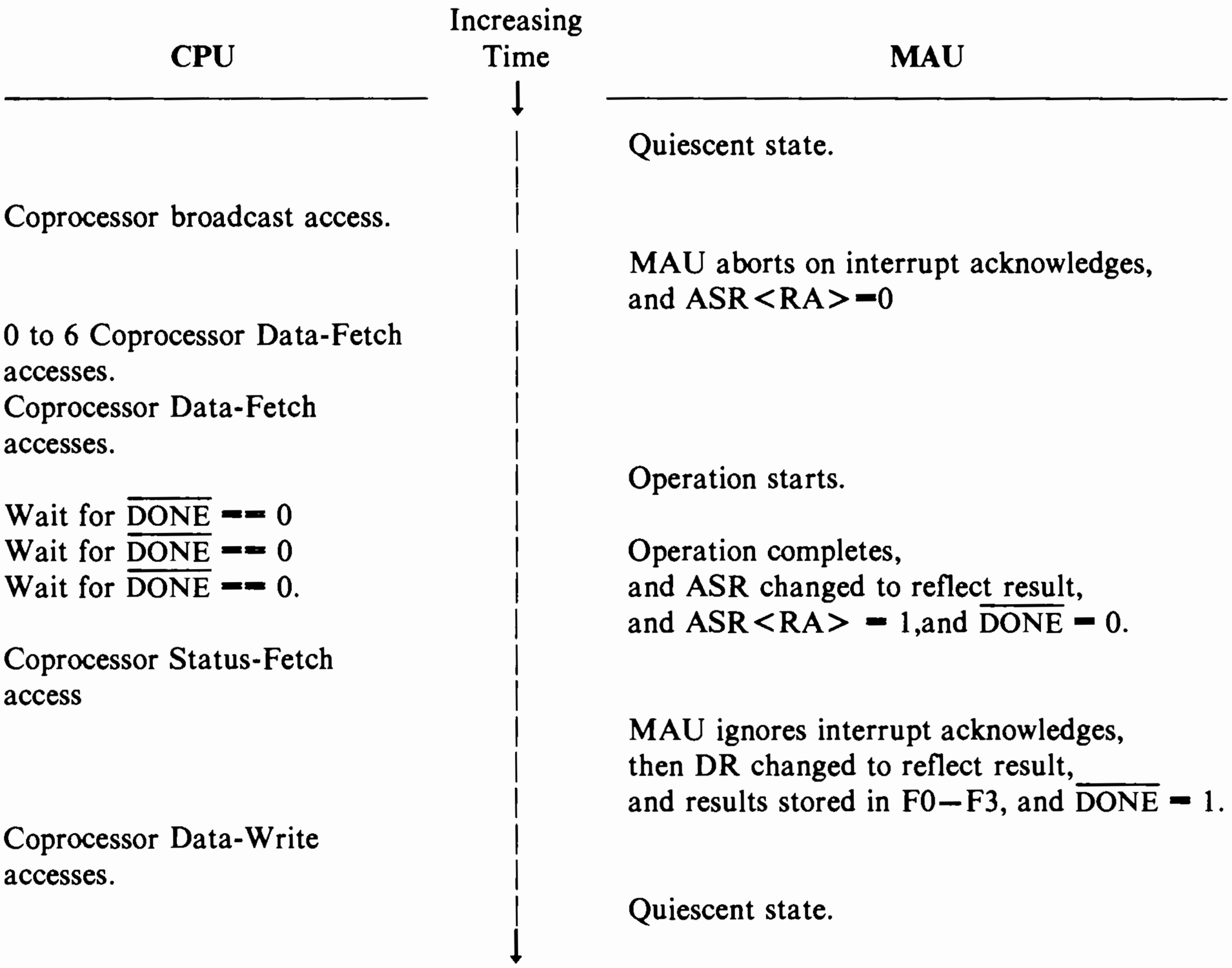


Figure 4-5. Example of Coprocessor Mode Transaction Sequence

Transaction Sequence

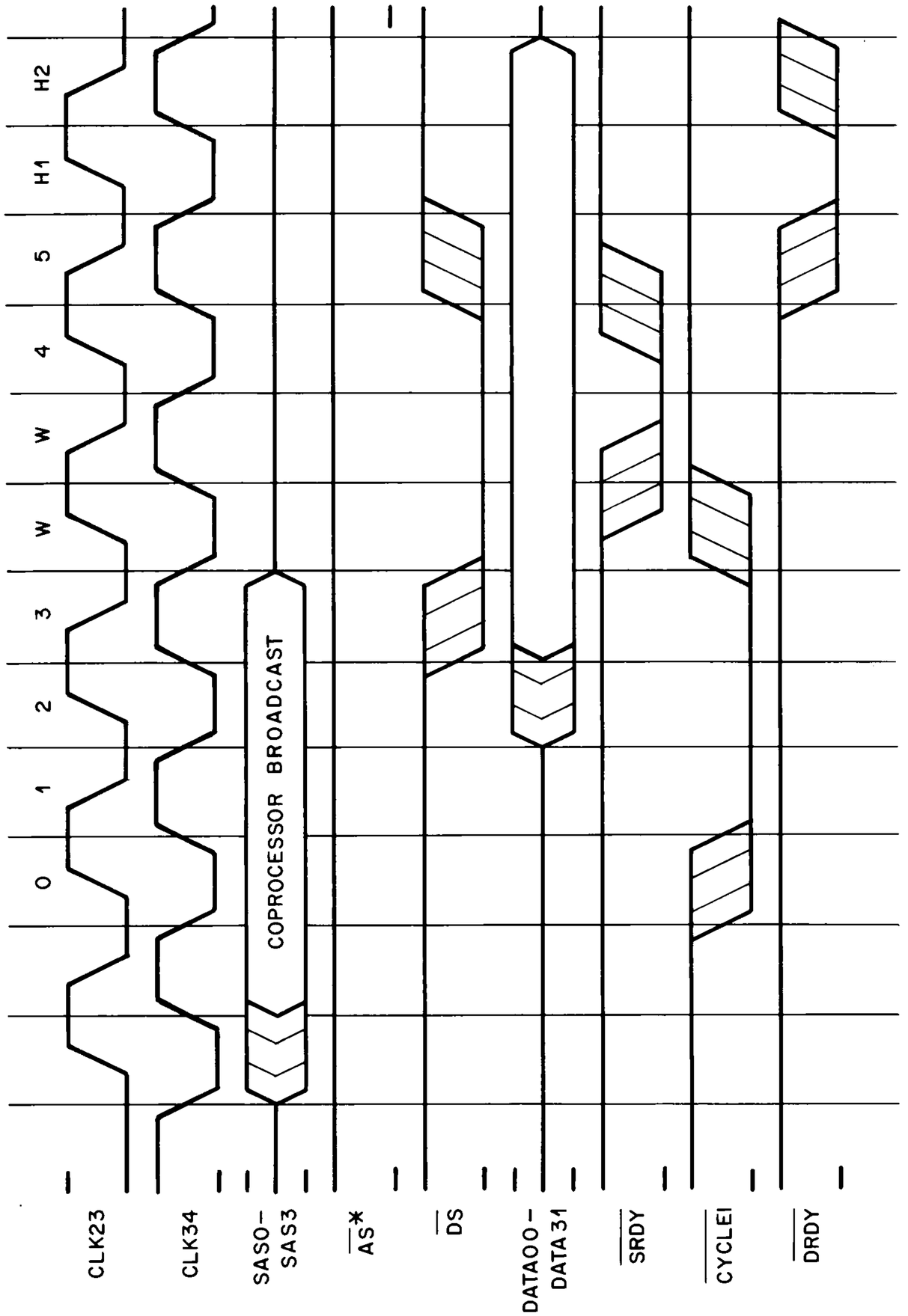
1. The CPU performs a coprocessor broadcast access with a command word on the data bus.
2. If the ID field of the command word is not equal to the MAU ID, the MAU enters its quiescent state (the rest of this sequence is not performed).
3. If the ID field of the command word is equal to the MAU ID, the MAU stores the command word in the command register and clears ASR<RA>. 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.
5. When the required number of coprocessor data fetch accesses have been performed, the MAU begins the specified operation. During the time, the CPU monitors the 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—F3), 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.
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).

