SECTION 9
APPLICATIONS INFORMATION

This section, which provides guidelines for using the MC68020/EC020, contains information on floating-point units, byte select logic, power and ground considerations, clock driver, memory interface, access time calculations, module support, and access levels.

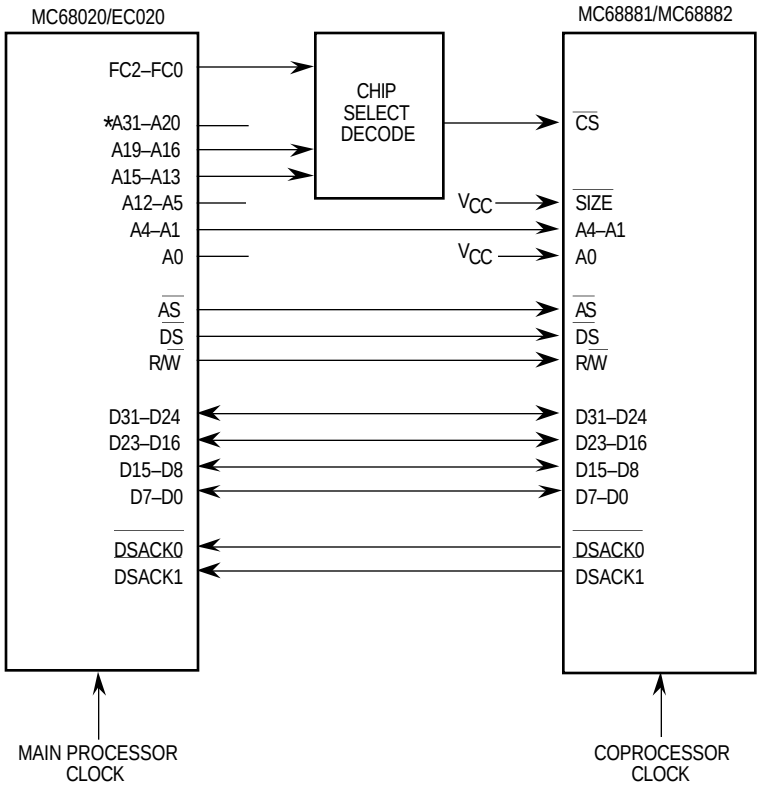
9.1 FLOATING-POINT UNITS

Floating-point support for the MC68020/EC020 is provided by the MC68881 floating-point coprocessor or the MC68882 enhanced floating-point coprocessor. Both devices offer a full implementation of the *IEEE Standard for Binary Floating-Point Arithmetic* (754). The MC68882 is a pin- and software-compatible upgrade of the MC68881, with an optimized MPU interface that provides over 1.5 times the performance of the MC68881 at the same clock frequency.

Both coprocessors provide a logical extension to the integer data processing capabilities of the main processor. They contain a high-performance floating-point arithmetic unit and a set of floating-point data registers that are utilized in a manner that is analogous to the use of the integer data registers of the processor. The MC68881/MC68882 instruction set, a natural extension of all earlier members of the M68000 family, supports all addressing modes and data types of the host MC68020/EC020. The programmer perceives the MC68020/EC020 coprocessor execution model as if both devices are implemented on one chip. In addition to supporting the full IEEE standard, the MC68881 and MC68882 provide a full set of trigonometric and transcendental functions, on-chip constants, and a full 80-bit extended-precision real data format.

The interface of the MC68020/EC020 to the MC68881 or MC68882 is easily tailored to system cost/performance needs. The MC68020/EC020 and the MC68881/MC68882 communicate via standard asynchronous M68000 bus cycles. All data transfers are performed by the main processor at the request of the MC68881/MC68882; thus, memory management, bus errors, address errors, and bus arbitration function as if the MC68881/MC68882 instructions are executed by the main processor. The floating-point unit and the processor can operate at different clock speeds, and up to seven floating-point coprocessors can simultaneously reside in an MC68020/EC020 system.

Figure 9-1 illustrates the coprocessor interface connection of an MC68881/MC68882 to an MC68020/EC020 (uses entire 32-bit data bus). The MC68881/MC68882 is configured to operate with a 32-bit data bus when both its A0 and SIZE pins are connected to V_{CC}. Refer to the MC68881UM/AD, *MC68881/MC68882 Floating-Point Coprocessor User's Manual*, for configuring the MC68881/MC68882 for smaller data bus widths.



*For the MC68EC020, A23–A0.

Figure 9-1. 32-Bit Data Bus Coprocessor Connection

The chip select (CS) decode circuitry is asynchronous logic that detects when a particular floating-point coprocessor is addressed. The MC68020/EC020 signals used by the logic include FC2–FC0 and A19–A13. Refer to **Section 7 Coprocessor Interface Description** for more information concerning the encoding of these signals. All or just a subset of these lines may be decoded, depending on the number of coprocessors in the system and the degree of redundant mapping allowed in the system.

For example, if a system has only one coprocessor, the full decoding of the ten signals (FC2–FC0 and A19–A13), provided by the PAL equations in Figure 9-3, is not absolutely necessary. It may be sufficient to use only FC1–FC0 and A17–A16. FC1–FC0 indicate when a bus cycle is operating in either CPU space ($7) or user-defined space ($3), and A17–A16 encode the CPU space type as coprocessor space ($2). A15–A13 can be ignored in this case because they encode the coprocessor identification code (CpID) used to differentiate between multiple coprocessors in a system. Motorola assemblers always default to a CpID of $1 for floating-point instructions; this can be controlled with assembler directives if a different CpID is desired or if multiple coprocessors exist in the system.

