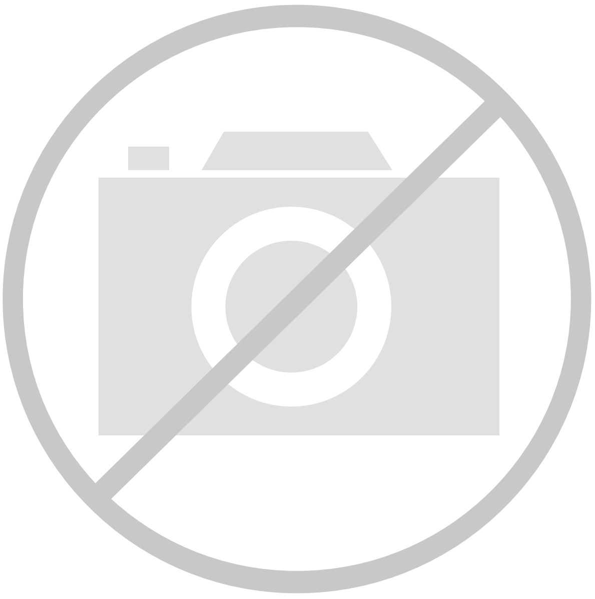
Motorola - MC68030CRC25C Datasheet



Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Active
Core Processor	68030
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	25MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5V
Operating Temperature	-40°C ~ 85°C (TA)
Security Features	-
Package / Case	128-BPGA
Supplier Device Package	128-PGA (34.55x34.55)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68030crc25c

LIST OF ILLUSTRATIONS (Continued)

2.4.18 Immediate Data

In this addressing mode, the operand is in one or two extension words:

Byte Operation
: Operand is in the low-order byte of the extension word

Word Operation
: Operand is in the extension word

Long-Word Operation
: The high-order 16 bits of the operand are in the first extension word; the low-order 16 bits are in the second extension word.

Coprocessor instructions can support immediate data of any size. The instruction word is followed by as many extension words as are required.

Generation:	Operand given
Assembler Syntax:	#xxx
Mode Field:	111
Register Field:	100
Number of Extension Words:	1 or 2, except for coprocessor instructions

MC68000/MC68008/MC68010 Address Extension Word

15	14 — 12	11	10	9	8	7 — 0
D/A	REGISTER	W/L	0	0	0	DISPLACEMENT INTEGER

D/A:	0 =	Data Register Select
	1 =	Address Register Select
W/L:	0 =	Word-Sized Operation
	1 =	Long-Word-Sized Operation

MC68020/MC68030 Address Extension Word

15	14 — 12	11	10 — 9	8	7 — 0
D/A	REGISTER	W/L	SCALE	0	DISPLACEMENT INTEGER

D/A:	0 =	Data Register Select
	1 =	Address Register Select
W/L:	0 =	Word-Sized Operation
	1 =	Long-Word-Sized Operation
SCALE:	00 =	Scale Factor 1 (Compatible with MC68000)
	01 =	Scale Factor 2 (Extension to MC68000)
	10 =	Scale Factor 4 (Extension to MC68000)
	11 =	Scale Factor 8 (Extension to MC68000)

Figure 2-15. M68000 Family Address Extension Words

mode. The term supervisor stack pointer (SSP) refers to the master or interrupt stack pointers, depending on the state of the M bit. When M=1, the term SSP (or A7) refers to the MSP address register. When M=0, the term SSP (or A7) refers to the ISP address register. The active system stack pointer is implicitly referenced by all instructions that use the system stack. Each system stack fills from high to low memory.

A subroutine call saves the program counter on the active system stack, and the return restores it from the active system stack. During the processing of traps and interrupts, both the program counter and the status register are saved on the supervisor stack (either master or interrupt). Thus, the execution of supervisor code is independent of user code and the condition of the user stack; conversely, user programs use the user stack pointer independently of supervisor stack requirements.

To keep data on the system stack aligned for maximum efficiency, the active stack pointer is automatically decremented or incremented by two for all byte-sized operands moved to or from the stack. In long-word-organized

3.3 INTEGER CONDITION CODES

The CCR portion of the SR contains five bits which indicate the results of many integer instructions. Program and system control instructions use certain combinations of these bits to control program and system flow.

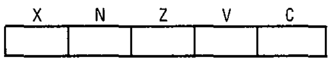
The first four bits represent a condition resulting from a processor operation. The X bit is an operand for multiprecision computations; when it is used, it is set to the value of the C bit. The carry bit and the multiprecision extend bit are separate in the M68000 Family to simplify programming techniques that use them (refer to Table 3-8 as an example).

