Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

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

Applications of Embedded - Microprocessors

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

Details	
Product Status	Obsolete
Core Processor	68020
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	20MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	-40°C ~ 85°C (TA)
Security Features	-
Package / Case	114-BPGA
Supplier Device Package	114-PGA (34.55x34.55)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68020crc20e

LIST OF ILLUSTRATIONS (Continued)

MC68020/EC020 ACRONYM LIST

BCD — Binary-Coded Decimal
CAAR — Cache Address Register
CACR — Cache Control Register
CCR — Condition Code Register
CIR — Coprocessor Interface Register
CMOS — Complementary Metal Oxide Semiconductor
CPU — Central Processing Unit
CQFP — Ceramic Quad Flat Pack
DDMA — Dual-Channel Direct Memory Access
DFC — Destination Function Code Register
DMA — Direct Memory Access
DRAM — Dynamic Random Access Memory
FPCP — Floating-Point Coprocessor
HCMOS — High-Density Complementary Metal Oxide Semiconductor
IEEE — Institute of Electrical and Electronic Engineers
ISP — Interrupt Stack Pointer
LMB — Lower Middle Byte
LRAR — Limited Rate Auto Request
LSB — Least Significant Byte
MMU — Memory Management Unit
MPU — Microprocessor Unit
MSB — Most Significant Byte
MSP — Master Stack Pointer
NMOS — n-Type Metal Oxide Semiconductor
PAL — Programmable Array Logic
PC — Program Counter
PGA — Pin Grid Array
PMMU — Paged Memory Management Unit
PPGA — Plastic Pin Grid Array
PQFP — Plastic Quad Flat Pack
RAM — Random Access Memory
SFC — Source Function Code Register
SP — Stack Pointer
SR — Status Register
SSP — Supervisor Stack Pointer
SSW — Special Status Word
UMB — Upper Middle Byte
USP — User Stack Pointer
VBR — Vector Base Register
VLSI — Very Large Scale Integration

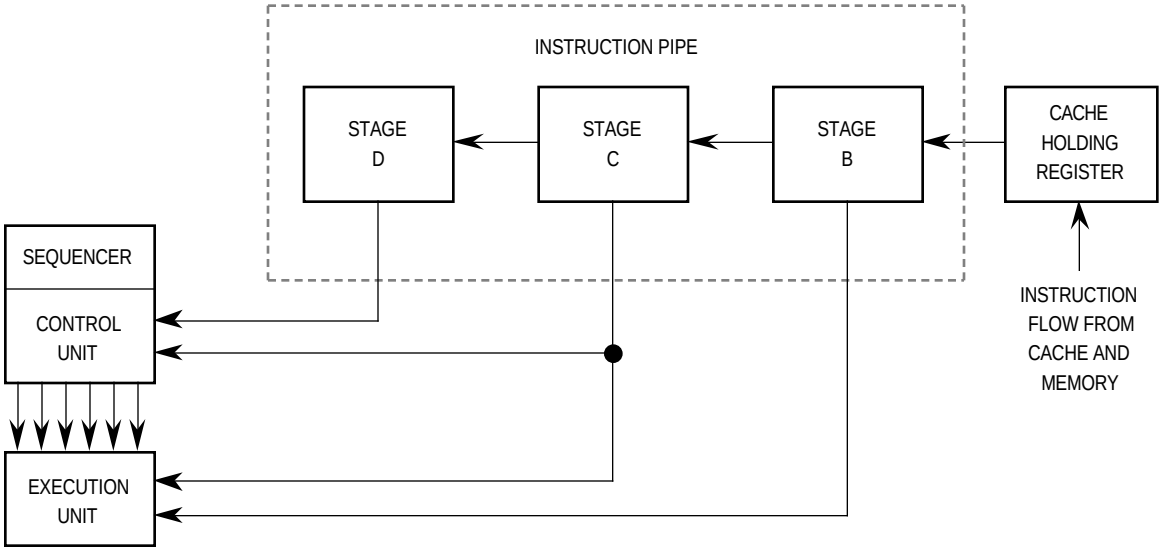


Figure 1-5. Instruction Pipe

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

Prefetch requests are simultaneously submitted to the cache holding register, the instruction cache, and the bus controller. Thus, even if the instruction cache is disabled, an instruction prefetch may hit in the cache holding register and cause an external bus cycle to be aborted.