The major concern of a system designer is to design a CS interface that meets the AC electrical specifications for both the MC68020/EC020 (MPU) and the MC68881/MC68882 (FPCP) without adding unnecessary wait states to FPCP accesses. The following maximum specifications (relative to CLK low) meet these objectives:

$$t_{CLK} \text{ low to AS low} \leq (\text{MPU Spec 1} - \text{MPU Spec 47A} - \text{FPCP Spec 19}) \quad (9\text{-}1)$$

$$t_{CLK} \text{ low to CS low} \leq (\text{MPU Spec 1} - \text{MPU Spec 47A} - \text{FPCP Spec 19}) \quad (9\text{-}2)$$

Even though requirement (9-1) is not met under worst-case conditions, if the MPU AS is loaded within specifications and the AS input to the FPCP is unbuffered, the requirement is met under typical conditions. Designing the CS generation circuit to meet requirement (9-2) provides the highest probability that accesses to the FPCP occur without unnecessary wait states. A PAL 16L8 (see Figure 9-2) with a maximum propagation delay of 10 ns, programmed according to the equations in Figure 9-3, can be used to generate CS. For a 25-MHz system, t_{CLK} low to CS low is less than or equal to 10 ns when this design is used. Should worst-case conditions cause t_{CLK} low to AS low to exceed requirement (1), one wait state is inserted in the access to the FPCP; no other adverse effects occur. Figure 9-4 shows the bus cycle timing for this interface. Refer to MC68881UM/AD, *MC68881/MC68882 Floating-Point Coprocessor User's Manual*, for FPCP specifications.

The circuit that generates CS must meet another requirement. When a nonfloating-point access immediately follows a floating-point access, CS (for the floating-point access) must be negated before AS and DS (for the subsequent access) are asserted. The PAL circuit previously described also meets this requirement.

