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	166MHz
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	128-BPGA
Supplier Device Package	128-PGA (34.55x34.55)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68030rc16c

1.6 VIRTUAL MEMORY AND VIRTUAL MACHINE CONCEPTS

The full addressing range of the MC68030 is 4 Gbytes (4,294,967,296 bytes) in each of eight address spaces. Even though most systems implement a smaller physical memory, the system can be made to appear to have a full 4 Gbytes of memory available to each user program by using virtual memory techniques.

In a virtual memory system, a user program can be written as if it has a large amount of memory available, when the physical memory actually present is much smaller. Similarly, a system can be designed to allow user programs to access devices that are not physically present in the system, such as tape drives, disk drives, printers, terminals, and so forth. With proper software emulation, a physical system can appear to be any other M68000 computer system to a user program, and the program can be given full access to all of the resources of that emulated system. Such an emulated system is called a virtual machine.

1.6.1 Virtual Memory

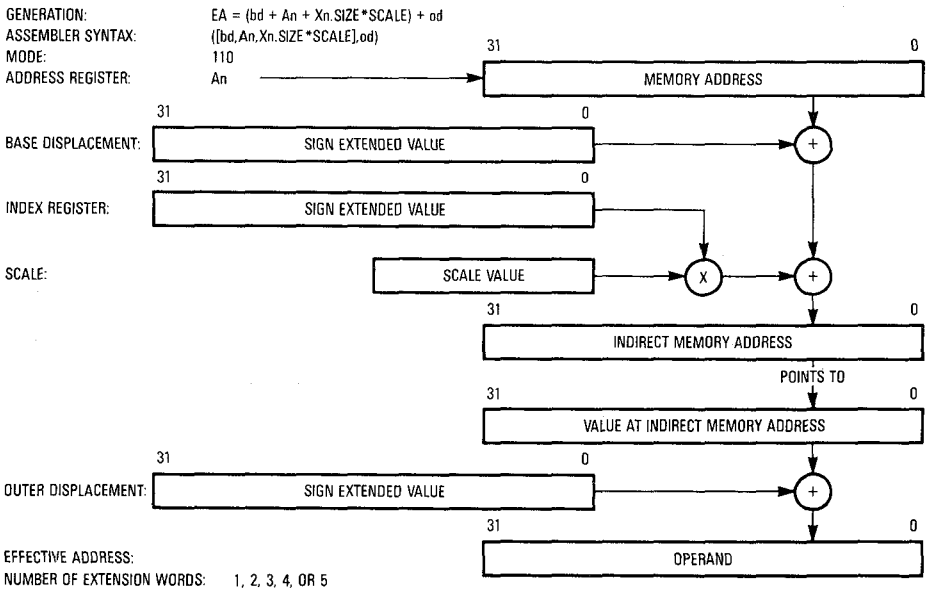
A system that supports virtual memory has a limited amount of high-speed physical memory that can be accessed directly by the processor and maintains an image of a much larger virtual memory on a secondary storage device such as a large-capacity disk drive. When the processor attempts to access a location in the virtual memory map that is not resident in physical memory, a page fault occurs. The access to that location is temporarily suspended while the necessary data is fetched from secondary storage and placed in physical memory. The suspended access is then either restarted or continued.

The MC68030 uses instruction continuation to support virtual memory. When a bus cycle is terminated with a bus error, the microprocessor suspends the current instruction and executes the virtual memory bus error handler. When the bus error handler has completed execution, it returns control to the program that was executing when the error was detected, reruns the faulted bus cycle (when required), and continues the suspended instruction.

2.4.10 Memory Indirect Preindexed Mode

In this mode, the operand and its address are in memory. The processor calculates an intermediate indirect memory address using the base register (An), a base displacement (bd), and the index operand (Xn.SIZE * SCALE). The processor accesses a long word at this address and adds the outer displacement to yield the effective address. Both displacements and the index register contents are sign-extended to 32 bits.

In the syntax for this mode, brackets enclose the values used to calculate the intermediate memory address. All four user-specified values are optional. Both the base and outer displacements may be null, word, or long word. When a displacement is omitted or an element is suppressed, its value is taken as zero in the effective address calculation.



For both the MC68020 and the MC68030, the register indirect modes can be extended further. Since displacements can be 32 bits wide, they can represent absolute addresses or the results of expressions that contain absolute addresses. This allows the general register indirect form to be (bd,Rn) or (bd,An,Rn) when the base register is not suppressed. Thus, an absolute address can be directly indexed by one or two registers (refer to Figure 2-6).

