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	Obsolete
Core Processor	68020
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	20MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	114-BPGA
Supplier Device Package	114-PGA (34.55x34.55)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68020rc20e

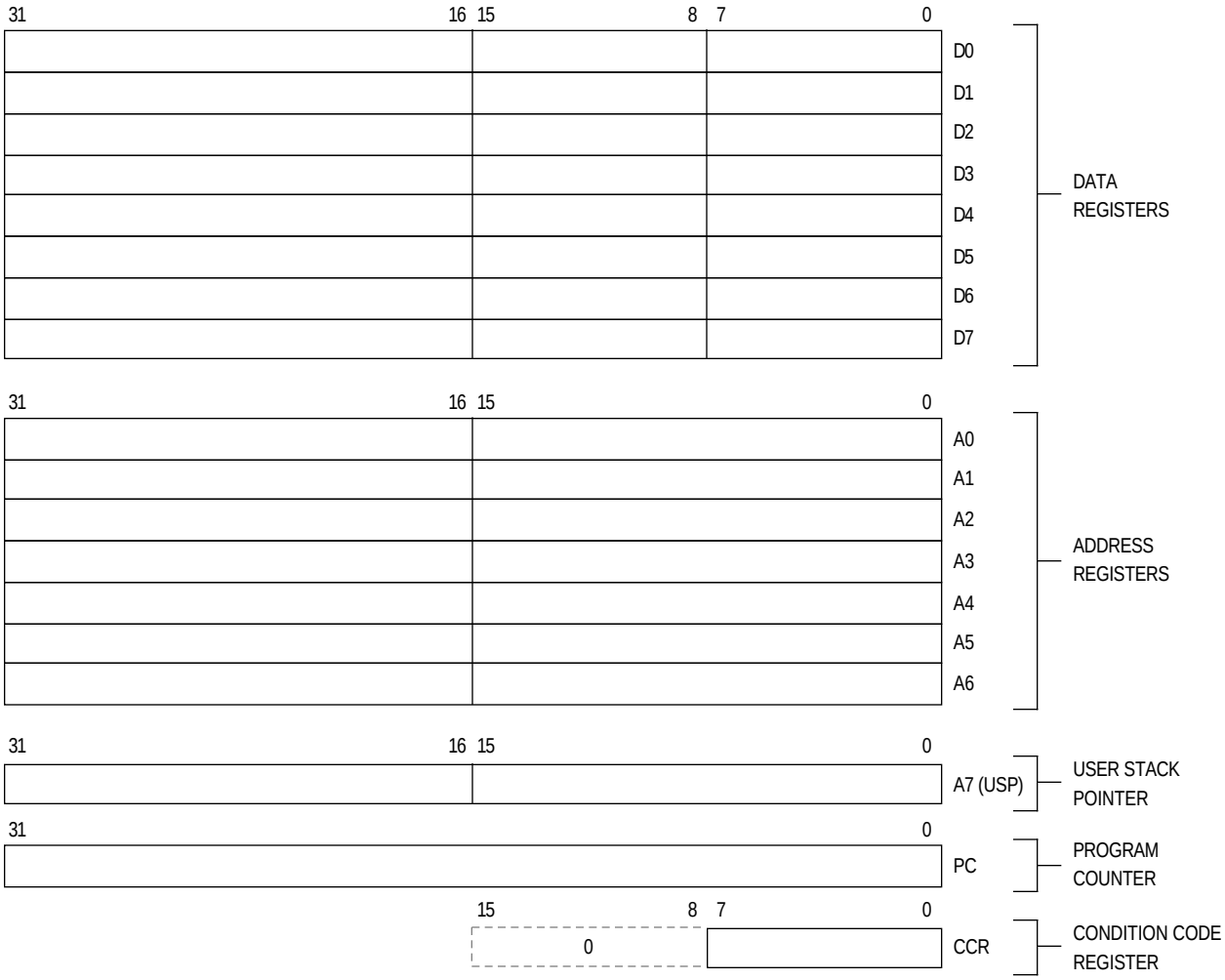


Figure 1-2. User Programming Model

2.3 EXCEPTION PROCESSING

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

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

2.3.1 Exception Vectors

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

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

2.3.2 Exception Stack Frame

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

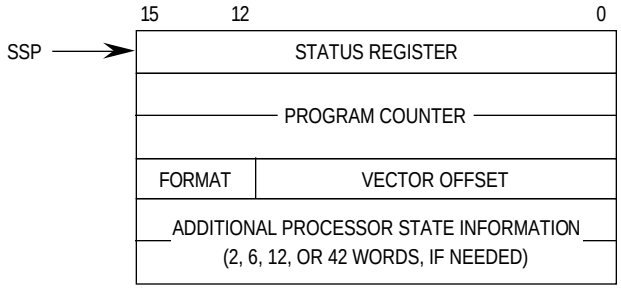


Figure 2-1. General Exception Stack Frame

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

Table 5-4. Data Bus Requirements for Read Cycles

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

5.2.2 Misaligned Operands

Since operands may reside at any byte boundary, they may be misaligned. A byte operand is properly aligned at any address; a word operand is misaligned at an odd address; a long word is misaligned at an address that is not evenly divisible by four. The MC68000, MC68008, and MC68010 implementations allow long-word transfers on odd-word boundaries but force exceptions if word or long-word operand transfers are attempted at odd-byte addresses. Although the MC68020/EC020 does not enforce any alignment restrictions for data operands (including PC relative data addresses), some performance degradation occurs when additional bus cycles are required for long-word or word operands that are misaligned. For maximum performance, data items should be aligned on their natural boundaries. All instruction words and extension words must reside on word boundaries. Attempting to prefetch an instruction word at an odd address causes an address error exception.