The condition codes were developed to meet two criteria:

- Consistency — across instructions, uses, and instances
- Meaningful Results — no change unless it provides useful information

Consistency across instructions means that all instructions that are special cases of more general instructions affect the condition codes in the same way. Consistency across instances means that all instances of an instruction affect the condition codes in the same way. Consistency across uses means that conditional instructions test the condition codes similarly and provide the same results, regardless of whether the condition codes are set by a compare, test, or move instruction.

In the instruction set definitions, the CCR is shown as follows:



where:

X (extend)
Set to the value of the C bit for arithmetic operations. Otherwise not affected or set to a specified result.

N (negative)
Set if the most significant bit of the result is set. Cleared otherwise.

Z (zero)
Set if the result equals zero. Cleared otherwise.

V (overflow)
Set if arithmetic overflow occurs. This implies that the result cannot be represented in the operand size. Cleared otherwise.

C (carry)
Set if a carry out of the most significant bit of the operand occurs for an addition. Also set if a borrow occurs in a subtraction. Cleared otherwise.

SECTION 6
ON-CHIP CACHE MEMORIES

The MC68030 microprocessor includes a 256-byte on-chip instruction cache and a 256-byte on-chip data cache that are accessed by logical (virtual) addresses. These caches improve performance by reducing external bus activity and increasing instruction throughput.

Reduced external bus activity increases overall performance by increasing the availability of the bus for use by external devices (in systems with more than one bus master, such as a processor and a DMA device) without degrading the performance of the MC68030. An increase in instruction throughput results when instruction words and data required by a program are available in the on-chip caches and the time required to access them on the external bus is eliminated. Additionally, instruction throughput increases when instruction words and data can be accessed simultaneously.

As shown in Figure 6-1, the instruction cache and the data cache are connected to separate on-chip address and data buses. The address buses are combined to provide the logical address to the memory management unit (MMU). The MC68030 initiates an access to the appropriate cache for the requested instruction or data operand at the same time that it initiates an access for the translation of the logical address in the address translation cache of the MMU. When a hit occurs in the instruction or data cache and the MMU validates the access on a write, the information is transferred from the cache (on a read) or to the cache and the bus controller (on a write). When a hit does not occur, the MMU translation of the address is used for an external bus cycle to obtain the instruction or operand. Regardless of whether or not the required operand is located in one of the on-chip caches, the address translation cache of the MMU performs logical-to-physical address translation in parallel with the cache lookup in case an external cycle is required.

On the initial access of a burst operation, a "retry" (indicated by the assertion of $\overline{\mathrm{BERR}}$ and $\overline{\mathrm{HALT}}$) causes the processor to retry the bus cycle and assert $\overline{\mathrm{CBREQ}}$ again. However, signaling a retry with simultaneous $\overline{\mathrm{BERR}}$ and $\overline{\mathrm{HALT}}$ during the second, third, or fourth cycle of a burst operation does not cause a retry operation, even if the requested operand is misaligned. Assertion of $\overline{\mathrm{BERR}}$ and $\overline{\mathrm{HALT}}$ during burst fill cycles of a burst operation causes independent bus error and halt operations. The processor remains halted until $\overline{\mathrm{HALT}}$ is negated, and then handles the bus error as described in the previous paragraphs.

6.2 CACHE RESET

When a hardware reset of the processor occurs, all valid bits of both caches are cleared. The cache enable bits, burst enable bits, and the freeze bits in the cache control register (CACR) for both caches (refer to Figure 6-14) are also cleared, effectively disabling both caches. The WA bit in the CACR is also cleared.

6.3 CACHE CONTROL

Only the MC68030 cache control circuitry can directly access the cache arrays, but the supervisor program can set bits in the CACR to exercise control over cache operations. The supervisor also has access to the cache address register (CAAR), which contains the address for a cache entry to be cleared.

6.3.1 Cache Control Register

The CACR, shown in Figure 6-14, is a 32-bit register that can be written or read by the MOVEC instruction or indirectly modified by a reset. Five of the bits (4–0) control the instruction cache; six other bits (13–8) control the data cache. Each cache is controlled independently of the other, although a similar operation can be performed for both caches by a single MOVEC instruction. For example, loading a long word in which bits 3 and 11 are set into the CACR clears both caches. Bits 31–14 and 7–5 are reserved for Motorola definition. They are currently read as zeros and are ignored when written. For future compatibility, writes should not set these bits.

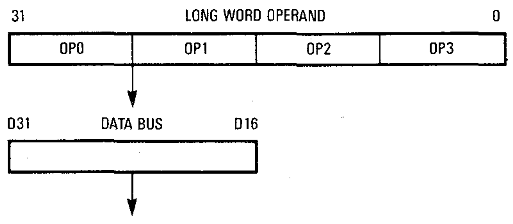
long-word or word operands that are misaligned. For maximum performance, data items should be aligned on their natural boundaries. All instruction words and extension words must reside on word boundaries. Attempting to prefetch an instruction word at an odd address causes an address error exception.

