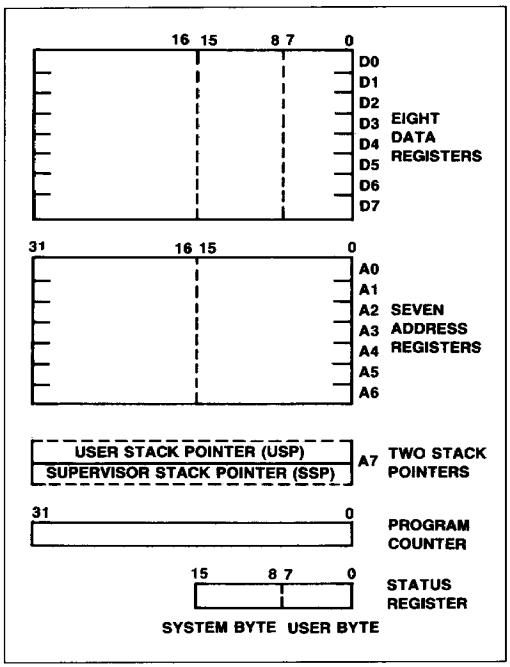
R68000
16-Bit Microprocessing Unit (MPU)

DESCRIPTION

The R68000 microprocessor is designed for high performance where operational computation and versatility is required. The R68000 provides powerful mass-memory handling capability and architectural features designed to fit the broad range of 16-bit needs. The Rockwell family of 16-bit products also includes a wide range of peripherals that will allow complete system design and manufacture.



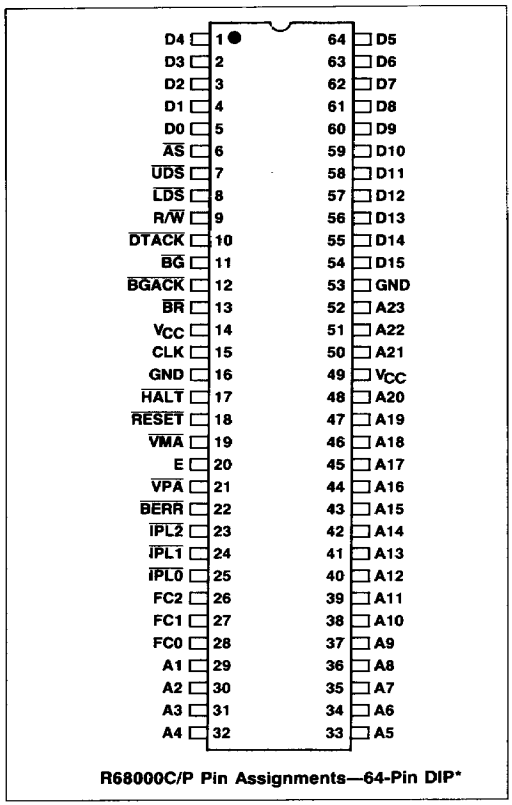
R68000 Registers

The R68000 offers seventeen 32-bit registers in addition to the 32-bit program counter and a 16-bit status register. The first eight registers (D0-D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) data operations. The second set of seven registers (A0-A6) and the system stack pointer may be used as software stack pointers and base address registers. In addition, these registers may be used for word and long word address operations. All 17 registers may be used as index registers.

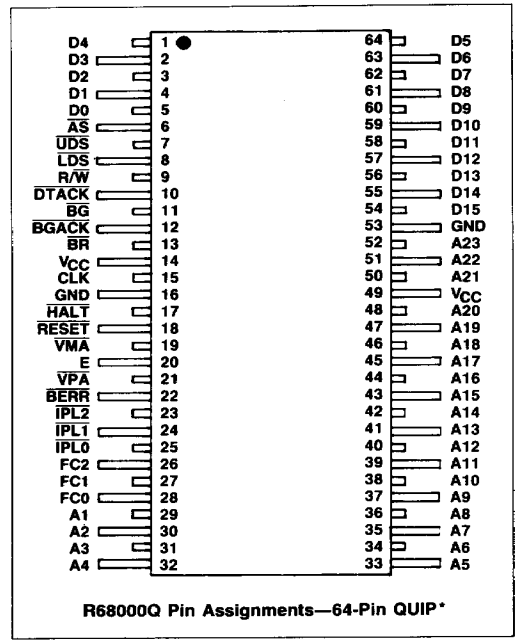
FEATURES

- 16M byte (8M word) Linear Addressing Range
- 14 Operand Addressing Modes
- 56 Powerful Instruction Types
- Instruction Set Supports Structured High-Level Languages
- Pipelining Instruction Execution
- 32-Bit Program Counter
- 16-Bit Data Bus
- 23-Line Address Bus
- 32-Bit Data and Address Registers Including:
  - — Eight General Purpose Data Registers
  - — Seven Address Registers
  - — Two Stack Pointers (User, Supervisory)
- All 17 Registers Can Be Index Registers
- Memory Mapped Peripheral Devices
- Vector Generated Exception Processing
- Seven Unique Autovectors for Interrupt Service Routines
- Trace Mode for Software Debugging
- Operations Occur on Five Main Data Types
  - — Bit
  - — BCD
  - — Byte
  - — Word
  - — Long Word
- Asynchronous and Synchronous Peripheral Interface Capability
- Many Peripheral Chips Available
  - — R68560/R68561 Multi-Protocol Communications Controller (MPCC)
  - — R68802 Local Network Controller (LNET)
  - — R68C552 Dual Asynchronous Communications Interface Adapter (DACIA)
- Up to 12.5 MHz Input Clock
- +5 VDC Power Supply

4



R68000C/P Pin Assignments—64-Pin DIP*



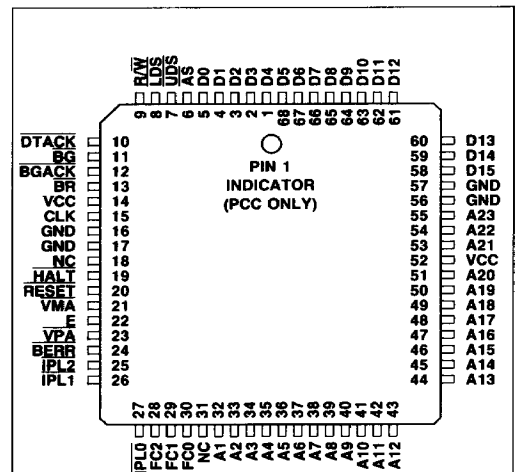
R68000Q Pin Assignments—64-Pin QUIP*

ORDERING INFORMATION

Part Number:

R68000

- Temperature Range (T_L to T_H) and Screening/Burn-in
  - Blank = 0°C to 70°C
  - M = −55°C to 125°C
  - R = −55°C to 125°C and burned-in 168 hours
  - B = −55°C to 125°C and screened to MIL-STD 883B
- Frequency
  - 6 = 6 MHz
  - 8 = 8 MHz
  - 10 = 10 MHz
  - 12 = 12.5 MHz
- Package
  - C = 64-Pin Ceramic DIP
  - P = 64-Pin Plastic DIP
  - Q = 64-Pin Plastic QUIP
  - J = 64-Pin Plastic Leaded Chip Carrier (PCC)
  - L = 64-Pin Leadless Ceramic Chip Carrier (LCCC)



R68000J/L Pin Assignments—68-Pin PCC and LCC*

*Pin Assignment package outlines are not actual size (refer to PACKAGE DIMENSIONS on pages 56–59).

SIGNAL DESCRIPTION

The following paragraphs briefly describe the input and output signals and also reference (if applicable) other paragraphs that contain more detail about the function being performed. Bus operation during the various machine cycles and operations is also discussed. The input and output signals can be functionally organized into the groups shown in Figure 1.

Note

The terms assertion and negation are used to avoid confusion when dealing with a mixture of "active-low" and "active-high" signals. The terms assert, or assertion, indicates that a signal is active, or true, independent of whether that voltage is low or high. The term negate, or negation, indicates that a signal is inactive or false.

ADDRESS BUS (A1 THROUGH A23). This 23-bit, unidirectional, three-state bus can address eight megawords of data. It provides the address for bus operation during all cycles except interrupt cycles. During interrupt cycles, address lines A1, A2, and A3 encode the interrupt level to be serviced while address lines A4 through A23 are all set high.

