Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

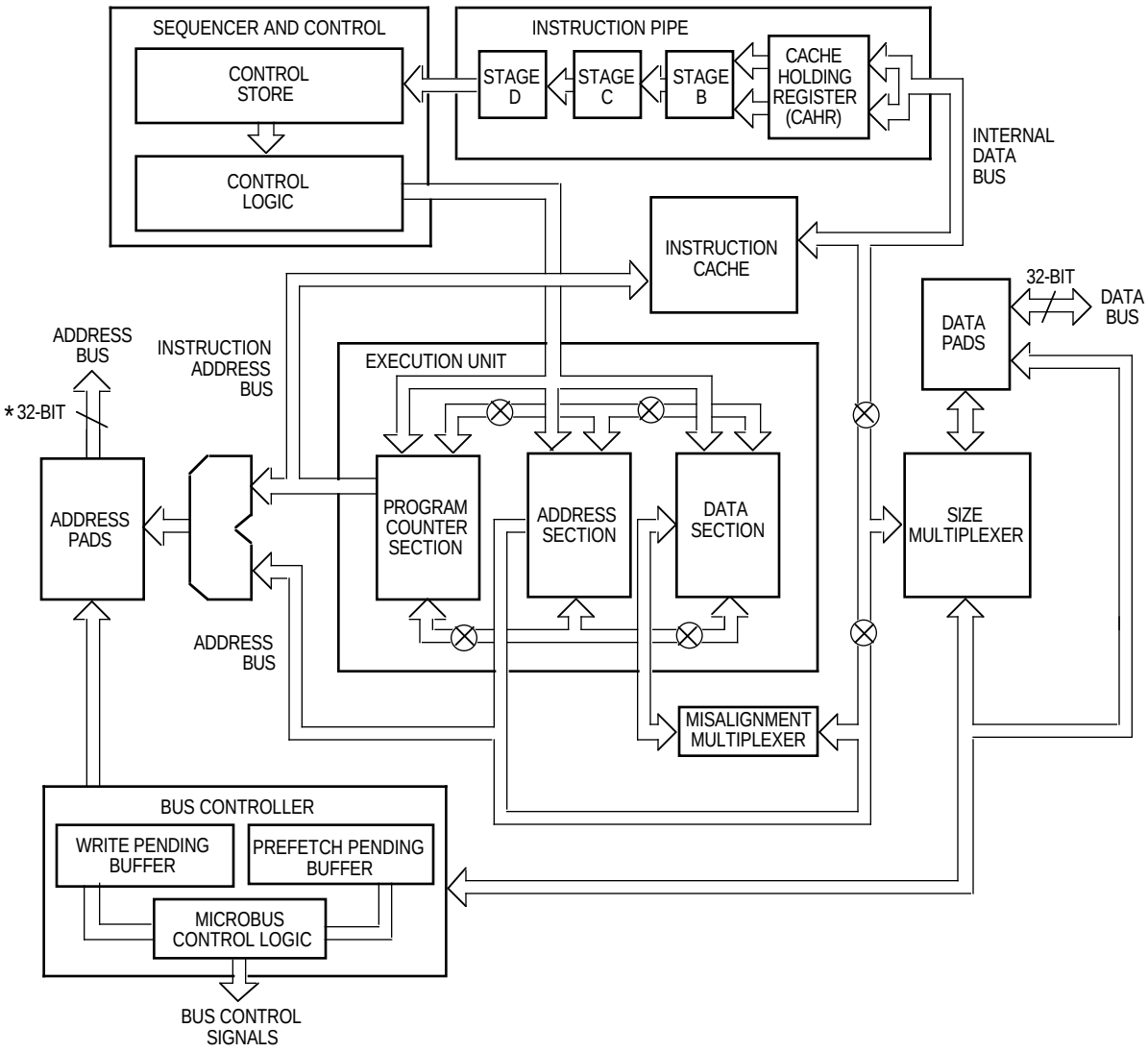
Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, 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	33MHz
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	132-BQFP Bumpered
Supplier Device Package	132-PQFP (46x46)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68020eh33e