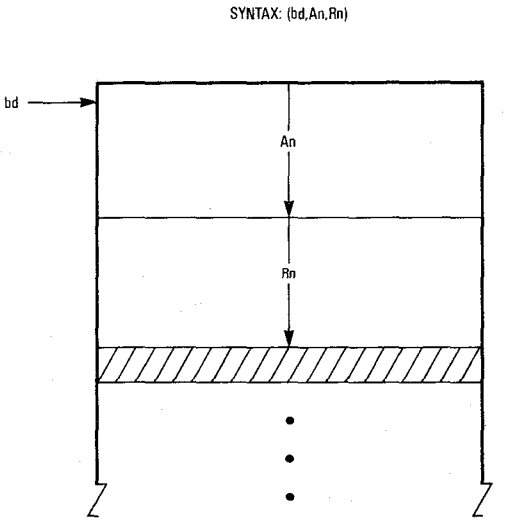


Figure 2-6. Using Absolute Address with Indexes

Scaling provides an optional shifting of the value in an index register to the left by zero, one, two, or three bits before using it in the effective address calculation (the actual value in the index register remains unchanged). This is equivalent to multiplying the register by one, two, four, or eight for direct subscripting into an array of elements of corresponding size using an arithmetic value residing in any of the 16 general registers. Scaling does not add to the effective address calculation time. However, when combined with the appropriate derived modes, it produces additional capabilities. Arrayed structures can be addressed absolutely and then subscripted, (bd,Rn*scale), for example. Optionally, an address register that contains a dynamic displacement can be included in the address calculation (bd,An,Rn*scale). Another variation that can be derived is (An,Rn*scale). In the first case, the array address is the sum of the contents of a register and a displacement, as shown in Figure 2-7. In the second example, An contains the address of an array and Rn contains a subscript.

X = extend (X) bit in CCR
N = negative (N) bit in CCR
Z = Zero (Z) bit in CCR
V = overflow (V) bit in CCR
C = carry (C) bit in CCR
+ = arithmetic addition or postincrement indicator
− = arithmetic subtraction or predecrement indicator
× = arithmetic multiplication
÷ = arithmetic division or conjunction symbol
~ = invert; operand is logically complemented
Λ = logical AND
V = logical OR
⊕ = logical exclusive OR
Dc = data register, D7–D0 used during compare
Du = data register, D7–D0 used during update
Dr, Dq = data registers, remainder or quotient of divide
Dh, Dl = data registers, high- or low-order 32 bits of product
MSW = most significant word
LSW = least significant word
MSB = most significant bit
FC = function code
{R/W} = read or write indicator
[An] = address extensions

3.2.1 Data Movement Instructions

The MOVE instructions with their associated addressing modes are the basic means of transferring and storing addresses and data. MOVE instructions transfer byte, word, and long-word operands from memory to memory, memory to register, register to memory, and register to register. Address movement instructions (MOVE or MOVEA) transfer word and long-word operands and ensure that only valid address manipulations are executed. In addition to the general MOVE instructions, there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), move quick (MOVEQ), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), and unlink stack (UNLK).

3.3.1 Condition Code Computation

Most operations take a source operand and a destination operand, compute, and store the result in the destination location. Single-operand operations take a destination operand, compute, and store the result in the destination location. Table 3-12 lists each instruction and how it affects the condition code bits.

Table 3-12. Condition Code Computations (Sheet 1 of 2)

Operations	X	N	Z	V	C	Special Definition
ABCD	*	U	?	U	?	C = Decimal Carry Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	V = Sm ∧ Dm ∧ $\overline{Rm}$ ∨ $\overline{Sm}$ ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ Dm ∨ $\overline{Rm}$ ∧ Dm ∨ Sm ∧ $\overline{Rm}$
ADDX	*	*	?	?	?	V = Sm ∧ Dm ∧ $\overline{Rm}$ ∨ $\overline{Sm}$ ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ Dm ∨ $\overline{Rm}$ ∧ Dm ∨ Sm ∧ $\overline{Rm}$ Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	—	*	*	0	0	
CHK	—	*	U	U	U	
CHK2, CMP2	—	U	?	U	?	Z = (R = LB) ∨ (R = UB) C = (LB < = UB) ∧ (IR < LB) ∨ (R > UB)) ∨ (UB < LB) ∧ (R > UB) ∧ (R < LB)
SUB, SUBI, SUBQ	*	*	*	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm
SUBX	*	*	?	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{R0}$
CAS, CAS2, CMP, CMPI, CMPM	—	*	*	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm
DIVS, DUVI	—	*	*	?	0	V = Division Overflow
MULS, MULU	—	*	*	?	0	V = Multiplication Overflow
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{Ro}$
NEG	*	*	*	?	?	V = Dm ∧ Rm C = Dm ∨ Rm
NEGX	*	*	?	?	?	V = Dm ∧ Rm C = Dm ∨ Rm Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{R0}$

5.2 FUNCTION CODE SIGNALS (FC0–FC2)

These three-state outputs identify the address space of the current bus cycle. Table 4-1 shows the relationship of the function code signals to the privilege levels and the address spaces. Refer to **4.2 ADDRESS SPACE TYPES** for more information.

5.3 ADDRESS BUS (A0–A31)

These three-state outputs provide the address for the current bus cycle, except in the CPU address space. Refer to **4.2 ADDRESS SPACE TYPES** for more information on the CPU address space. A31 is the most significant address signal. Refer to **7.1.2 Address Bus** for information on the address bus and its relationship to bus operation.

5

5.4 DATA BUS (D0–D31)

These three-state bidirectional signals provide the general-purpose data path between the MC68030 and all other devices. The data bus can transfer 8, 16, 24, or 32 bits of data per bus cycle. D31 is the most significant bit of the data bus. Refer to **7.1.4 Data Bus** for more information on the data bus and its relationship to bus operation.

5.5 TRANSFER SIZE SIGNALS (SIZ0, SIZ1)

These three-state outputs indicate the number of bytes remaining to be transferred for the current bus cycle. With A0, A1, $\overline{\text{DSACK0}}$, $\overline{\text{DSACK1}}$, and $\overline{\text{STERM}}$, SIZ0 and SIZ1 define the number of bits transferred on the data bus. Refer to **7.2.1 Dynamic Bus Sizing** for more information on the size signals and their use in dynamic bus sizing.

5.7 CACHE CONTROL SIGNALS

The following signals relate to the on-chip caches.