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

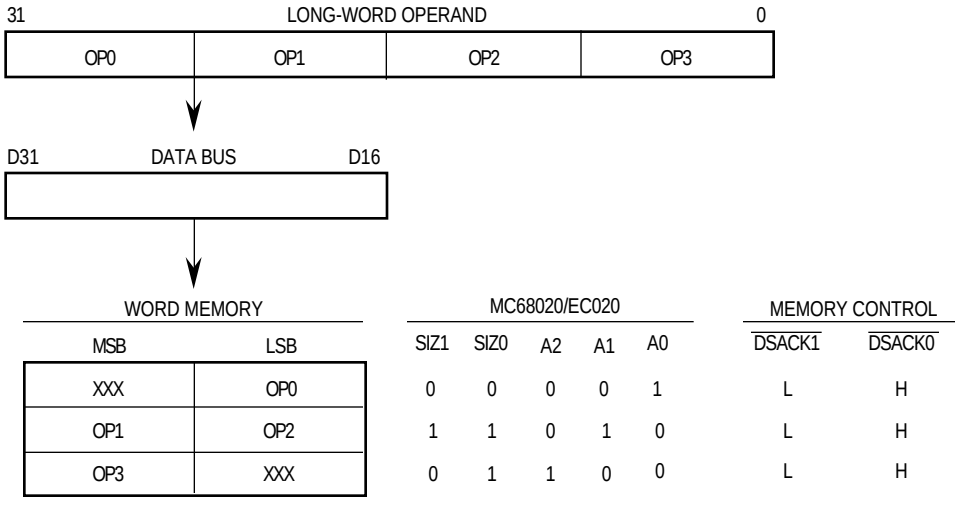
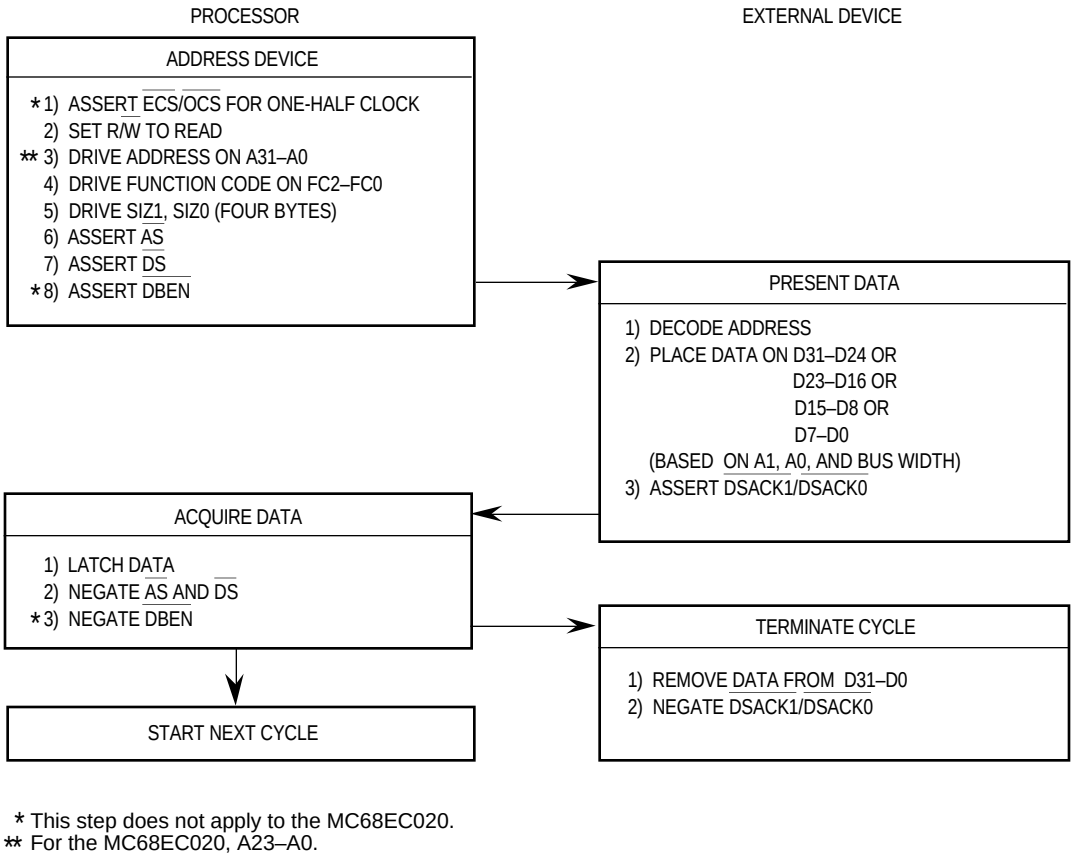