DATA BUS (D0 THROUGH D15). This 16-bit, bidirectional, three-state bus is the general purpose data path. It transfers and accepts data in either word or byte length. During an interrupt acknowledge cycle, an external device supplies the vector number on data lines D0-D7.

ASYNCHRONOUS BUS CONTROL. Asynchronous data transfers are handled using the following control signals: address strobe, read/write, upper and lower data strobes, and data transfer acknowlege. These signals are explained in the following paragraphs.

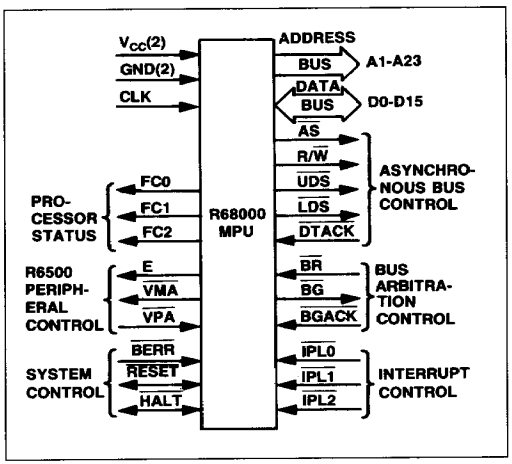


Figure 1. Input and Output Signals

Address Strobe ($\overline{AS}$). The $\overline{AS}$ output indicates that there is a valid address on the address bus.

Read/Write (R/$\overline{W}$). The R/$\overline{W}$ output defines the data bus transfer as a read or write cycle. The R/$\overline{W}$ signal also works in conjunction with the upper and lower data strobes as explained in the following paragraph.

Upper and Lower Data Strobes ($\overline{UDS}$, $\overline{LDS}$). The $\overline{UDS}$ and $\overline{LDS}$ outputs control the data on the data bus, as shown in Table 1. When the R/$\overline{W}$ line is high, the processor reads from the data bus as indicated. When the R/$\overline{W}$ line is low, the processor writes to the data bus as shown.

Data Transfer Acknowledge ($\overline{DTACK}$). The $\overline{DTACK}$ input indicates that the data transfer is completed. When the processor recognizes $\overline{DTACK}$ during a read cycle, data is latched and the bus cycle terminated. When $\overline{DTACK}$ is recognized during a write cycle, the bus cycle terminates. Refer to **ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION.**

BUS ARBITRATION CONTROL. These three signals form a bus arbitration circuit to determine which device will be the bus master device.

Bus Request ($\overline{BR}$). The $\overline{BR}$ input indicates to the processor that some other device desires to become the bus master. This input can be externally ORed with all other devices that could be bus masters.

Bus Grant ($\overline{BG}$). The $\overline{BG}$ output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.

Bus Grant Acknowledge ($\overline{BGACK}$). The $\overline{BGACK}$ input indicates that some other device has become the bus master. This signal cannot be asserted until the following four conditions are met:

1. a bus grant ($\overline{BG}$) has been received,
2. address strobe ($\overline{AS}$) is inactive which indicates that the processor is not using the bus

Table 1. Data Strobe Control of Data Bus

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	D8-D15	D0-D7
High	High	—	No valid data	No valid data
Low	Low	High	Valid data bits 8-15	Valid data bits 0-7
High	Low	High	No valid data	Valid data bits 0-7
Low	High	High	Valid data bits 8-15	No valid data
Low	Low	Low	Valid data bits 8-15	Valid data bits 0-7
High	Low	Low	Valid data bits 0-7*	Valid data bits 0-7
Low	High	Low	Valid data bits 8-15	Valid data bits 8-15*

*These conditions are a result of current implementation and may not appear on future devices.

3. data transfer acknowledge (DTACK) is inactive which indicates that neither memory nor peripherals are using the bus, and
4. bus grant acknowledge (BGACK) is inactive which indicates that no other device is still claiming bus mastership.

INTERRUPT CONTROL (IPL0, IPL1, IPL2). These input pins indicate the encoded priority level of the device requesting an interrupt. Level seven is the highest priority while level zero indicates that no interrupts are requested. Level seven cannot be masked. IPL0 is the least significant bit while IPL2 is the most significant bit. To insure an interrupt is recognized, the interrupt control lines (IPLX) must remain stable until the processor signals interrupt acknowledge (FC0, FC1, and FC2 all high).

SYSTEM CONTROL. The system control inputs either reset or halt the processor or indicate to the processor that bus errors have occurred. The three system control inputs are explained in the following paragraphs.

Bus Error (BERR). The BERR input informs the processor that a problem exists with the cycle currently being executed. Problems may be a result of:
1. nonresponding devices,
2. interrupt vector number acquisition failure,
3. illegal access request as determined by a memory management unit, or
4. other application dependent errors.

The Bus Error (BERR) signal interacts with the HALT signal to determine if exception processing should be performed or the current bus cycle should be retried.

Refer to BUS ERROR AND HALT OPERATION paragraph for additional information about the interaction of the bus error and halt signals.

Reset (RESET). This bidirectional signal line acts to reset (initiate a system initialization sequence) the processor and system in response to an external reset signal. An internally generated reset (result of a RESET instruction) resets all external devices while not affecting the internal state of the processor. A total system reset (processor and external devices) is the result of external HALT and RESET signals applied simultaneously. Refer to RESET OPERATION paragraph for additional information.

Halt (HALT). The bidirectional HALT line, when driven by an external device, will cause the processor to stop at the completion of the current bus cycle. Halting the processor using HALT causes all control signals to go inactive and all three-state lines to go to their high-impedance state. Refer to BUS ERROR AND HALT OPERATION paragraph for additional information about the interaction between the HALT and BERR signals.

When the processor has stopped executing instructions, such as in a double bus fault condition, the HALT line is driven by the processor to indicate to external devices that the processor has stopped. Refer to paragaph on Double Bus Faults.

R6500 PERIPHERAL CONTROL. These control signals are used to allow the interfacing of synchronous R6500 peripheral devices with the asynchronous R68000. These signals are explained in the following paragraphs.

Enable (E). The E output signal is the standard enable signal (ϕ2 clock) common to all R6500 type peripheral devices. The period for this output is ten R68000 clock periods (six clocks low; four clocks high). Enable is generated by an internal ring counter which may come up in any state (i.e., at power on, it is impossible to guarantee phase relationship of E to CLK). E is a free-running clock and runs regardless of the state of the bus on the MPU.

Valid Peripheral Address (VPA). The VPA input indicates that the device or region addressed is a R6500 family device and that data transfer should be synchronized with the enable (E) signal. This input also indicates that the processor should use automatic vectoring for an interrupt. Refer to INTERFACE WITH R6500 PERIPHERALS.

Valid Memory Address (VMA). The VMA output indicates to R6500 peripheral devices that there is a valid address on the address bus and that the processor is synchronized to enable. This signal only responds to a valid peripheral address (VPA) input which indicates that the peripheral is a R6500 family device.

PROCESSOR STATUS (FC0, FC1, FC2). These function code outputs indicate the state (user or supervisor) and the cycle type currently being executed, as shown in Table 2. The information indicated by the function code outputs is valid whenever address strobe (AS) is active.

CLOCK (CLK). The clock input is a TTL-compatible signal that is internally buffered for development of the internal clocks needed by the processor. The clock input should not be gated off at any time and the clock signal must conform to minimum and maximum pulse width times.

SIGNAL SUMMARY. Table 3 summarizes all the signals discussed in the previous paragraphs.