5.7.1 Cache Inhibit Input ($\overline{\text{CIIN}}$)

This input signal prevents data from being loaded into the MC68030 instruction and data caches. It is a synchronous input signal and is interpreted on a bus-cycle-by-bus-cycle basis. $\overline{\text{CIIN}}$ is ignored during all write cycles. Refer to **6.1 ON-CHIP CACHE ORGANIZATION AND OPERATION** for information on the relationship of $\overline{\text{CIIN}}$ to the on-chip caches.

5.7.2 Cache Inhibit Output ($\overline{\text{CIOUT}}$)

This three-state output signal reflects the state of the CI bit in the address translation cache entry for the referenced logical address, indicating that an external cache should ignore the bus transfer. When the referenced logical address is within an area specified for transparent translation, the CI bit of the appropriate transparent translation register controls the state of $\overline{\text{CIOUT}}$. Refer to **SECTION 9 MEMORY MANAGEMENT UNIT** for more information about the address translation cache and transparent translation. Also, refer to **SECTION 6 ON-CHIP CACHE MEMORIES** for the effect of $\overline{\text{CIOUT}}$ on the internal caches.

5.7.3 Cache Burst Request ($\overline{\text{CBREQ}}$)

This three-state output signal requests a burst mode operation to fill a line in the instruction or data cache. Refer to **6.1.3 Cache Filling** for filling information and **7.3.7 Burst Operation Cycles** for bus cycle information pertaining to burst mode operations.

5.7.4 Cache Burst Acknowledge ($\overline{\text{CBACK}}$)

This input signal indicates that the accessed device can operate in the burst mode and can supply at least one more long word for the instruction or data cache. Refer to **7.3.7 Burst Operation Cycles** for information about burst mode operation.

Table 5-2. Signal Summary

Signal Function	Signal Name	Input/Output	Active State	Three-State
Function Codes	FC0–FC2	Output	High	Yes
Address Bus	A0–A31	Output	High	Yes
Data Bus	D0–D31	Input/Output	High	Yes
Transfer Size	SIZ0/SIZ1	Output	High	Yes
Operand Cycle Start	$\overline{\text{OCS}}$	Output	Low	No
External Cycle Start	$\overline{\text{ECS}}$	Output	Low	No
Read/Write	R/$\overline{\text{W}}$	Output	High/Low	Yes
Read-Modify-Write Cycle	$\overline{\text{RMC}}$	Output	Low	Yes
Address Strobe	$\overline{\text{AS}}$	Output	Low	Yes
Data Strobe	$\overline{\text{DS}}$	Output	Low	Yes
Data Buffer Enable	$\overline{\text{DBEN}}$	Output	Low	Yes
Data Transfer and Size Acknowledge	$\overline{\text{DSACK0}}$/ $\overline{\text{DSACK1}}$	Input	Low	—
Synchronous Termination	$\overline{\text{STERM}}$	Input	Low	—
Cache Inhibit In	$\overline{\text{CIIN}}$	Input	Low	—
Cache Inhibit Out	$\overline{\text{CIOUT}}$	Output	Low	Yes
Cache Burst Request	$\overline{\text{CBREQ}}$	Output	Low	Yes
Cache Burst Acknowledge	$\overline{\text{CBACK}}$	Input	Low	—
Interrupt Priority Level	$\overline{\text{IPL0}}$–$\overline{\text{IPL2}}$	Input	Low	—
Interrupt Pending	$\overline{\text{IPEND}}$	Output	Low	No
Autovector	$\overline{\text{AVEC}}$	Input	Low	—
Bus Request	$\overline{\text{BR}}$	Input	Low	—
Bus Grant	$\overline{\text{BG}}$	Output	Low	No
Bus Grant Acknowledge	$\overline{\text{BGACK}}$	Input	Low	—
Reset	$\overline{\text{RESET}}$	Input/Output	Low	No
Halt	$\overline{\text{HALT}}$	Input	Low	—
Bus Error	$\overline{\text{BERR}}$	Input	Low	—
Cache Disable	$\overline{\text{CDIS}}$	Input	Low	—
MMU Disable	$\overline{\text{MMUDIS}}$	Input	Low	—
Pipeline Refill	$\overline{\text{REFILL}}$	Output	Low	No
Microsequencer Status	$\overline{\text{STATUS}}$	Output	Low	No
Clock	CLK	Input	—	—
Power Supply	V_{CC}	Input	—	—
Ground	GND	Input	—	—

address bus that specifies the address for the transfer and a data bus that transfers the data. Control signals indicate the beginning of the cycle, the address space and the size of the transfer, and the type of cycle. The selected device then controls the length of the cycle with the signal(s) used to terminate the cycle. Strobe signals, one for the address bus and another for the data bus, indicate the validity of the address and provide timing information for the data.

The bus can operate in an asynchronous mode identical to the MC68020 bus for any port width. The bus and control input signals used for asynchronous operation are internally synchronized to the MC68030 clock, introducing a delay. This delay is the time period required for the MC68030 to sample an asynchronous input signal, synchronize the input to the internal clocks of the processor, and determine whether it is high or low. Figure 7-1 shows the relationship between the clock signal and the associated internal signal of a typical asynchronous input.