4.2.3 Coprocessor Broadcast Transaction

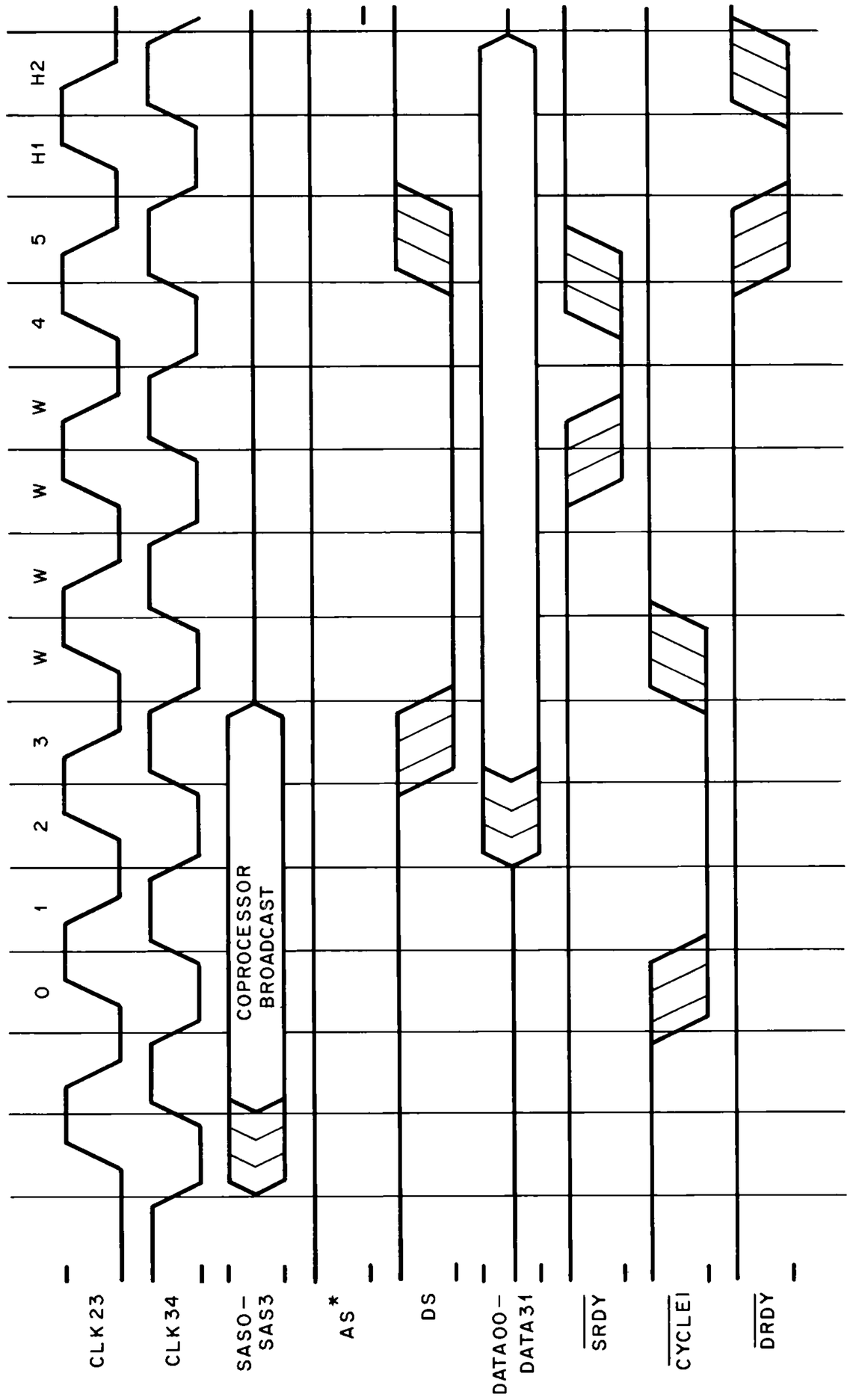
The MAU initially remains in an idle state and monitors the $\overline{\text{CYCLEI}}$ signal on every falling edge of CLK23. If $\overline{\text{CYCLEI}}$ is asserted and the access status code (SAS0—SAS3) signals indicate that the current transaction is a coprocessor broadcast transaction, the MAU 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, then 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, then 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, the 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 4-6 through 4-8.



*SHOWN ONLY FOR ADDITIONAL CLARITY.

Figure 4-6. Coprocessor Broadcast (1 Wait State)



* SHOWN ONLY FOR ADDITIONAL CLARITY

Figure 4-7. Coprocessor Broadcast (2 Wait States)

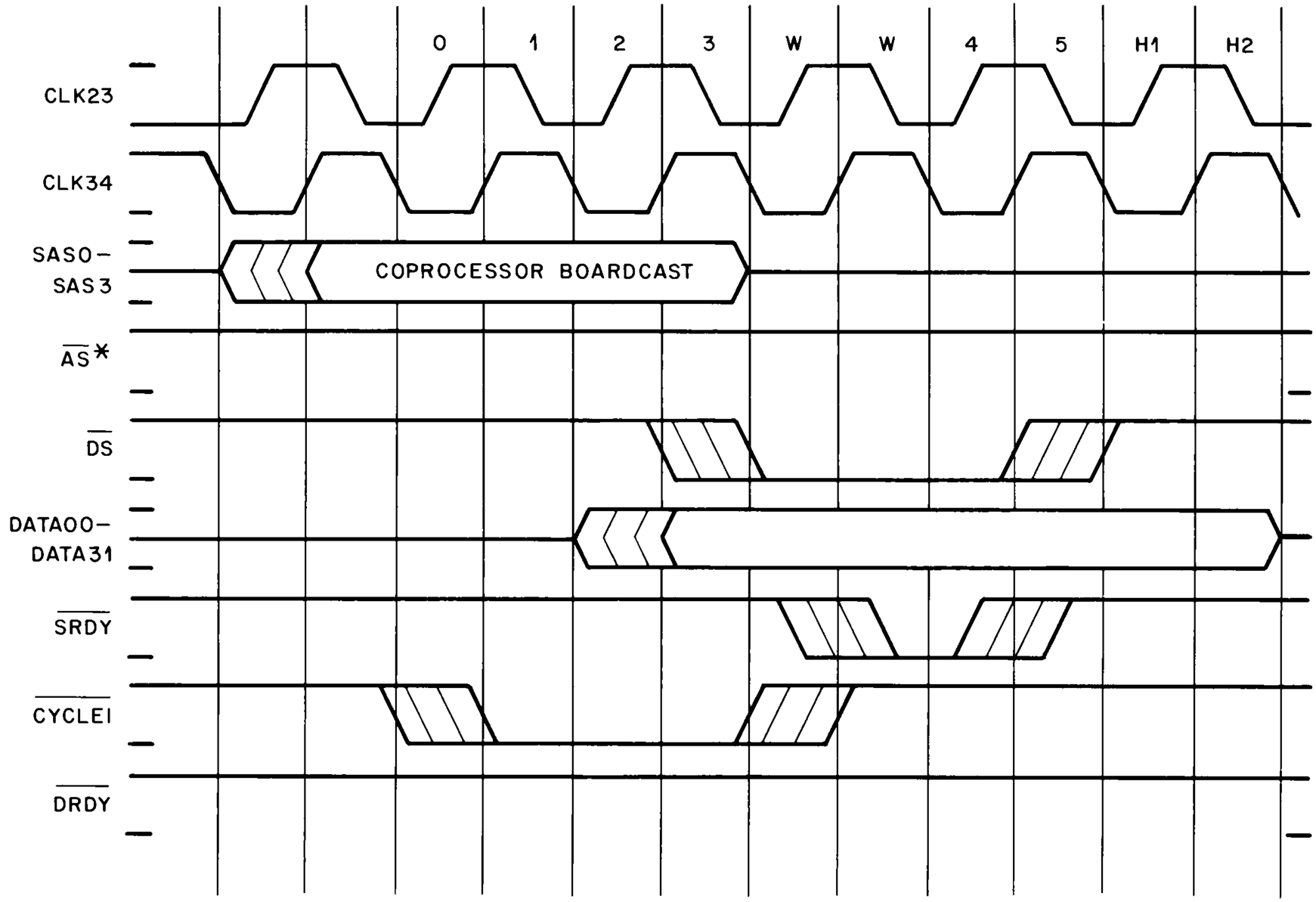


Figure 4-8. Coprocessor Broadcast with Bus Exceptions (1 Wait State)

4.2.4 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, then, 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, then 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, R/$\overline{\text{W}}$ = 1, and bus exceptions is shown on Figures 4-9 through 4-12.