Table 2. Function Code Outputs

FC2	FC1	FC0	Cycle Type
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

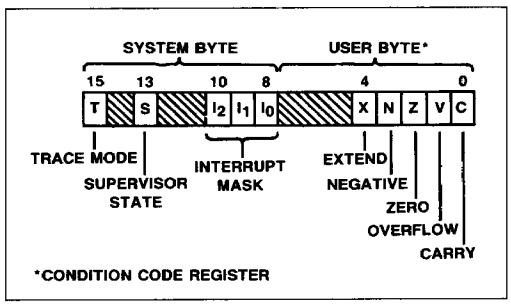
Table 3. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Hi-Z	
				On $\overline{HALT}$	On $\overline{BGACK}$
Address Bus	A1-A23	Output	High	Yes	Yes
Data Bus	D0-D15	Input/Output	High	Yes	Yes
Address Strobe	$\overline{AS}$	Output	Low	No	Yes
Read/Write	R/$\overline{W}$	Output	Read-High Write-Low	No	Yes
Upper and Lower Data Strobes	$\overline{UDS}$, $\overline{LDS}$	Output	Low	No	Yes
Data Transfer Acknowledge	$\overline{DTACK}$	Input	Low	No	No
Bus Request	$\overline{BR}$	Input	Low	No	No
Bus Grant	$\overline{BG}$	Output	Low	No	No
Bus Grant Acknowledge	$\overline{BGACK}$	Input	Low	No	No
Interrupt Priority Level	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	Input	Low	No	No
Bus Error	$\overline{BERR}$	Input	Low	No	No
Reset	$\overline{RESET}$	Input/Output	Low	No*	No*
Halt	$\overline{HALT}$	Input/Output	Low	No*	No*
Enable	E	Output	High	No	No
Valid Memory Address	$\overline{VMA}$	Output	Low	No	Yes
Valid Peripheral Address	$\overline{VPA}$	Input	Low	No	No
Function Code Output	FC0, FC1, FC2	Output	High	No	Yes
Clock	CLK	Input	High	No	No
Power Input	V_{CC}	Input	—	—	—
Ground	GND	Input	—	—	—

*Open drain.

REGISTER DESCRIPTION AND DATA ORGANIZATION

STATUS REGISTER. The status register contains the eight level interrupt mask as well as the condition codes; extend (X), negative (N), zero (Z), overflow (V), and carry (C). Additional status bits indicate that the processor is in a trace (T) mode and/or in a supervisor (S) state.



Status Register

OPERAND SIZE

Operand sizes are defined as follows: a byte equals 8 bits, a word equals 16 bits, and a long word equals 32 bits. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Implicit instructions support some subset of all three sizes.

DATA ORGANIZATION IN REGISTERS

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers together with the active stack pointer support address operands of 32 bits.

DATA REGISTERS. Each data register is 32 bits wide. Byte operands occupy the low order 8 bits, word operands the low order 16 bits, and long word operands the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31. When a data register is used as either a source or destination operand, only the appropriate low-order portion is changed; the remaining high order portion is neither used nor changed.

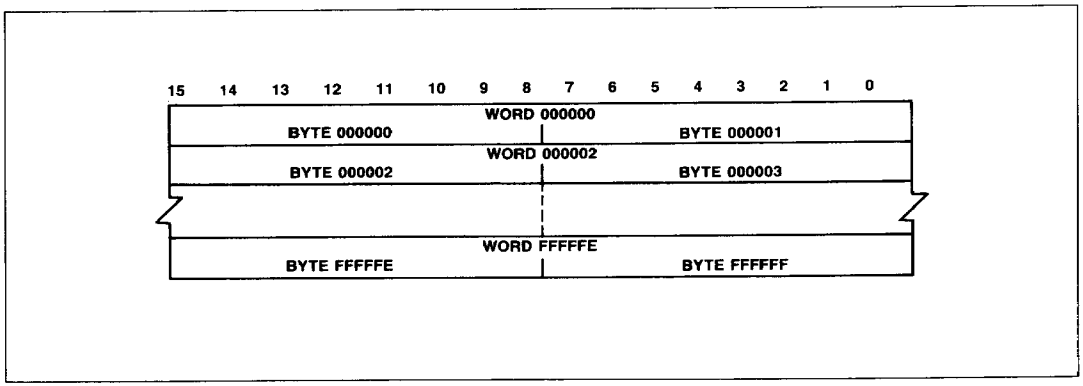


Figure 2. Word Organization In Memory

ADDRESS REGISTERS. Each address register and the stack pointer is 32 bits wide and holds a full 32-bit address. Address registers do not support byte sized operands. Therefore, when an address register is used as a source operand, either the low order word or the entire long word operand is used depending upon the operation size. When an address register is used as the destination operand, the entire register is affected regardless of the operation size. If the operation size is word, any other operands are sign extended to 32 bits before the operation is performed.

DATA ORGANIZATION IN MEMORY

Bytes are individually addressable with the high order byte having an even address the same as the word, as shown in Figure 2. The low order byte has an odd address that is one higher than the word address. Instructions and multi-byte data are accessed only on word (even byte) boundaries. If a long word datum is located at address n (n even), then the second word of that datum is located at address n + 2.

The data types supported by the R68000 are: bit data, integer data of 8, 16, or 32 bits, 32-bit addresses and binary coded decimal data. Each of these data types is put in memory, as shown in Figure 3. The numbers indicate the order in which data is accessed from the processor.

BUS OPERATION

The following paragraphs explain control signal and bus operation during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation.

DATA TRANSFER OPERATIONS. Transfer of data between devices involves the following signals:

- Address Bus A1 through A23
- Data Bus D0 through D15
- Control Signals

The address and data buses are separate parallel buses which transfer data using an asynchronous bus structure. In all cycles, the bus master assumes responsibility for deskewing all signals it issues at both the start and end of a cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave device.

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the R68000 for interlocked multiprocessor communications.

Read Cycle. During a read cycle, the processor receives data from memory or a peripheral device. The processor reads bytes of data in all cases, and for a word (or double word) operation, the processor reads both upper and lower bytes simultaneously by asserting both upper and lower data strobes. When the instruction specifies byte operation, the processor uses an internal AO bit to determine which byte to read and then issues the data strobe required for that byte. When the AO bit equals zero, the upper data strobe is issued, and when the AO bit equals one, the lower data strobe is issued. The processor correctly positions the received data internally.

A word read cycle flow chart is given in Figure 4. A byte read cycle flow chart is given in Figure 5. Read cycle timing is given in Figure 6. Figure 7 details word and byte read cycle operations.

Write Cycle. During a write cycle, the processor sends bytes of data to memory or a peripheral device. If the instruction specifies a word operation, the processor writes both bytes. When the instruction specifies a byte operation, the processor uses an internal AO bit to determine which byte to write and then issues the data strobe required for that byte. When the AO bit equals zero, the upper data strobe is issued and when the AO bit equals one, the lower data strobe is issued. A word write cycle flow chart is given in Figure 8. A byte write cycle flow chart is given in Figure 9. Write cycle timing is given in Figure 6. Figure 10 details word and byte write cycle operation.