Figure 7-9 shows the transfer of a long-word operand to an odd address in word-organized memory, which requires three bus cycles. For the first cycle, the size signals specify a long-word transfer, and the address offset (A2:A0) is 001. Since the port width is 16 bits, only the first byte of the long word is transferred. The slave device latches the byte and acknowledges the data transfer, indicating that the port is 16 bits wide. When the processor starts the second cycle, the size signals specify that three bytes remain to be transferred with an address offset (A2:A0) of 010. The next two bytes are transferred during this cycle. The processor then initiates the third cycle, with the size signals indicating one byte remaining to be transferred. The address offset (A2:A0) is now 100; the port latches the final byte; and the operation is complete. Figure 7-10 shows the associated bus transfer signal timing.

Figure 7-11 shows the equivalent operation for a cachable data read cycle.

Figures 7-12 and 7-13 show a word transfer to an odd address in word-organized memory. This example is similar to the one shown in Figures 7-9 and 7-10 except that the operand is word sized and the transfer requires only two bus cycles.

Figure 7-14 shows the equivalent operation for a cachable data read cycle.



WORD MEMORY		MC68030					MEMORY CONTROL	
MSB	LSB	SIZ1	SIZ0	A2	A1	A0	$\overline{DSACK1}$	$\overline{DSACK0}$
XXX	OP0	0	0	0	0	1	L	H
OP1	OP2	1	1	0	1	0	L	H
OP3	XXX	0	1	1	0	0	L	H

Figure 7-9. Misaligned Long-Word Transfer to Word Port Example

is hitting in both caches and if the bus controller is free. Note that, if the bus controller is executing other cycles, these aborted cycles due to cache hits may not be seen externally. Also, $\overline{\text{OCS}}$ is asserted for the first *external* cycle of an operand transfer. Therefore, in the case of a misaligned data transfer where the first portion of the operand results in a cache hit (but the bus controller did not begin an external cycle and then abort it) and the second portion in a cache miss, $\overline{\text{OCS}}$ is asserted for the second portion of the operand.

7.2.8 Asynchronous Operation

The MC68030 bus may be used in an asynchonous manner. In that case, the external devices connected to the bus can operate at clock frequencies different from the clock for the MC68030. Asynchronous operation requires using only the handshake line ($\overline{\text{AS}}$, $\overline{\text{DS}}$, $\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$) to control data transfers. Using this method, $\overline{\text{AS}}$ signals the start of a bus cycle, and $\overline{\text{DS}}$ is used as a condition for valid data on a write cycle. Decoding the size outputs and lower address lines (A0 and A1) provides strobes that select the active portion of the data bus. The slave device (memory or peripheral) then responds by placing the requested data on the correct portion of the data bus for a read cycle or latching the data on a write cycle, and asserting the $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ combination that corresponds to the port size to terminate the cycle. If no slave responds or the access is invalid, external control logic asserts the $\overline{\text{BERR}}$ or $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ signal(s) to abort or retry the bus cycle, respectively.

The $\overline{\text{DSACKx}}$ signals can be asserted before the data from a slave device is valid on a read cycle. The length of time that $\overline{\text{DSACKx}}$ may precede data is given by parameter #31, and it must be met in any asynchronous system to insure that valid data is latched into the processor. (Refer to MC68030EC/D, *MC68030 Electrical Specifications* for timing parameters.) Notice that no maximum time is specified from the assertion of $\overline{\text{AS}}$ to the assertion of $\overline{\text{DSACKx}}$. Although the processor can transfer data in a minimum of three clock cycles when the cycle is terminated with $\overline{\text{DSACKx}}$, the processor inserts wait cycles in clock period increments until $\overline{\text{DSACKx}}$ is recognized.

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

7.4.1.1 INTERRUPT ACKNOWLEDGE CYCLE — TERMINATED NORMALLY. When the MC68030 processes an interrupt exception, it performs an interrupt acknowledge cycle to obtain the number of the vector that contains the starting location of the interrupt service routine.

Some interrupting devices have programmable vector registers that contain the interrupt vectors for the routines they use. The following paragraphs describe the interrupt acknowledge cycle for these devices. Other interrupting conditions or devices cannot supply a vector number and use the autovector cycle described in **7.4.1.2 AUTOVECTOR INTERRUPT ACKNOWLEDGE CYCLE.**

The interrupt acknowledge cycle is a read cycle. It differs from the asynchronous read cycle described in **7.3.1 Asynchronous Read Cycle** or the synchronous read cycle described in **7.3.4 Synchronous Read Cycle** in that it accesses the CPU address space. Specifically, the differences are:

1. FC0–FC2 are set to seven (FC0/FC1/FC2 = 111) for CPU address space.
2. A1, A2, and A3 are set to the interrupt request level (the inverted values of $\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$, and $\overline{\text{IPL2}}$, respectively).
3. The CPU space type field (A16–A19) is set to \$F, the interrupt acknowledge code.
4. A20–A31, A4–A15, and A0 are set to one.