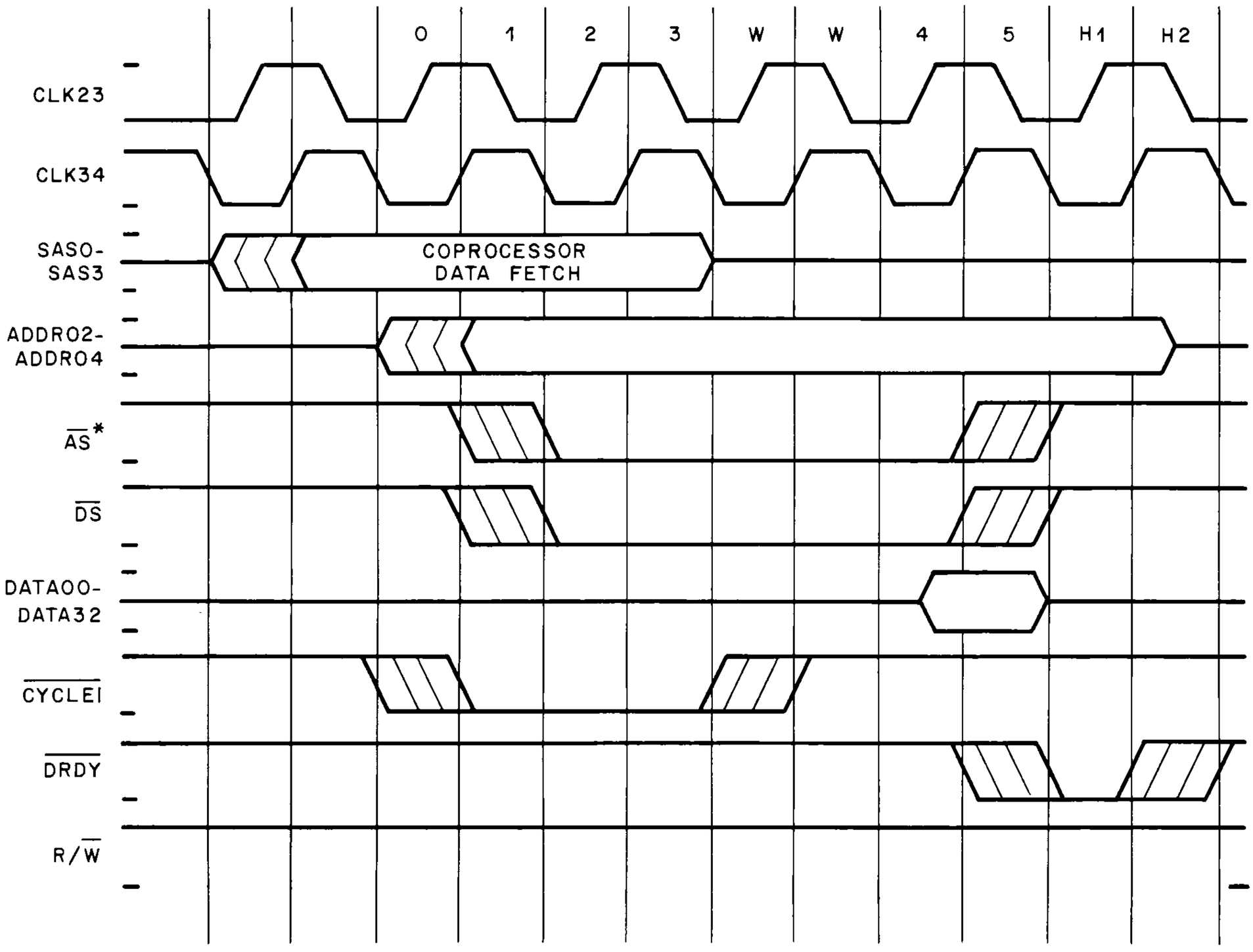
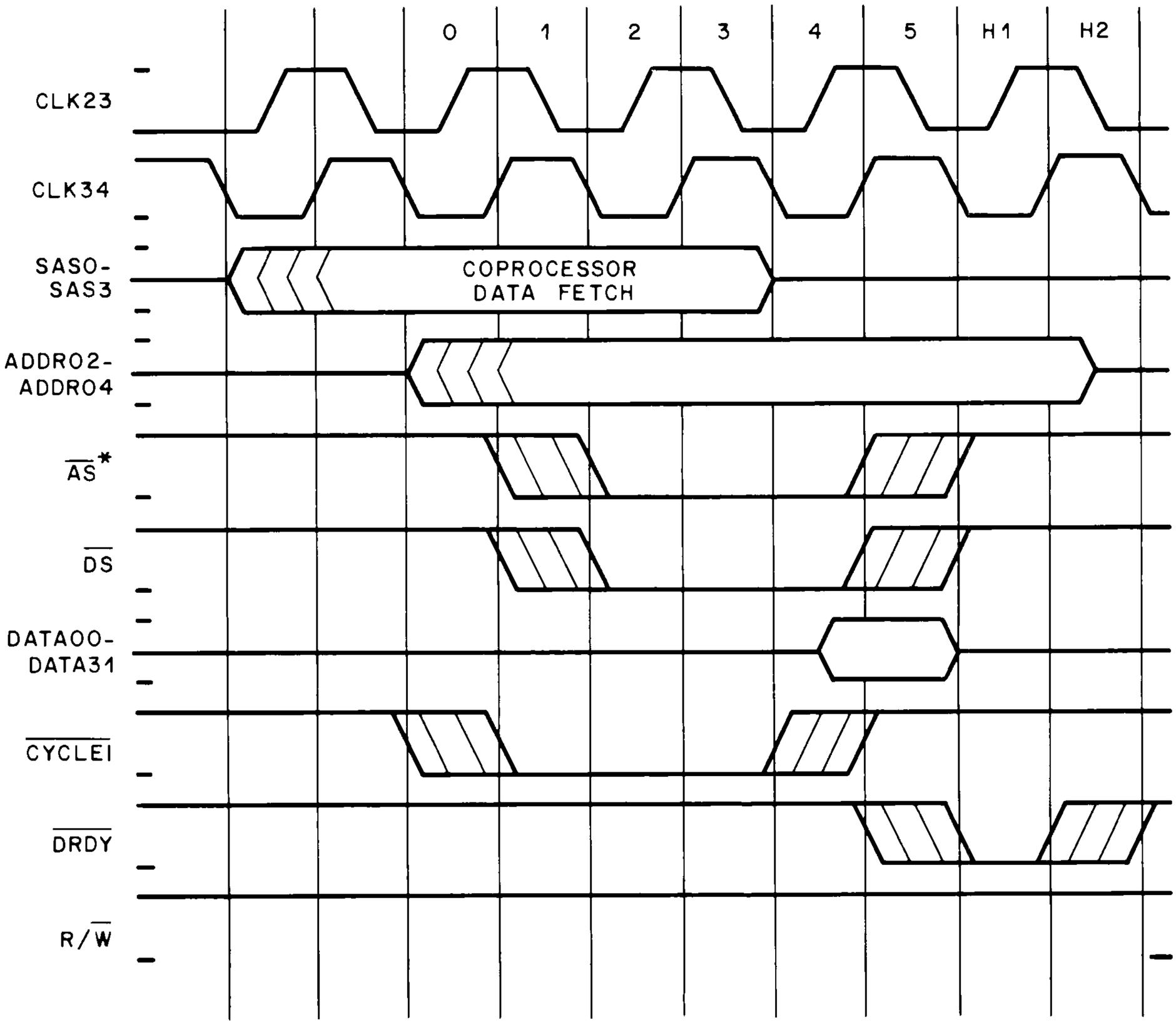


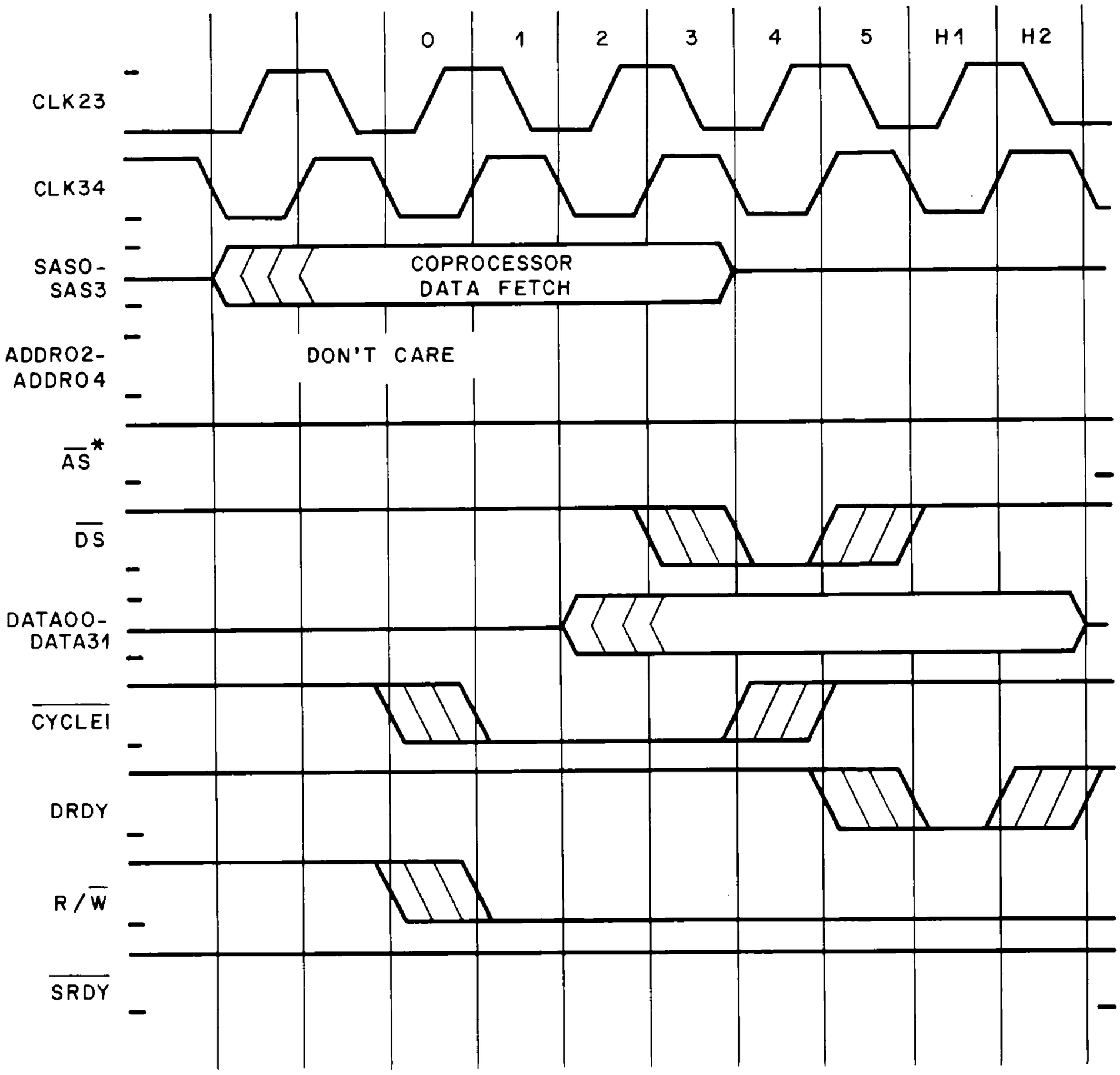
Figure 4-9. Coprocessor Broadcast Data Fetch (1 Wait State)