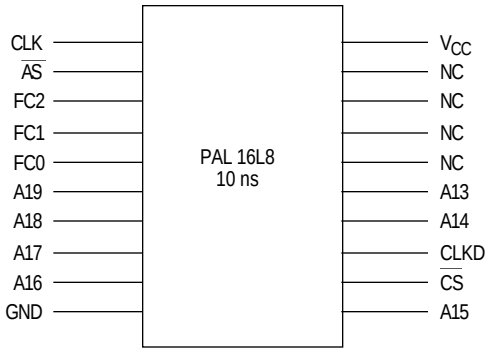


Figure 9-2. Chip Select Generation PAL

```
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

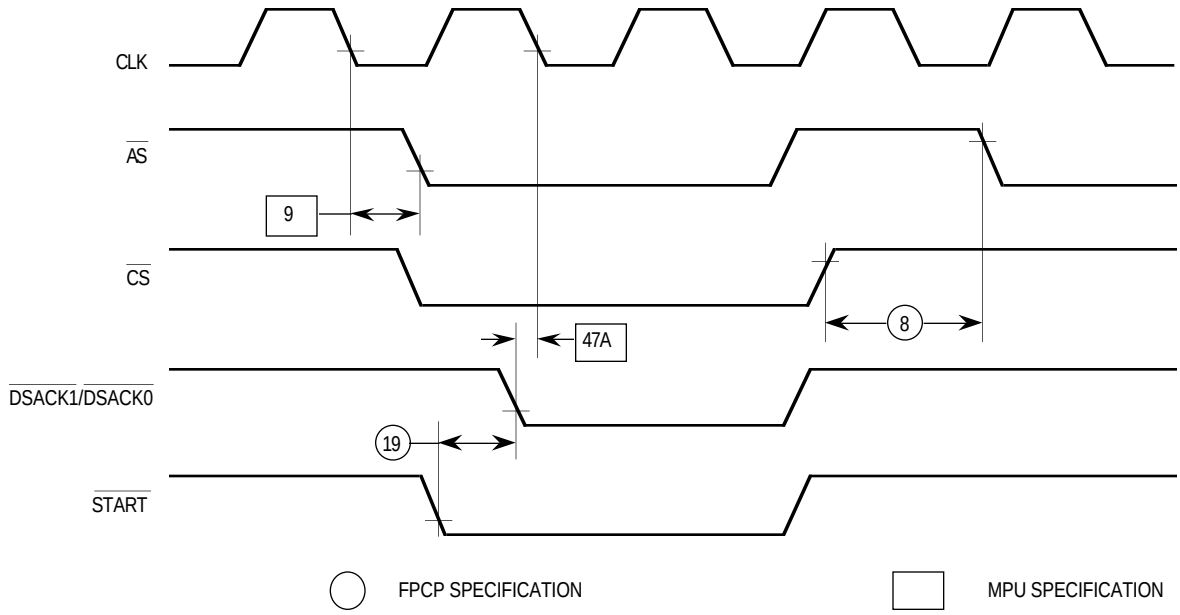


Figure 9-4. Bus Cycle Timing Diagram

9.2 BYTE SELECT LOGIC FOR THE MC68020/EC020

The MC68020/EC020 architecture supports byte, word, and long-word operand transfers to any 8-, 16-, or 32-bit data port, regardless of alignment. This feature allows the programmer to write code that is not bus-width specific. When accessed, the peripheral or memory subsystem reports its actual port size to the controller, and the MC68020/EC020 then dynamically sizes the data transfer accordingly, using multiple bus cycles when necessary. The following paragraphs describe the generation of byte select control signals that enable the dynamic bus sizing mechanism, the transfer of differently sized operands, and the transfer of misaligned operands to operate correctly.

The following signals control the MC68020/EC020 operand transfer mechanism:

A1, A0 — Address signals. The most significant byte of the operand to be transferred is addressed directly.

SIZ1, SIZ0 — Transfer size signals. Output of the MC68020/EC020. These indicate the number of bytes of an operand remaining to be transferred during a given bus cycle.

R/W — Read/Write signal. Output of the MC68020/EC020. For byte select generation in MC68020/EC020 systems.

DSACK1, DSACK0 — Data transfer and size acknowledge signals. Driven by an asynchronous port to indicate the actual bus width of the port.

The MC68020/EC020 assumes that 16-bit ports are situated on data lines D31–D16, and that 8-bit ports are situated on data lines D31–D24. This ensures that the following logic works correctly with the MC68020/EC020's on-chip internal-to-external data bus multiplexer. Refer to **Section 5 Bus Operation** for more details on the dynamic bus sizing mechanism.

The need for byte select signals is best illustrated by an example. Consider a long-word write cycle to an odd address in word-organized memory. The transfer requires three bus cycles to complete. The first bus cycle transfers the most significant byte of the long word on D23–D16. The second bus cycle transfers a word on D31–D16, and the last bus cycle transfers the least significant byte of the original long word on D31–D24. To prevent overwriting those bytes that are not used in these transfers, a unique byte data strobe must be generated for each byte when using devices with 16- and 32-bit port widths.

For noncachable read cycles and all write cycles, the required active bytes of the data bus for any given bus transfer are a function of the SIZ1, SIZ0 and A1, A0 outputs (see Table 9-1). Individual strobes or select signals can be generated by decoding these four signals for every bus cycle. Devices residing on 8-bit ports can utilize DS or AS since there is only one valid byte for any transfer.

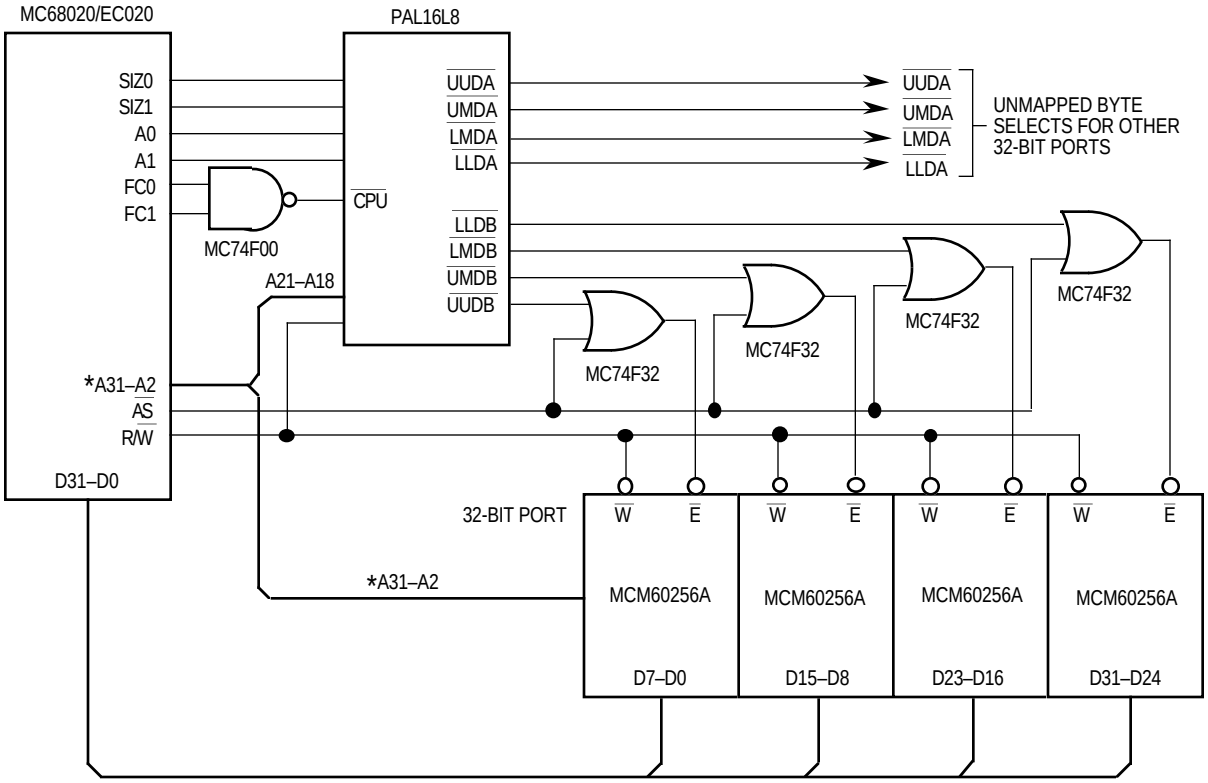
Table 9-1. Data Bus Activity for Byte, Word, and Long-Word Ports

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

During cachable read cycles, the addressed device must provide valid data over its full bus width as indicated by DSACK1/DSACK0. While instructions are always prefetched as long-word-aligned accesses, data fetches can occur with any alignment and size. Because the MC68020/EC020 assumes that the entire data bus port size contains valid data, cachable data read bus cycles must provide as much data as signaled by the port size during a bus cycle. To satisfy this requirement, the R/W signal must be included in the byte select logic for the MC68020/EC020.

Figure 9-5 shows a block diagram of an MC68020/EC020 system with a single memory bank. The PAL provides memory-mapped byte select signals for an asynchronous 32-bit port and unmapped byte select signals for other memory banks or ports. Figure 9-6 provides sample equations for the PAL.

The PAL equations and circuits presented here cannot be the optimal implementation for every system. Depending on the CPU clock frequency, memory access times, and system architecture, different circuits may be required.



*For the MC68EC020, A23–A2.

Figure 9-5. Example MC68020/EC020 Byte Select PAL System Configuration

```
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

9.3 POWER AND GROUND CONSIDERATIONS

The MC68020/EC020 is fabricated in Motorola's advanced HCMOS process and is capable of operating at clock frequencies of up to 25 MHz. While the use of CMOS for a device containing such a large number of transistors allows significantly reduced power consumption compared to an equivalent NMOS circuit, the high clock speed makes the characteristics of power supplied to the device very important. The power supply must be able to furnish large amounts of instantaneous current when the MC68020/EC020 performs certain operations, and it must remain within the rated specification at all times. To meet these requirements, more detailed attention must be given to the power supply connection to the MC68020/EC020 than is required for NMOS devices operating at slower clock rates.