The responding device places the vector number on the data bus during the interrupt acknowledge cycle. Beyond this, the cycle is terminated normally with either $\overline{\text{STERM}}$ or $\overline{\text{DSACKx}}$. Figure 7-43 is the flowchart of the interrupt acknowledge cycle.

To properly control termination of a bus cycle for a retry or a bus error condition, $\overline{\text{DSACKx}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ can be asserted and negated with the rising edge of the MC68030 clock. This assures that when two signals are asserted simultaneously, the required setup time (#47A) and hold time (#47B) for both of them is met for the same falling edge of the processor clock. (Refer to MC68030EC/D, *MC68030 Electrical Specifications* for timing requirements.) This or some equivalent precaution should be designed into the external circuitry that provides these signals.

The acceptable bus cycle terminations for asynchronous cycles are summarized in relation to $\overline{\text{DSACKx}}$ assertion as follows (case numbers refer to Table 7-8):

Normal Termination:
: $\overline{\text{DSACKx}}$ is asserted; $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ remain negated (case 1).

Halt Termination:
: $\overline{\text{HALT}}$ is asserted at same time or before $\overline{\text{DSACKx}}$, and $\overline{\text{BERR}}$ remains negated (case 2).

Bus Error Termination:
: $\overline{\text{BERR}}$ is asserted in lieu of, at the same time, or before $\overline{\text{DSACKx}}$ (case 3) or after $\overline{\text{DSACKx}}$ (case 4), and $\overline{\text{HALT}}$ remains negated; $\overline{\text{BERR}}$ is negated at the same time or after $\overline{\text{DSACKx}}$.

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

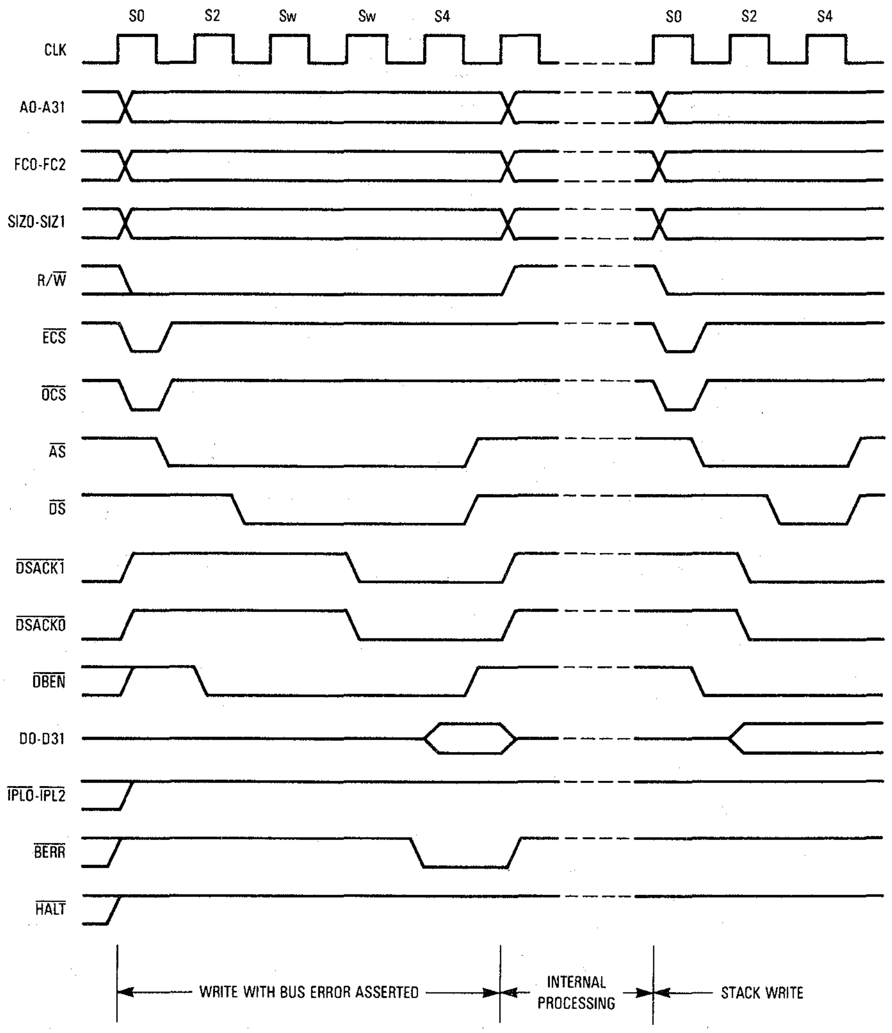
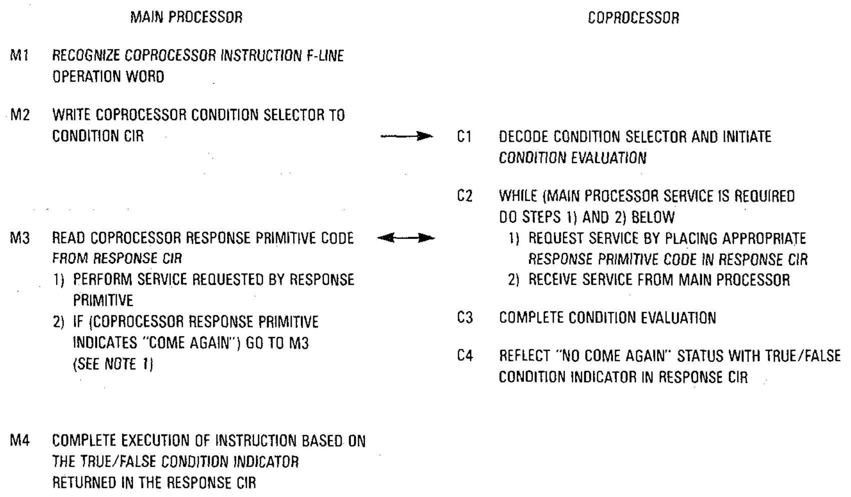