TABLE OF CONTENTS



* 24-Bit for MC68EC020

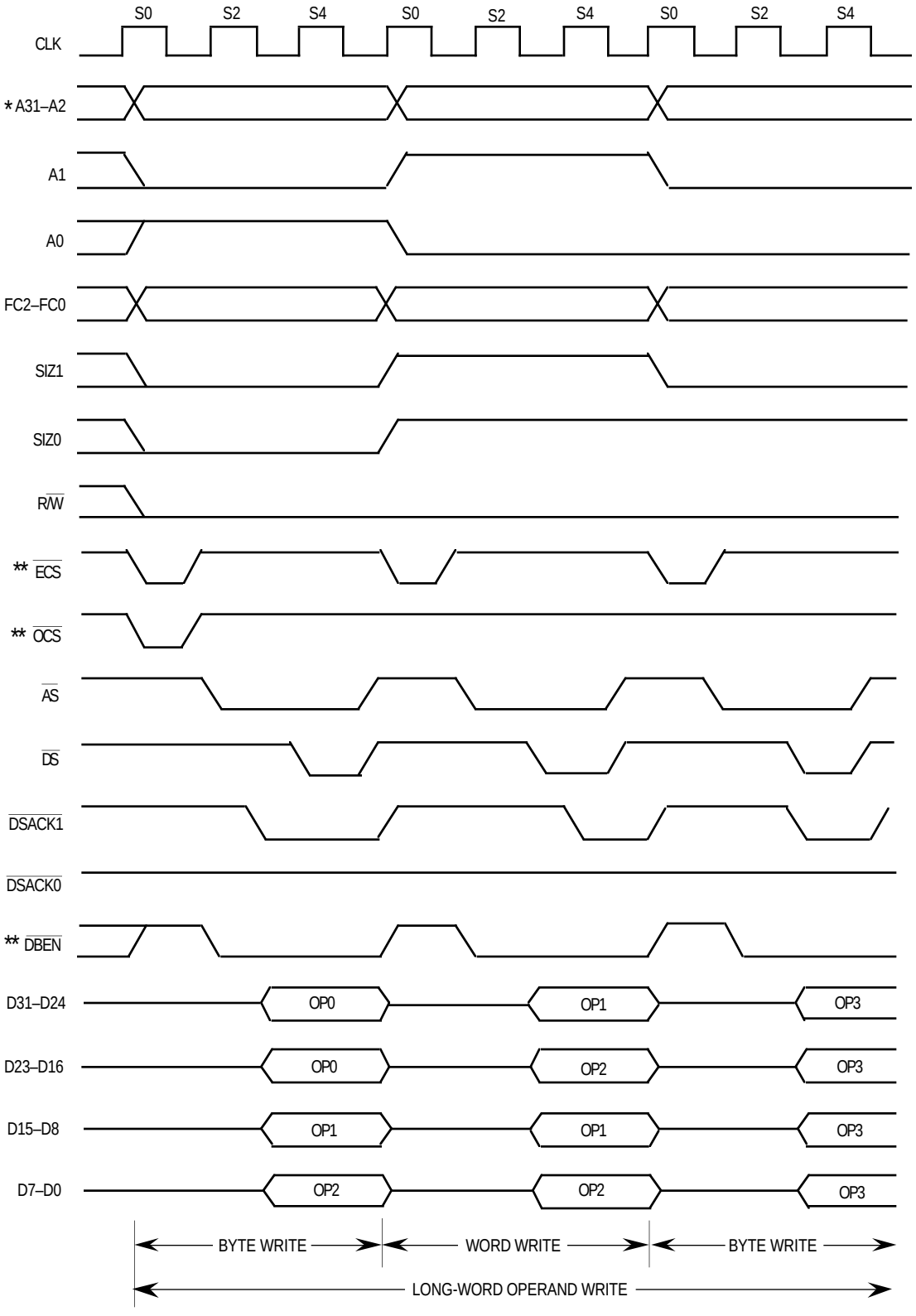
Figure 1-1. MC68020/EC020 Block Diagram

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

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

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

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



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

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

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

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

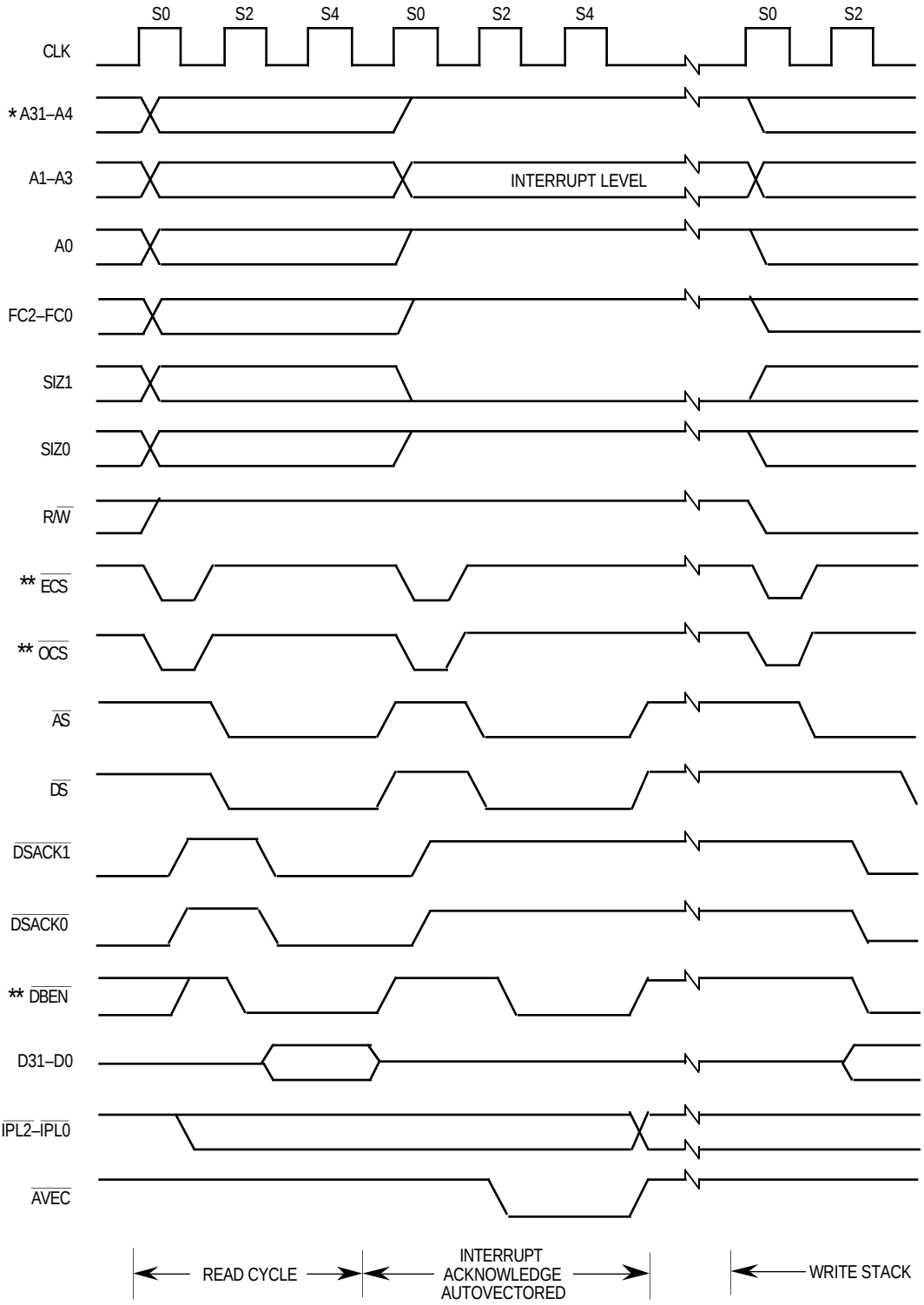
Figure 5-18 shows a logic diagram of one method for generating byte enable signals for 16- and 32-bit ports from the SIZ1, SIZ0, A1, and A0 encodings and the R/$\overline{W}$ signal.

5.2.5 Cache Interactions

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

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

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



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

Figure 5-34. Autovector Operation Timing

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

Normal Termination:

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

Halt Termination:

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

Bus Error Termination:

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

Retry Termination:

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

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

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

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

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

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

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

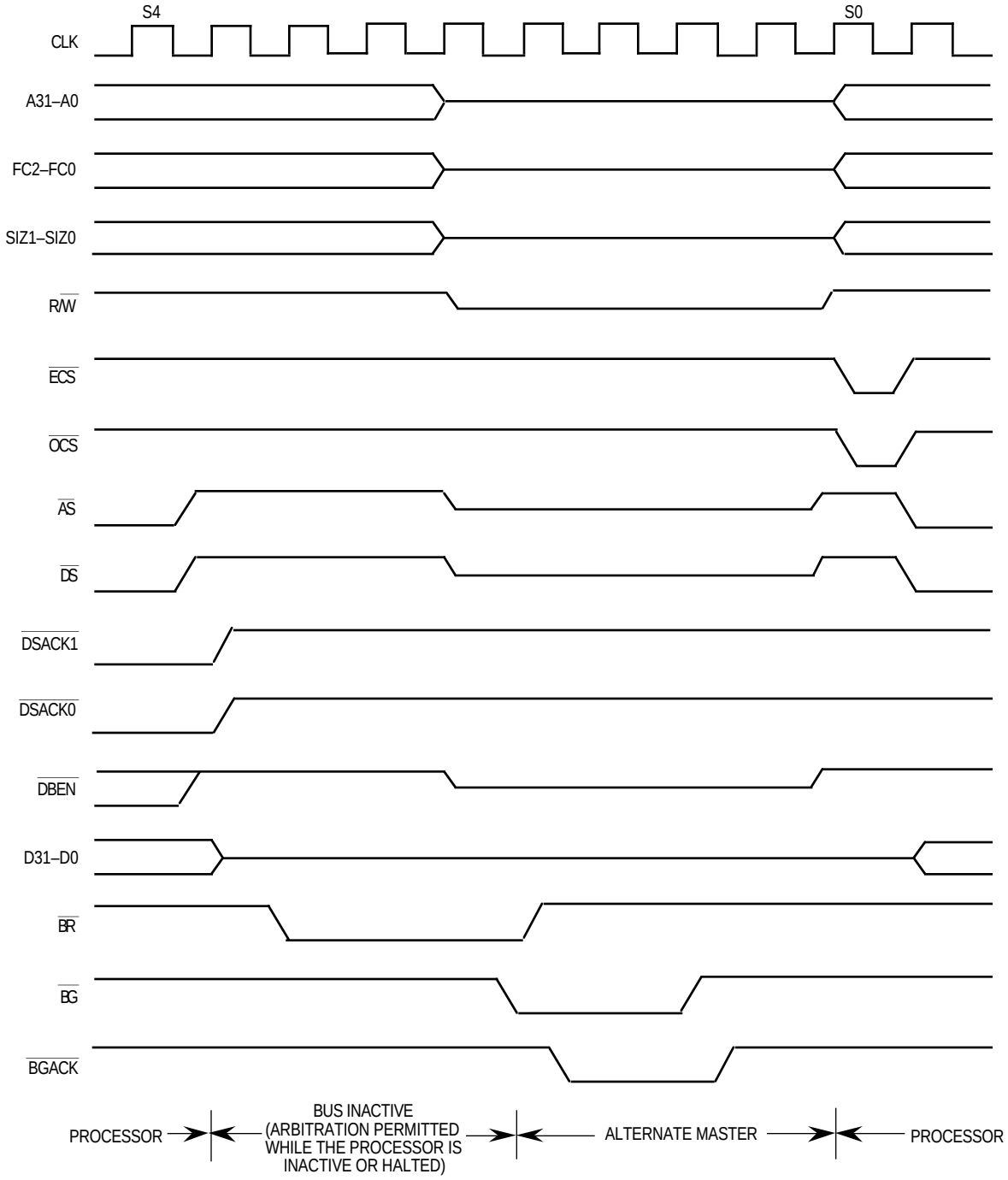


Figure 5-45. MC68020 Bus Arbitration Operation Timing—Bus Inactive

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

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

Bits 11–9—Reserved by Motorola

DF—Fault/Rerun Flag

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

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

RM—Read-Modify-Write

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

RW—Read/Write

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

SIZE—Size Code

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

Bit 3—Reserved by Motorola

FC2–FC0—Specifies the address space for data cycle

6.2.2 Using Software to Complete the Bus Cycles

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

information to the main processor during the execution of these instructions. These coprocessor format codes are discussed in detail in **7.2.3.2 Coprocessor Format Words**.

7.2.3.1 COPROCESSOR INTERNAL STATE FRAMES. The context save (cpSAVE) and context restore (cpRESTORE) instructions transfer an internal coprocessor state frame between memory and a coprocessor. This internal coprocessor state frame represents the state of coprocessor operations. Using the cpSAVE and cpRESTORE instructions, it is possible to interrupt coprocessor operation, save the context associated with the current operation, and initiate coprocessor operations with a new context.

A cpSAVE instruction stores a coprocessor internal state frame as a sequence of long-word entries in memory. Figure 7-14 shows the format of a coprocessor state frame. The format and length fields of the coprocessor state frame format comprise the format word. During execution of the cpSAVE instruction, the MC68020/EC020 calculates the state frame effective address from information in the operation word of the instruction and stores a format word at this effective address. The processor writes the long words that form the coprocessor state frame to descending memory addresses, beginning with the address specified by the sum of the effective address and the length field multiplied by four. During execution of the cpRESTORE instruction, the MC68020/EC020 reads the state frame from ascending addresses beginning with the effective address specified in the instruction operation word.