To reduce the amount of noise in the power supply connected to the MC68020/EC020 and to provide for the instantaneous current requirements, common capacitive decoupling techniques should be observed. While there is no recommended layout for this capacitive decoupling, it is essential that the inductance and distance between these devices and the MC68020/EC020 be minimized to provide sufficiently fast response time to satisfy momentary current demands and to maintain a constant supply voltage. It is suggested that high-frequency, high-quality capacitors be placed as close to the MC68020/EC020 as possible. Table 9-2 lists the V_{CC} and GND pin assignments for the MC68EC020 PPGA (RP suffix) package. Table 9-3 lists the V_{CC} and GND pin assignments for the MC68EC020 PQFP (FG suffix) package. Refer to **Section 11 Ordering Information and Mechanical Data** for the V_{CC} and GND pin assignments for the MC68020 packages. When assigning capacitors to the V_{CC} and GND pins, the noisier pins (address and data buses) should be heavily decoupled from the internal logic pins. Typical decoupling practices include a high-frequency, high-quality capacitor to decouple every device on the printed circuit board; however, due to the power requirements and drive capability of the MC68020/EC020, each V_{CC} pin should be decoupled with an individual capacitor. Motorola recommends using a capacitor in the range of 0.01 μF to 0.1 μF on each V_{CC} pin on each device to provide filtering for most frequencies prevalent in a digital system. In addition to the individual decoupling, several bulk decoupling capacitors should be placed onto the printed circuit board with typical values in the range of 33 μF to 330 μF. When power and ground planes are used with an adequate number of high-frequency, high-quality capacitors, the system noise will be reduced to the required levels, and the MC68020/EC020 will function properly. Similar decoupling techniques should also be observed for other VLSI devices in the system.

In addition to the capacitive decoupling of the power supply, care must be taken to ensure a low-impedance connection between all MC68020/EC020 V_{CC} and GND pins and the system power supply. A solid power supply connection from the power and ground planes to the MC68020/EC020 V_{CC} and GND pins, respectively, will meet this requirement. Failure to provide connections of sufficient quality between the MC68020/EC020 power pins and the system power supplies will result in increased assertion delays for external signals, decreased voltage noise margins, increased system noise, and possible errors in MC68020/EC020 internal logic.

Table 9-2. V_{CC} and GND Pin Assignments—MC68EC020 PPGA (RP Suffix)

Pin Group	V_{CC}	GND
Address Bus	B7, C7	A1, A7, C8, D13
Data Bus	K12, M9, N9	J13, L8, M1, M8
Internal Logic	D1, D2, E12, E13	F11, F12, J1, J2
Clock	—	B1

Table 9-3. V_{CC} and GND Pin Assignments—MC68EC020 PQFP (FG Suffix)

Pin Group	V_{CC}	GND
Address Bus	90	72, 89, 100
Data Bus	44, 57	26, 43, 58, 59
Internal Logic	7, 8, 70, 71	3, 20, 21, 68, 69
Clock	—	4

9.4 CLOCK DRIVER

The MC68020/EC020 is designed to sustain high performance while using low-cost memory subsystems. The MC68020/EC020 requires a stable clock source that is free of ringing and ground bounce, has sufficient rise and fall times, and meets the minimum and maximum high and low cycle times. The individual system may require additional clocks for peripherals with a minimum amount of clock skew. Two possible clock solutions are provided with the MC88916 and MC74F803. Many other clock solutions can be used. Some crystal clock drivers are capable of driving the MC68020/EC020 directly. For slower speed designs, a simple 74F74 flip-flop meets the clocking needs of the MC68020/EC020. Coupled with the MC88916 or MC74F803 clock generation and distribution circuit, the MC68020/EC020 provides simple interface to lower speed memory subsystems. The MC88916 (see Figure 9-7) and MC74F803 (see Figure 9-8) generate the clock signals required to minimize the skew between different clocks to multiple devices such as coprocessors, synchronous state machines, DRAM controllers, and memory subsystems. The MC88916 clock driver can be used in doubling and synchronizing a low-frequency clock source. The MC74F803 will provide a controlled skew output for clocking other peripherals.