Figure 5-9. Misaligned Long-Word Operand Write to Word Port Example



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

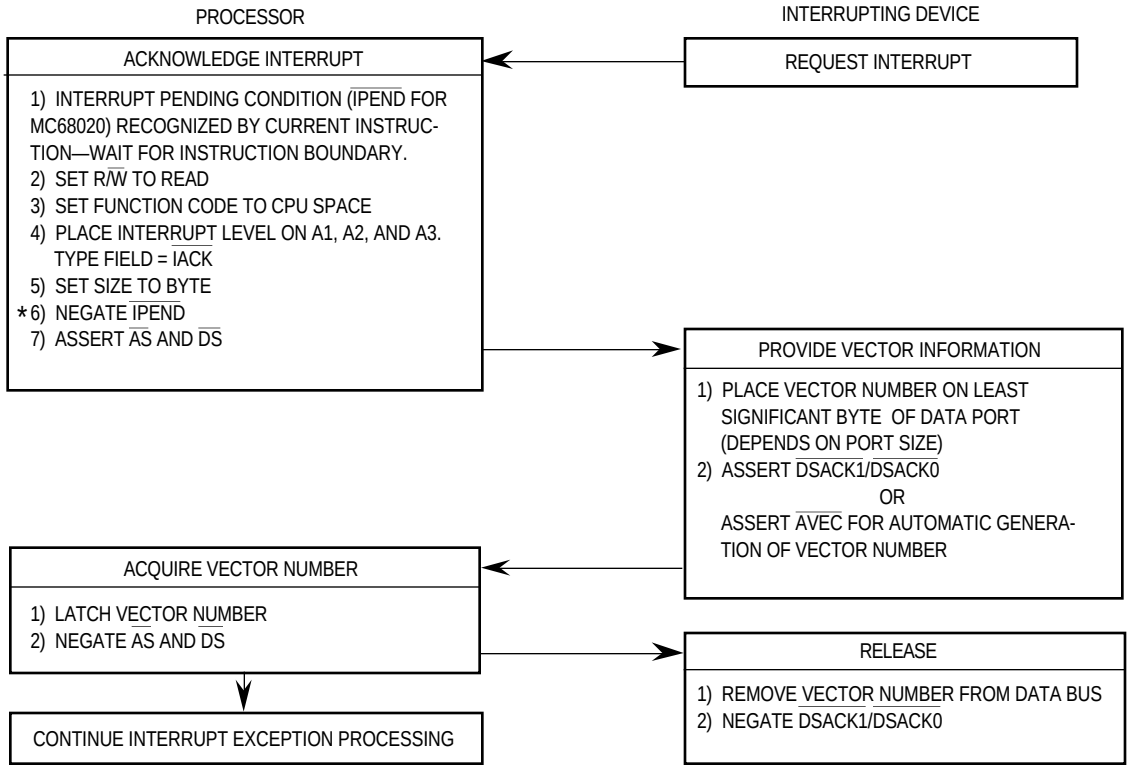
Figure 5-20. Byte Read Cycle Flowchart

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

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

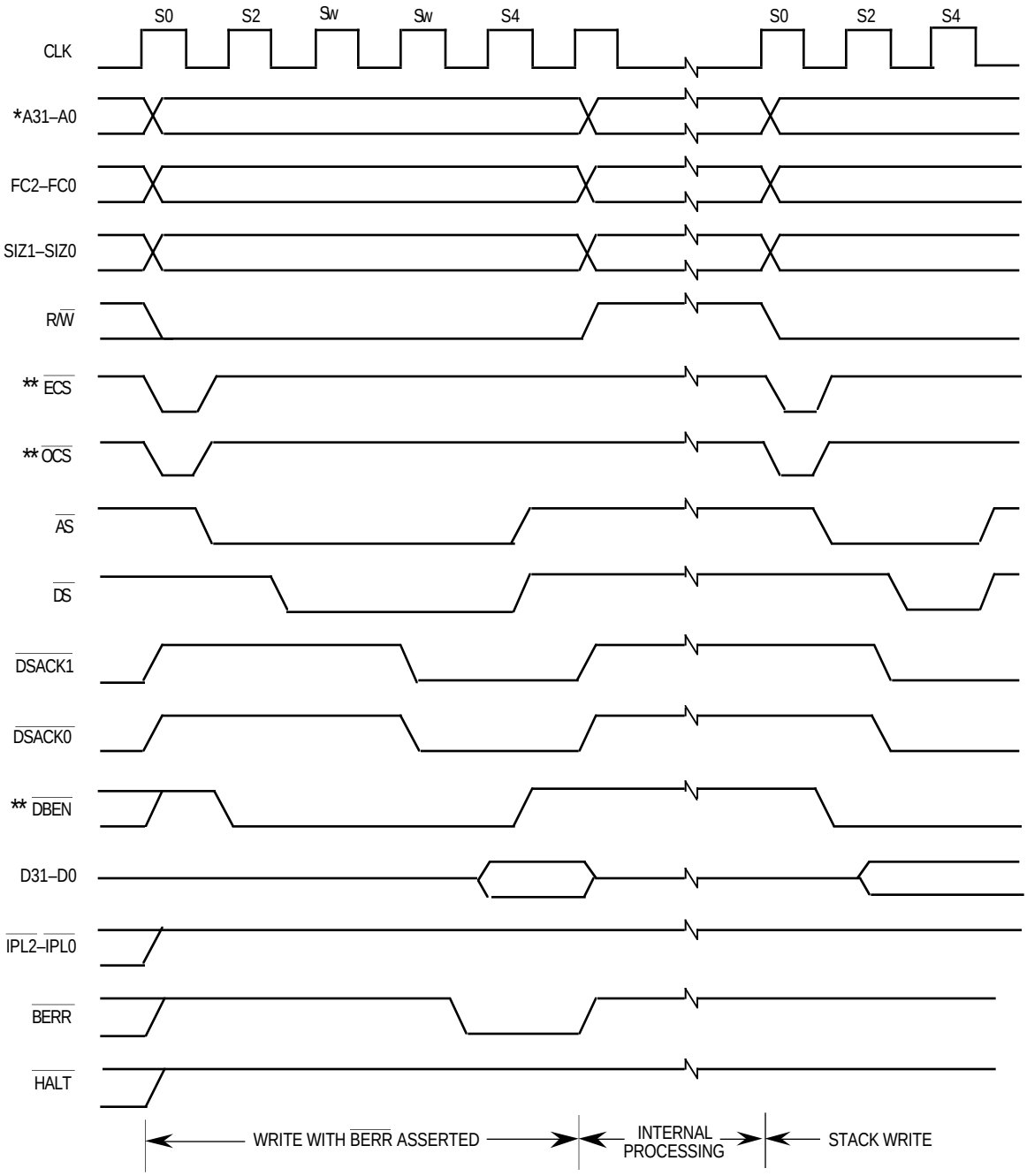
The responding device places the vector number on the data bus during the interrupt acknowledge cycle. Beyond this, the cycle is terminated normally with $\overline{DSACK1}/\overline{DSACK0}$. Figure 5-32 is the flowchart of the interrupt acknowledge cycle.

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



* This step does not apply to the MC68EC020.