1.7 CACHE MEMORY

Due to locality of reference, instructions that are used in a program have a high probability of being reused within a short time. Additionally, instructions that reside in proximity to the instructions currently in use also have a high probability of being utilized within a short period. To exploit these locality characteristics, the MC68020/EC020 contains an on-chip instruction cache.

The cache improves the overall performance of the system by reducing the number of bus cycles required by the processor to fetch information from memory and by increasing the bus bandwidth available for other bus masters in the system.

2.1 PRIVILEGE LEVELS

The processor operates at one of two privilege levels: the user level or the supervisor level. The supervisor level has higher privileges than the user level. Not all processor or coprocessor instructions are permitted to execute at the lower privileged user level, but all are available at the supervisor level. This arrangement allows a separation of supervisor and user so the supervisor can protect system resources from uncontrolled access. The S-bit in the SR is used to select either the user or supervisor privilege level and either the USP or an SSP for stack operations. The processor identifies a bus access (supervisor or user mode) via the function codes so that differentiation between supervisor level and user level can be maintained.

In many systems, the majority of programs execute at the user level. User programs can access only their own code and data areas and can be restricted from accessing other information. The operating system typically executes at the supervisor privilege level. It has access to all resources, performs the overhead tasks for the user-level programs, and coordinates user-level program activities.

2.1.1 Supervisor Privilege Level

The supervisor level is the higher privilege level. The privilege level is determined by the S-bit of the SR; if the S-bit is set, the supervisor privilege level applies, and all instructions are executable. The bus cycles for instructions executed at the supervisor level are normally classified as supervisor references, and the values of the FC2–FC0 signals refer to supervisor address spaces.

In a multitasking operating system, it is more efficient to have a supervisor stack space associated with each user task and a separate stack space for interrupt-associated tasks. The MC68020/EC020 provides two supervisor stacks, master and interrupt; the M bit of the SR selects which of the two is active. When the M-bit is set, references to the SSP implicitly or to address register seven (A7) explicitly, access the MSP. The operating system sets the MSP for each task to point to a task-related area of supervisor data space. This arrangement separates task-related supervisor activity from asynchronous, I/O-related supervisor tasks that may be only coincidental to the currently executing task. The MSP can separately maintain task control information for each currently executing user task, and the software updates the MSP when a task switch is performed, providing an efficient means for transferring task-related stack items. The other supervisor stack pointer, the ISP, can be used for interrupt control information and workspace area as interrupt handling routines require.

When the M-bit is clear, the MC68020/EC020 is in the interrupt mode of the supervisor privilege level, and operation is the same as supervisor mode in the MC68000, MC68HC001, MC68008, and MC68010. (The processor is in this mode after a reset operation.) All SSP references access the ISP in this mode.

Data Buffer Enable (DBEN, MC68020 only)

This output signal is an enable signal for external data buffers. This signal may not be required in all systems. Refer to **Section 5 Bus Operation** for more information about the relationship of DBEN to bus operation.

DBEN is not implemented in the MC68EC020.

Data Transfer and Size Acknowledge (DSACK1, DSACK0)

These input signals indicate the completion of a requested data transfer operation. In addition, they indicate the size of the external bus port at the completion of each cycle. These signals apply only to asynchronous bus cycles. Refer to **Section 5 Bus Operation** for more information on these signals and their relationship to dynamic bus sizing.

3.7 INTERRUPT CONTROL SIGNALS

The following signals are the interrupt control signals for the MC68020/EC020. Note that IPEND is implemented in the MC68020 and not implemented in the MC68EC020.

Interrupt Priority Level Signals (IPL2–IPL0)

These input signals provide an indication of an interrupt condition and the encoding of the interrupt level from a peripheral or external prioritizing circuitry. IPL2 is the most significant bit of the level number. For example, since the IPL2–IPL0 signals are active low, IPL2–IPL0 equal to $5 corresponds to an interrupt request at interrupt level 2. Refer to **Section 6 Exception Processing** for information on MC68020/EC020 interrupts.