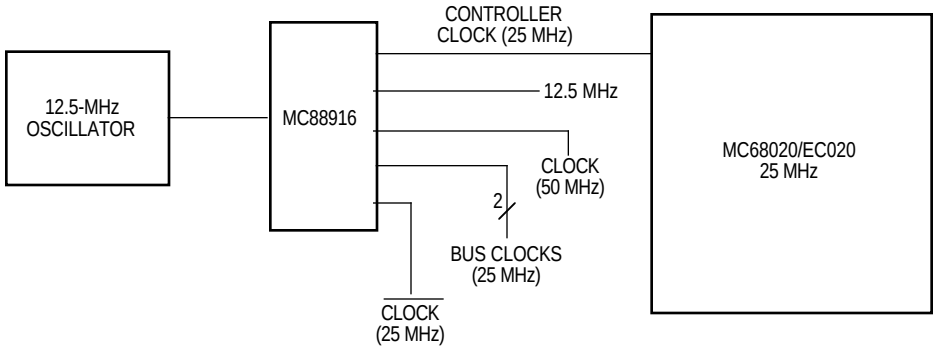


Figure 9-7. High-Resolution Clock Controller

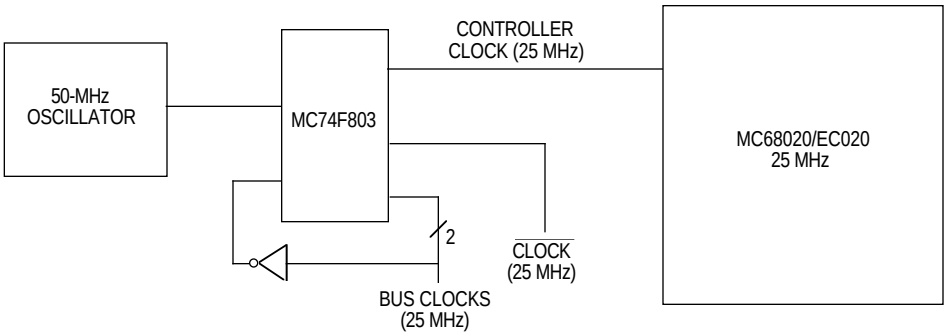


Figure 9-8. Alternate Clock Solution

9.5 MEMORY INTERFACE

The MC68020/EC020 is capable of running an external bus cycle in a minimum of three clocks (refer to **Section 5 Bus Operation**). The MC68020/EC020 runs an asynchronous bus cycle, terminated by the DSACK1/DSACK0 signals, and has a minimum duration of three controller clock periods in which up to four bytes (32 bits) are transferred.

During read operations, the MC68020/EC020 latches data on the last falling clock edge of the bus cycle, one-half clock before the bus cycle ends. Latching data here, instead of the next rising clock edge, helps to avoid data bus contention with the next bus cycle and allows the MC68020/EC020 to receive the data into its execution unit sooner for a net performance increase.

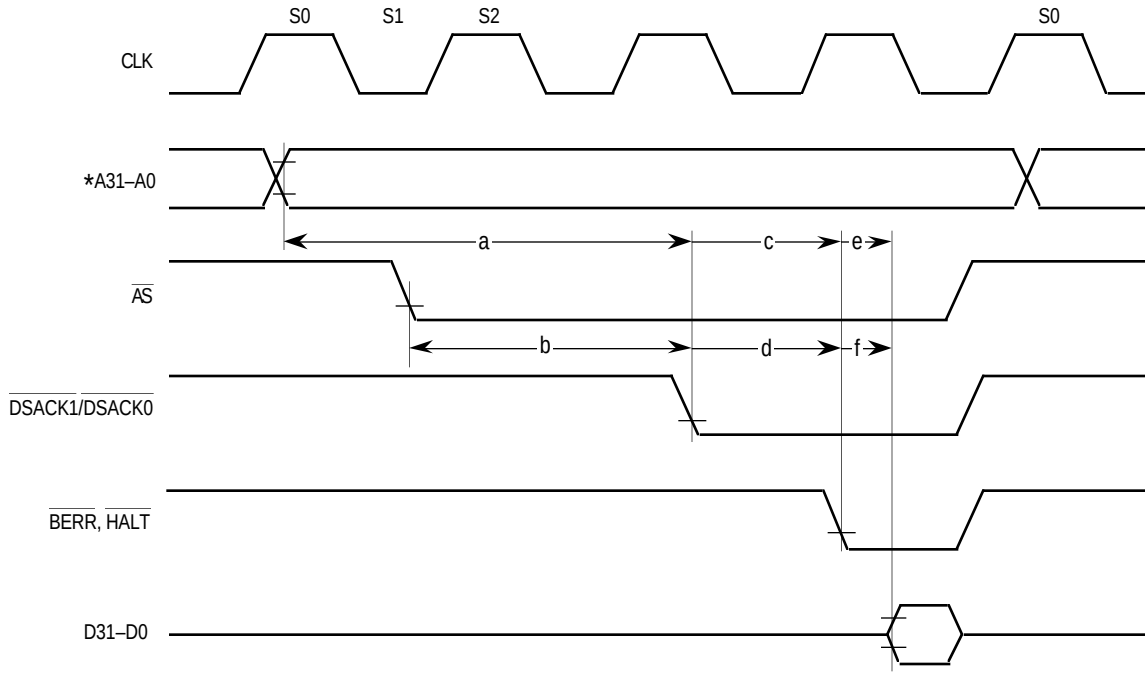
Write operations also use this data bus timing to allow data hold times from the negating strobes and to avoid any bus contention with the following bus cycle. This MC68020/EC020 characteristic allows the system to be designed with a minimum of bus buffers and latches.

One benefit of the MC68020/EC020 on-chip instruction cache is that the effect of external wait states on performance is lessened because the caches are always accessed in fewer than “no wait states,” regardless of the external memory configuration.

9.6 ACCESS TIME CALCULATIONS

The timing paths that are critical in any memory interface are illustrated and defined in Figure 9-9.

The type of device that is interfaced to the MC68020/EC020 determines exactly which of the paths is most critical. The address-to-data paths are typically the critical paths for static devices since there is no penalty for initiating a cycle to these devices and later validating that access with the appropriate bus control signal. Conversely, the address-strobe-to-data-valid path is often most critical for dynamic devices since the cycle must be validated before an access can be initiated. For devices that signal termination of a bus cycle before data is validated (e.g., error detection and correction hardware and some external caches), to improve performance, the critical path may be from the address or strobes to the assertion of BERR (or BERR and HALT). Finally, the address-valid-to-DSACK1/DSACK0-asserted path is most critical for very fast devices and external caches, since the time available between the address becoming valid and the DSACK1/DSACK0 assertion to terminate the bus cycle is minimal. Table 9-4 provides the equations required to calculate the various memory access times assuming a 50-percent duty cycle clock.