Figure 5-32. Interrupt Acknowledge Cycle Flowchart



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

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

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

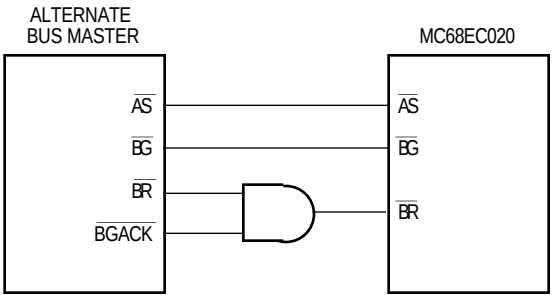


Figure 5-50. Interface for Three-Wire to Two-Wire Bus Arbitration

5.8 RESET OPERATION

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

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

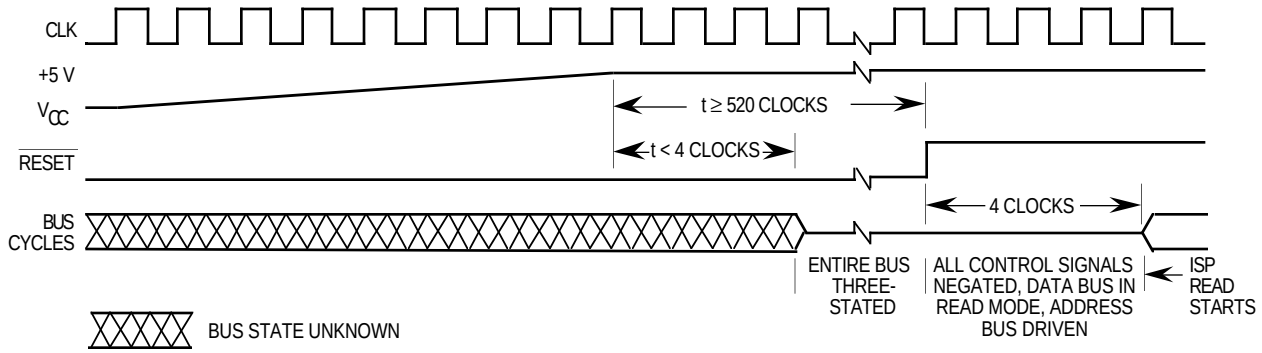


Figure 5-51. Initial Reset Operation Timing

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

When a RESET instruction is executed, the processor drives the RESET signal for 512 clock cycles. In this case, the processor resets the external devices of the system, and the internal registers of the processor are unaffected. The external devices connected to the RESET signal are reset at the completion of the RESET instruction. An external RESET signal that is asserted to the processor during execution of a RESET instruction must extend beyond the reset period of the instruction by at least eight clock cycles to reset the processor. Figure 5-52 shows the timing information for the RESET instruction.

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

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

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

7.1.3 Coprocessor Instruction Format

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

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

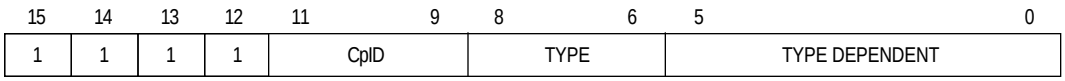
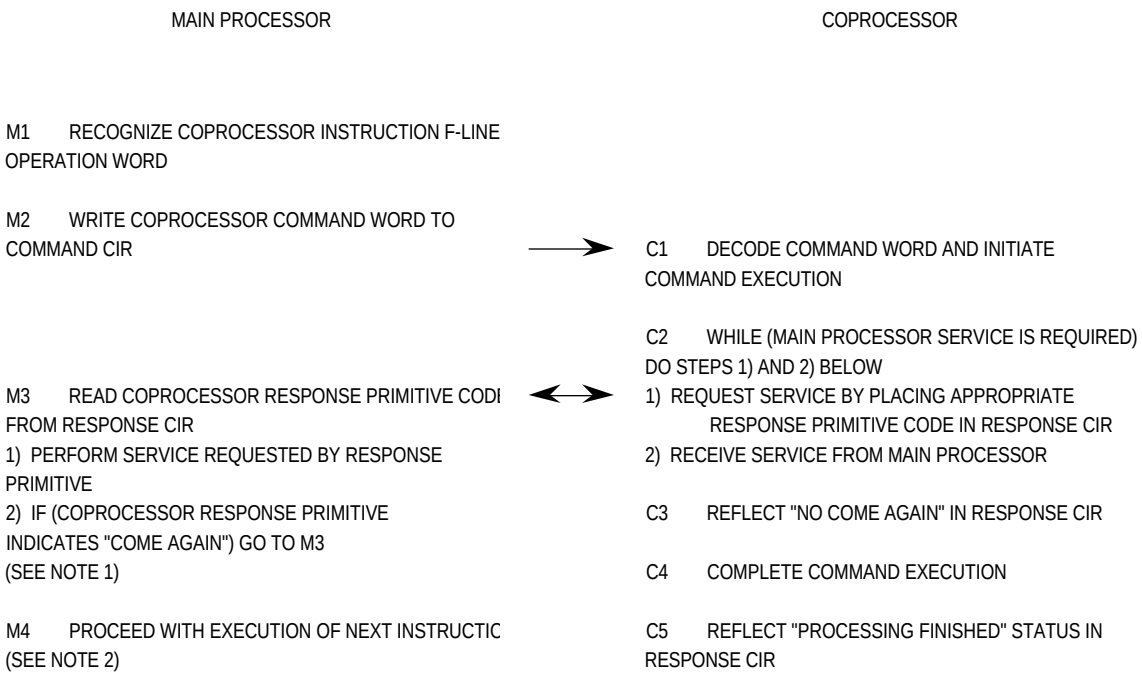


Figure 7-1. F-Line Coprocessor Instruction Operation Word

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



NOTES: 1. "Come Again" indicates that further service of the main processor is being requested by the coprocessor.
2. The next instruction should be the operation word pointed to by the ScanPC at this point. The operation of the MC68020/EC020 ScanPC is discussed in **7.4.1 ScanPC**.

Figure 7-7. Coprocessor Interface Protocol for General Category Instructions

7.2.2 Coprocessor Conditional Instructions

The conditional instruction category provides program control based on the operations of the coprocessor. The coprocessor evaluates a condition and returns a true/false indicator to the main processor. The main processor completes the execution of the instruction based on this true/false condition indicator.

The implementation of instructions in the conditional category promotes efficient use of both the main processor and the coprocessor hardware. The condition specified for the instruction is related to the coprocessor operation and is therefore evaluated by the coprocessor. However, the instruction completion following the condition evaluation is directly related to the operation of the main processor. The main processor performs the change of flow, the setting of a byte, or the TRAP operation, since its architecture explicitly implements these operations for its instruction set.

Figure 7-8 shows the protocol for a conditional category coprocessor instruction. The main processor initiates execution of an instruction in this category by writing a condition selector to the condition CIR. The coprocessor decodes the condition selector to determine the condition to evaluate. The coprocessor can use response primitives to request that the main processor provide services required for the condition evaluation.

7.3.7 Condition CIR

The main processor initiates a conditional category instruction by writing the condition selector to bits 5–0 of the 16-bit condition CIR. Bits 15–6 are undefined and reserved by Motorola. The offset from the base address of the CIR set for the condition CIR is $0E. Figure 7-20 shows the format of the condition CIR.

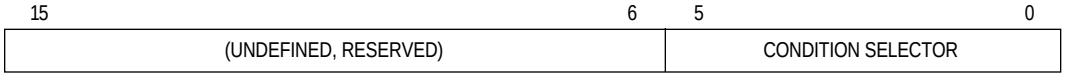


Figure 7-20. Condition CIR Format

7.3.8 Operand CIR

When the coprocessor requests the transfer of an operand, the main processor performs the transfer by reading from or writing to the 32-bit operand CIR. The offset from the base address of the CIR set for the operand CIR is $10.