Interrupt Pending (IPEND, MC68020 only)

This output signal indicates that an interrupt request exceeding the current interrupt priority mask in the SR has been recognized internally. This output is for use by external devices (coprocessors and other bus masters, for example) to predict processor operation on the following instruction boundaries. Refer to **Section 6 Exception Processing** for interrupt information. Also, refer to **Section 5 Bus Operation** for bus information related to interrupts.

IPEND is not implemented in the MC68EC020.

Autovector (AVEC)

This input signal indicates that the MC68020/EC020 should generate an automatic vector during an interrupt acknowledge cycle. Refer to **Section 5 Bus Operation** for more information about automatic vectors.

Figure 5-7 shows a word write to an 8-bit bus port. Like the preceding example, this example requires two bus cycles. Each bus cycle transfers a single byte. SIZ1 and SIZ0 for the first cycle specify two bytes; for the second cycle, one byte. Figure 5-8 shows the associated bus transfer signal timing.

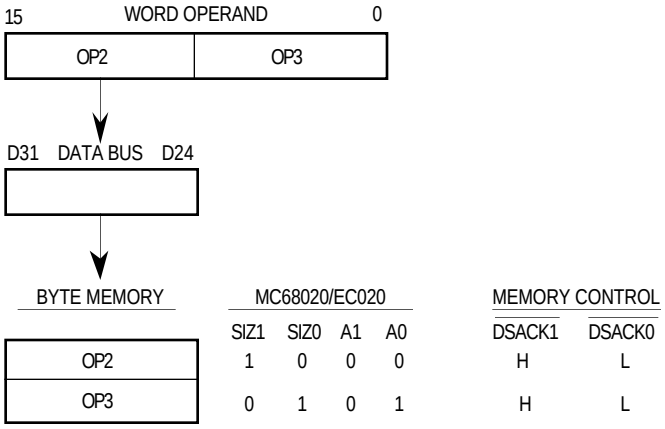


Figure 5-7. Word Operand Write to Byte Port Example

5.4.3 Coprocessor Communication Cycles

The MC68020/EC020 coprocessor interface provides instruction-oriented communication between the processor and as many as eight coprocessors. Coprocessor accesses use the MC68020/EC020 bus protocol except that the address bus supplies access information rather than a 32-bit address. The CPU space type field (A19–A16) for a coprocessor operation is 0010. A15–A13 contain the coprocessor identification number (CpID), and A5–A0 specify the coprocessor interface register to be accessed. The memory management unit of an MC68020/EC020 system is always identified by a CpID of zero and has an extended register select field (A7–A0) in CPU space 0001 for use by the CALLM and RTM access level checking mechanism. Refer to **Section 9 Applications Information** for more details.

5.5 BUS EXCEPTION CONTROL CYCLES

The MC68020/EC020 bus architecture requires assertion of $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ from an external device to signal that a bus cycle is complete. $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ or $\overline{\text{AVEC}}$ is not asserted if:

- The external device does not respond,
- No interrupt vector is provided, or
- Various other application-dependent errors occur.

External circuitry can assert $\overline{\text{BERR}}$ when no device responds by asserting $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ or $\overline{\text{AVEC}}$ within an appropriate period of time after the processor asserts $\overline{\text{AS}}$. Assertion of $\overline{\text{BERR}}$ allows the cycle to terminate and the processor to enter exception processing for the error condition.

$\overline{\text{HALT}}$ is also used for bus exception control. $\overline{\text{HALT}}$ can be asserted by an external device for debugging purposes to cause single bus cycle operation or can be asserted in combination with $\overline{\text{BERR}}$ to cause a retry of a bus cycle in error.

To properly control termination of a bus cycle for a retry or a bus error condition, $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ can be asserted and negated with the rising edge of the MC68020/EC020 clock. This procedure ensures that when two signals are asserted simultaneously, the required setup time (#47A) and hold time (#47B) for both of them is met for the same falling edge of the processor clock. (Refer to **Section 10 Electrical Characteristics** for timing requirements.) This or some equivalent precaution should be designed into the external circuitry that provides these signals.