Coprocessor Data Fetch Transaction



*SHOWN ONLY FOR ADDITIONAL CLARITY
MEMORY RESPONDS WITH SRDY OR DTACK

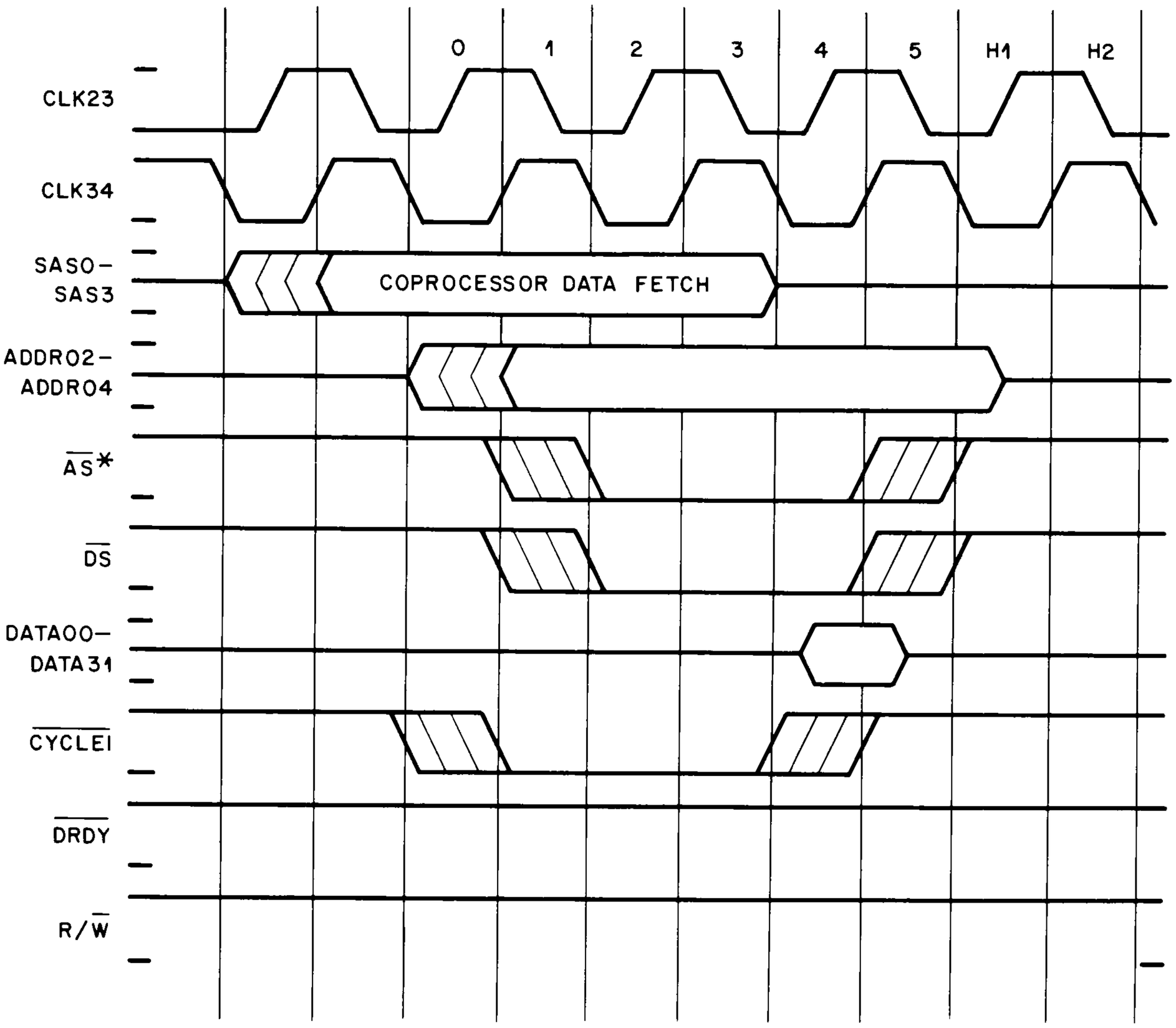
Figure 4-10. Coprocessor Broadcast Data Fetch (0 Wait States)



*SHOWN ONLY FOR ADDITIONAL CLARITY

Note: This transaction never occurs with the *WE* 32100 CPU

Figure 4-11. Coprocessor Broadcast Data Fetch (R/$\overline{W}$ = 0)



*SHOWN ONLY FOR ADDITIONAL CLARITY

MEMORY RESPONDS WITH FAULT, RETRY, OR RRREQ TERMINATING THE ACCESS

Figure 4-12. Coprocessor Broadcast Data Fetch with Bus Exceptions (0 Wait States)

4.2.5 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 $\overline{\text{CYCLE}}$, the MAU knows the start of this access. The MAU puts its condition codes on the data bus on 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 three-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 three-states its data outputs (DATA00–DATA31). If an exception has occurred during the execution of the instruction, the MAU asserts the $\overline{\text{FAULT}}$ signal along with the $\overline{\text{SRDY}}$ signal.

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

Figures 4-13 and 4-14 show the protocol for a coprocessor status read, with and without exceptions.