Figure 7-50. Late Bus Error with $\overline{\text{DSACKx}}$

A bus error occurring during a burst fill operation is a special case. If a bus error occurs during the first cycle of a burst, the data is ignored, the entire cache line is marked invalid, and the burst operation is aborted. If the cycle is for an instruction fetch, a bus error exception is made pending. This bus error is processed only if the execution unit attempts to use either of the two

Figure 10-8 shows the protocol for a conditional category coprocessor instruction. The main processor initiates execution of an instruction in this category by writing a condition selector to the condition CIR. The coprocessor decodes the condition selector to determine the condition to evaluate. The coprocessor can use response primitives to request that the main processor provide services required for the condition evaluation. After evaluating the condition, the coprocessor returns a true/false indicator to the main processor by placing a null primitive (refer to **10.4.4 Null Primitive**) in the response CIR. The main processor completes the coprocessor instruction execution when it receives the condition indicator from the coprocessor.



MAIN PROCESSOR

M1 RECOGNIZE COPROCESSOR INSTRUCTION F-LINE OPERATION WORD

M2 WRITE COPROCESSOR CONDITION SELECTOR TO CONDITION CIR

M3 READ COPROCESSOR RESPONSE PRIMITIVE CODE FROM RESPONSE CIR
1) PERFORM SERVICE REQUESTED BY RESPONSE PRIMITIVE
2) IF (COPROCESSOR RESPONSE PRIMITIVE INDICATES "COME AGAIN") GO TO M3 (SEE NOTE 1)

M4 COMPLETE EXECUTION OF INSTRUCTION BASED ON THE TRUE/FALSE CONDITION INDICATOR RETURNED IN THE RESPONSE CIR

COPROCESSOR

C1 DECODE CONDITION SELECTOR AND INITIATE CONDITION EVALUATION

C2 WHILE (MAIN PROCESSOR SERVICE IS REQUIRED DO STEPS 1) AND 2) BELOW
1) REQUEST SERVICE BY PLACING APPROPRIATE RESPONSE PRIMITIVE CODE IN RESPONSE CIR
2) RECEIVE SERVICE FROM MAIN PROCESSOR

C3 COMPLETE CONDITION EVALUATION

C4 REFLECT "NO COME AGAIN" STATUS WITH TRUE/FALSE CONDITION INDICATOR IN RESPONSE CIR

NOTES: 1. *All coprocessor response primitives, except the Null primitive, that allow the "Come Again" primitive attribute must indicate "Come Again" when used during the execution of a conditional category instruction. If a "Come Again" attribute is not indicated in one of these primitives, the main processor will initiate protocol violation exception processing (see 10.6.2.1 PROTOCOL VIOLATIONS)*

Figure 10-8. Coprocessor Interface Protocol for Conditional Category Instructions

10.2.2.1 BRANCH ON COPROCESSOR CONDITION INSTRUCTION. The conditional instruction category includes two formats of the M68000 Family branch instruction. These instructions branch on conditions related to the coprocessor operation. They execute similarly to the conditional branch instructions provided in the M68000 Family instruction set.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	1	1	0	1	0	0	0	0	0	0	0	0

Figure 10-34. Transfer Main Processor Control Register Primitive Format

When the main processor receives this primitive, it reads a control register select code from the register select CIR. This code determines which main processor control register is transferred. Table 10-5 lists the valid control register select codes. If the control register select code is not valid, the MC68030 initiates protocol violation exception processing (refer to **10.5.2.1 PROTOCOL VIOLATIONS**).