The acceptable bus cycle terminations for asynchronous cycles are summarized in relation to $\overline{DSACK1}/\overline{DSACK0}$ 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

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 SR, enters the supervisor privilege level (by setting the S bit in the SR), and clears the T1 and T0 bits in the SR, 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 PC, and copy of the SR are saved on the supervisor stack, with the saved value of the PC 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 PC if the instruction is emulated in software or is to be skipped on return from the handler.

6.1.6 Privilege Violation Exception

To provide system security, the following instructions are privileged:

ANDI to SR
EORI to SR
cpRESTORE
cpSAVE
MOVE from SR
MOVE to SR
MOVE USP
MOVEC
MOVES
ORI to SR
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.

Exception processing for privilege violations is similar to that for illegal instructions. When the processor identifies a privilege violation, it begins exception processing before executing the instruction. The processor copies the SR, enters the supervisor privilege level by setting the S-bit in the SR, and clears the T1 and T0 bits in the SR. The processor generates vector number 8, the privilege violation exception vector, and saves the privilege violation vector offset, the current PC value, and the internal copy of the SR on the supervisor stack. The saved value of the PC is the logical address of the first word of the instruction that caused the privilege violation. Instruction execution resumes after the required prefetches from the address in the privilege violation exception vector.

6.1.10 Breakpoint Instruction Exception

To use the MC68020/EC020 in a hardware emulator, it must provide a means of inserting breakpoints in the emulator code and of performing appropriate operations at each breakpoint. For the MC68000 and MC68008, this can be done by inserting an illegal instruction at the breakpoint and detecting the illegal instruction exception from its vector location. However, since the VBR on M68000 family processors MC68010 and later allows arbitrary relocation of exception vectors, the exception address cannot reliably identify a breakpoint. The MC68020/EC020 processor provides a breakpoint capability with a set of breakpoint instructions, $4848–$484F, for eight unique breakpoints. The breakpoint facility also allows external hardware to monitor the execution of a program residing in the on-chip instruction cache without severe performance degradation.

When the MC68020/EC020 executes a breakpoint instruction, it performs a breakpoint acknowledge cycle (read cycle) from CPU space type $0 with address lines A4–A2 corresponding to the breakpoint number. Refer to **Section 5 Bus Operation** for a description of the breakpoint acknowledge cycle. The external hardware can return either BERR or DSACK1/DSACK0 with an instruction word on the data bus. If the bus cycle terminates with BERR, the processor performs illegal instruction exception processing. If the bus cycle terminates with DSACK1/DSACK0, the processor uses the data returned to replace the breakpoint instruction in the internal instruction pipe and begins execution of that instruction. The remainder of the pipe remains unaltered. In addition, no stacking or vector fetching is involved with the execution of the instruction. Figure 6-6 is a flowchart of the breakpoint instruction execution.

6.1.11 Multiple Exceptions

When several exceptions occur simultaneously, they are processed according to a fixed priority. Table 6-4 lists the exceptions grouped by characteristics. Each group has a priority from 4–0. Priority 0 has the highest priority.

As soon as the MC68020/EC020 has completed exception processing for a condition when another exception is pending, it begins exception processing for the pending exception instead of executing the exception handler for the original exception condition. Also, whenever a bus error or address error occurs, its exception processing takes precedence over lower priority exceptions and occurs immediately. For example, if a bus error occurs during the exception processing for a trace condition, the system processes the bus error and executes its handler before completing the trace exception processing. However, most exceptions cannot occur during exception processing, and very few combinations of the exceptions shown in Table 6-4 can be pending simultaneously.

The take address and transfer data primitive described in **7.4.11 Take Address and Transfer Data Primitive** does not replace the effective address value that has been calculated by the MC68020/EC020. The address that the main processor obtains in response to the take address and transfer data primitive is not available to the write to previously evaluated effective address primitive.

A coprocessor can issue an evaluate effective address and transfer data primitive followed by this primitive to perform a read-modify-write operation that is not indivisible. The bus cycles for this operation are normal bus cycles that can be interrupted, and the bus can be arbitrated between the cycles.

7.4.11 Take Address and Transfer Data Primitive

The take address and transfer data primitive transfers an operand between the coprocessor and an address supplied by the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-31 shows the format of the take address and transfer data primitive.