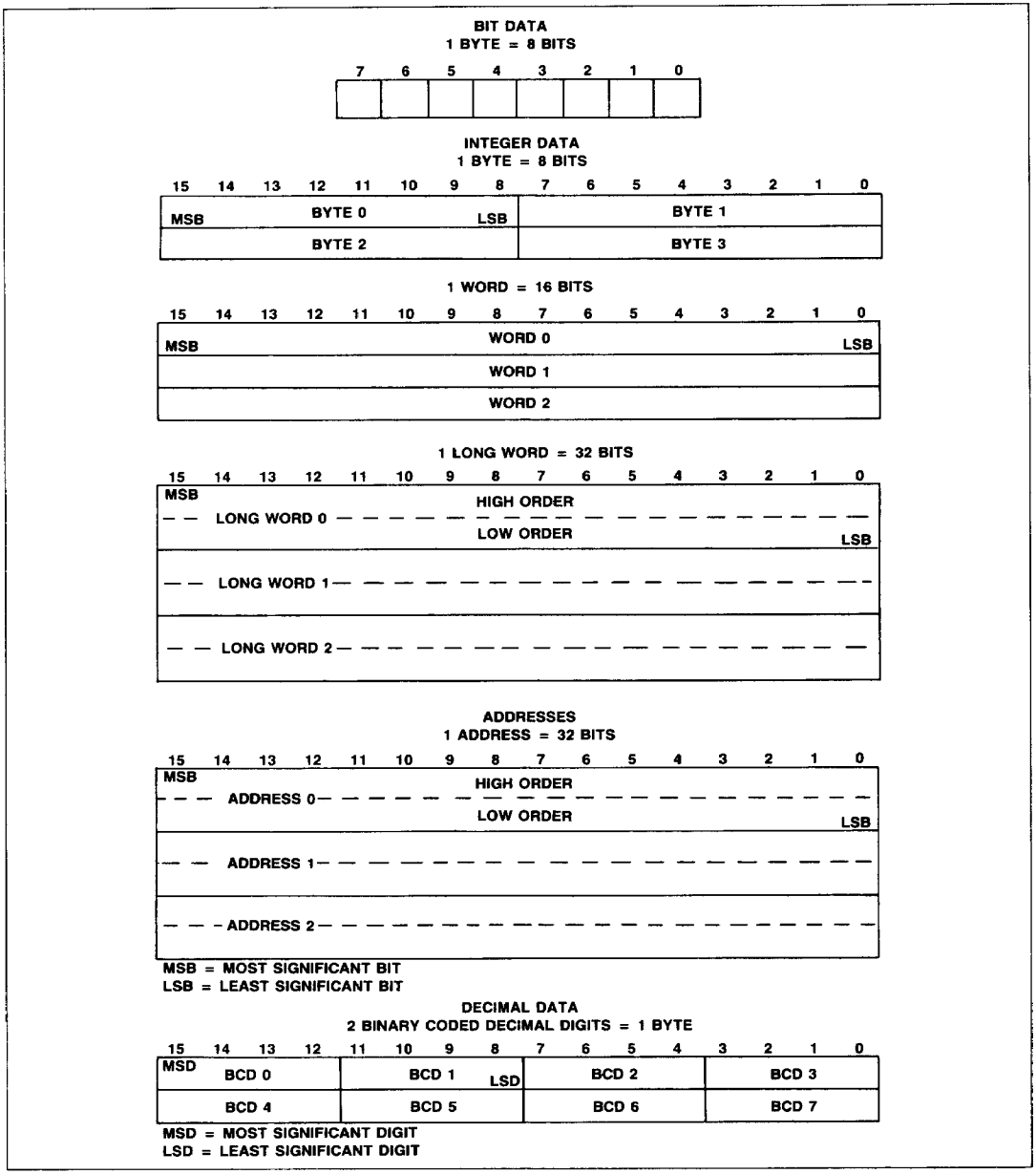


Figure 3. Data Organization In Memory

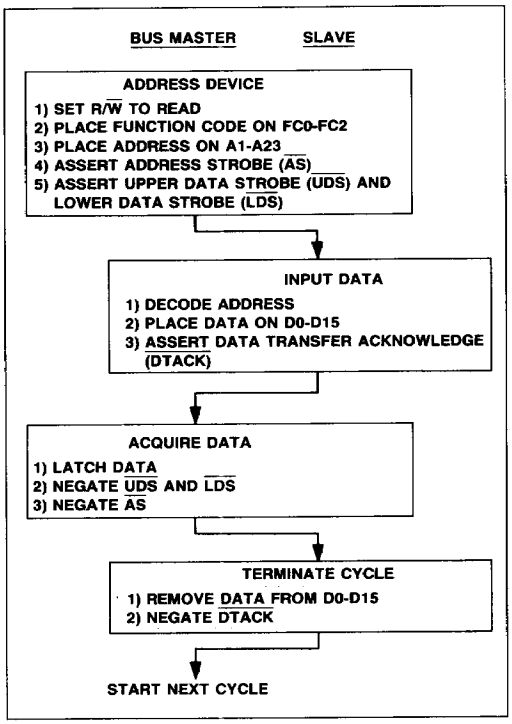


Figure 4. Word Read Cycle Flow Chart

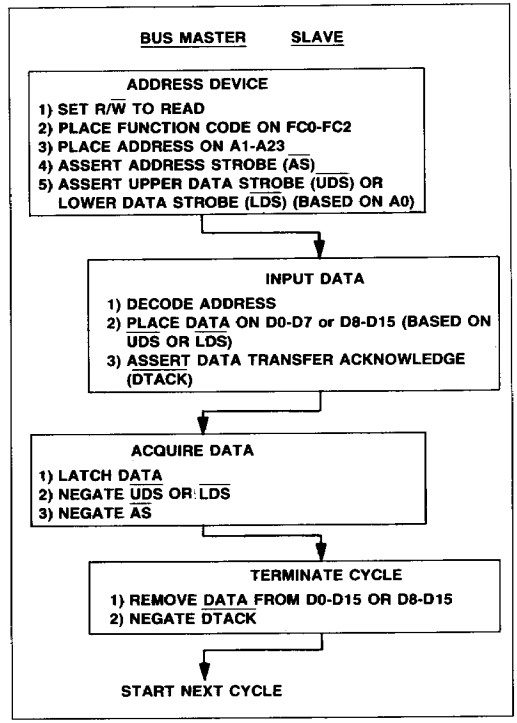


Figure 5. Byte Read Cycle Flow Chart

Read-Modify-Write Cycle. The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the R68000 this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. TAS is the only instruction that uses the read-modify-write cycles. Since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write cycle flow chart is given in Figure 11 and a timing diagram is given in Figure 12.

BUS ARBITRATION. Bus arbitration is a technique used by master-type devices to request, be granted, and acknowledge bus mastership. In its simplest form, it consists of:

1. asserting a bus mastership request,
2. receiving a grant that the bus is available at the end of the current cycle, and
3. acknowledging that mastership has been assumed.

Figure 13 is a flow chart showing the detail involved in a request from a single device. Figure 14 is a timing diagram for the same operation. This technique allows processing of bus requests during data transfer cycles.

The timing diagram shows that the bus request is negated at the time that an acknowledge is asserted. This is true for a system consisting of the processor and one device capable of bus mastership. However, in systems having a number of devices capable of bus mastership, the bus request line from each device is ORed to the processor. In this system, it is easy to see that there could be more than one bus request being made. The timing diagram shows that the bus grant signals negate a few clock cycles after the transition of the acknowledge (BGACK) signal.

However, if the bus requests are still pending, the processor will assert another bus grant within a few clock cycles after negation. This additional assertion of bus grant allows external arbitration circuitry to select the next bus master before the current bus master has completed its requirements. The following paragraphs provide additional information about the three steps in the arbitration process.