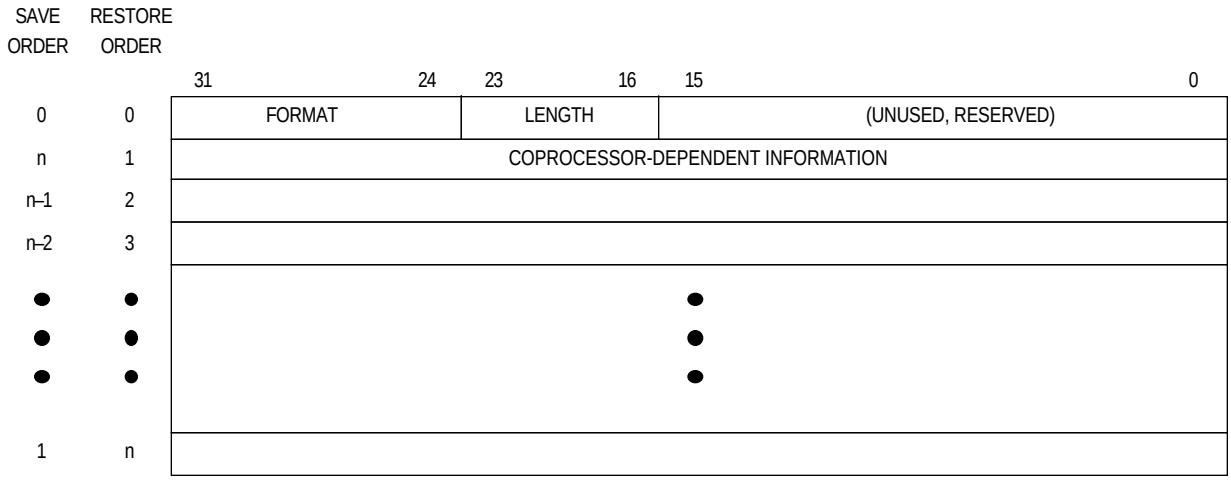


Figure 7-14. Coprocessor State Frame Format in Memory

The processor stores the coprocessor format word at the lowest address of the state frame in memory, and this word is the first word transferred for both the cpSAVE and cpRESTORE instructions. The word following the format word does not contain information relevant to the coprocessor state frame, but serves to keep the information in the state frame a multiple of four bytes in size. The number of entries following the format word (at higher addresses) is determined by the format word length for a given coprocessor state.

Figure 7-19. Control CIR Format

When the MC68020/EC020 receives one of the three take exception coprocessor response primitives, it acknowledges the primitive by setting the exception acknowledge bit (XA) in the control CIR. The MC68020/EC020 sets the abort bit (AB) in the control CIR to abort any coprocessor instruction in progress. (The 14 most significant bits of both masks are undefined.) The MC68020/EC020 aborts a coprocessor instruction when it detects one of the following exception conditions:

- An F-line emulator exception condition after reading a response primitive
- A privilege violation exception as it performs a supervisor check in response to a supervisor check primitive
- A format error exception when it receives an invalid format word or a valid format word that contains an invalid length

7.3.3 Save CIR

The coprocessor uses the 16-bit save CIR to communicate status and state frame format information to the main processor while executing a cpSAVE instruction. The main processor reads the save CIR to initiate execution of the cpSAVE instruction by the coprocessor. The offset from the base address of the CIR set for the save CIR is $04. Refer to **7.2.3.2 Coprocessor Format Words** for more information on the save CIR.

7.3.4 Restore CIR

The main processor initiates the cpRESTORE instruction by writing a coprocessor format word to the 16-bit restore register. During the execution of the cpRESTORE instruction, the coprocessor communicates status and state frame format information to the main processor through the restore CIR. The offset from the base address of the CIR set for the restore CIR is $06. Refer to **7.2.3.2 Coprocessor Format Words** for more information on the restore CIR.

7.3.5 Operation Word CIR

The main processor writes the F-line operation word of the instruction in progress to the 16-bit operation word CIR in response to a transfer operation word coprocessor response primitive (refer to **7.4.6 Transfer Operation Word Primitive**). The offset from the base address of the CIR set for the operation word CIR is $08.

7.3.6 Command CIR

The main processor initiates a coprocessor general category instruction by writing the instruction command word, which follows the instruction F-line operation word in the instruction stream, to the 16-bit command CIR. The offset from the base address of the CIR set for the command CIR is $0A.

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

Table 7-6. Exceptions Related to Primitive Processing

Primitive	Protocol	F-Line	Other
Busy			
Null			
Supervisory Check* Other: Privilege Violation if S-Bit in the SR = 0			X
Transfer Operation Word*			
Transfer from Instruction Stream* Protocol: If Length Field Is Odd (Zero Length Legal)	X		
Evaluate and Transfer Effective Address Protocol: If Used with Conditional Instruction F-Line: If EA in Opword Is NOT Control Alterable	X	X	
Evaluate Effective Address and Transfer Data Protocol: 1. If Used with Conditional Instructions 2. Length Is Not 1, 2, or 4 and EA = Register Direct 3. If EA = Immediate and Length Odd and Greater Than 1 4. Attempt to Write to Unalterable Address Even if Address Declared Legal in Primitive F-Line: Valid EA Field Does Not Match EA in Opword	X	X	
Write to Previously Evaluated Effective Address Protocol: If Used with Conditional Instruction	X		
Take Address and Transfer Data*			
Transfer to/from Top of Stack* Protocol: Length Field Other Than 1, 2, or 4	X		
Transfer Single Main Processor Register*			
Transfer Main Processor Control Register Protocol: Invalid Control Register Select Code	X		
Transfer Multiple Main Processor Registers*			
Transfer Multiple Coprocessor Registers Protocol: 1. If Used with Conditional Instructions 2. Odd Length Value F-Line: 1. EA Not Control Alterable or (An)+ for CP to Memory Transfer 2. EA Not Control Alterable or –(An) for Memory to CP Transfer	X	X	
Transfer Status and ScanPC Protocol: If Used with Conditional Instruction Other: 1. Trace—Trace Made Pending if MC68020/EC020 in “Trace on Change of Flow” Mode and DR = 1 2. Address Error—If Odd Value Written to ScanPC	X		X
Take Preinstruction, Midinstruction, or Postinstruction Exception Exception Depends on Vector Supplies in Primitive	X	X	X

*Use of this primitive with CA = 0 will cause protocol violation on conditional instructions.