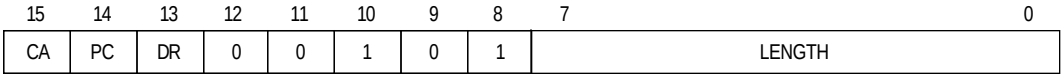


Figure 7-31. Take Address and Transfer Data Primitive Format

The take address and transfer data primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If the coprocessor issues this primitive with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

The length field of the primitive format specifies the operand length, which can be from 0–255 bytes.

The main processor reads a 32-bit address from the operand address CIR. Using a series of long-word transfers, the processor transfers the operand between this address and the operand CIR. The DR bit determines the direction of the transfer. The processor reads or writes the operand parts to ascending addresses, starting at the address from the operand address CIR. If the operand length is not a multiple of four bytes, the final operand part is transferred using a one-, two-, or three-byte transfer as required.

The function code used with the address read from the operand address CIR indicates either supervisor or user data space according to the value of the S-bit in the MC68020/EC020 SR.

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

7.4.15 Transfer Multiple Main Processor Registers Primitive

The transfer multiple main processor registers primitive transfers long-word operands between one or more of its data or address registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 7-35 shows the format of the transfer multiple main processor registers primitive.

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

Figure 7-35. Transfer Multiple Main Processor Registers Primitive Format

The transfer multiple main processor registers primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**. If the coprocessor issues this primitive with CA = 0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the main processor receives this primitive, it reads a 16-bit register select mask from the register select CIR. The format of the register select mask is shown in Figure 7-36. A register is transferred if the bit corresponding to the register in the register select mask is set. The selected registers are transferred in the order D7–D0 and then A7–A0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

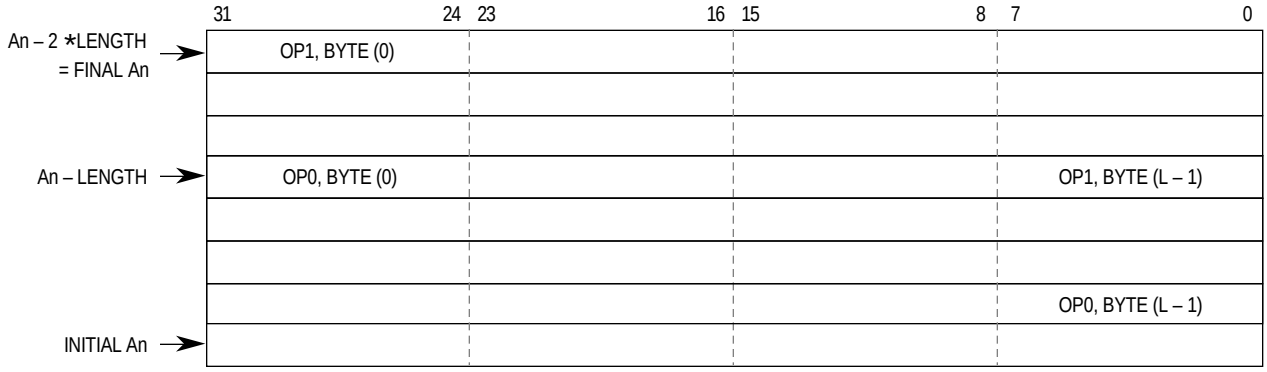
Figure 7-36. Register Select Mask Format

If DR = 0, the main processor writes the contents of each register indicated in the register select mask to the operand CIR using a sequence of long-word transfers. If DR = 1, the main processor reads a long-word operand from the operand CIR into each register indicated in the register select mask. The registers are transferred in the same order, regardless of the direction of transfer indicated by the DR bit.

7.4.16 Transfer Multiple Coprocessor Registers Primitive

The transfer multiple coprocessor registers primitive transfers from 0–16 operands between the effective address specified in the coprocessor instruction and the coprocessor. This primitive applies to general category instructions. If the coprocessor issues this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-37 shows the format of the transfer multiple coprocessor registers primitive.