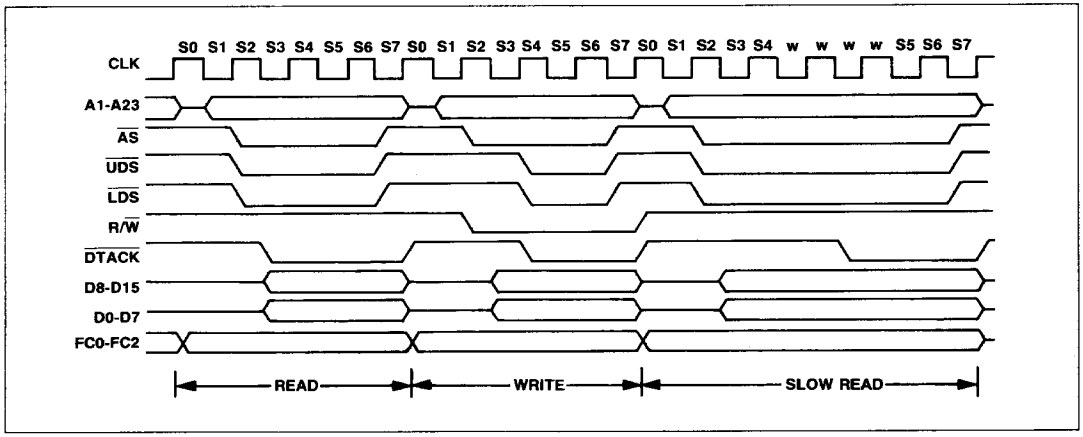


Figure 6. Read and Write Cycle Timing Diagram

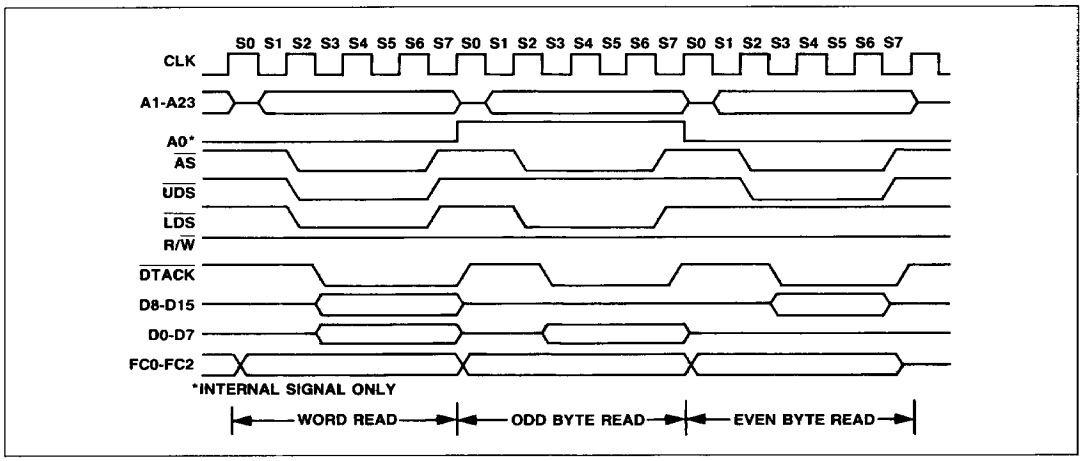


Figure 7. Word and Byte Read Cycle Timing Diagram

Requesting the Bus. External devices capable of becoming bus masters request the bus by asserting the bus request ($\overline{BR}$) signal. This ORed signal (although it need not be constructed from open collector devices) indicates to the processor that some external device requires control of the external bus. The processor, at a lower bus priority level than the external device, will relinquish the bus after it has completed the last bus cycle it has started. If no acknowledge is received before the bus request signal goes inactive, the processor will continue processing when it detects that the bus request is inactive. This allows ordinary processing to continue if the arbitration circuitry inadvertently responded to noise.

Receiving the Bus Grant. Normally the processor asserts bus grant ($\overline{BG}$) as soon as possible after internal synchronization. The only exception occurs when the processor has made an internal decision to execute the next bus cycle but has not progressed far enough into the cycle to have asserted the address strobe ($\overline{AS}$) signal. In this case, bus grant will not be asserted until one clock after address strobe is asserted to indicate to external devices that a bus cycle is being executed.

The bus grant signal may be routed through a daisy-chained network or through a specific priority-encoded network. The processor is not affected by the external method of arbitration as long as the protocol is obeyed.