The MC68020/EC020 aligns all operands transferred to and from the operand CIR to the most significant byte of this CIR. The processor performs a sequence of long-word transfers to read or write any operand larger than four bytes. If the operand size is not a multiple of four bytes, the portion remaining after the initial long-word transfer is aligned to the most significant byte of the operand CIR. Figure 7-21 shows the operand alignment used by the MC68020/EC020 when accessing the operand CIR.

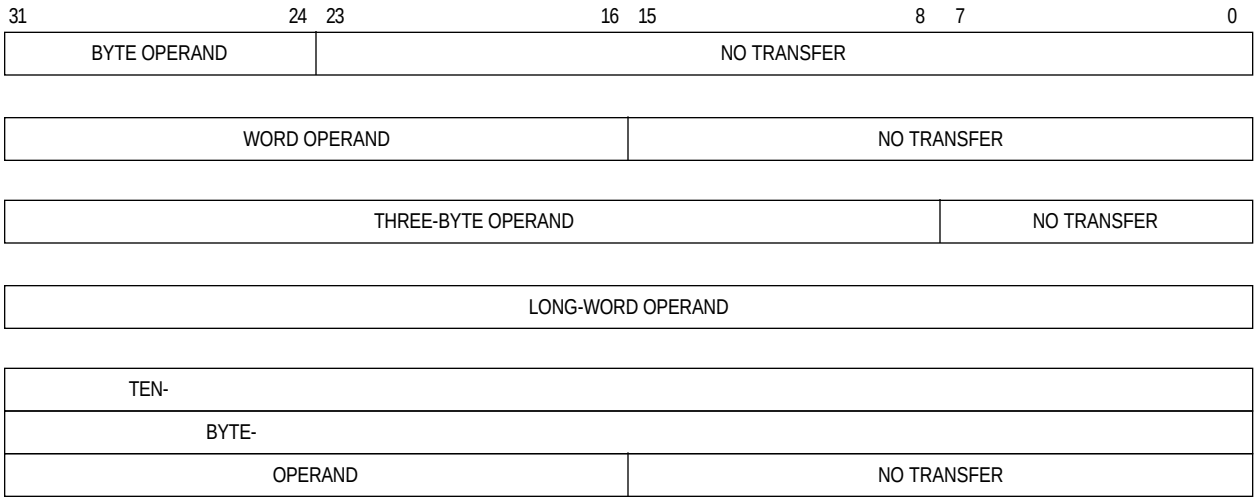


Figure 7-21. Operand Alignment for Operand CIR Accesses

7.4.3 Busy Primitive

The busy response primitive causes the main processor to reinitiate a coprocessor instruction. This primitive applies to instructions in the general and conditional categories. Figure 7-23 shows the format of the busy primitive.

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

Figure 7-23. Busy Primitive Format

The busy primitive uses the PC bit as described in **7.4.2 Coprocessor Response Primitive General Format**.

Coprocessors that can operate concurrently with the main processor but cannot buffer write operations to their command or condition CIR use the busy primitive. A coprocessor may execute a cpGEN instruction concurrently with an instruction in the main processor. If the main processor attempts to initiate an instruction in the general or conditional instruction category while the coprocessor is executing a cpGEN instruction, the coprocessor can place the busy primitive in the response CIR. When the main processor reads this primitive, it services pending interrupts using a preinstruction exception stack frame (refer to Figure 7-41). The processor then restarts the general or conditional coprocessor instruction that it had attempted to initiate earlier.

The busy primitive should only be used in response to a write to the command or condition CIR. It should be the first primitive returned after the main processor attempts to initiate a general or conditional category instruction. In particular, the busy primitive should not be issued after program-visible resources have been altered by the instruction. (Program-visible resources include coprocessor and main processor program-visible registers and operands in memory, but not the scanPC.) The restart of an instruction after it has altered program-visible resources causes those resources to have inconsistent values when the processor reinitiates the instruction.

The MC68020/EC020 responds to the busy primitive differently in a special case that can occur during a breakpoint operation (refer to **Section 6 Exception Processing**). This special case occurs when a breakpoint acknowledge cycle initiates a coprocessor F-line instruction, the coprocessor returns the busy primitive in response to the instruction initiation, and an interrupt is pending. When these three conditions are met, the processor reexecutes the breakpoint acknowledge cycle after completion of interrupt exception processing. A design that uses a breakpoint to monitor the number of passes through a loop by incrementing or decrementing a counter may not work correctly under these conditions. This special case may cause several breakpoint acknowledge cycles to be executed during a single pass through a loop.

and PF = 1, and then performs trace exception processing. When IA = 1, the main processor services pending interrupts before reading the response CIR again.

A coprocessor can be designed to execute a cpGEN instruction concurrently with the execution of main processor instructions and, also, buffer one write operation to either its command or condition CIR. This type of coprocessor issues a null primitive with CA = 1 when it is concurrently executing a cpGEN instruction, and the main processor initiates another general or conditional coprocessor instruction. This primitive indicates that the coprocessor is busy and the main processor should read the response CIR again without reinitiating the instruction. The IA bit of this null primitive usually should be set to minimize interrupt latency while the main processor is waiting for the coprocessor to complete the general category instruction.

Table 7-3 summarizes the encodings of the null primitive.