For the predecrement addressing mode, the operands are written to memory with descending addresses, but the bytes within each operand are written to memory with ascending addresses. As an example, Figure 7-38 shows the format in long-word-oriented memory for two 12-byte operands transferred from the coprocessor to the effective address using the –(An) addressing mode. The processor decrements the address register by the size of an operand before the operand is transferred. It writes the bytes of the operand to ascending memory addresses. When the transfer is complete, the address register has been decremented by the total number of bytes transferred. The MC68020/EC020 transfers the data using long-word transfers whenever possible.



NOTE: OP0, Byte (0) is the first byte written to memory
OP0, Byte (L–1) is the last byte of the first operand written to memory
OP1, Byte (0) is the first byte of the second operand written to memory
OP1, Byte (L–1) is the last byte written to memory

Figure 7-38. Operand Format in Memory for Transfer to –(An)

7.4.17 Transfer Status Register and ScanPC Primitive

The transfer status register and the scanPC primitive transfers values between the coprocessor and the MC68020/EC020 SR. On an optional basis, the scanPC also makes transfers. This primitive applies to general category instructions. If the coprocessor issues this primitive during the execution of a conditional category instruction, the main processor initiates protocol violation exception processing. Figure 7-39 shows the format of the transfer status register and scanPC primitive.

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

Figure 7-39. Transfer Status Register and ScanPC Primitive Format

The transfer status register and scanPC primitive uses the CA, PC, and DR bits as described in **7.4.2 Coprocessor Response Primitive General Format**.

The SP bit selects the scanPC option. If SP = 1, the primitive transfers both the scanPC and SR. If SP = 0, only the SR is transferred.

SECTION 8
INSTRUCTION EXECUTION TIMING

This section describes the instruction execution and operations (table searches, etc.) of the MC68020/EC020 in terms of external clock cycles. It provides accurate execution and operation timing guidelines but not exact timings for every possible circumstance. This approach is used since exact execution time for an instruction or operation is highly dependent on memory speeds and other variables. The timing numbers presented in this section allow the assembly language programmer or compiler writer to predict timings needed to evaluate the performance of the MC68020/EC020.

In this section, instruction and operation times are shown in clock cycles, which eliminates clock frequency dependencies.

8.1 TIMING ESTIMATION FACTORS

The advanced architecture of the MC68020/EC020 makes exact instruction timing calculations difficult due to the effects of:

1. An On-Chip Instruction Cache and Instruction Prefetch
2. Operand Misalignment
3. Bus Controller/Sequence Concurrency
4. Instruction Execution Overlap

These factors make MC68020/EC020 instruction set timing difficult to calculate on a single instruction basis since instructions vary in execution time from one context to another. A detailed explanation of each of these factors follows.

8.1.1 Instruction Cache and Prefetch

The on-chip cache of the MC68020/EC020 is an instruction-only cache. Its purpose is to increase execution efficiency by providing a quick store for instructions.

Instruction prefetches that hit in the cache will occur with no delay in instruction execution. Instruction prefetches that miss in the cache will cause an external memory cycle to be performed, which may overlap with internal instruction execution. Thus, while the execution unit of the microprocessor is busy, the bus controller prefetches the next instruction from external memory. Both cases are illustrated in later examples.

8.2.9 Immediate Arithmetic/Logical Instructions

The immediate arithmetic/logical instructions table indicates the number of clock periods needed for the processor to fetch the source immediate data value and perform the specified arithmetic/logical operation using the specified destination addressing mode. Footnotes indicate when to add appropriate fetch effective or fetch immediate effective address time. This computation will give the total execution time needed to perform the appropriate immediate arithmetic/logical operation. 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
	MOVEQ	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
	ADDQ	#<data>,Rn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	ADDQ	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	SUBQ	#<data>,Rn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	SUBQ	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	ADDI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	ADDI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	ANDI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	ANDI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	EORI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	EORI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	ORI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	ORI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	SUBI	#<data>,Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
**	SUBI	#<data>,Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
**	CMPI	#<data>,EA	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)