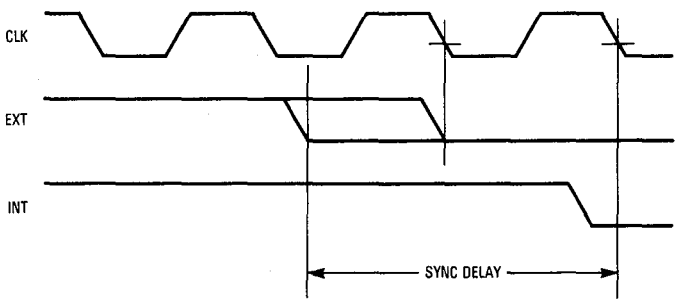


Figure 7-1. Relationship between External and Internal Signals

Furthermore, for all asynchronous inputs, the processor latches the level of the input during a sample window around the falling edge of the clock signal. This window is illustrated in Figure 7-2. To ensure that an input signal is recognized on a specific falling edge of the clock, that input must be stable during the sample window. If an input makes a transition during the window time period, the level recognized by the processor is not predictable; however, the processor always resolves the latched level to either a logic high or low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

terminates the bus cycle. It then starts a new bus cycle with SIZ0_SIZ1_A0_A1 = 1010 to transfer the remaining 16 bits. SIZ0 and SIZ1 indicate that a word remains to be transferred; A0 and A1 indicate that the word corresponds to an offset of two from the base address. The multiplexer follows the pattern corresponding to this configuration of the size and address signals and places the two least significant bytes of the long word on the word portion of the bus (D16–D31). The bus cycle transfers the remaining bytes to the word-size port. Figure 7-6 shows the timing of the bus transfer signals for this operation.

Figure 7-7 shows a word transfer to an 8-bit bus port. Like the preceding example, this example requires two bus cycles. Each bus cycle transfers a single byte. The size signals for the first cycle specify two bytes; for the second cycle, one byte. Figure 7-8 shows the associated bus transfer signal timing.

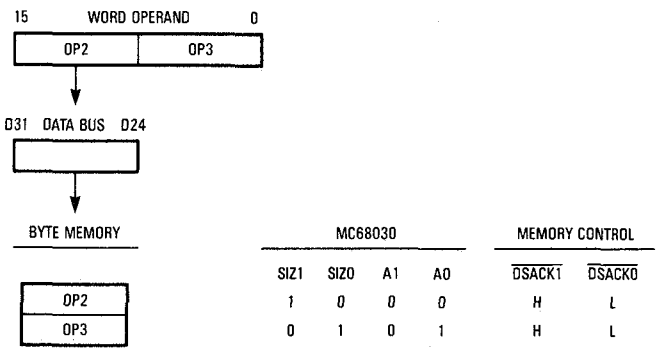
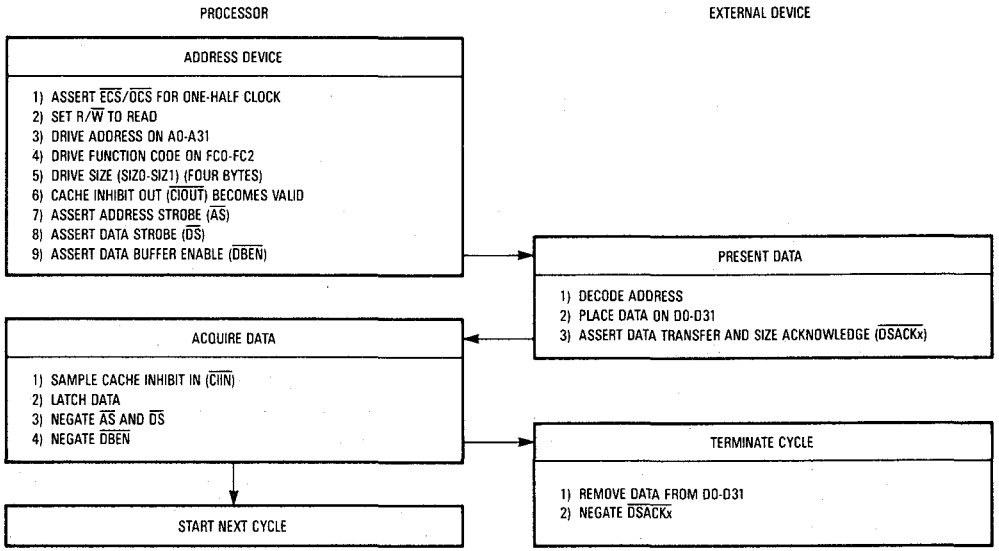


Figure 7-7. Example of Word Transfer to Byte Port

7.2.2 Misaligned Operands

Since operands may reside at any byte boundaries, 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 MC68030 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