Table 7-3. Null Coprocessor Response Primitive Encodings

CA	PC	IA	PF	TF	General Instructions	Conditional Instructions
x	1	x	x	x	Pass Program Counter to Instruction Address CIR, Clear PC Bit, and Proceed with Operation Specified by CA, IA, PF, and TF Bits	Same as General Category
1	0	0	x	x	Reread Response CIR, Do Not Service Pending Interrupts	Same as General Category
1	0	1	x	x	Service Pending Interrupts and Reread the Response CIR	Same as General Category
0	0	0	0	c	If (Trace Pending) Reread Response CIR; Else, Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c
0	0	1	0	c	If (Trace Pending) Service Pending Interrupts and Reread Response CIR; Else, Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c
0	0	x	1	c	Coprocessor Instruction Completed; Service Pending Exceptions or Execute Next Instruction	Main Processor Completes Instruction Execution Based on TF = c.

x = Don't Care

c = 1 or 0 Depending on Coprocessor Condition Evaluation

8.2.2 Fetch Immediate Effective Address

The fetch immediate effective address table indicates the number of clock periods needed for the processor to fetch the immediate source operand and calculate and fetch the specified destination operand. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Address Mode	Best Case	Cache Case	Worst Case
#<data>.W,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
#<data>.L,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
#<data>.W,(An)	**3**(1/0/0)	**4**(1/0/0)	**4**(1/1/0)
#<data>.L,(An)	**3**(1/0/0)	**4**(1/0/0)	**7**(1/1/0)
#<data>.W,(An)+	**4**(1/0/0)	**6**(1/0/0)	**7**(1/1/0)
#<data>.L,(An)+	**5**(1/0/0)	**8**(1/0/0)	**9**(1/1/0)
#<data>.W,–(An)	**3**(1/0/0)	**5**(1/0/0)	**6**(1/1/0)
#<data>.L,–(An)	**4**(1/0/0)	**7**(1/0/0)	**8**(1/1/0)
#<data>.W,(bd,An)	**3**(1/0/0)	**5**(1/0/0)	**7**(1/1/0)
#<data>.L,(bd,An)	**4**(1/0/0)	**7**(1/0/0)	**10**(1/2/0)
#<data>.W,xxx.W	**3**(1/0/0)	**5**(1/0/0)	**7**(1/1/0)
#<data>.L,xxx.W	**4**(1/0/0)	**7**(1/0/0)	**10**(1/2/0)
#<data>.W,xxx.L	**3**(1/0/0)	**6**(1/0/0)	**10**(1/2/0)
#<data>.L,xxx.L	**4**(1/0/0)	**8**(1/0/0)	**12**(1/2/0)
#<data>.W,#<data>.B,W	**0**(0/0/0)	**4**(0/0/0)	**6**(0/2/0)
#<data>.L,#<data>.B,W	**1**(0/0/0)	**6**(0/0/0)	**8**(0/2/0)
#<data>.W,#<data>.L	**0**(0/0/0)	**6**(0/0/0)	**8**(0/2/0)
#<data>.L,#(data>.L	**1**(0/0/0)	**8**(0/0/0)	**10**(0/2/0)
#<data>.W,(d_8,An,Xn) or (d_8,PC,Xn)	**4**(1/0/0)	**9**(1/0/0)	**11**(1/2/0)
#<data>.L,(d_8,An,Xn) or (d_8,PC,Xn)	**5**(1/0/0)	**11**(1/0/0)	**13**(1/2/0)
#<data>.W,(d_{16},An,Xn) or (d_{16},PC,Xn)	**4**(1/0/0)	**9**(1/0/0)	**12**(1/2/0)
#<data>.L,(d_{16},An,Xn) or (d_{16},PC,Xn)	**5**(1/0/0)	**11**(1/0/0)	**15**(1/2/0)
#<data>.W,(B)	**4**(1/0/0)	**9**(1/0/0)	**12**(1/1/0)
#<data>.L,(B)	**5**(1/0/0)	**11**(1/0/0)	**14**(1/2/0)
#<data>.W,(bd,PC)	**10**(1/0/0)	**15**(1/0/0)	**19**(1/3/0)
#<data>.L,(bd,PC)	**11**(1/0/0)	**17**(1/0/0)	**21**(1/3/0)
#<data>.W,(d_{16},B)	**6**(1/0/0)	**11**(1/0/0)	**15**(1/2/0)
#<data>.L,(d_{16},B)	**7**(1/0/0)	**13**(1/0/0)	**17**(1/2/0)
#<data>.W,(d_{32},B)	**10**(1/0/0)	**15**(1/0/0)	**19**(1/3/0)
#<data>.L,(d_{32},B)	**11**(1/0/0)	**17**(1/0/0)	**21**(1/3/0)
#<data>.W,([B],I)	**9**(2/0/0)	**14**(2/0/0)	**16**(2/2/0)
#<data>.L,([B],I)	**10**(2/0/0)	**16**(2/0/0)	**18**(2/2/0)
#<data>.W,([B],I,d_{16})	**11**(2/0/0)	**16**(2/0/0)	**19**(2/2/0)
#<data>.L,([B],I,d_{16})	**12**(2/0/0)	**18**(2/0/0)	**21**(2/2/0)

CACHE CASE (Continued)