*Add Fetch Effective Address Time

**Add Fetch Immediate Address Time

8.2.11 Single-Operand Instructions

The single-operand instructions table indicates the number of clock periods needed for the processor to perform the specified 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
	CLR	Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
†	CLR	Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	NEG	Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	NEG	Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	NEGX	Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	NEGX	Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	NOT	Dn	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)
*	NOT	Mem	**3**(0/0/1)	**4**(0/0/1)	**6**(0/1/1)
	EXT	Dn	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)
	NBCD	Dn	**6**(0/0/0)	**6**(0/0/0)	**6**(0/1/0)
	Scc	Dn	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)
†	Scc	Mem	**6**(0/0/1)	**6**(0/0/1)	**6**(0/1/1)
	TAS	Dn	**1**(0/0/0)	**4**(0/0/0)	**4**(0/1/0)
†	TAS	Mem	**12**(1/0/1)	**12**(1/0/1)	**13**(1/1/1)
*	TST	EA	**0**(0/0/0)	**2**(0/0/0)	**3**(0/1/0)

*Add Fetch Effective Address Time

†Add Calculate Effective Address Time

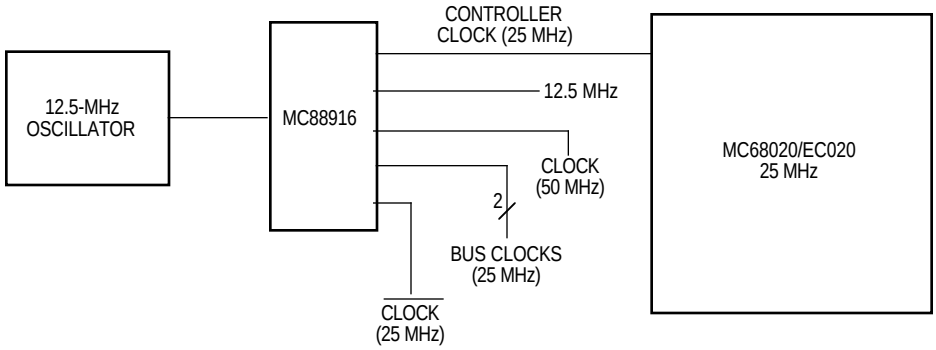


Figure 9-7. High-Resolution Clock Controller

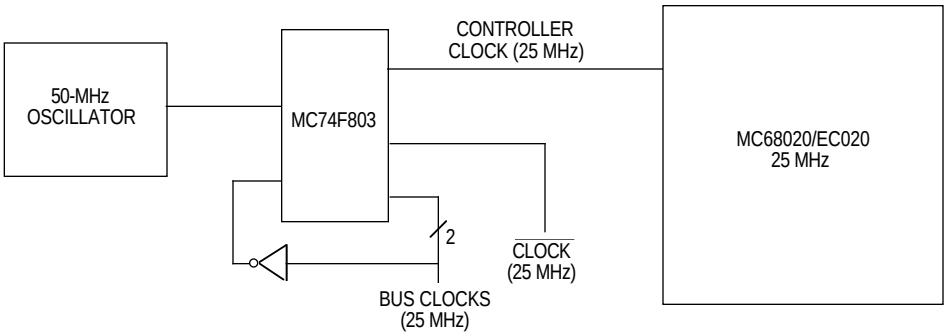


Figure 9-8. Alternate Clock Solution

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.