7

Figure 7-19. Asynchronous Long-Word Read Cycle Flowchart

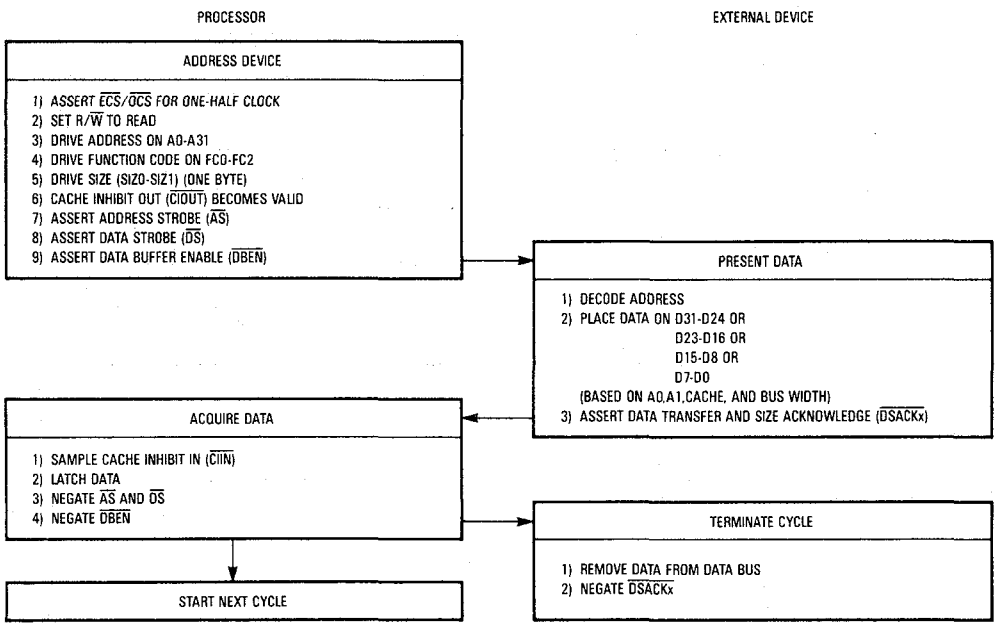


Figure 7-20. Asynchronous Byte Read Cycle Flowchart

7.5.2 Retry Operation

When the $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ signals are both asserted by an external device during a bus cycle, the processor enters the retry sequence. A delayed retry, similar to the delayed bus error signal described previously, can also occur, both for synchronous and asynchronous cycles.

The processor terminates the bus cycle, places the control signals in their inactive state, and does not begin another bus cycle until the $\overline{\text{HALT}}$ signal is negated by external logic. After a synchronization delay, the processor retries the previous cycle using the same access information (address, function code, size, etc.) The $\overline{\text{BERR}}$ signal should be negated before S2 of the read cycle to ensure correct operation of the retried cycle. Figure 7-54 shows a retry operation of an asynchronous cycle, and Figure 7-55 shows a retry operation of a synchronous cycle.

The processor retries any read or write cycle of a read-modify-write operation separately; $\overline{\text{RMC}}$ remains asserted during the entire retry sequence.

On the initial access of a burst operation, a retry (indicated by the assertion of $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$) causes the processor to retry the bus cycle and assert $\overline{\text{CBREQ}}$ again. Figure 7-56 shows a late retry operation that causes an initial burst operation to be repeated. However, signaling a retry with simultaneous $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ 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 $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ during a subsequent cycle of a burst operation causes independent $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ operations. The external bus activity remains halted until $\overline{\text{HALT}}$ is negated and the processor acts as previously described for the bus error during a burst operation.

Asserting $\overline{\text{BR}}$ along with $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ provides a relinquish and retry operation. The MC68030 does not relinquish the bus during a read-modify-write operation, except during the first read cycle. Any device that requires the processor to give up the bus and retry a bus cycle during a read-modify-write cycle must either assert $\overline{\text{BERR}}$ and $\overline{\text{BR}}$ only ($\overline{\text{HALT}}$ must not be included) or use the single wire arbitration method discussed in **7.7.4 Bus Arbitration Control**. The bus error handler software should examine the read-modify-write bit in the special status word (refer to **8.2.1 Special Status Word**) and take the appropriate action to resolve this type of fault when it occurs.

Exception processing for illegal and unimplemented instructions is similar to that for instruction traps. When the processor has identified an illegal or unimplemented instruction, it initiates exception processing instead of attempting to execute the instruction. The processor copies the status register, enters the supervisor privilege level, and clears the trace bits, disabling further tracing. The processor generates the vector number, either 4, 10, or 11, according to the exception type. The illegal or unimplemented instruction vector offset, current program counter, and copy of the status register are saved on the supervisor stack, with the saved value of the program counter being the address of the illegal or unimplemented instruction. Instruction execution resumes at the address contained in the exception vector. It is the responsibility of the handling routine to adjust the stacked program counter if the instruction is emulated in software or is to be skipped on return from the handler.

8.1.6 Privilege Violation Exception

To provide system security, the following instructions are privileged:

ANDI TO SR
EOR to SR
cpRESTORE
cpSAVE
MOVE from SR
MOVE to SR
MOVE USP
MOVEC
MOVES
ORI to SR
PFLUSH
PLOAD
PMOVE
PTEST
RESET
RTE
STOP

An attempt to execute one of the privileged instructions while at the user privilege level causes a privilege violation exception. Also, a privilege violation exception occurs if a coprocessor requests a privilege check and the processor is at the user level.

if $\overline{\text{MMUDIS}}$ is asserted during this type of operation, the disabling of address translation does not become effective until the entire transfer is complete. Note that the assertion of $\overline{\text{MMUDIS}}$ does not affect the operation of the transparent translation registers.

9.3 TRANSPARENT TRANSLATION

Two independent transparent translation registers (TT0 and TT1) in the MMU optionally define two blocks of the logical address space that are directly translated to the physical address spaces. The MMU does not explicitly check write protection for the addresses in these blocks, but a block can be specified as transparent only for read cycles. The blocks of addresses defined by the TTx registers include at least 16M bytes of logical address space; the two blocks can overlap, or they can be separate.

The following description of the address comparison assumes that both TT0 and TT1 are enabled; however, each TTx register can be independently disabled. A disabled TTx register is completely ignored.