Source Address Mode	Destination							
	(d$_8$,An,Xn)	**(d$_{16}$,An,Xn)**	**(B)**	**(d$_{16}$,B)**	**(d$_{32}$,B)**	**([B],I)**	**([B],I,d$_{16}$)**	**([B],I,d$_{32}$)**
Rn	**7**(0/0/1)	**9**(0/0/1)	**8**(0/0/1)	**10**(0/0/1)	**14**(0/0/1)	**12**(1/0/1)	**14**(1/0/1)	**15**(1/0/1)
#<data>.B,W	**7**(0/0/1)	**9**(0/0/1)	**8**(0/0/1)	**10**(0/0/1)	**14**(0/0/1)	**12**(1/0/1)	**14**(1/0/1)	**15**(1/0/1)
#<data>.L	**9**(0/0/1)	**11**(0/0/1)	**10**(0/0/1)	**12**(0/0/1)	**16**(0/0/1)	**14**(1/0/1)	**16**(1/0/1)	**17**(1/0/1)
(An)	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(An)+	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
–(An)	**10**(1/0/1)	**12**(1/0/1)	**11**(1/0/1)	**13**(1/0/1)	**17**(1/0/1)	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)
(d$_{16}$,An) or (d$_{16}$,PC)	**10**(1/0/1)	**12**(2/0/1)	**11**(1/0/1)	**13**(1/0/1)	**17**(1/0/1)	**15**(2/0/1)	**17**(2/0/1)	**18**(2/0/1)
(xxx).W	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(xxx).L	**9**(1/0/1)	**11**(1/0/1)	**10**(1/0/1)	**12**(1/0/1)	**16**(1/0/1)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/0/1)
(d$_8$,An,Xn) or (d$_8$,PC,Xn)	**12**(1/0/1)	**14**(1/0/1)	**13**(1/0/1)	**15**(1/0/1)	**19**(1/0/1)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)
(d$_{16}$,An,Xn) or (d$_{16}$,PC,Xn)	**12**(1/0/1)	**14**(1/0/1)	**13**(1/0/1)	**15**(1/0/1)	**19**(1/0/1)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)
(B)	**12**(1/0/1)	**14**(1/0/1)	**13**(1/0/1)	**15**(1/0/1)	**19**(1/0/1)	**17**(2/0/1)	**19**(2/0/1)	**20**(2/0/1)
(d$_{16}$,B)	**14**(1/0/1)	**16**(1/0/1)	**15**(1/0/1)	**17**(1/0/1)	**21**(1/0/1)	**19**(2/0/1)	**21**(2/0/1)	**22**(2/0/1)
(d$_{32}$,B)	**18**(1/0/1)	**20**(1/0/1)	**19**(1/0/1)	**21**(1/0/1)	**25**(1/0/1)	**23**(2/0/1)	**25**(2/0/1)	**26**(2/0/1)
([B],I)	**17**(2/0/1)	**19**(2/0/1)	**18**(2/0/1)	**20**(2/0/1)	**24**(2/0/1)	**22**(3/0/1)	**24**(3/0/1)	**25**(3/0/1)
([B],I,d$_{16}$)	**19**(2/0/1)	**21**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**26**(2/0/1)	**24**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)
([B],I,d$_{32}$)	**19**(2/0/1)	**21**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**26**(2/0/1)	**24**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)
([d$_{16}$,B],I)	**19**(2/0/1)	**21**(2/0/1)	**20**(2/0/1)	**22**(2/0/1)	**26**(2/0/1)	**24**(3/0/1)	**26**(3/0/1)	**27**(3/0/1)
([d$_{16}$,B],I,d$_{16}$)	**21**(2/0/1)	**23**(2/0/1)	**22**(2/0/1)	**24**(2/0/1)	**28**(2/0/1)	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)
([d$_{16}$,B],I,d$_{32}$)	**21**(2/0/1)	**23**(2/0/1)	**22**(2/0/1)	**24**(2/0/1)	**28**(2/0/1)	**26**(3/0/1)	**28**(3/0/1)	**29**(3/0/1)
([d$_{32}$,B],I)	**23**(2/0/1)	**25**(2/0/1)	**24**(2/0/1)	**26**(2/0/1)	**30**(2/0/1)	**28**(3/0/1)	**30**(3/0/1)	**31**(3/0/1)
([d$_{32}$,B],I,d$_{16}$)	**25**(2/0/1)	**27**(2/0/1)	**26**(2/0/1)	**28**(2/0/1)	**32**(2/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)
([d$_{32}$,B],I,d$_{32}$)	**25**(2/0/1)	**27**(2/0/1)	**26**(2/0/1)	**28**(2/0/1)	**32**(2/0/1)	**30**(3/0/1)	**32**(3/0/1)	**33**(3/0/1)

WORST CASE (Continued)