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

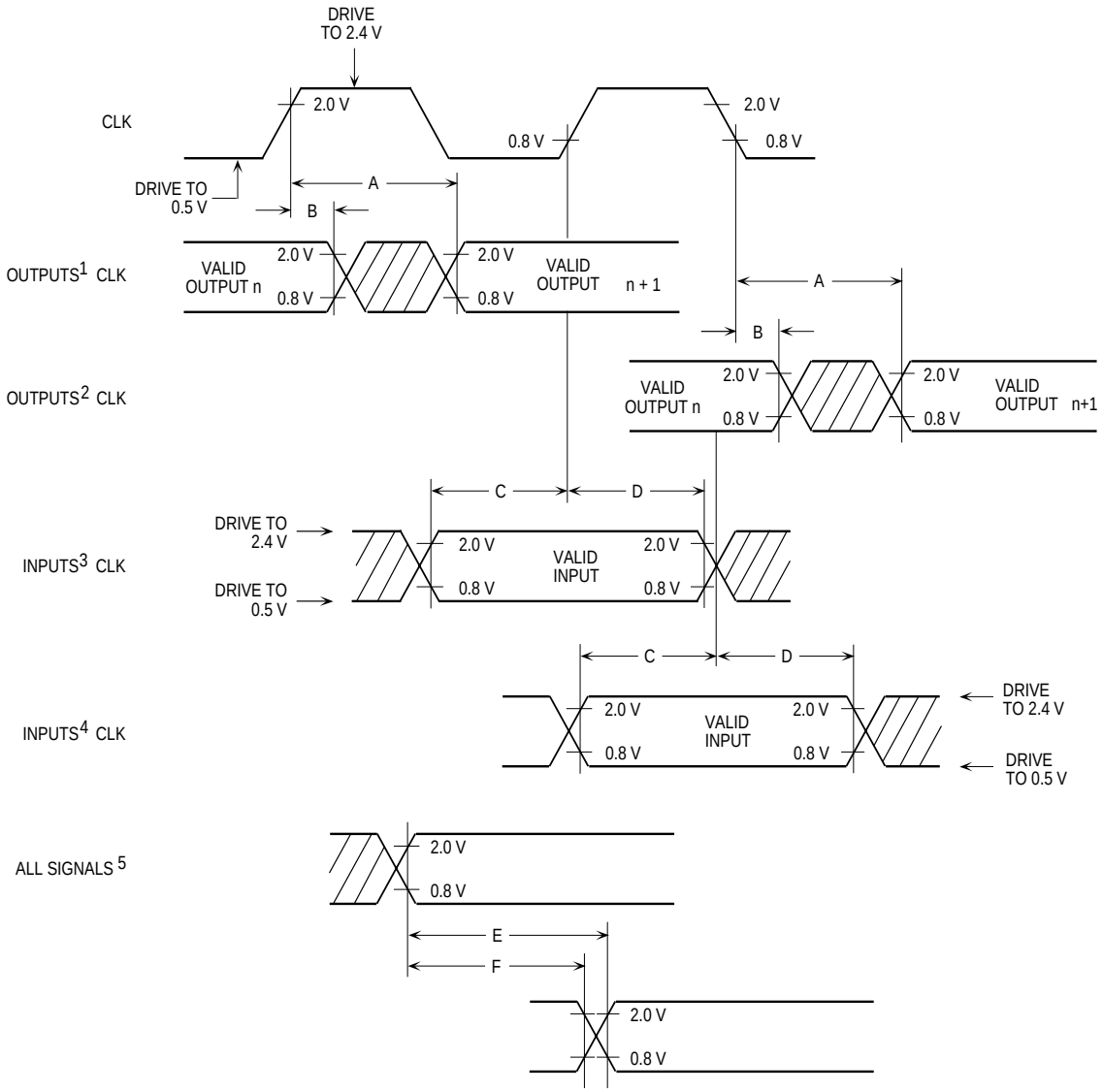
Equation	25 MHz	N = 3	N = 4	N = 5	N = 6	N = 7	Unit
9-3	t_{AVDL} = (N – 1) • t1 – t2 – t6 – t47A	31	71	111	151	191	ns
9-4	t_{SADL} = (N – 1) • t1 – t9 – t60	17	57	97	137	177	ns
9-5	t_{AVBHL} = N • t1 – t2 – t6 – t27A	22	41	60	79	98	ns
9-6	t_{SABHL} = (N – 1) • t1 – t9 – t27A	26	44	62	80	98	ns
9-7	t_{AVDV}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



NOTES:
1. This output timing is applicable to all parameters specified relative to the rising edge of the clock.
2. This output timing is applicable to all parameters specified relative to the falling edge of the clock.
3. This input timing is applicable to all parameters specified relative to the rising edge of the clock.
4. This input timing is applicable to all parameters specified relative to the falling edge of the clock.
5. This timing is applicable to all parameters specified relative to the assertion/negation of another signal.

LEGEND:
A. Maximum output delay specification.
B. Minimum output hold time.
C. Minimum input setup time specification.
D. Minimum input hold time specification.
E. Signal valid to signal valid specification (maximum or minimum).
F. Signal valid to signal invalid specification (maximum or minimum).

FIGURE 10-1
MC68020UM

Figure 10-1. Drive Levels and Test Points for AC Specifications