Abbreviations:

EA—Effective Address
CP—Coprocessor

counter field of the saved stack frame to point to the next instruction operation word and executes the RTE instruction. The MC68020/EC020 then executes the instruction following the instruction that was emulated.

The exception handler should also check the copy of the SR on the stack to determine whether tracing is enabled. If tracing is enabled, the trace exception processing should also be emulated. Refer to **Section 6 Exception Processing** for additional information.

7.5.2.3 PRIVILEGE VIOLATIONS. Privilege violations can result from the cpSAVE and cpRESTORE instructions and from the supervisor check coprocessor response primitive. The MC68020/EC020 initiates privilege violation exception processing if it attempts to execute either the cpSAVE or cpRESTORE instruction when it is in the user state (S = 0 in the SR). The main processor initiates this exception processing prior to any communication with the coprocessor associated with the cpSAVE or cpRESTORE instructions.

If the main processor is executing a coprocessor instruction in the user state when it reads the supervisor check primitive, it aborts the coprocessor instruction in progress by writing an abort mask to the control CIR (refer to **7.3.2 Control CIR**). The main processor then performs privilege violation exception processing.

If a privilege violation occurs, the main processor initiates exception processing using the four-word preinstruction stack frame (refer to Figure 7-41) and the privilege violation exception vector number 8. Thus, if the exception handler does not modify the stack frame, the main processor attempts to restart the instruction during which the exception occurred after it executes an RTE to return from the handler.

7.5.2.4 cpTRAPcc INSTRUCTION TRAPS. If, during the execution of a cpTRAPcc instruction, the coprocessor returns the TRUE condition indicator to the main processor with a null primitive, the main processor initiates trap exception processing. The main processor uses the six-word postinstruction exception stack frame (refer to Figure 7-45) and the trap exception vector number 7. The scanPC field of this stack frame contains the address of the instruction following the cpTRAPcc instruction. The processing associated with the cpTRAPcc instruction can then proceed, and the exception handler can locate any immediate operand words encoded in the cpTRAPcc instruction using the information contained in the six-word stack frame. If the exception handler does not modify the stack frame, the main processor executes the instruction following the cpTRAPcc instruction after it executes an RTE instruction to exit from the handler.

7.5.2.5 TRACE EXCEPTIONS. The MC68020/EC020 supports two modes of instruction tracing, as discussed in **Section 6 Exception Processing**. In the trace on instruction execution mode, the MC68020/EC020 takes a trace exception after completing each instruction. In the trace on change of flow mode, the MC68020/EC020 takes a trace exception after each instruction that alters the SR or places an address other than the address of the next instruction in the PC.

8.2.10 Binary-Coded Decimal Operations

The binary-coded decimal operations table indicates the number of clock periods needed for the processor to perform the specified operation using the given addressing modes, with complete execution times given. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction		Best Case	Cache Case	Worst Case
ABCD	Dn,Dn	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
ABCD	–(An),–(An)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/1/1)
SBCD	Dn,Dn	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
SBCD	–(An),–(An)	**14**(2/0/1)	**16**(2/0/1)	**17**(2/1/1)
ADDX	Dn,Dn	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
ADDX	–(An),–(An)	**10**(2/0/1)	**12**(2/0/1)	**13**(2/1/1)
SUBX	Dn,Dn	**2**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
SUBX	–(An),–(An)	**10**(2/0/1)	**12**(2/0/1)	**13**(2/1/1)
CMPM	(An)+,(An)+	**8**(2/0/0)	**9**(2/0/0)	**10**(2/1/0)
PACK	Dn,Dn,#<data>	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
PACK	–(An),–(An),#<data>	**11**(1/0/1)	**13**(1/0/1)	**13**(1/1/1)
UNPK	Dn,Dn,#<data>	**5**(0/0/0)	**8**(0/0/0)	**9**(0/1/0)
UNPK	–(An),–(An),#<data>	**11**(1/0/1)	**13**(1/0/1)	**13**(1/1/1)