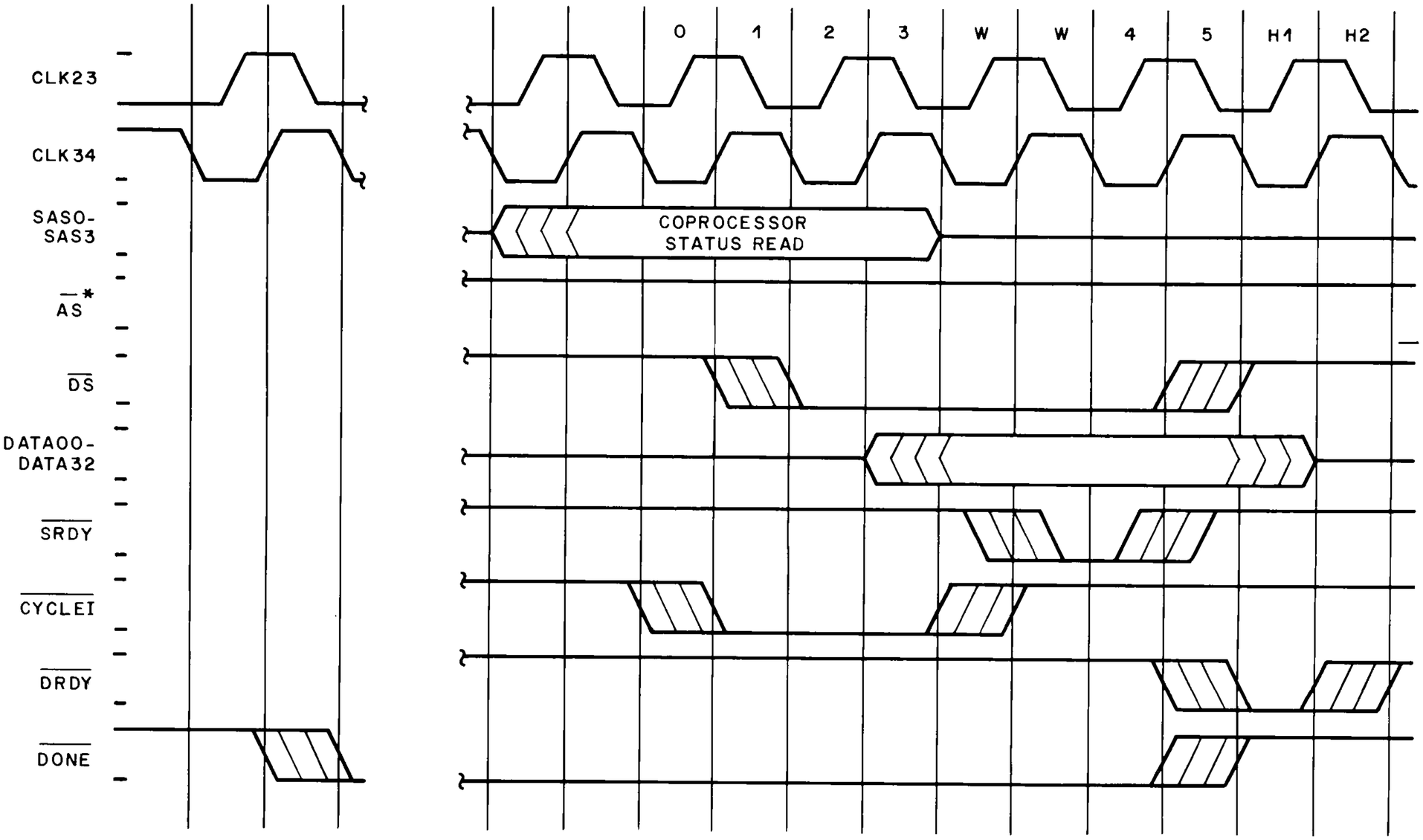
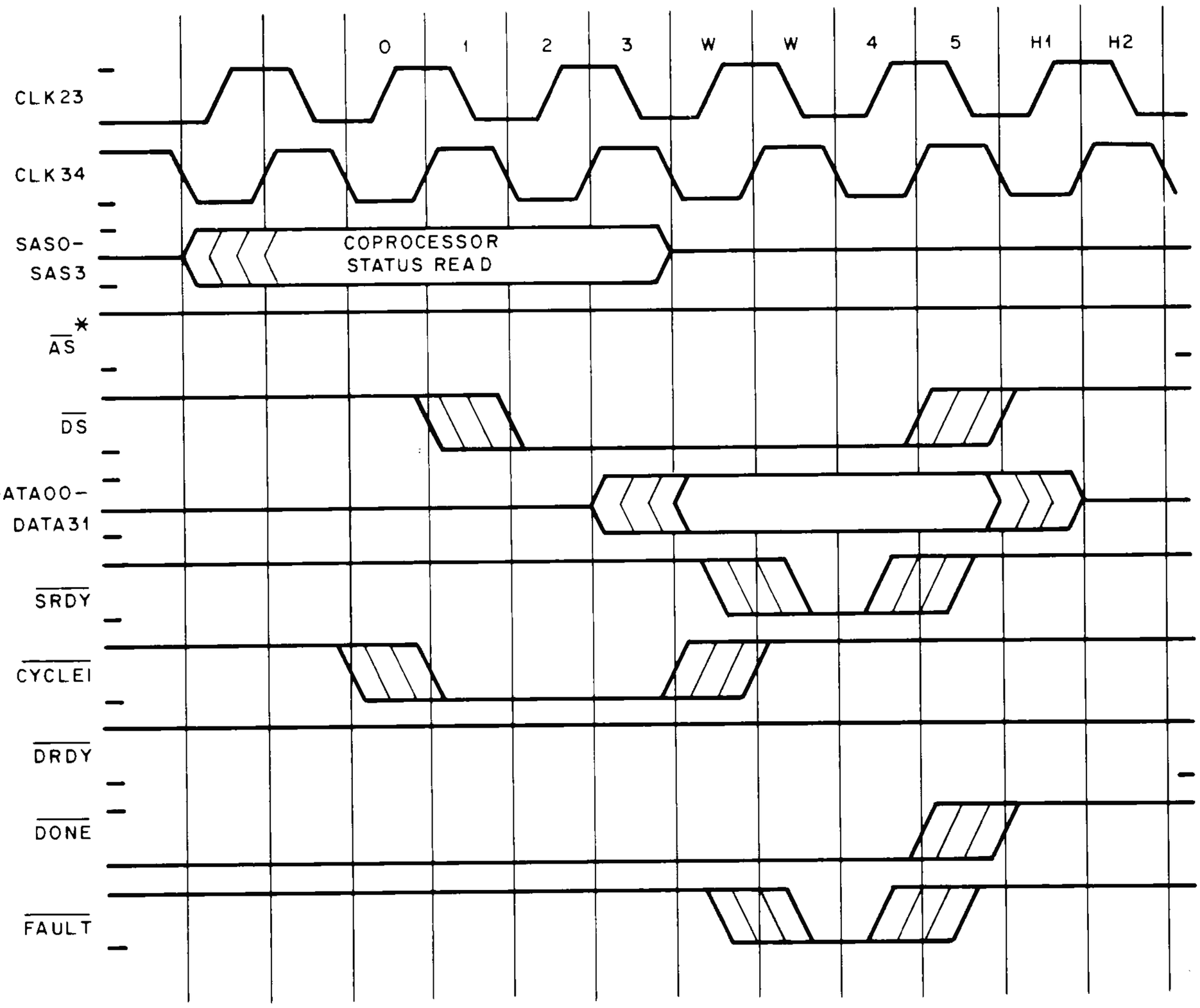


Figure 4-13. $\overline{\text{DONE}}$ Assertion and Coprocessor Status Read (No Exceptions)



*SHOWN ONLY FOR ADDITIONAL CLARITY

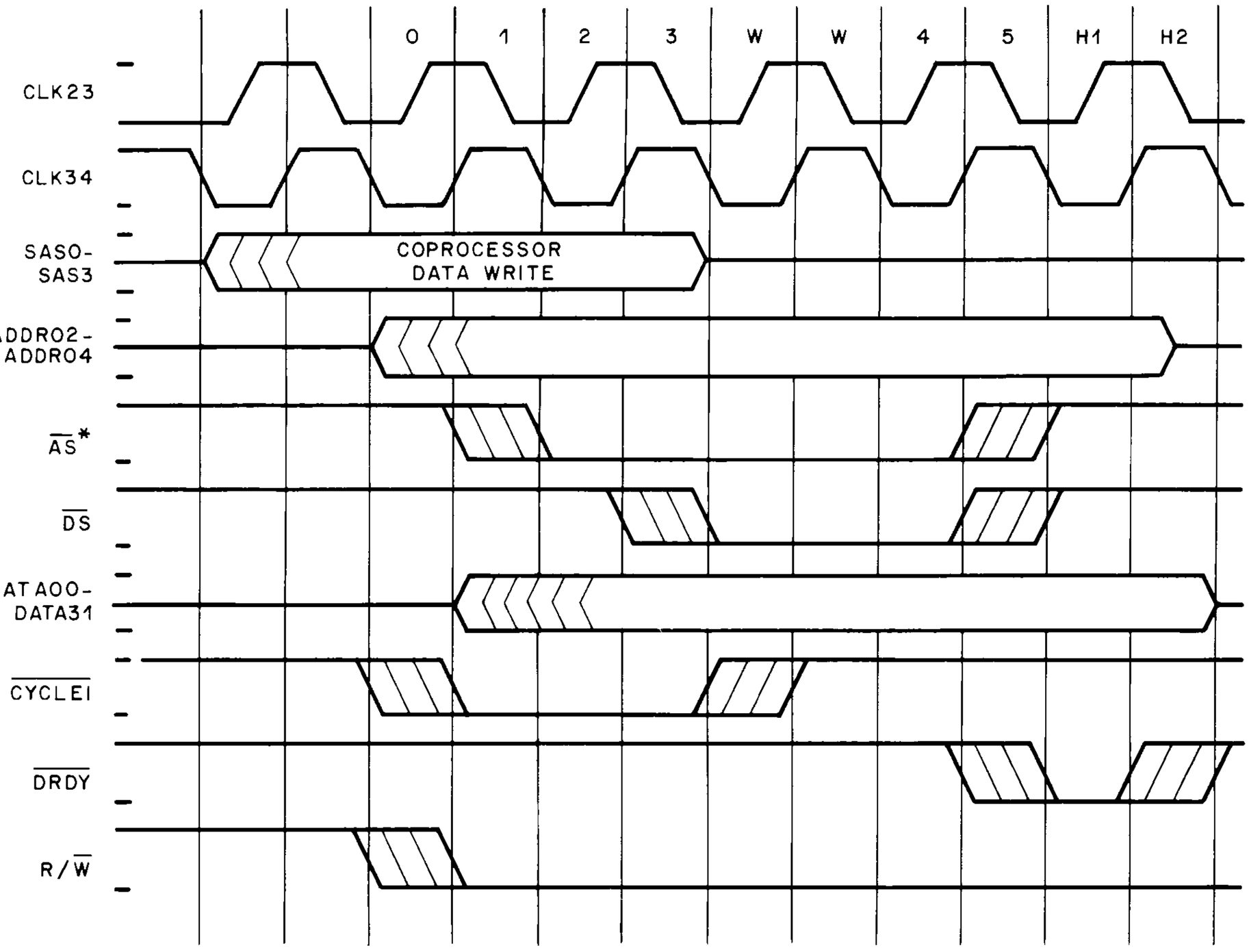
Figure 4-14. Coprocessor Status Read (With Exceptions)

4.2.6 Coprocessor Data Write Transaction

Figure 4-15 through Figure 4-19 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., three-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.