Source Address Mode	Destination							
	(d_8,An,Xn)	(d_{16},An,Xn)	(B)	(d_{16},B)	(d_{32},B)	([B],I)	([B],I,d_{16})	([B],I,D_{32})
Rn	**9**(0/1/1)	**12**(0/2/1)	**10**(0/1/1)	**14**(0/2/1)	**19**(0/2/1)	**14**(1/1/1)	**17**(1/2/1)	**20**(1/2/1)
#<data>.B,W	**9**(0/1/1)	**12**(0/2/1)	**10**(0/1/1)	**14**(0/2/1)	**19**(0/2/1)	**14**(1/1/1)	**17**(1/2/1)	**20**(1/2/1)
#<data>.L	**11**(0/1/1)	**14**(0/2/1)	**12**(0/1/1)	**16**(0/2/1)	**21**(0/2/1)	**16**(1/1/1)	**19**(1/2/1)	**22**(1/2/1)
(An)	**11**(1/1/1)	**14**(1/2/1)	**12**(1/1/1)	**16**(1/2/1)	**21**(1/2/1)	**12**(2/1/1)	**19**(2/2/1)	**22**(2/2/1)
(An)+	**11**(1/1/1)	**14**(1/2/1)	**12**(1/1/1)	**16**(1/2/1)	**21**(1/2/1)	**12**(2/1/1)	**19**(2/2/1)	**22**(2/2/1)
–(An)	**12**(1/1/1)	**15**(1/2/1)	**13**(1/1/1)	**17**(1/2/1)	**22**(1/2/1)	**13**(2/1/1)	**20**(2/2/1)	**23**(2/2/1)
(d_{16},An) or (d_{16},PC)	**13**(1/2/1)	**16**(1/3/1)	**14**(1/2/1)	**18**(1/3/1)	**23**(1/3/1)	**14**(2/2/1)	**21**(2/3/1)	**24**(2/3/1)
(xxx).W	**12**(1/2/1)	**15**(1/3/1)	**13**(1/2/1)	**17**(1/3/1)	**22**(1/3/1)	**13**(2/2/1)	**20**(2/3/1)	**23**(2/3/1)
(xxx).L	**14**(1/2/1)	**17**(1/3/1)	**15**(1/2/1)	**19**(1/3/1)	**24**(1/3/1)	**15**(2/2/1)	**22**(2/3/1)	**25**(2/3/1)
(d_8,An,Xn) or (d_8,PC,Xn)	**15**(1/2/1)	**18**(1/3/1)	**16**(1/2/1)	**20**(1/3/1)	**25**(1/3/1)	**16**(2/2/1)	**23**(2/3/1)	**26**(2/3/1)
(d_{16},An,Xn) or (d_{16},PC,Xn)	**16**(1/2/1)	**19**(1/3/1)	**17**(1/2/1)	**21**(1/3/1)	**26**(1/3/1)	**17**(2/2/1)	**24**(2/3/1)	**27**(2/3/1)
(B)	**16**(1/2/1)	**19**(1/3/1)	**17**(1/2/1)	**21**(1/3/1)	**26**(1/3/1)	**17**(2/2/1)	**24**(2/3/1)	**27**(2/3/1)
(d_{16},B)	**19**(1/2/1)	**22**(1/3/1)	**20**(1/2/1)	**24**(1/3/1)	**29**(1/3/1)	**20**(2/2/1)	**27**(2/3/1)	**30**(2/3/1)
(d_{32},B)	**23**(1/3/1)	**26**(1/4/1)	**24**(1/3/1)	**28**(1/4/1)	**33**(1/4/1)	**24**(2/3/1)	**31**(2/4/1)	**34**(2/4/1)
([B],I)	**20**(2/2/1)	**23**(2/3/1)	**21**(2/2/1)	**25**(2/3/1)	**30**(2/3/1)	**21**(3/2/1)	**28**(3/3/1)	**31**(3/3/1)
([B],I,d_{16})	**23**(2/2/1)	**26**(2/3/1)	**24**(2/2/1)	**28**(2/3/1)	**33**(2/3/1)	**24**(3/2/1)	**31**(3/3/1)	**34**(3/3/1)
([B],I,d_{32})	**24**(2/3/1)	**27**(2/4/1)	**25**(2/3/1)	**29**(2/4/1)	**34**(2/4/1)	**25**(3/3/1)	**32**(3/4/1)	**35**(3/4/1)
([d_{16},B],I)	**23**(2/2/1)	**26**(2/3/1)	**24**(2/2/1)	**28**(2/3/1)	**33**(2/3/1)	**24**(3/2/1)	**31**(3/3/1)	**34**(3/3/1)
([d_{16},B],I,d_{16})	**26**(2/3/1)	**29**(2/4/1)	**27**(2/3/1)	**31**(2/4/1)	**36**(2/4/1)	**27**(3/3/1)	**34**(3/4/1)	**37**(3/4/1)
([d_{16},B],I,d_{32})	**27**(2/3/1)	**30**(2/4/1)	**28**(2/3/1)	**32**(2/4/1)	**37**(2/4/1)	**28**(3/3/1)	**35**(3/4/1)	**38**(3/4/1)
([d_{32},B],I)	**27**(2/3/1)	**30**(2/4/1)	**28**(2/3/1)	**32**(2/4/1)	**37**(2/4/1)	**28**(3/3/1)	**35**(3/4/1)	**38**(3/4/1)
([d_{32},B],I,d_{16})	**29**(2/3/1)	**32**(2/4/1)	**30**(2/3/1)	**34**(2/4/1)	**39**(2/4/1)	**30**(3/3/1)	**37**(3/4/1)	**40**(3/4/1)
([d_{32},B],I,d_{32})	**31**(2/4/1)	**34**(2/5/1)	**32**(2/4/1)	**36**(2/5/1)	**41**(2/5/1)	**32**(3/4/1)	**39**(3/5/1)	**42**(3/5/1)

```
PAL16L8
FPCP CS GENERATION CIRCUITRY FOR 25 MHz OPERATION
MOTOROLA INC., AUSTIN, TEXAS
INPUTS:     CLK   ~AS   FC2   FC1   FC0   A19   A18   A17   A16   A15   A14   A13
OUTPUTS:    ~CS   CLKD

!~CS        = FC2          *FC1          *FC0          *!A16     ;cpu space = $7
            *!A19          *!A18         *A17          *!A16     ;coprocessor access = $2
            *!A15          *!A14         *A13                    ;coprocessor id = $1
            *!CLK                                                ;qualified by MPU clock low
            +FC2           *FC1          *FC0                    ;cpu space = $7
            *!A19          *!A18         *A17          *!A16     ;coprocessor access = $2
            *!A15          *!A14         *A13                    ;coprocessor id = $1
            *!~AS                                                ;qualified by address strobe low
            +FC2           *FC1          *FC0                    ;cpu space = $7
            *!A19          *!A18         *A17          *!A16     ;coprocessor access = $2
            *!A15          *!A14         *A13                    ;coprocessor id = $1
            *CLKD                                                ;qualified by CLKD (delayed CLK)

CLKD = CLK
```

Description: There are three terms to the CS generation. The first term denotes the earliest time CS can be asserted. The second term is used to assert CS until the end of the FPCP access. The third term is to ensure that no race condition occurs in case of a late AS.

Figure 9-3. Chip Select PAL Equations

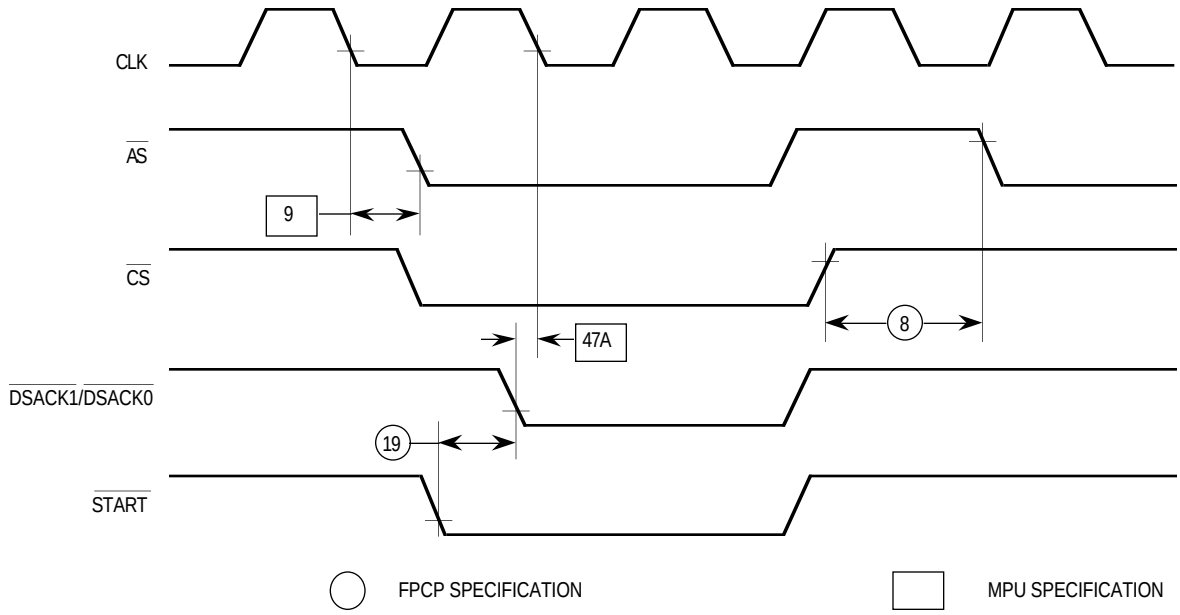


Figure 9-4. Bus Cycle Timing Diagram

Figure 9-9. Access Time Computation Diagram