8.2.13 Bit Manipulation Instructions

The bit manipulation instructions table indicates the number of clock periods needed for the processor to perform the specified bit operation on the given addressing mode. Footnotes indicate when it is necessary to add another table entry to calculate the total effective execution time for the instruction. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction			Best Case	Cache Case	Worst Case
	BTST	#<data>,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BTST	Dn,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
**	BTST	#<data>,Mem	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
*	BTST	Dn,Mem	**4**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BCHG	#<data>,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BCHG	Dn,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
**	BCHG	#<data>,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
*	BCHG	Dn,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
	BCLR	#<data>,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BCLR	Dn,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
**	BCLR	#<data>,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
*	BCLR	Dn,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
	BSET	#<data>,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
	BSET	Dn,Dn	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
**	BSET	#<data>,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)
*	BSET	Dn,Mem	**4**(0/0/1)	**4**(0/0/1)	**5**(0/1/1)

*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

8.2.15 Conditional Branch Instructions

The conditional branch instructions table indicates the number of clock periods needed for the processor to perform the specified branch on the given branch size, with complete execution times given. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses; the number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). These cycles are included in the total clock cycle number.

Instruction	Best Case	Cache Case	Worst Case
Bcc (Taken)	**3**(0/0/0)	**6**(0/0/0)	**9**(0/2/0)
Bcc.B (Not Taken)	**1**(0/0/0)	**4**(0/0/0)	**5**(0/1/0)
Bcc.W (Not Taken)	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)
Bcc.L (Not Taken)	**3**(0/0/0)	**6**(0/0/0)	**9**(0/2/0)
DBcc (cc = False, Count Not Expired)	**3**(0/0/0)	**6**(0/0/0)	**9**(0/2/0)
DBcc (cc = False, Count Expired)	**7**(0/0/0)	**10**(0/0/0)	**10**(0/3/0)
DBcc (cc = True)	**3**(0/0/0)	**6**(0/0/0)	**7**(0/1/0)