*For the MC68EC020, A23–A0.

Parameter	Description	System	Equation
a	Address Valid to DSACK1/DSACK0 Asserted	t_{AVDL}	9-3
b	AS Asserted to DSACK1/DSACK0 Asserted	t_{SADL}	9-4
c	Address Valid to BERR/HALT Asserted	t_{AVBHL}	9-5
d	AS Asserted to BERR/HALT Asserted	t_{SABHL}	9-6
e	Address Valid to Data Valid	t_{AVDV}	9-7
f	AS Asserted to Data Valid	t_{SADV}	9-8

Figure 9-9. Access Time Computation Diagram

Table 9-4. Memory Access Time Equations at 16.67 and 25 MHz

Equation	16.667 MHz	N = 3	N = 4	N = 5	N = 6	N = 7	Unit
9-3	t_{AVDL} = (N – 1) • t1 – t2 – t6 – t47A	61	121	181	241	301	ns
9-4	t_{SADL} = (N – 1) • t1 – t9 – t60	25	85	145	205	265	ns
9-5	t_{AVBHL} = N • t1 – t2 – t6 – t27A	22	46	70	94	118	ns
9-6	t_{SABHL} = (N – 1) • t1 – t9 – t27A	40	70	100	130	160	ns
9-7	t_{AVDV} = N • t1 – t2 – t6 – t27	121	181	241	301	361	ns
9-8	t_{SADV} = (N – 1) • t1 – t9 – t27	85	145	205	265	325	ns

Equation	25 MHz	N = 3	N = 4	N = 5	N = 6	N = 7	Unit
9-3	t_{AVDL} = (N – 1) • t1 – t2 – t6 – t47A	31	71	111	151	191	ns
9-4	t_{SADL} = (N – 1) • t1 – t9 – t60	17	57	97	137	177	ns
9-5	t_{AVBHL} = N • t1 – t2 – t6 – t27A	22	41	60	79	98	ns
9-6	t_{SABHL} = (N – 1) • t1 – t9 – t27A	26	44	62	80	98	ns
9-7	t_{AVDV} = N • t1 – t2 – t6 – t27	71	111	151	191	231	ns
9-8	t_{SADV} = (N – 1) • t1 – t9 – t27	57	97	137	177	217	ns

Where:

tX = Refers to AC Electrical Specification X
t1 = The Clock Period
t2 = The Clock Low Time
t3 = The Clock High Time
t6 = The Clock High to Address Valid Time
t9 = The Clock Low to AS Low Delay
t27 = The Data-In to Clock Low Setup Time
t27A = The BERR/HALT to Clock Low Setup Time
t47A = The Asynchronous Input Setup Time
N = The Total Number of Clock Periods in the Bus Cycle (N ≥ 3 Cycles)

During asynchronous bus cycles, DSACK1/DSACK0 are used to terminate the current bus cycle. In true asynchronous operations, such as accesses to peripherals operating at a different clock frequency, either or both signals may be asserted without regard to the clock, and then data must be valid a certain amount of time later as defined by specification 31. With a 25-MHz controller, this time is 32 ns after DSACK1/DSACK0 asserts; with a 16.67-MHz controller, this time is 50 ns after DSACK1/DSACK0 asserts (both numbers vary with the actual clock frequency).

However, many local memory systems do not operate in a truly asynchronous manner because either the memory control logic can be related to the MC68020/EC020 clock or worst-case propagation delays are known; thus, asynchronous setup times for the DSACK1/DSACK0 signals can be guaranteed. The timing requirements for this pseudo-synchronous DSACK1/DSACK0 generation is governed by the equation for t_{AVDL}.

Another way to optimize the CPU-to-memory access times in a system is to use a clock frequency less than the rated maximum of the specific MC68020/EC020 device. Table 9-5 provides calculated t_{AVDV} (see Equation 9-7 of Table 9-4) results for a 16 MHz MC68020/EC020 and a 25 MHz MC68020/EC020 operating at various clock frequencies. If the system uses other clock frequencies, the above equations can be used to calculate the exact access times.

Table 9-5. Calculated t_{AVDV} Values for Operation at Frequencies Less Than or Equal to the CPU Maximum Frequency Rating

Equation 9-7 t_{AVDV}		16-MHz MC68020/EC020		25-MHz MC68020/EC020		
Clocks Per (N) and Type Bus Cycle	Wait States	Clock at 12.5 MHz	Clock at 16.67 MHz	Clock at 16.67 MHz	Clock at 20 MHz	Clock at 25 MHz
3 Clock Asynchronous	0	181	121	131	101	71
4 Clock Asynchronous	1	261	181	191	151	111
5 Clock Asynchronous	2	341	241	251	201	151
6 Clock Asynchronous	3	421	301	311	251	191

9.7 MODULE SUPPORT

The MC68020/EC020 includes support for modules with the CALLM and RTM instructions. The CALLM instruction references a module descriptor. This descriptor contains control information for entry into the called module. The CALLM instruction creates a module stack frame and stores the current module state in that frame and loads a new module state from the referenced descriptor. The RTM instruction recovers the previous module state from the stack frame and returns to the calling module.

The module interface facilitates finer resolution of access control by external hardware. Although the MC68020/EC020 does not interpret the access control information, it communicates with external hardware when the access control is to be changed and relies on the external hardware to verify that the changes are legal.