When the MMU receives an address to be translated, the function code and the eight high-order bits of the address are compared to the block of addresses defined by TT0 and TT1. The address space block for each TTx register is defined by the base function code, the function code mask, the logical base address, and the logical address mask. When a bit in a mask field is set, the corresponding bit of the base function code or logical base address is ignored in the function code and address comparison. Setting successively higher order bits in the address mask increases the size of the transparently translated block.

The address for the current bus cycle and a TTx register address match when the function code bits and address bits (not including masked bits) are equal. Each TTx register can specify read accesses or write accesses as transparent. In that case, the internal read/write signal must match the R/$\overline{\text{W}}$ bit in the TTx register for the match to occur. The selection of the type of access (read or write) can also be masked. The read/write mask bit (RWM) must be set for transparent translation of addresses used by instructions that execute read-modify-write operations. Otherwise, neither the read nor write portions of read-modify-write operations are mapped transparently with the TTx registers, regardless of the function code and address bits for the individual cycles within a read-modify-write operation.

The bits in the MMUSR have different meanings for the two kinds of PTEST instructions, as shown in Table 9-3.

Table 9-3. MMUSR Bit Definitions

MMUSR Bit	PTEST, Level 0	PTEST, Level 1–7
Bus Error (B)	This bit is set if the bus error bit is set in the ATC entry for the specified logical address.	This bit is set if a bus error is encountered during the table search for the PTEST instruction.
Limit (L)	This bit is cleared.	This bit is set if an index exceeds a limit during the table search.
Supervisor Violation (S)	This bit is cleared.	This bit is set if the S bit of a long (S) format table descriptor or long format page descriptor encountered during the search is set, and the FC2 bit of the function code specified by the PTEST instruction is not equal to one. The S bit is undefined if the I bit is set.
Write Protected (W)	This bit is set if the WP bit of the ATC entry is set. It is undefined if the I bit is set.	This bit is set if a descriptor or page descriptor is encountered with the WP bit set during the table search. The W bit is undefined if the I bit is set.
Invalid (I)	This bit indicates an invalid translation. The I bit is set if the translation for the specified logical address is not resident in the ATC or if the B bit of the corresponding ATC entry is set.	This bit indicates an invalid translation. The I bit is set if the DT field of a table or a page descriptor encountered during the serach is set to invalid or if either the B or L bits of the MMUSR are set during the table search.
Modified (M)	This bit is set if the ATC entry corresponding to the specified address has the modified bit set. It is undefined if the I bit is set.	This bit is set if the page descriptor for the specified address has the modified bit set. It is undefined if I is set.
Transparent (T)	This bit is set if a match occurred in either (or both) of the transparent translation registers (TT0 or TT1). If the T bit is set, all remaining MMUSR bits are undefined.	This bit is set to zero.
Number of Levels (N)	This 3-bit field is cleared to zero.	This 3-bit field contains the actual number of tables accessed during the search.

10.5.1.4 COPROCESSOR SYSTEM-RELATED EXCEPTIONS. System-related exceptions detected by a DMA coprocessor include those associated with bus activity and any other exceptions (interrupts, for example) occurring external to the coprocessor. The actions taken by the coprocessor and the main processor depend on the type of exception that occurs.

When an address or bus error is detected by a DMA coprocessor, the coprocessor should store any information necessary for the main processor exception handling routines in system-accessible registers. The coprocessor should place one of the three take exception primitives encoded with an appropriate exception vector number in the response CIR. Which of the three primitives is used depends upon the point in the coprocessor instruction at which the exception was detected and the point in the instruction execution at which the main processor should continue after exception processing.

10.5.1.5 FORMAT ERRORS. Format errors are the only coprocessor-detected exceptions that are not signaled to the main processor with a response primitive. When the main processor writes a format word to the restore CIR during the execution of a cpRESTORE instruction, the coprocessor decodes this word to determine if it is valid (refer to **10.2.3.3 COPROCESSOR CONTEXT SAVE INSTRUCTION**). If the format word is not valid, the coprocessor places the invalid format code in the restore CIR. When the main processor reads the invalid format code, it aborts the coprocessor instruction by writing an abort mask (refer to **10.3.2 Control CIR**) to the control CIR. The main processor then performs exception processing using a four-word pre-instruction stack frame and the format error exception vector number 14. Thus, if the exception handler does not modify the stack frame, the MC68030 restarts the cpRESTORE instruction when the RTE instruction in the handler is executed. If the coprocessor returns the invalid format code when the main processor reads the save CIR to initiate a cpSAVE instruction, the main processor performs format error exception processing as outlined for the cpRESTORE instruction.

10.5.2.5 TRACE EXCEPTIONS. The MC68030 supports two modes of instruction tracing, discussed in **8.1.7 Trace Exception**. In the trace on instruction execution mode, the MC68030 takes a trace exception after completing each instruction. In the trace on change of flow mode, the MC68030 takes a trace exception after each instruction that alters the status register or places an address other than the address of the next instruction in program counter.

The protocol used to execute coprocessor cpSAVE, cpRESTORE, or conditional category instructions does not change when a trace exception is pending in the main processor. The main processor performs a pending trace on instruction execution exception after completing the execution of that instruction. If the main processor is in the trace on change of flow mode and an instruction places an address other than that of the next instruction in the program counter, the processor takes a trace exception after it executes the instruction.