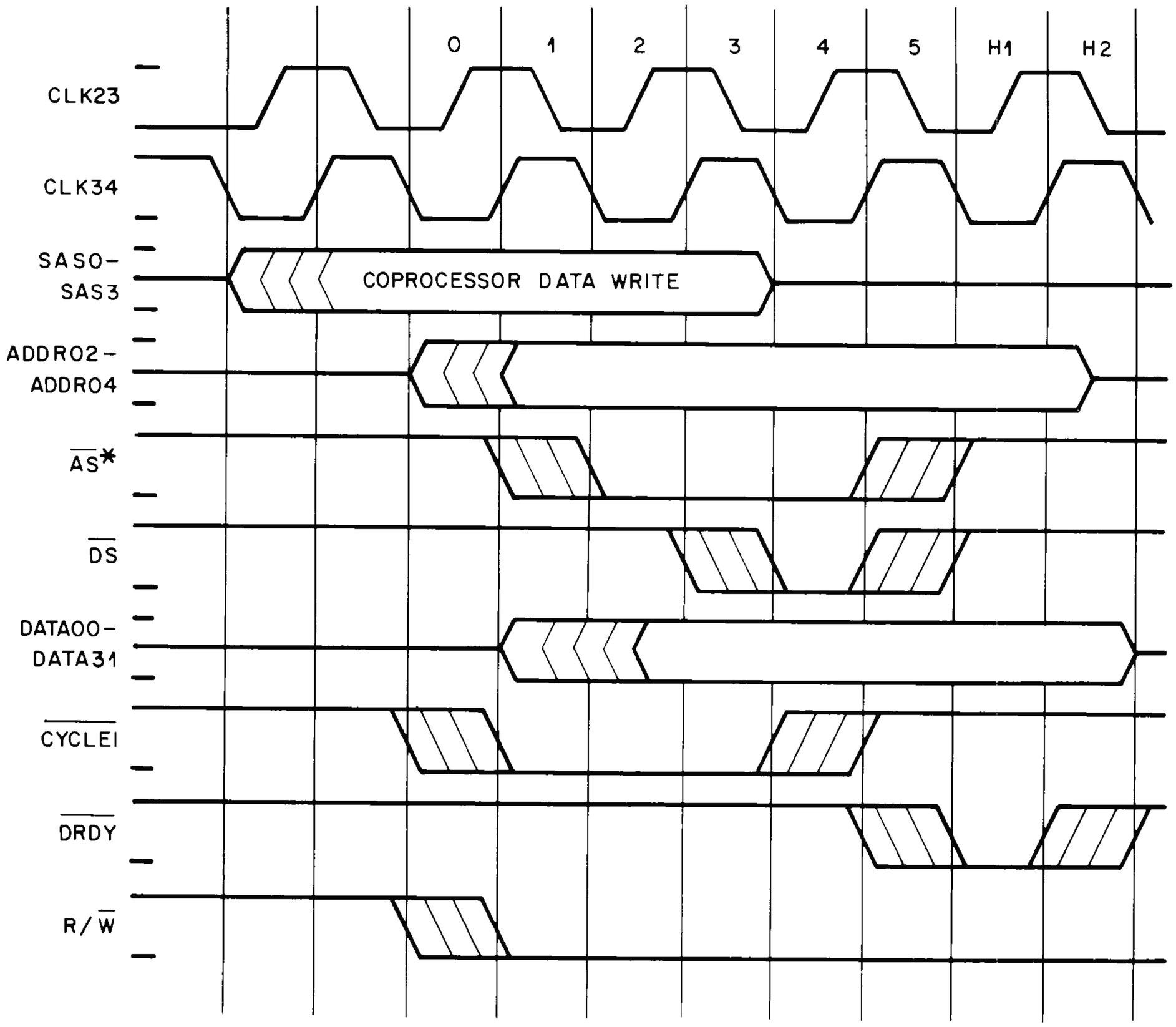
Assertion of data bus shadow, $\overline{DSHAD}$, causes the MAU to immediately three-state its data outputs until it detects that DSHAD is negated, upon which it immediatedly drives its data outputs.



*SHOWN ONLY FOR ADDITIONAL CLARITY
MEMORY RESPONDS WITH SRDY OR DTACK

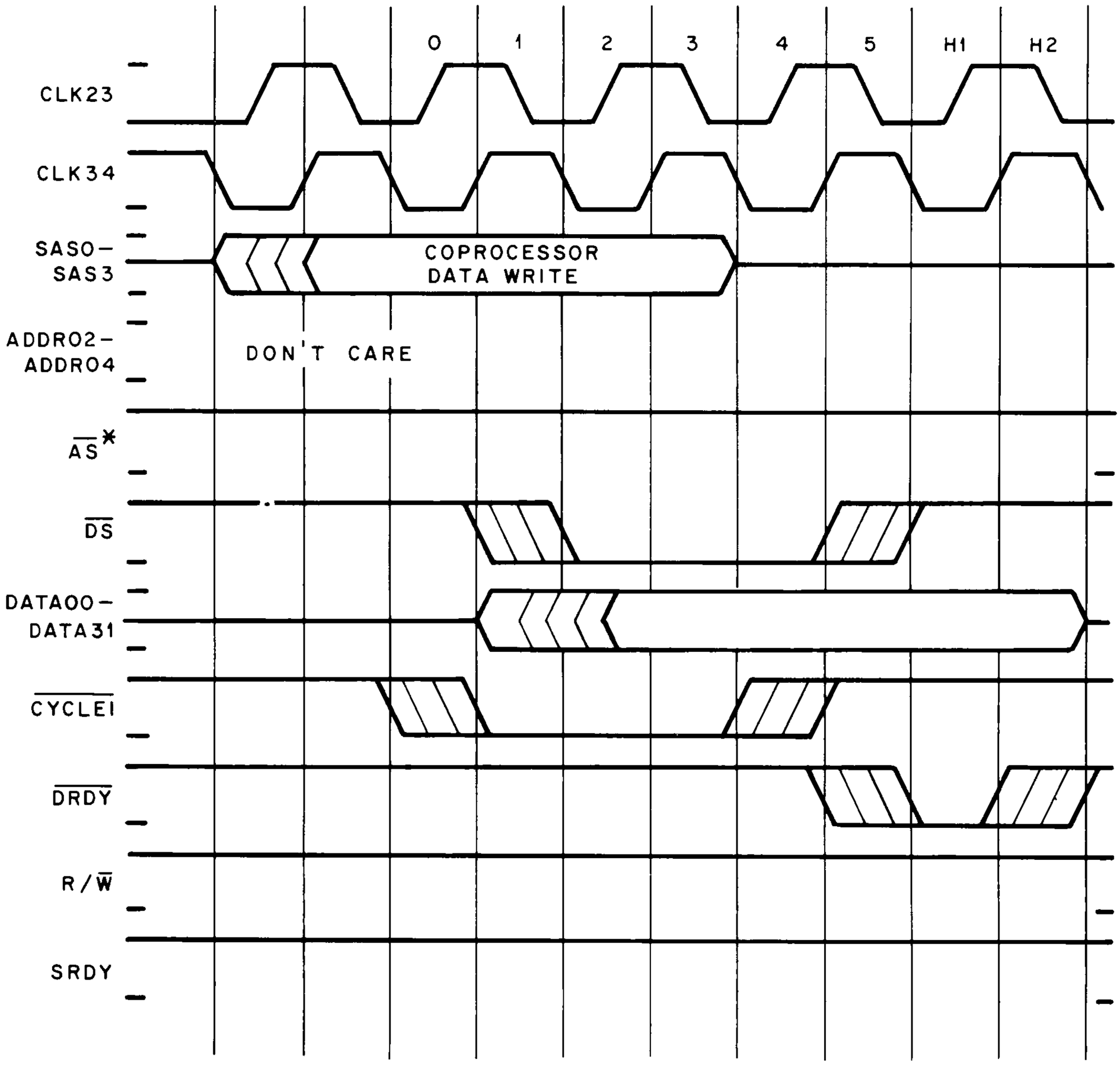
Figure 4-15. Coprocessor Data Write (1 Wait State)