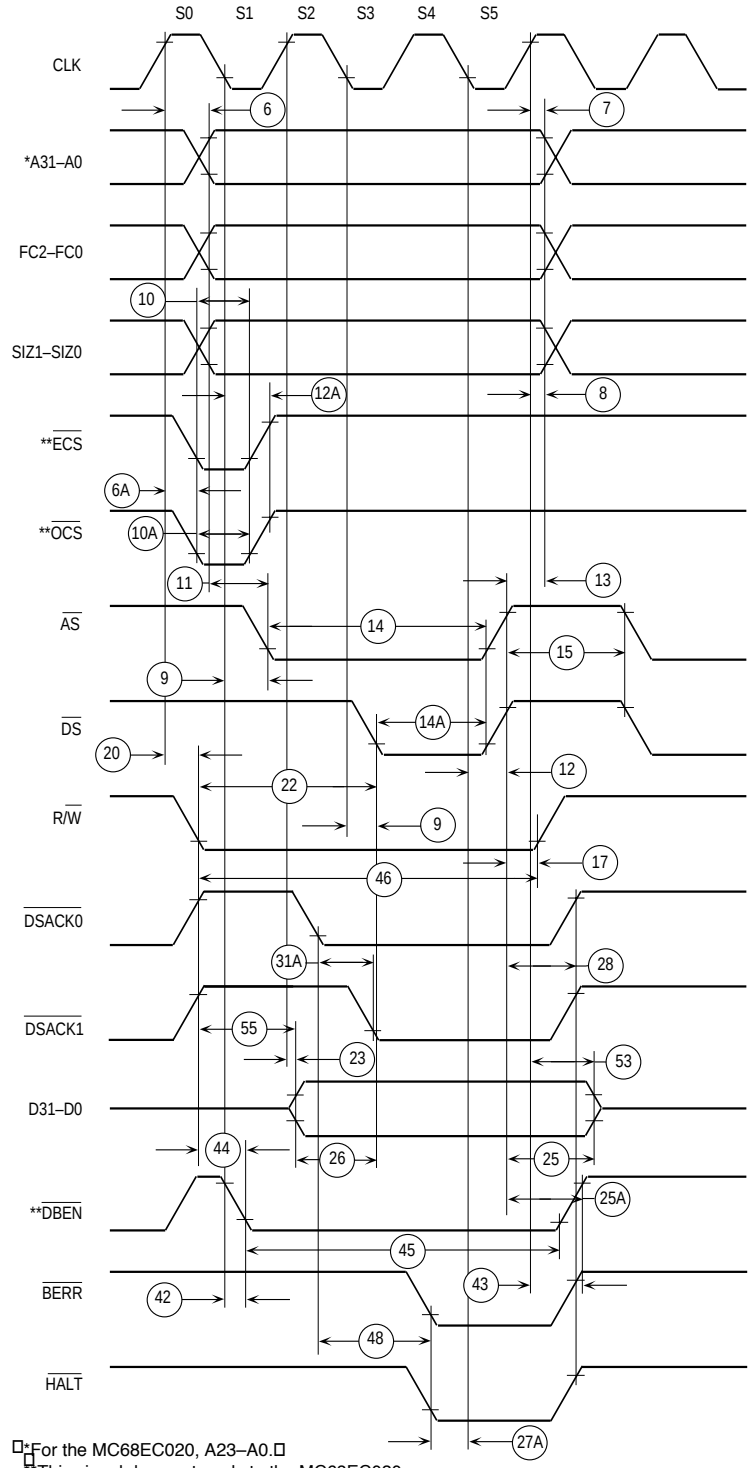
```
PAL16L8
BYTE_SELECT
MC68020/EC020 BYTE DATA SELECT GENERATION FOR 32-BIT PORTS, MAPPED AND UNMAPPED.
MOTOROLA INC., AUSTIN, TEXAS
INPUTS:     A0      A1      SIZ0    SIZ1    RW      A18     A19     A20     A21     ~CPU
OUTPUTS:    ~UUDA       ~UMDA       ~LMDA       ~LLDA       ~UUDA       ~UMDB       ~LMDB       ~LLDB

!~UUDA      = RW                                    ;enable upper byte on read of 32-bit port
            +!A0 *!A1                               ;directly addressed, any size
!~UMDA      = RW                                    ;enable upper middle byte on read of 32-bit port
            +A0 *!A1                                ;directly addressed, any size
            +!A1 *!SIZ0                             ;even word aligned, size word or long word
            +!A1 *SIZ1                              ;even word aligned, size is word or three byte
!~LMDA      = RW                                    ;enable lower middle byte on read of 32-bit port    +!A0 *A1
            ;directly addressed, any size
            +!A1 *!SIZ0 *!SIZ1                      ;even word aligned, size is long word
            +!A1 *SIZ0 *SIZ1                        ;even word aligned, size is three byte
            +!A1 *A0 *!SIZ0                         ;even word aligned, size is word or long word
!~LLDA      = RW                                    ;enable lower byte on read of 32-bit port
            +A0 *A1                                 ;directly addressed, any size
            +A0 *SIZ0 *SIZ1                         ;odd byte alignment, three byte size
            +!SIZ0 *!SIZ1                           ;size is long word, any address
            +A1 *SIZ1                               ;odd word aligned, word or three byte size

!~UUDB      = RW *!~CPU * (addressb)                ;enable upper byte on read of 32-bit port
            +!A0 *!A1 *!~CPU * (addressb)           ;directly addressed, any size
!~UMDB      = RW *!~CPU * (addressb)                ;enable upper middle byte on read of 32-bit port
            + A0 *!A1 *!~CPU * (addressb)           ;directly addressed, any size
            +!A1 *!SIZ0 *!~CPU * (addressb)         ;even word aligned, size word or long word
            +!A1 *SIZ1 *!~CPU * (addressb)          ;even word aligned, size is word or three byte
!~LMDB      =RW *!~CPU * (addressb)                 ;enable lower middle byte on read of 32-bit port
            +!A0 * A1 *!~CPU * (addressb)           ;directly addressed, any size
            +!A1 *!SIZ0 *!SIZ1 *!~CPU * (addressb)  ;even word aligned, size is long word
            +!A1 * SIZ0 * SIZ1 *!~CPU * (addressb)  ;even word aligned, size is three byte
            +!A1 * A0 *!SIZ0 *!~CPU * (addressb)    ;even word aligned, size is word or long word
!~LLDB      =RW *!~CPU * (addressb)                 ;enable lower byte on read of 32-bit port
            +A0 * A1 *!~CPU * (addressb)            ;directly addressed, any size
            + A0 * SIZ0 * SIZ1 *!~CPU * (addressb)  ;odd byte alignment, three byte size
            +!SIZ0 *!SIZ1 *!~CPU * (addressb)       ;size is long word, any address
            +A1 * SIZ1 *!~CPU * (addressb)          ;odd word aligned, word or three byte size
```

DESCRIPTION: Byte select signals for writing. On reads, all byte selects are asserted if the respective memory block is addressed. The input signal CPU prevents byte select assertion during CPU space cycles and is derived from NANDing FC1–FC0 or FC2–FC0. The label (addressb) is a designer-selectable combination of address lines used to generate the proper address decode for the system's memory bank. With the address lines given here, the decode block size is 256 Kbytes to 2 Mbytes. A similar address might be included in the equations for UUDA, UMDA, etc. if the designer wishes them to be memory mapped also.

Figure 9-6. MC68020/EC020 Byte Select PAL Equations



*For the MC68EC020, A23–A0.

**This signal does not apply to the MC68EC020.

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 V and 2.0 V.

FIGURE 10-4
MC68020UM

Figure 10-4. Write Cycle Timing Diagram