9.7.1 Module Descriptor

Figure 9-10 illustrates the format of the module descriptor. The first long word contains control information used during execution of the CALLM instruction. The remaining locations contain data that can be loaded into processor registers by the CALLM instruction.

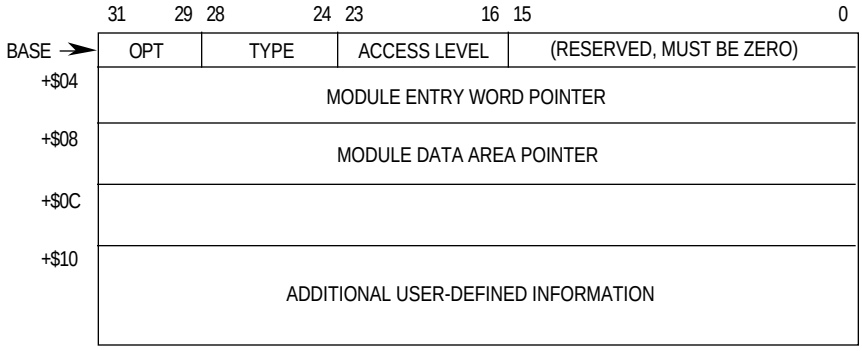


Figure 9-10. Module Descriptor Format

The opt field specifies how arguments are to be passed to the called module; the MC68020/EC020 recognizes only the options of 000 and 100; all others cause a format exception. The 000 option indicates that the called module expects to find arguments from the calling module on the stack just below the module stack frame. In cases where there is a change of stack pointer during the call, the MC68020/EC020 will copy the arguments from the old stack to the new stack. The 100 option indicates that the called module will access the arguments from the calling module through an indirect pointer in the stack of the calling module. Hence, the arguments are not copied, but the MC68020/EC020 puts the value of the stack pointer from the calling module in the module stack frame.

The type field specifies the type of the descriptor; the MC68020/EC020 only recognizes descriptors of type $00 and $01; all others cause a format exception. The $00 type descriptor defines a module for which there is no change in access rights, and the called module builds its stack frame on top of the stack used by the calling module. The $01 type descriptor defines a module for which there may be a change in access rights; such a called module may have a separate stack area from that of the calling module.

The access level field is used only with the type $01 descriptor and is passed to external hardware to change the access control.

The module entry word pointer specifies the entry address of the called module. The first word at the entry address (see Figure 9-11) specifies the register to be saved in the module stack frame and then loaded with the module descriptor data area pointer; the first instruction of the module starts with the next word. The module descriptor data area pointer field contains the address of the called module data area.

If the access change requires a change of stack pointer, the old value is saved in the module stack frame, and the new value is taken from the module descriptor stack pointer field. Any further information in the module descriptor is user defined.

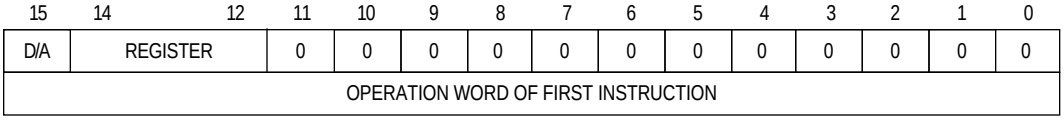


Figure 9-11. Module Entry Word

All module descriptor types $10–$1F are reserved for user definition and cause a format error exception. This provides the user with a means of disabling any module by setting a single bit in its descriptor without loss of any descriptor information.

If the called module does not wish the module data area pointer to be loaded into a register, the module entry word can select register A7, and the loaded value will be overwritten with the correction stack pointer value after the module stack frame is created and filled.

9.7.2 Module Stack Frame

Figure 9-12 illustrates the format of the module stack frame. This frame is constructed by the CALLM instruction and is removed by the RTM instruction. The first and second long words contain control information passed by the CALLM instruction to the RTM instruction. The module descriptor pointer contains the address of the descriptor used during the module call. All other locations contain information to be restored on return to the calling module.

The PC is the saved address of the instruction following the CALLM instruction. The opt and type fields, which specify the argument options and type of module stack frame, are copied to the frame from the module descriptor by the CALLM instruction; the RTM instruction will cause a format error if the opt and type fields do not have recognizable values. The access level is the saved access control information, which is saved from external hardware by the CALLM instruction and restored by the RTM instruction. The argument count field is set by the CALLM instruction and is used by the RTM instruction to remove arguments from the stack of the calling module. The contents of the CCR are saved by the CALLM instruction and restored by the RTM instruction. The saved stack pointer field contains the value of the stack pointer when the CALLM instruction started execution, and that value is restored by RTM. The saved module data area pointer field contains the saved value of the module data area pointer register from the calling module.

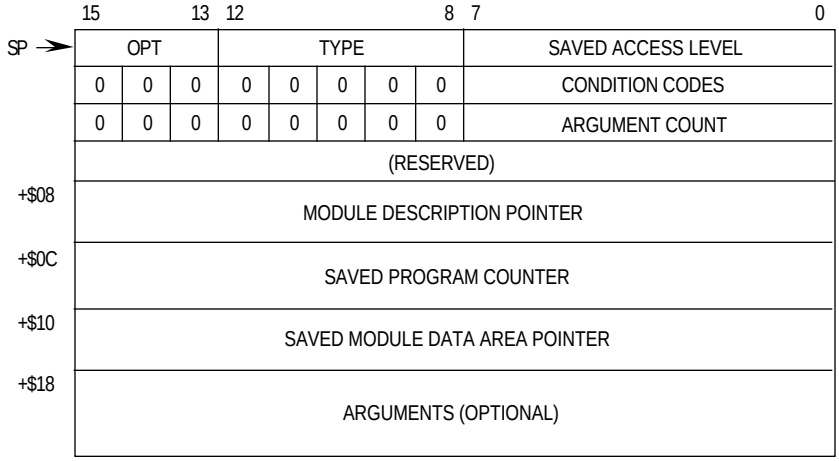


Figure 9-12. Module Call Stack Frame

9.8 ACCESS LEVELS

The MC68020/EC020 module mechanism supports a finer level of access control beyond the distinction between user and supervisor privilege levels. The module mechanism allows a module with limited access rights to call a module with greater access rights. With the help of external hardware, the processor can verify that an increase in access rights is allowable or can detect attempts by a module to gain access rights to which it is not entitled.