Table 10-5. Main Processor Control Register Selector Codes

Hex	Control Register
x000	Source Function Code (SFC) Register
x001	Destination Function Code (DFC) Register
x002	Cache Control Register (CACR)
x800	User Stack Pointer (USP)
x801	Vector Base Register (VBR)
x802	Cache Address Register (CAAR)
x803	Master Stack Pointer (MSP)
x804	Interrupt Stack Pointer (ISP)
All other codes cause a protocol violation exception	

After reading a valid code from the register select CIR, if DR=0, the main processor writes the long-word operand from the specified control register to the operand CIR. If DR=1, the main processor reads a long-word operand from the operand CIR and places it in the specified control register.

TAKE MID-INSTRUCTION EXCEPTION

15	14	13	12	11	10	9	8	7 … 0
0	PC	0	1	1	1	0	1	VECTOR NUMBER

TAKE POST-INSTRUCTION EXCEPTION

15	14	13	12	11	10	9	8	7 … 0
0	PC	0	1	1	1	1	0	VECTOR NUMBER

WRITE TO PREVIOUSLY EVALUATED EFFECTIVE ADDRESS

15	14	13	12	11	10	9	8	7 … 0
CA	PC	1	0	0	0	0	0	LENGTH

sequencer activity. The execution of an instruction that only accesses on-chip registers can be overlapped entirely with a concurrent data write generated by a previous instruction, if prefetches generated by that instruction are resident in the instruction cache.

11.2 RESOURCE SCHEDULING

Some of the variability in instruction execution timings results from the overlap of resource utilization. The processor can be viewed as consisting of eight independently scheduled resources. Since very little of the scheduling is directly related to instruction boundaries, it is impossible to make accurate estimates of the time required to execute a particular instruction without knowing the complete context within which the instruction is executing. The position of these resources within the MC68030 is shown in Figure 11-1.

11.2.1 Microsequencer

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

11.2.2 Instruction Pipe

The MC68030 contains a three-word instruction pipe where instruction opcodes are decoded. As shown in Figure 11-1, instruction words (instruction operation words and all extension words) enter the pipe at stage B and proceed to stages C and D. An instruction word is completely decoded when it reaches stage D of the pipe. Each of the pipe stages has a status bit that reflects whether the word in the stage was loaded with data from a bus cycle that was terminated abnormally. Stages of the pipe are only filled in response to specific prefetch requests issued by the microsequencer.

Words are loaded into the instruction pipe from the cache holding register. While the individual stages of the pipe are only 16 bits wide, the cache holding register is 32 bits wide and contains the entire long word. This long word is obtained from the instruction cache or the external bus in response to a prefetch request from the microsequencer. When the microsequencer re-

The instructions that require the instruction-cache case, head, and tail of an effective address (CCea, Hea, and Tea) to be overlapped with CCop, Hop, and Top are footnoted in **11.6 INSTRUCTION TIMING TABLES**.

The actual instruction-cache-case execution time for a stream of instructions can be computed using Equation (11-1) or the general Equation (11-2). Equation (11-1) is used unless the instruction-cache case, head, and tail of an effective address are required.

An example using a series of instructions that require Equation (11-1) to calculate the instruction-cache-case execution time follows. The assumptions referred to in **11.6 INSTRUCTION TIMING TABLES** apply.

	Instruction	
1.	ADD.L	A1,D1
2.	SUBA.L	D1,A2

Referring to the timing table in **11.6.8 Arithmetic/Logical Instructions**, the head, tail, and instruction-cache-case (CC) times for ADD.L A1,D1 and SUBA.L D1,A2 are found. There is no footnote directing the user to add an effective address time for either instruction. Since both of the instructions use register operands only, there is no need to add effective address calculation times. Therefore, the general Equation (11-1) can be used for both.

		Head	**Tail**	**CC**
1.	ADD.L A1,D1	2	$\underline{0}$	2
2.	SUBA.L D1,A2	$\underline{4}$	0	4

NOTE

The underlined numbers show the typical pattern for the comparison of head and tail in the following equation.