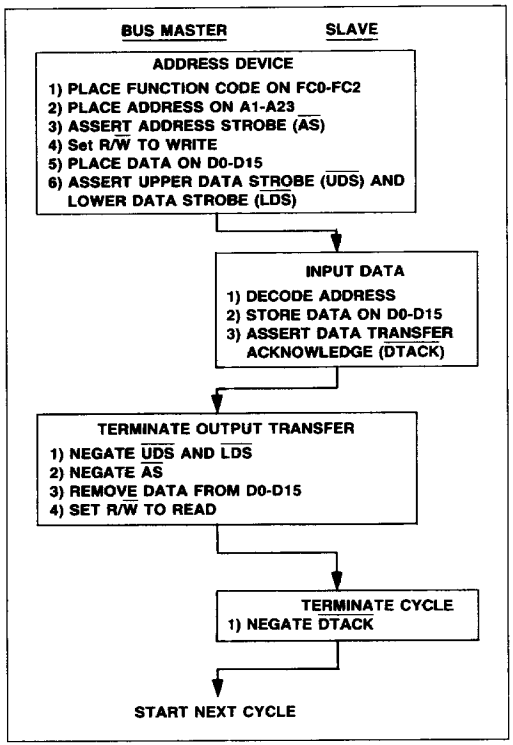


Figure 8. Word Write Cycle Flow Chart

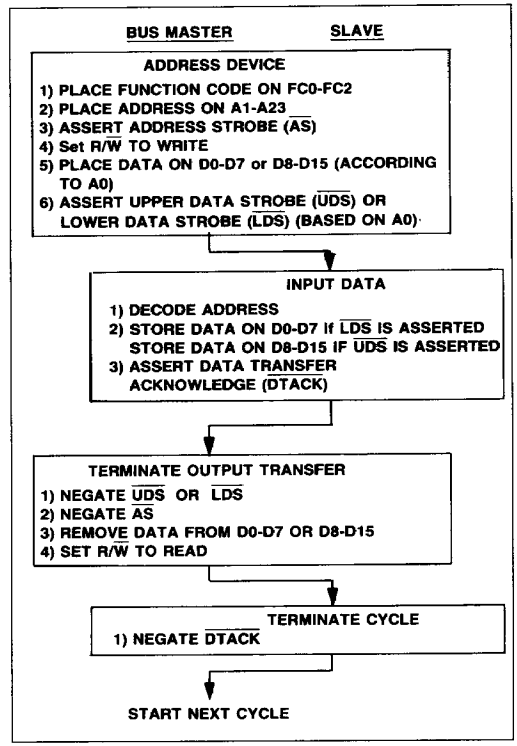


Figure 9. Byte Write Cycle Flow Chart

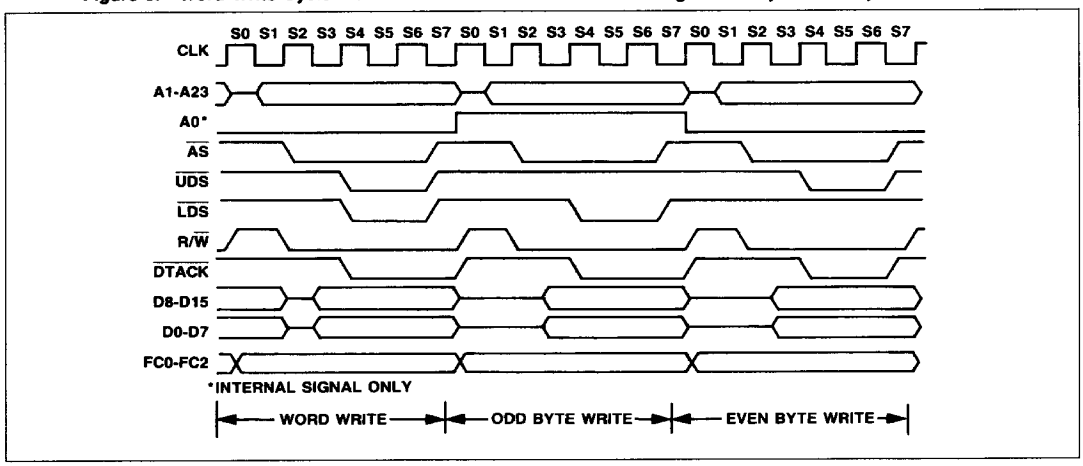


Figure 10. Word and Byte Write Cycle Timing Diagram

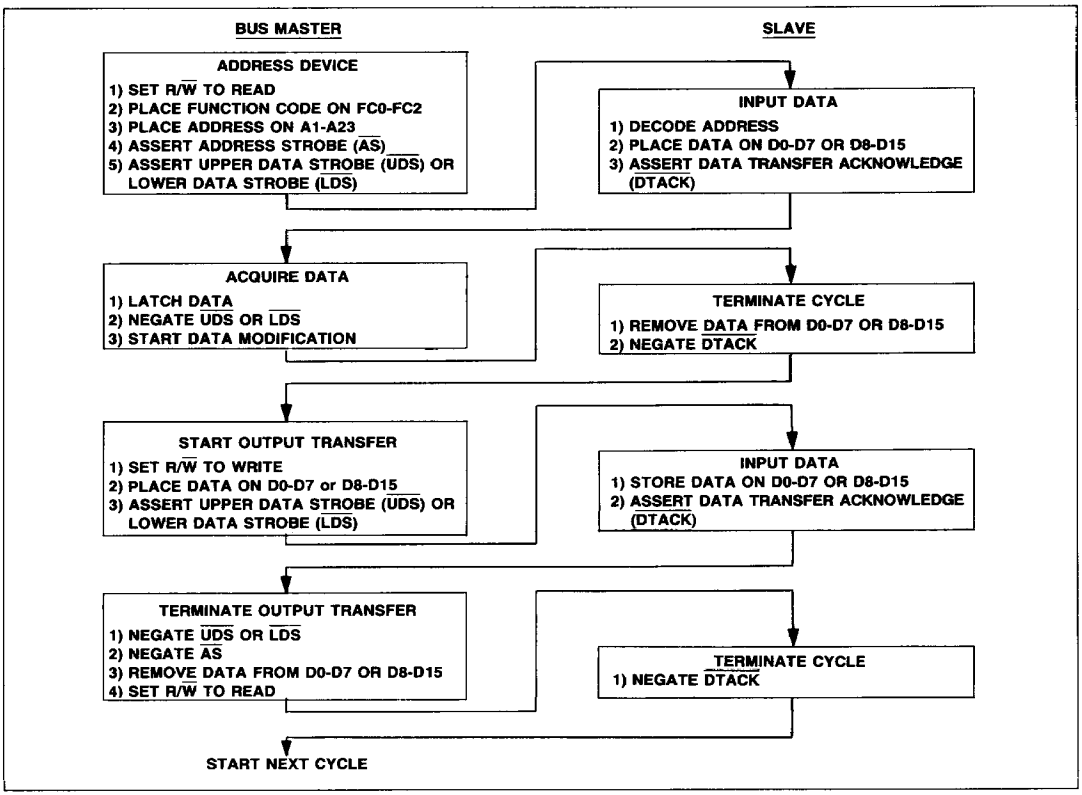


Figure 11. Read-Modify-Write Cycle Flow Chart

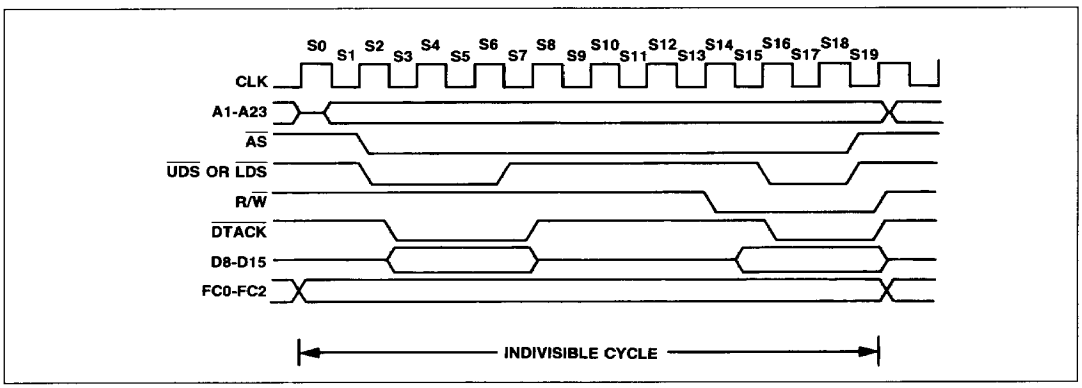


Figure 12. Read-Modify-Write Cycle Timing Diagram

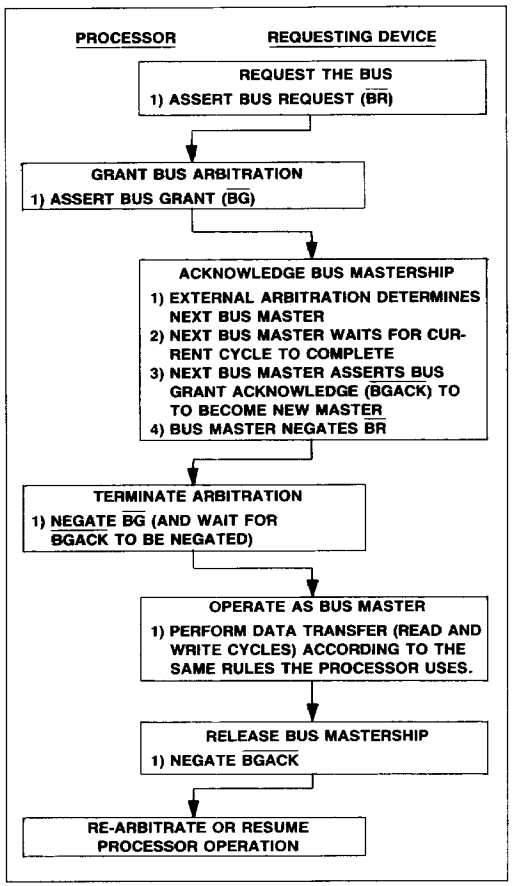


Figure 13. Bus Arbitration Cycle Flow Chart

Acknowledgment of Mastership. Upon receiving a bus grant ($\overline{BG}$), the requesting device waits until address strobe ($\overline{AS}$), data transfer acknowledge ($\overline{DTACK}$), and bus grant acknowledge ($\overline{BGACK}$) are negated before issuing its own $\overline{BGACK}$. The negation of the address strobe indicates that the previous master has completed its cycle, while the negation of bus grant acknowledge indicates that the previous master has released the bus. (If address strobe is asserted no device is allowed to "break into" a cycle.) The negation of data transfer acknowledge indicates the previous slave has terminated its connection to the previous master. In some applications data transfer acknowledge may not be required. In this case the devices would use the address strobe. When bus grant acknowledge is issued the device is bus master. Only after the bus cycle(s) is (are) completed should bus grant acknowledge be negated to terminate bus mastership.

The bus request from the granted device should be dropped after bus grant acknowledge is asserted. If a bus request is still pending, another bus grant will be asserted within a few clocks of the negation of bus grant. Refer to Bus Arbitration Control section. The processor does not perform any external bus cycles before it reasserts bus grant.

BUS ARBITRATION CONTROL. The bus arbitration control unit in the R68000 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 15. All asynchronous signals to the R68000 are synchronized before being used internally. This synchronization is accomplished in a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (#47) has been met (see Figure 16). The input signal is sampled on the falling edge of the clock and is valid internally after the next falling edge. If $\overline{BR}$ and $\overline{BGACK}$ meet the asynchronous set-up time tASI (#47), then tBGKBR (#37A) can be ignored. If $\overline{BR}$ and $\overline{BGACK}$ are asserted asynchronously with respect to the clock, $\overline{BGACK}$ has to be asserted before $\overline{BR}$ is negated.

As shown in Figure 15, input signals labeled R and A are internally synchronized on the bus request and bus grant acknowledge pins respectively. The bus grant output is labeled G and the internal three-state control signal T. If T is true, the address, data, and control buses are placed in a high-impedance state when $\overline{AS}$ is negated. All signals are shown in positive logic (active high) regardless of their true active voltage level.

State changes (valid outputs) occur on the next rising clock edge after the internal signal is valid.

A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 17. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 18.

If a bus request ($\overline{BR}$) is made at a time when the MPU has already begun a bus cycle but $\overline{AS}$ has not been asserted (bus state S0), $\overline{BG}$ will not be asserted on the next rising edge. Instead $\overline{BG}$ will be delayed until the second rising edge following its internal assertion. This sequence is shown in Figure 19.

BUS ERROR AND HALT OPERATION. In a bus architecture that requires a handshake from an external device, the possibility exists that the handshake might not occur. Since different systems will require a different maximum response time, a bus error input is provided.

External circuitry must be used to determine the duration between address strobe and data transfer acknowledge before issuing a bus error signal. When a bus error signal is received, the processor has two options: initiate a bus error exception sequence or try running the bus cycle again.