Type $01 module descriptors and module stack frames indicate a request to change access levels. While processing a type $01 descriptor or frame, the CALLM and RTM instructions communicate with external access control hardware via accesses in the CPU space. For these accesses, A19–A16 equal 0001. Figure 9-13 shows the address map for these CPU space accesses. If the processor receives a bus error on any of these CPU space accesses during the execution of a CALLM or RTM instruction, the processor will take a format error exception.

	31 — 24	23 — 0
$00	CAL	(UNUSED, RESERVED)
$04	ACCESS STATUS REGISTER	(UNUSED, RESERVED)
$08	IAL	(UNUSED, RESERVED)
$0C	DAL	(UNUSED, RESERVED)

$40	FUNCTION CODE 0 DESCRIPTOR ADDRESS
$44	FUNCTION CODE 1 DESCRIPTOR ADDRESS (USER DATA)
$48	FUNCTION CODE 2 DESCRIPTOR ADDRESS (USER PROGRAM)
$4C	FUNCTION CODE 3 DESCRIPTOR ADDRESS
$50	FUNCTION CODE 4 DESCRIPTOR ADDRESS (SUPERVISOR DATA)
$54	FUNCTION CODE 5 DESCRIPTOR ADDRESS (SUPERVISOR PROGRAM)
$58	FUNCTION CODE 6 DESCRIPTOR ADDRESS
$5C	FUNCTION CODE 7 DESCRIPTOR ADDRESS (CPU SPACE)

Figure 9-13. Access Level Control Bus Registers

The current access level register (CAL) contains the access level rights of the currently executing module. The increase access level register (IAL) is the register through which the processor requests increased access rights. The decrease access level register (DAL) is the register through which the processor requests decreased access rights. The formats of these three registers are undefined to the main processor, but the main processor assumes that information read from the module descriptor stack frame or the CAL can be meaningfully written to the IAL or the DAL. The access status register allows the processor to query the external hardware as to the legality of intended access level transitions. Table 9-6 lists the valid values of the access status register.

Table 9-6. Access Status Register Codes

Value	Validity	Processor Action
$00	Invalid	Format Error
$01	Valid	No Change in Access Rights
$02–$03	Valid	Change Access Rights with No Change of Stack Pointer
$04–$07	Valid	Change Access Rights and Change Stack Pointer
Other	Undefined	Undefined (Take Format Error Exception)

The processor uses the descriptor address registers during the CALLM instruction to communicate the address of the type $01 descriptor, allowing external hardware to verify that the address is a valid address for a type $01 descriptor. This validation prevents a module from creating a type $01 descriptor to increase its access rights.

9.8.1 Module Call

The CALLM instruction is used to make the module call. For the type $00 module descriptor, the processor creates and fills the module stack frame at the top of the active system stack. The condition codes of the calling module are saved in the CCR field of the frame. If opt is equal to 000 (arguments passed on the stack) in the module descriptor, the MC68020/EC020 does not save the stack pointer or load a new stack pointer value. The processor uses the module entry word to save and load the module data area pointer register and then begins execution of the called module.

For the type $01 module descriptor, the processor must first obtain the current access level from external hardware. It also verifies that the calling module has the right to read from the area pointed to by the current value of the stack pointer by reading from that address. It passes the descriptor address and increase access level to external hardware for validation and then reads the access status. If external hardware determines that the change in access rights should not be granted, the access status is zero, and the processor takes a format error exception. No visible processor registers are changed, nor should the current access level enforced by external hardware be changed. If external hardware determines that a change should be granted, the external hardware changes its access level, and the processor proceeds. If the access status register indicates that a change in the stack pointer is required, the stack pointer is saved internally, a new value is loaded from the module descriptor, and arguments are copied from the calling stack to the new stack. Finally, the module stack frame is created and filled on the top of the current stack. The condition codes of the calling module are saved in the CCR field of the frame. Execution of the called module then begins as with a type $00 descriptor.

9.8.2 Module Return

The RTM instruction is used to return from a module. For the type $00 module stack frame, the processor reloads the condition codes, the PC, and the module data area pointer register from the frame. The frame is removed from the top of the stack, the argument count is added to the stack pointer, and execution returns to the calling module.

For the type $01 module stack frame, the processor reads the access level, condition codes, PC, saved module data area pointer, and saved stack pointer from the module stack frame. The access level is written to the DAL for validation by external hardware; the processor then reads the access status to check the validation. If the external hardware determines that the change in access right should not be granted, the access status is zero, and the processor takes a format error exception. No visible processor registers are changed, nor should the current access level enforced by external hardware be changed. If the external hardware determines that the change in access rights should be granted, the external hardware changes its access level, the values read from the module stack frame are loaded into the corresponding processor registers, the argument count is added to the new stack pointer value, and execution returns to the calling module.

If the called module does not wish the saved module data pointer to be loaded into a register, the RTM instruction word can select register A7, and the loaded value will be overwritten with the correct stack pointer value after the module stack frame is deallocated.