If a trace exception is not pending during a general category instruction, the main processor terminates communication with the coprocessor after reading any primitive with CA = 0. Thus, the coprocessor can complete a cpGEN instruction concurrently with the execution of instructions by the main processor. When a trace exception is pending, however, the main processor must ensure that all processing associated with a cpGEN instruction has been completed before it takes the trace exception. In this case, the main processor continues to read the response CIR and to service the primitives until it receives either a null, CA = 0, PF = 1 primitive, or until exception processing caused by a take post-instruction exception primitive has completed. The coprocessor should return the null, CA = 0 primitive with PF = 0, while it is completing the execution of the cpGEN instruction. The main processor may service pending interrupts between reads of the response CIR if IA = 1 in these primitives (refer to Table 10-3). This protocol ensures that a trace exception is not taken until all processing associated with a cpGEN instruction has completed.

If T1:T0 = 01 in the MC68030 status register (trace on change of flow) when a general category instruction is initiated, a trace exception is taken for the instruction only when the coprocessor issues a transfer status register and scanPC primitive with DR = 1 during the execution of that instruction. In this case, it is possible that the coprocessor is still executing the cpGEN instruction concurrently when the main processor begins execution of the trace exception handler. A cpSAVE instruction executed during the trace on change of flow exception handler could thus suspend the execution of a concurrently operating cpGEN instruction.

11.6.9 Immediate Arithmetical/Logical Instructions

The immediate arithmetical/logical operation timing table indicates the number of clock periods needed for the processor to fetch the source immediate data value and to perform the specified arithmetic/logical operation using the specified destination addressing mode. Footnotes indicate when to account for the appropriate fetch effective or fetch immediate effective address times. For instruction-cache case and for no-cache case, 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). The read, prefetch, and write cycles are included in the total clock cycle number.

All timing data assumes two-clock reads and writes.

	Instruction		Head	Tail	I-Cache Case	No-Cache Case
	MOVEQ	#⟨data⟩,Dn	2	0	2(0/0/0)	2(0/1/0)
	ADDQ	#⟨data⟩,Rn	2	0	2(0/0/0)	2(0/1/0)
*	ADDQ	#⟨data⟩,Mem	0	1	3(0/0/1)	4(0/1/1)
	SUBQ	#⟨data⟩,Rn	2	0	2(0/0/0)	2(0/1/0)
*	SUBQ	#⟨data⟩,Mem	0	1	3(0/0/1)	4(0/1/1)
**	ADDI	#⟨data⟩,Dn	2	0	2(0/0/0)	2(0/1/0)
**	ADDI	#⟨data⟩,Mem	0	1	3(0/0/1)	4(0/1/1)
**	ANDI	#⟨data⟩,Dn	2	0	2(0/0/0)	2(0/1/0)
**	ANDI	#⟨data⟩,Mem	0	1	3(0/0/1)	4(0/1/1)
**	EORI	#⟨data⟩,Dn	2	0	2(0/0/0)	2(0/1/0)
**	EORI	#⟨data⟩,Mem	0	1	3(0/0/1)	4(0/1/1)
**	ORI	#⟨data⟩,Dn	2	0	2(0/0/0)	2(0/1/0)
**	ORI	#⟨data⟩,Mem	0	1	3(0/0/1)	4(0/1/1)
**	SUBI	#⟨data⟩,Dn	2	0	2(0/0/0)	2(0/1/0)
**	SUBI	#⟨data⟩,Mem	0	1	3(0/0/1)	4(0/1/1)
**	CMPI	#⟨data⟩,Dn	2	0	2(0/0/0)	2(0/1/0)
**	CMPI	#⟨data⟩,Mem	0	0	2(0/0/0)	2(0/1/0)