The following computations use Equation (11-1):

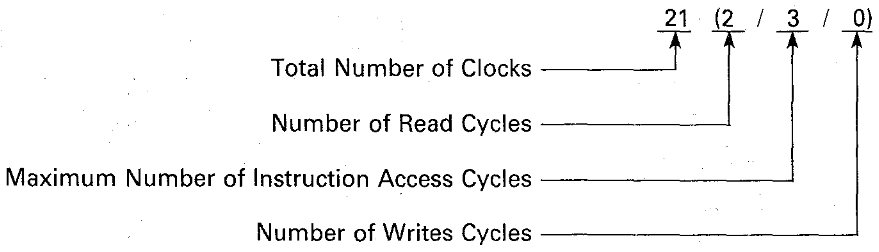
$$\begin{aligned}\text{Execution Time} &= CC_1 + [CC_2 - \min(H_2,T_1)]\\ &= 2 + [4 - \min(4,0)]\\ &= 2 + [4 - 0]\\ &= 6 \text{ clocks}\end{aligned}$$

Instructions that require the addition of an effective address calculation time from an appropriate table use the general Equation (11-2) to calculate the actual CC time. The CCea, Hea, and Tea values must be extracted from the appropriate effective address table (either fetch effective address, fetch im-

		Head	Tail	CC
1.	ADD.L −(A1),D1			
	fea −(An)	2	2	4(1/0/0)
	*	2	1	3(1/0/0)
	**	**2**	**1**	**3(1/0/0)**
	ADD.L EA,Dn	0	0	2(0/1/0)
	*	0	0	2(0/1/0)
	**	**0**	**0**	**2(0/1/0)**
2.	AND.L D1,([A1])			
	fea ([B])	4	0	10(1/0/0)
	*	4	0	10(1/0/0)
	***	**4**	**0**	**12(1/0/0)**
	AND Dn,EA	0	1	3(0/0/1)
	*	0	1	3(0/0/1)
	**	**0**	**3**	**5(0/0/1)**
3.	MOVE.L (A6),(8,A1)			
	fea (An)	1	1	3(1/0/0)
	*	1	0	2(1/0/0)
	**	**1**	**0**	**2(1/0/0)**
	MOVE Source,(d_{16},An)	2	0	4(0/0/1)
	*	2	0	4(0/0/1)
	**	**2**	**2**	**6(0/0/1)**
4.	TAS (A3)+			
	Cea (An)	0	0	2(0/0/0)
	*	0	0	2(0/0/0)
	**	**0**	**0**	**2(0/0/0)**
	TAS Mem	3	0	12(1/0/1)
	*	3	0	12(1/0/1)
	**	**3**	**0**	**14(1/0/1)**

NOTES:
*Corrected for data cache hits.
**Corrected for wait states also (only on data writes).
***No data cache hit assumed for address fetch.

The instruction-cache-case and average no-cache-case columns of the instruction timing tables contain four sets of numbers, three of which are enclosed in parentheses. The outer number is the total number of clocks for the given cache case and instruction. The first number inside the parentheses is the number of operand read cycles performed by the instruction. The second value inside the parentheses is the maximum number of instruction bus cycles performed by the instruction, including all prefetches to keep the instruction pipe filled. Because the second value is the average of the odd-word-aligned case and the even-word-aligned case (rounded up to an integral number of bus cycles), it is always greater than or equal to the actual number of bus cycles (one bus cycle per two instruction prefetches). The third value within the parentheses is the number of write cycles performed by the instruction. One example from the instruction timing table is:



The total numbers of bus-activity clocks and internal clocks (not overlapped by bus activity) of the instruction in this example are derived as follows:

(2 Reads•2 Clocks/Read) + (3 Instruction Accesses•2 Clocks/Access) +
(0 Writes•2 Clocks/Write) = 10 Clocks of Bus Activity
21 Total Clocks − 10 Bus Activity Clocks = 11 Internal Clocks

The example used here is taken from a no-cache-case 'fetch effective address' time. The addressing mode is ([d_{32},B],I,d_{32}). The same addressing mode under the instruction-cache-case execution time entry is 18(2/0/0). For the instruction-cache-case execution time, no instruction accesses are required because the cache is enabled and the sequencer does not have to access external memory for the instruction words.

The first five timing tables deal exclusively with fetching and calculating effective addresses and immediate operands. The remaining tables are instruction and operation timings. Some instructions use addressing modes that are not included in the corresponding instruction timings. These cases refer to footnotes that indicate the additional table needed for the timing calculation. All read and write accesses are assumed to take two clock periods.