Coprocessor Data Write Transaction



*SHOWN ONLY FOR ADDITIONAL CLARITY
MEMORY RESPONDS WITH SRDY OR DTACK

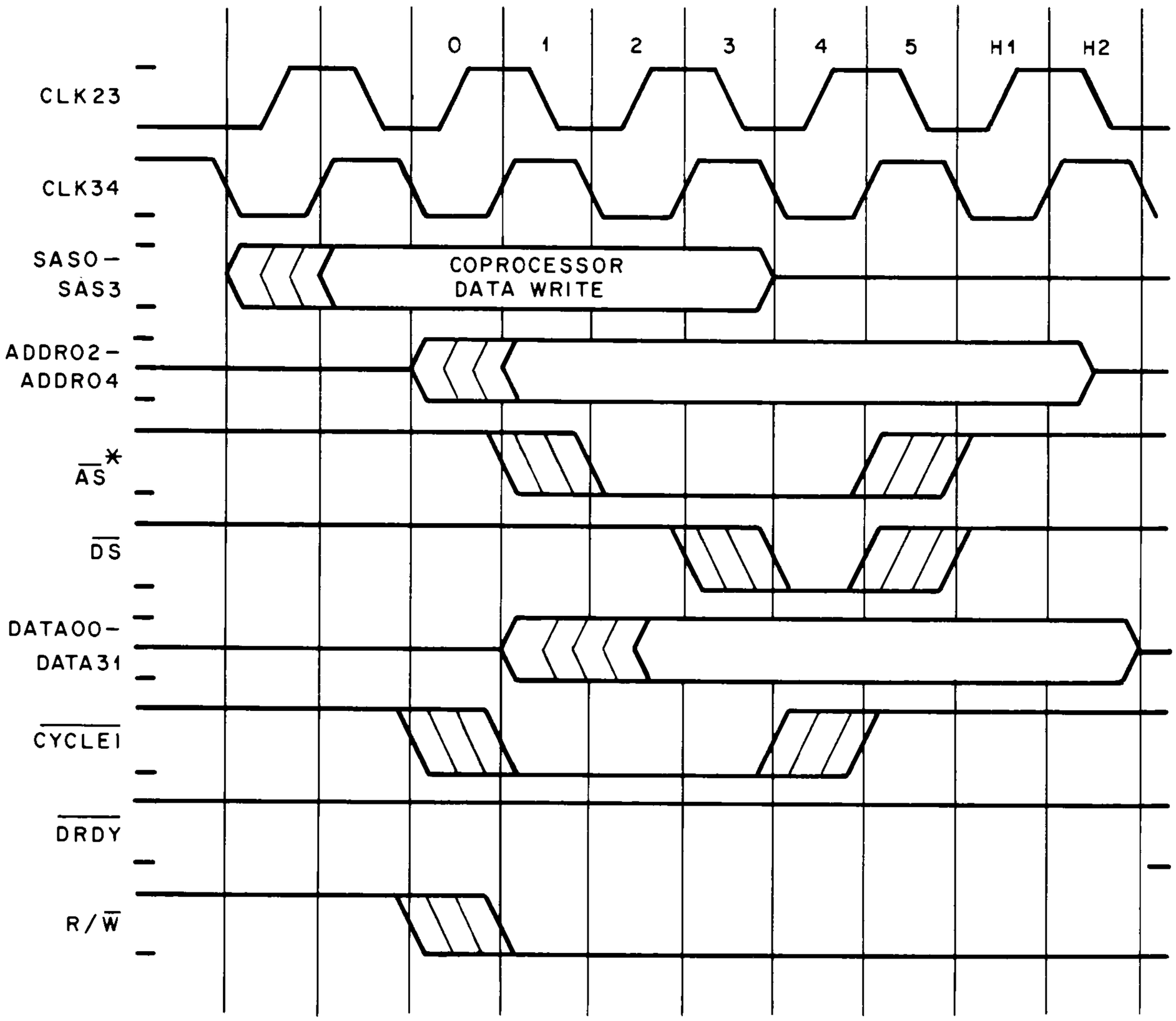
Figure 4-16. Coprocessor Data Write (0 Wait States)



*SHOWN ONLY FOR ADDITIONAL CLARITY.

Note: This transaction never occurs with the *WE* 32100 CPU

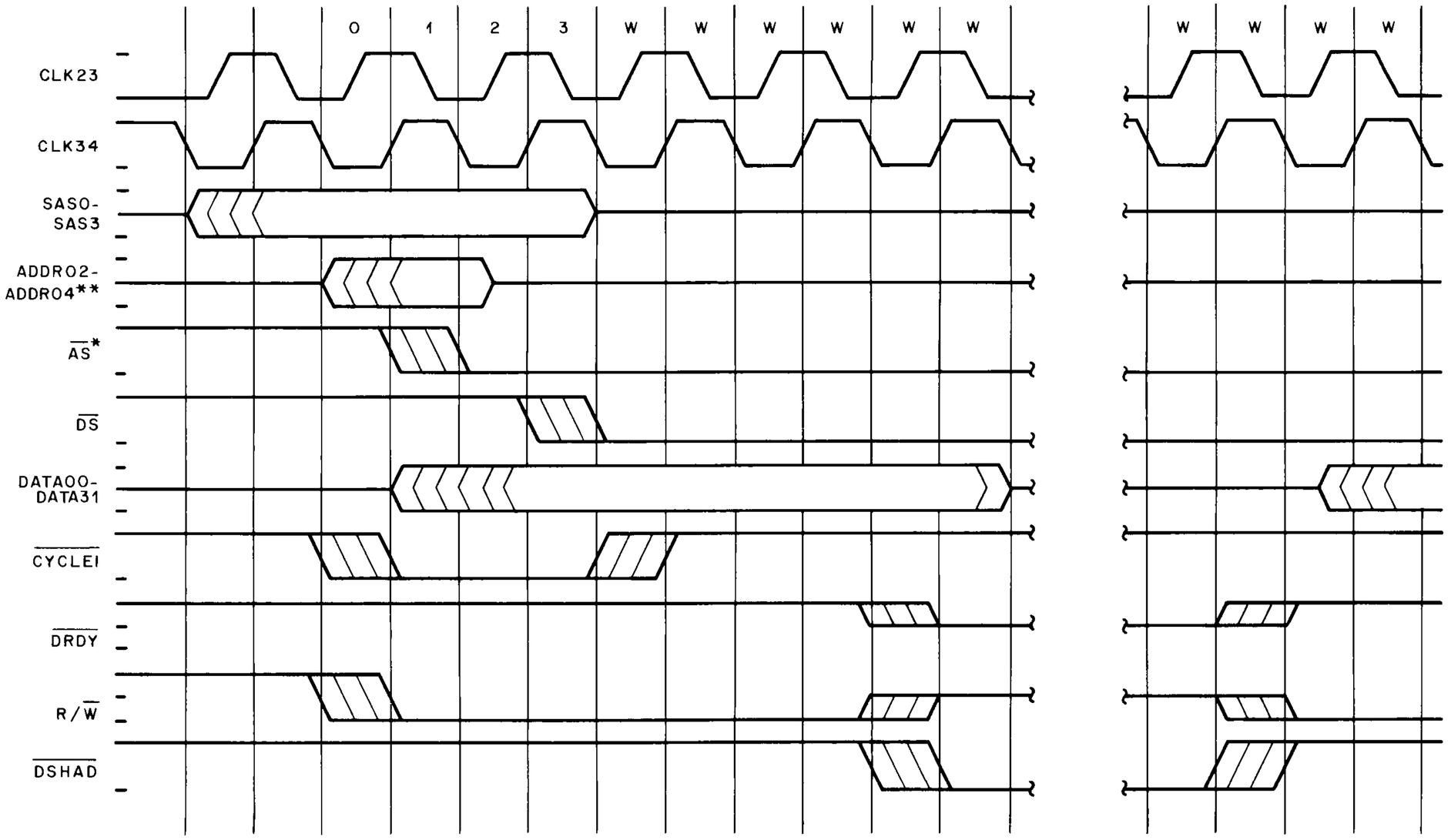
Figure 4-17. Coprocessor Data Write (R/$\overline{W}$ = 1)



*SHOWN ONLY FOR ADDITIONAL CLARITY

MEMORY RESPONDS WITH FAULT, RETRY, OR RRREQ TERMINATING THE ACCESS

Figure 4-18. Coprocessor Data Write With Bus Exceptions (0 Wait States)



*SHOWN ONLY FOR ADDITIONAL CLARITY
**TRANSACTION PROCEEDS WITH TRANSLATED ADDRESS

Figure 4-19. Coprocessor Data Write (DSHAD Assertion)

Chapter 5

Instruction Set

CHAPTER 5. INSTRUCTION SET

CONTENTS

5. INSTRUCTION SET

The *WE* 32106 Math Acceleration Unit (MAU) instruction set includes arithmetic, logical, data transfer, and format conversion. A detailed description of these instructions is given in **5.2.2 Instruction Set Descriptions**. Tables 5-1 through 5-5 list the instructions according to mnemonics, opcodes, and functional groups.

The MAU supports operations consisting of zero to three operands. The *WE* 32100 CPU allows from zero to two operands in memory for each coprocessor protocol instruction; therefore, it 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 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* 32106 Math Acceleration Unit instruction set may be separated into five functional groups: arithmetic, compare, data transfer, conversion, and miscellaneous instructions. This section contains a description of each group, along with a listing for each group.

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	CMPES	0x1B

5.1.3 Data Transfer Group

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	DTOF	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 Miscellaneous Instructions

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

Table 5-5. 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 Instruction Set Descriptions** presents descriptions of each member of the instruction set for the *WE* 32106 Math Acceleration Unit. The descriptions are listed in alphabetical order. (For quick reference to the instructions by function, mnemonic, or opcode see **5.2.3 Instruction Set Summary by Mnemonic** and **5.2.4 Instruction Set Summary by Opcode.)**

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 5-6 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 the applicable mnemonic, then applied to Table 5-6.

Condition flags. Identifies the effect of the instruction on the condition flags of the Auxiliary Status Register (ASR). Refer to **2.2.1 Auxiliary Status Register** for a description of the condition flags.

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

Table 5-6. Notation for Special Cases

Number	Action

1	ASR<IS> = 1;
	If (ASR<IM> == 1)
	ASR<ECP> = 1;
	signal invalid operation exception;*
	else if (ASR<NTNC> == 1)
	ASR<ECP> = 1;
	Signal exception to generate quiet (non trapping)
	NaN with Virtual PC as significand via software;*
	else
	Op3 = quiet (non trapping) NaN, sign = 0;
	
	(On a fault the ASR N & Z flags are unspecified. However, as implemented, the Z flag is cleared to 0 and the N flag in the ASR remains unchanged from the previous operation.)
	
	*When Op1 is a trapping NaN then register DR=Op1 else DR=Op2.
	In addition, for single or double precision, set J=1.

1	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.
	
	ASR<IS> = 1;
	if (ASR<IM> == 1)
	data register = source operand;
	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;**
	else
	Op3 = quiet (nontrapping) NaN, sign = 0;
	
	**DR = Op1.

Table 5-6. Notation for Special Cases (continued)

Number	Action

1	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.
	