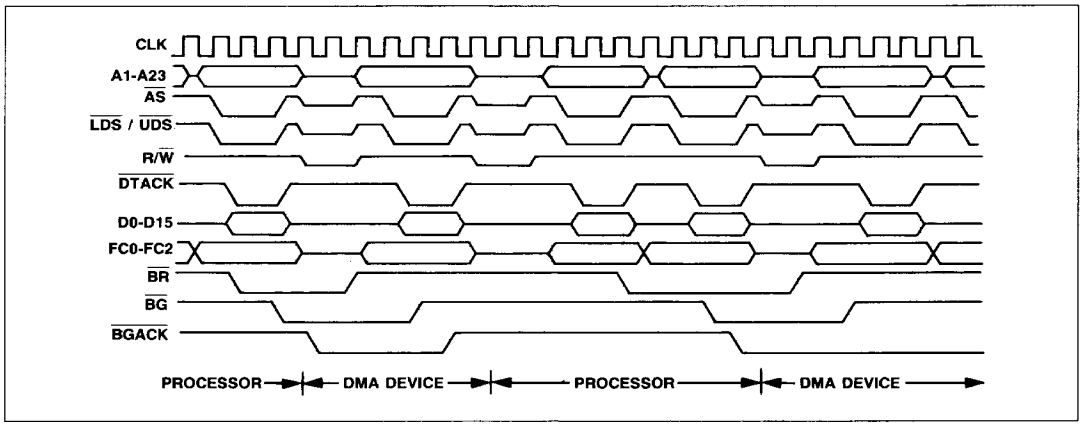
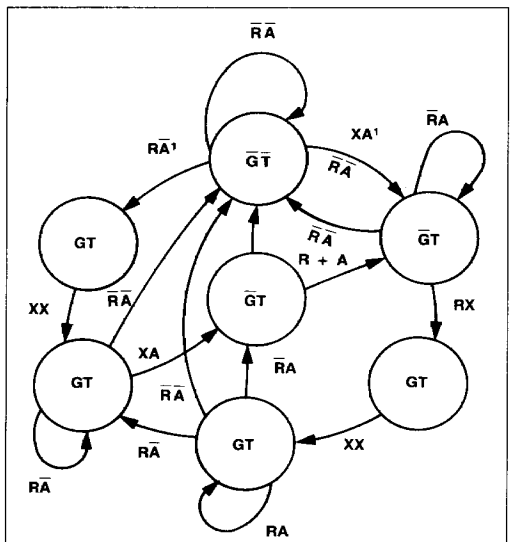


Figure 14. Bus Arbitration Cycle Timing Diagram



R = BUS REQUEST INTERNAL
A = BUS GRANT ACKNOWLEDGE INTERNAL
G = BUS GRANT
T = THREE-STATE CONTROL TO BUS CONTROL LOGIC[2]
X = DON'T CARE

1. STATE MACHINE WILL NOT CHANGE STATE IF BUS IS IN S0 OR S1. REFER TO BUS ARBITRATION CONTROL FOR ADDITIONAL INFORMATION.
2. THE ADDRESS BUS WILL BE PLACED IN THE HIGH IMPEDANCE STATE IF T IS ASSERTED AND AS NEGATED.

Figure 15. State Diagram of R68000 Bus Arbitration Unit

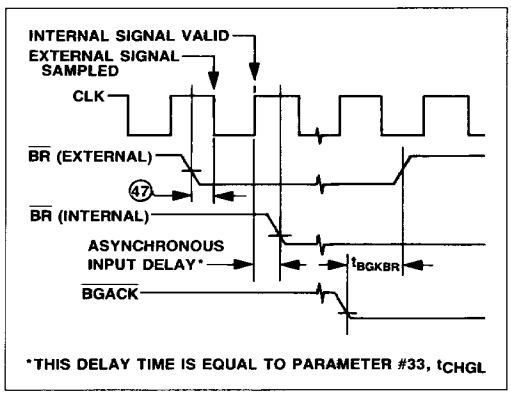


Figure 16. Timing Relationship of External Asynchronous Inputs to Internal Signals

Bus Error Operation. When BERR is asserted, the current bus cycle is terminated. If BERR is asserted before the falling edge of S2, AS will be negated in S7 in either a read or write cycle. As long as BERR remains asserted, the data and address buses will be in the high-impedance state. When BERR is negated, the processor will begin stacking for exception processing. Figure 20 is a timing diagram for the exception sequence. The sequence is composed of the following elements:

1. stacking the program counter and status register,
2. stacking the error information,
3. reading the bus error vector table entry, and
4. executing the bus error handler routine.