*Add Fetch Effective Address Time
**Add Fetch Immediate Effective Address Time

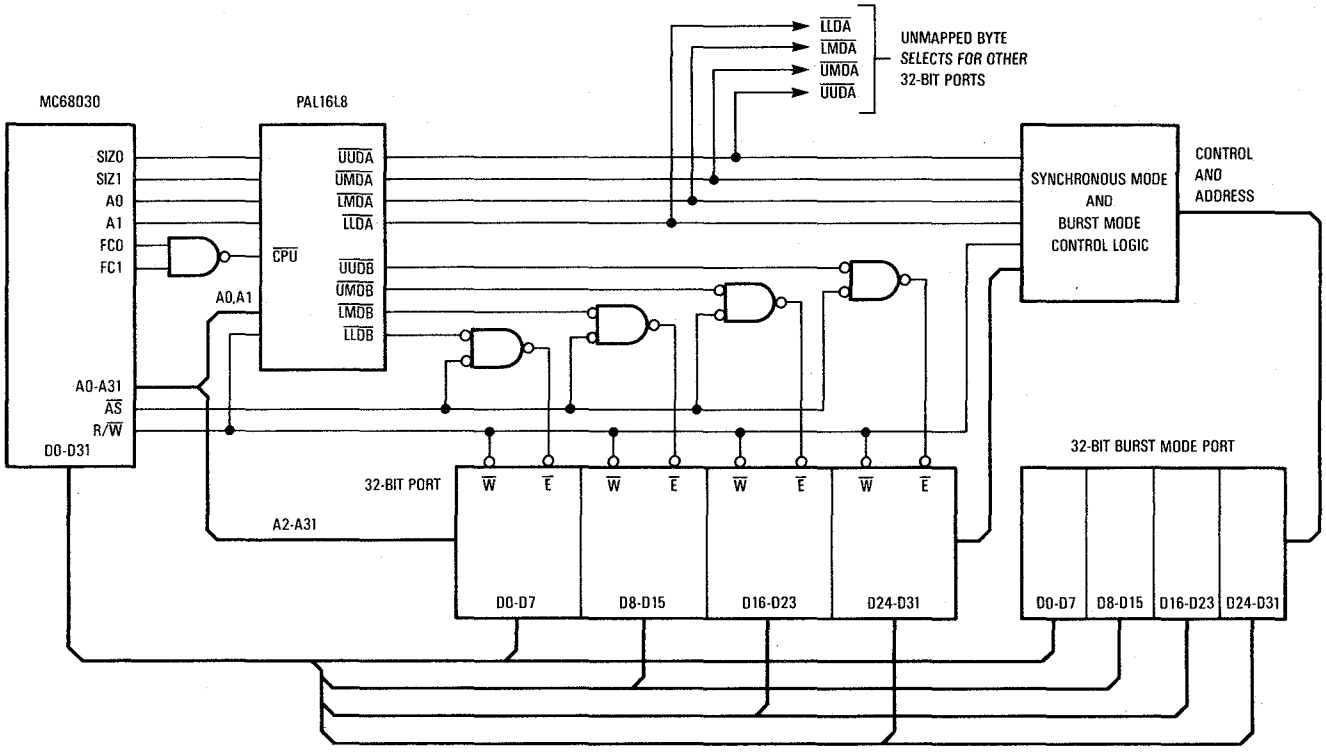


Figure 12-6. Example MC68030 Byte Select PAL System Configuration

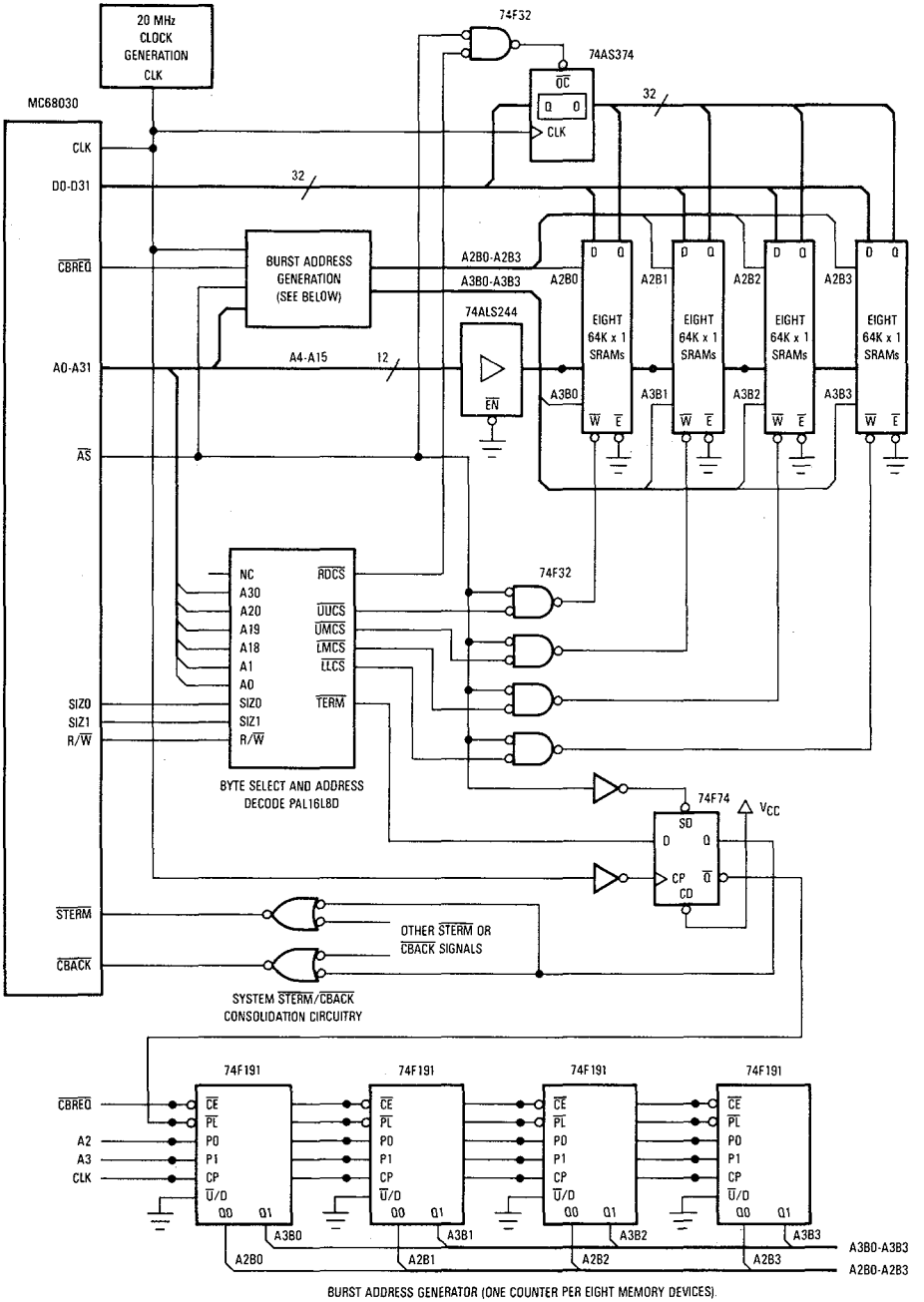


Figure 12-15. Example 3-1-1-1 Pipelined Burst Mode Memory Bank at 20 MHz, 256K Bytes