	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;*
	else
	Op3 = Quiet (nontrapping) NaN, sign = 0;
	
	*DR = unspecified.

2	ASR<QS> = 1;
	if (ASR<QM> == 1)
	signal exception;
	else
	Op3 = infinity, normal sign calculation;

3	Op3 = Op1; (see note)
4	Op3 = 0, normal sign calculation;
5	Op3 = Infinity, normal sign calculation;
6	Op3 = Op2; (see note)

7	if (sign(Op1) == sign(Op2)) (see note)
	Op3 = Op1;
	else if (ASR<RC> == RM)
	Op3 = (−0);
	else
	Op3 = (+0);

8	Perform Operation;
9	Op3 = (decimal equivalent of floating Op1);
10	Op3 = (floating equivalent of decimal Op1);

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.

5.2.2 Instruction Set Descriptions

The instruction sets are described in detail on the following pages.

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 cause where Op1 is a special case value. Each action is specified as a number. The meaning of each number is specified in Table 5-6.

Op1	Action
Infinity	8
NaN	8*
TNaN	1
± zero	8
Unnormal zero	4
Normal	8

* Return NaN with sign bit cleared.

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

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 5-6.

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

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

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 Flags of the ASR

N ← 1 if Op1 < Op2, else 0
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 stick 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 Flags of the ASR

N ← 1 if Op1 < Op2, else 0
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 stick 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 Flags of the ASR

N ← 1 if Op1 == Op2, else 0
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 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 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.

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 Flags of the ASR

N ← 1 if Op1 == Op2, else 0
Z ← 1 if Op1 < Op2, else 0
IO ← 0
PR ← 0
UO ← 1 if unordered condition exists, else 0

Exceptions Invalid operation

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.

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 5-6.

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

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
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 then 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, 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 5-6.

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

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
Inexact

EXTRACT RESULT ON FAULT

Mnemonic EROF

Opcode 0x14

Operation Op3 = Data Register

Description The value of the data register is placed in operand 3. This is a triple word transfer.

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.

If this instruction is executed under normal conditions, the value returned is indeterminate.

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 is 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 to 1, and the destination is be updated with an undefined value. By controlling the overflow enable/disable bit in the *WE* 32100 CPU's PSW, the user can choose to mask or unmask the decimal overflow exception.

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 5-6.

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

FTOD

Condition Flags of the ASR

N ← 1 if value stored in Op3 is negative, else 0
Z ← 1 if 0 stored in Op3, else 0
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 to be an MAU register. If the invalid operation exception is enabled then Op1 (after converting to double-extended precision of necessary) is stored in DR.

If the intermediate result overflows the destination, ASR<IO> is set to 1, and the destination will be updated with an undefined value. By controlling the overflow enable/disable bit in the *WE* 32100 CPU's PSW, the user can choose to mask or unmask the integer overflow exception.

Sizes other than the single-word for Op3 yield indeterminate values.

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 5-6.

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

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
Integer overflow

CONVERT INTEGER TO FLOAT

Mnemonic ITOF

Opcode 0x10

Operation Op3 = float(Op1)

Description Operand 1 is converted from 2's complement integer format to floating-point format and copied into operand 3. Operand 1 is left unaffected.

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.

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

An inexact condition can occur when Op3 is a single-word operand.

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.

Special Cases: None

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

Inexact
Invalid operation

LOAD DATA REGISTER

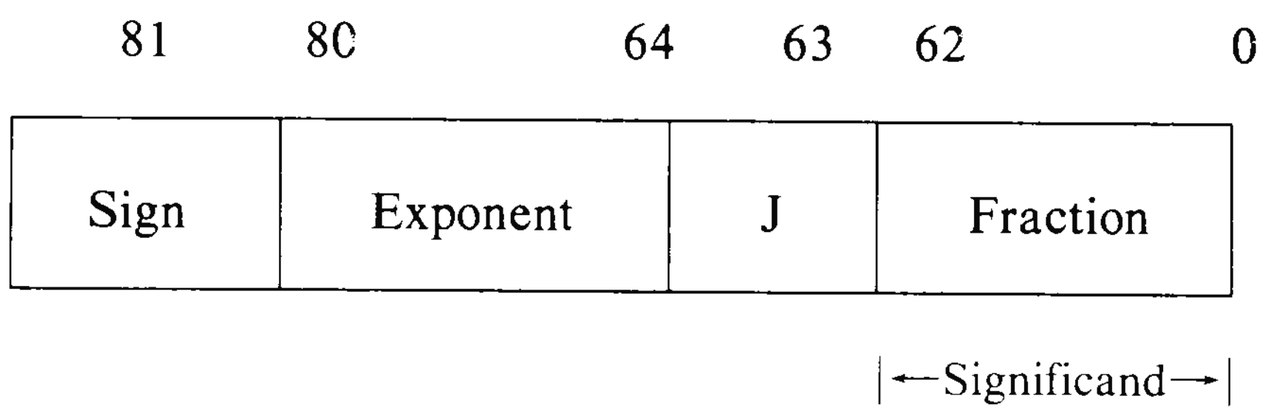
Mnemonic LDR

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 (DR). 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.

Special Cases: None

Condition Flags of the ASK Unchanged

Exceptions None

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* 32100 CPU does not support this because it requires all memory-based operands to be of the same size. With the *WE* 32100 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* 32100 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 5-6.

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

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 5-6.

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

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

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 5-6.

Op3	Action
Infinity	8
NaN	8*
TNaN	1
± zero	8
Normal	8

* Complement sign bit of NaN

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

NO OPERATION

Mnemonic NOP

Opcode 0x13

Operation Stop current operation, if any, and return to quiescent state.

Description 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.

Any unassigned opcodes cause behavior identical to that of the NOP opcode, but these opcodes are reserved for future MAU use.

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.

Special Cases: None

Condition Flags of the ASR Unchanged

Exceptions None

MOVE FROM ASR

Mnemonic RDASR

Opcode 0x08

Operation Op3 = ASR

Description The ASR is copied into operand 3. The ASR is 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.

Special Cases: None

Condition Flags of the ASR Unchanged

Exceptions None

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 5-6.

Op1	Op2					
	Infinity	**NaN**	**TNaN**	**Zero**	**Unnormal Zero**	**Normal**
Infinity	1	3	1	1	1	1
NaN	6	3	1	6	6	6
TNaN	1	1	1	1	1	1
Zero	6	3	1	1	1	6
Unnormal Zero	4	3	1	1	1	4
Normal	6	3	1	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.

REM

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 ← 1 if result is partial remainder, else 0
UO ← 0

Exceptions

Invalid operation
Overflow
Underflow
Inexact

ROUND TO INTEGRAL VALUE

Mnemonic RTOI

Opcode 0x0E

Operation Op3 = integer(Op1)

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

The inexact condition exists if the final result is not exactly equal to the input argument and no other exceptions have occurred.

The RTOI operation is performed in double-extended precision. If the destination is single or double then this operation may perform rounding twice. That is, first, when generating round to integral value in double-extended and, second, rounding may occur at the time of converting double extended values to single or double precision.

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 5-6.

Op1	Action
Infinity	3
NaN	3
TNaN	1
Zero	3
Un-Normalized Zero	4
Normalized Number	8

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
Inexact

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 5-6.

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 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

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 5-6.

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

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

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

SUB

Exceptions

Invalid operation
Overflow
Underflow
Inexact

MOVE TO ASR

Mnemonic WRASR

Opcode 0x09

Operation ASR = Op1

Description Operand 1 is copied to the ASR. Operand 1 is left unaffected. 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.

Special Cases: None

Condition Flags of the ASR

N	←	bit in N-bit position in Op1
Z	←	bit in Z-bit position in Op1
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 5-7.

Table 5-7. Instruction Set Summary by Mnemonic

Mnemonic	Opcode	Instructioin	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
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
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

Table 5-7. Instruction Set Summary by Mnemonic (Continued)

Mnemonic	Opcode	Instruction	Operation	Condition Codes Affected
EROF	0x14	Extract result on fault	Place value of data register 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 two'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 two'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
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

Table 5-7. Instruction Set Summary by Mnemonic (Continued)

Mnemonic	Opcode	Instruction	Operation	Condition Codes Affected
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
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

5.2.4 Instruction Set Summary by Opcode

Table 5-8 lists the instructions by Opcode.

Table 5-8. 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	CMPES	Compare with exceptions and flags swapped

Glossary

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 two raised to the power of its exponent.

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 its exponent incremented until the exponent equals 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 two 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 (CR). 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 can not be represented in the destination format, and no other exceptions have occurred.

Invalid operation – An exception that occurs if a result can not 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 to "1", an exception may occur. If it is set to "0", 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.

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

GLOSSARY

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

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 the 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 zeros.

Index

INDEX

Notes

Notes

Notes

For additional information, contact your AT&T Account Manager, or call:

☐ AT&T Technologies
Dept. 50AL203140
555 Union Boulevard
Allentown, PA 18103
1-800-372-2447

In Europe contact:
☐ AT&T Microelectronics
Freischützstrasse 92,
8000 München 81,
West Germany
Tel. 0 89/95 97 0 Telex 5 216 884

November 1986

MN86-31MCP

AT&T
The right choice.

COMCODE 105195390