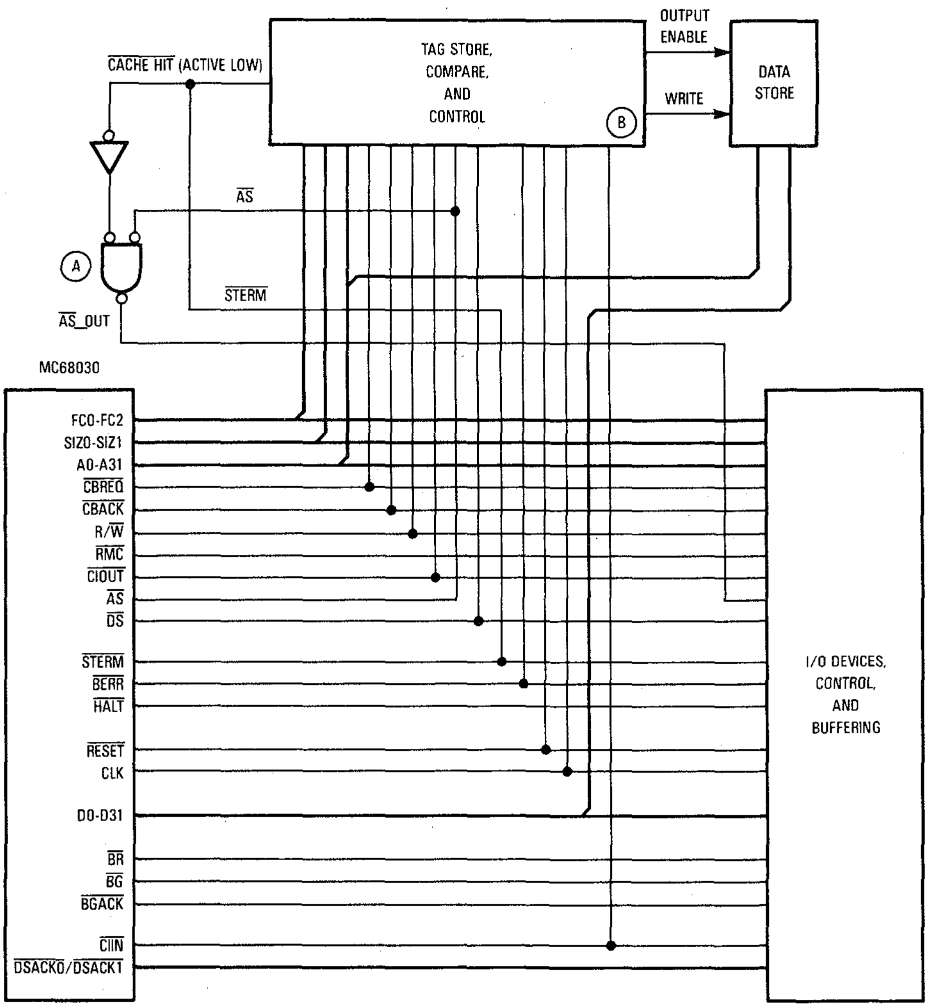


Figure 12-17. Example MC68030 Hardware Configuration with External Physical Cache

Normally, as soon as $\overline{\text{AS}}$ is asserted, circuit (C) immediately asserts the $\overline{\text{STERM}}$ signal to terminate the bus cycle, assuming that the cache will produce a valid hit later in the cycle. Circuit (C) also prevents the early termination from occurring from those cycles that access operands that are noncachable or had missed in the cache on the previous cycle (and have not already been retried). In this example, (C) prevents early termination of all CPU space accesses, all write cycles (assuming a writethrough cache is implemented), To minimize the potential for delays in retrying a bus cycle, the negation path of the bus error and halt signals should be carefully controlled. Light capacitive loading of these signals lines as well as the use of a properly sized pullup resistor for any open collector drivers, or some equivalent method, is recommended.

The available cache tag lookup, compare, and logic delay (D) and (E) time for this implementation is given by Equation 12-5 of Table 12-2 (40 ns at 20.0 MHz with no wait states).

A further design consideration is the response of the main memory controller to accesses that miss in the cache and are retried. During a retry operation and in the absence of arbitration for the logical bus, the MC68030 continuously drives the address bus with the address that caused the retry to be signaled. This presents the designer with the opportunity to utilize this information to continue (or initiate) the access in the main memory (by latching the state of the $\overline{\text{AS}}$ signal during the initial bus cycle and holding it asserted for the duration of the retry), thus decreasing the overhead associated with retrying the cycle.

12.6.2 Instruction-Only External Cache Implementations

In some cases, particularly in multiprocessing systems where cache coherence is a concern, it is desirable to store only instruction operands since they are not considered to be alterable and, hence, cannot generate stale data. In general, this is feasible with the MC68000 architecture as long as PC relative addressing modes are not used. This restriction allows program and data accesses to be distinguished externally by decoding the function code signals.

12.7 DEBUGGING AIDS

The MC68030 supports the monitoring of internal microsequencer activity with the $\overline{\text{STATUS}}$ and $\overline{\text{REFILL}}$ signals. The use of these signals is described in the following paragraph. A useful device to aid programming debugging is described in **12.7.2 Real-Time Instruction Trace**.