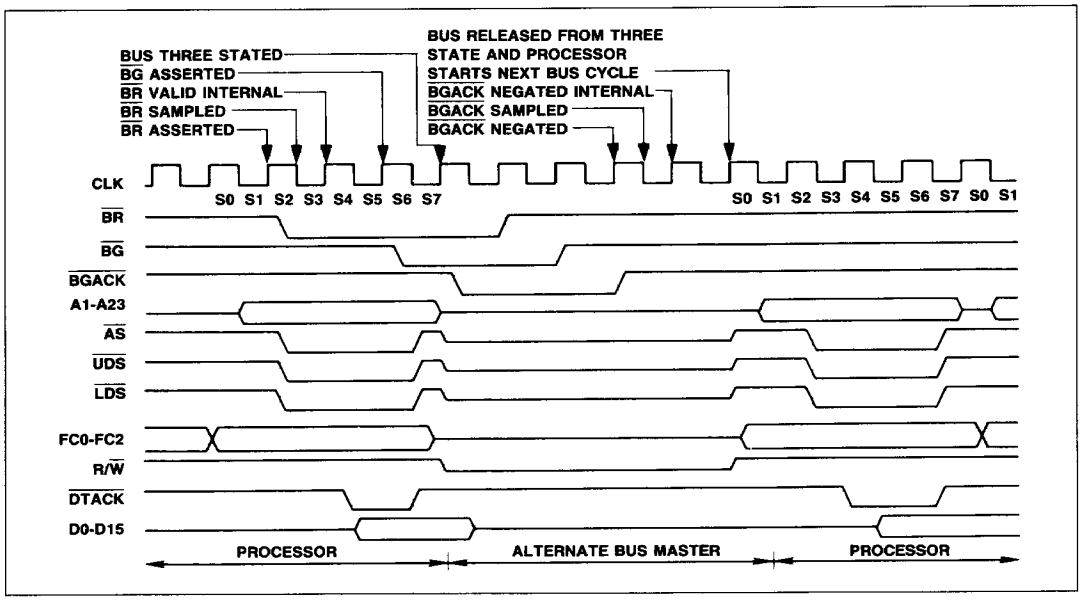


Figure 17. Bus Arbitration During Processor Bus Cycle

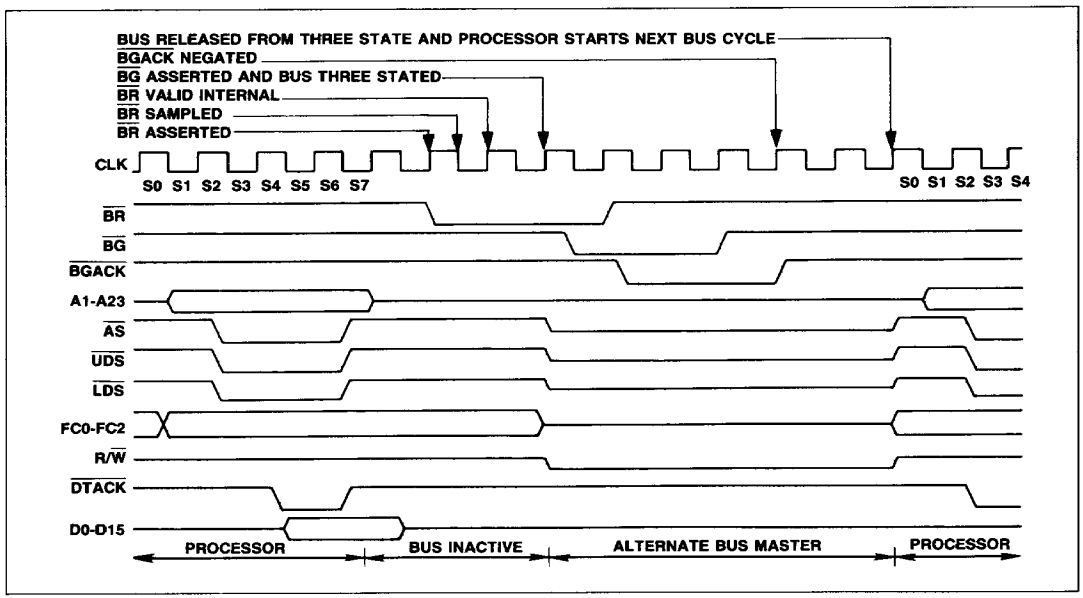


Figure 18. Bus Arbitration with Bus Inactive

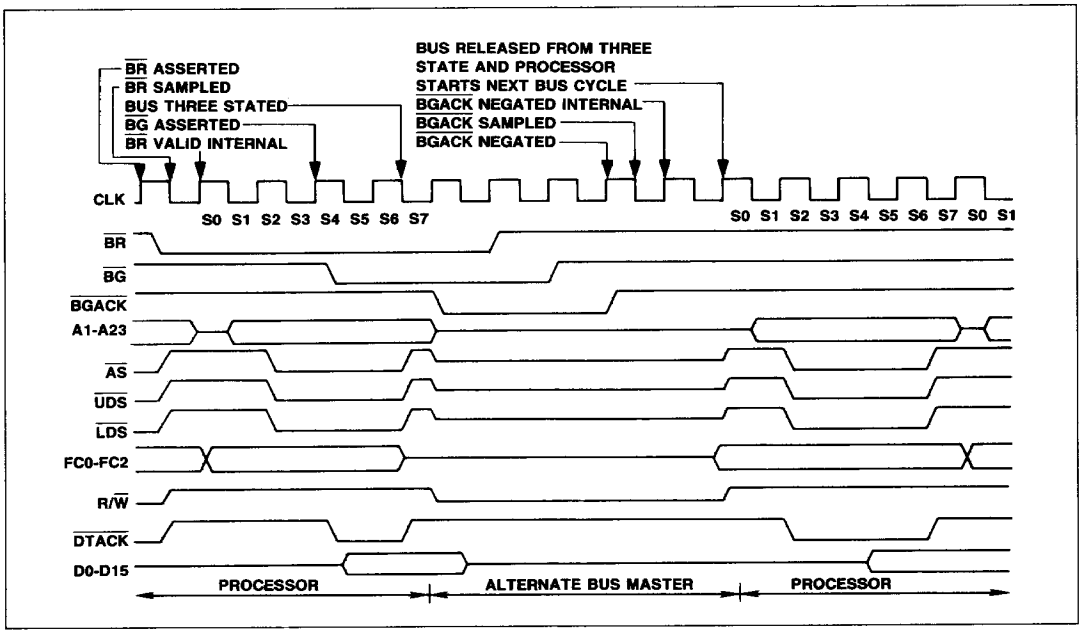


Figure 19. Bus Arbitration During Processor Bus Cycle Special Case

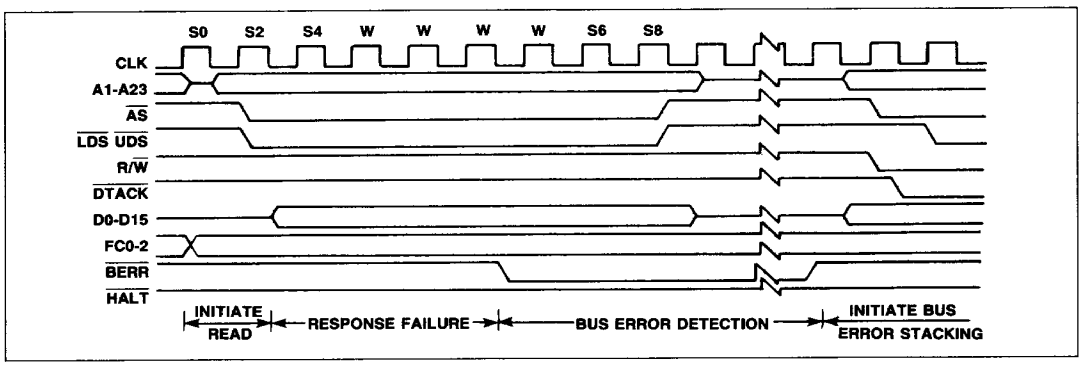


Figure 20. Bus Error Timing Diagram

The stacking of the program counter and the status register is identical to the interrupt sequence. Several additional items are stacked when a bus error occurs. These items are used to determine the nature of the error and correct it, if possible. The bus error vector is vector number two located at address $000008. The processor loads the new program counter from this location. A software bus error handler routine is then executed by the processor. Refer to EXCEPTION PROCESSING for additional information.

Re-Running the Bus Cycle. When, during a bus cycle, the processor receives a $\overline{BERR}$, and $\overline{HALT}$ is being driven by an external device, the processor enters the re-run sequence. Figure 21 is a timing diagram for re-running the bus cycle.

The processor terminates the bus cycle, then puts the address and data output lines in the high-impedance state. The processor remains "halted" and will not run another bus cycle until external logic negates $\overline{HALT}$. Then the processor will re-run the previous bus cycle using the same address, the same function codes, the same data (for a write operation), and the same controls. $\overline{BERR}$ should be negated at least one clock cycle before $\overline{HALT}$ is negated.

Note

The processor will not re-run a read-modify-write cycle. This restriction is made to guarantee that the entire cycle runs correctly and that the write operation of a Test-and-Set operation is performed without ever releasing $\overline{AS}$. If $\overline{BERR}$ and $\overline{HALT}$ are asserted during a read-modify-write bus cycle, a bus error operation results.

Halt Operation with No Bus Error. The $\overline{HALT}$ input signal to the R68000 performs a Halt/Run/Single-Step function in a similar fashion to the R6500 halt functions. When the $\overline{HALT}$ signal is constantly active the processor "halts" (does nothing) and when the $\overline{HALT}$ signal is constantly inactive the processor "runs" (does something).

The single-step mode, derived from correctly timed transitions on the $\overline{HALT}$ signal input, forces the processor to execute a single bus-cycle by entering the "run" mode until the processor starts a bus cycle then changing to the "halt" mode. Thus, the single-step mode allows the user to proceed through (and therefore debug) processor operations one bus cycle at a time.

Figure 22 details the timing required for correct single-step operations. Some care must be exercised to avoid harmful interactions between $\overline{BERR}$ and $\overline{HALT}$ when using the single cycle mode as a debugging tool. This is also true of interactions between the $\overline{HALT}$ and $\overline{RESET}$ lines since these can reset the machine.

When the processor completes a bus cycle after recognizing that $\overline{HALT}$ is active, most three-state signals are put in the high-impedance state. These include:

1. address lines, and
2. data lines.

This is required for correct performance of the re-run bus cycle operation.

Honoring the halt request has no effect on bus arbitration. Only the bus arbitration function removes the control signals from the bus.

Total debugging flexibility is derived from the software debugging package, the halt function, and the hardware trace capability. These processor capabilities allow the hardware debugger to trace single bus cycles or single instructions at a time.

Double Bus Faults. When a bus error exception occurs, the processor will attempt to stack several words containing information about the state of the machine. If a bus error exception occurs during the stacking operation, there have been two bus errors in a row, or a double bus fault. A double bus fault causes the processor to halt. Once a bus error exception has occurred, any bus error exception occurring before the execution of the next instruction constitutes a double bus fault.

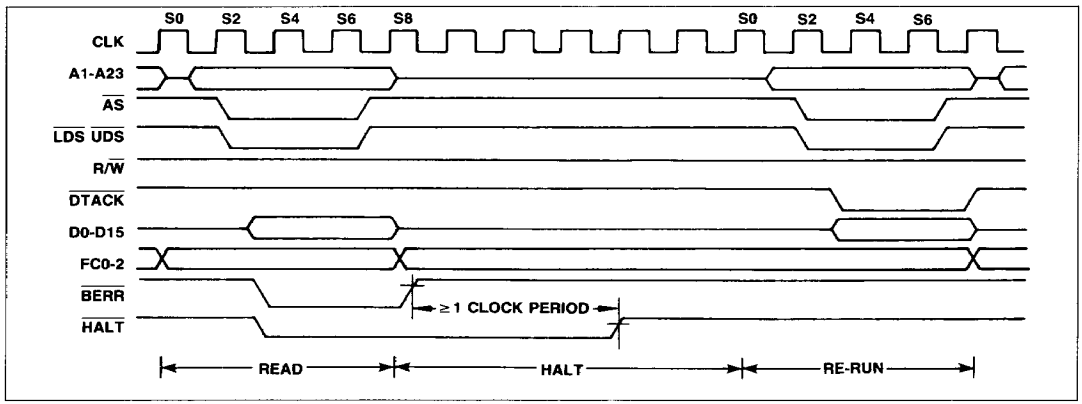


Figure 21. Re-Run Bus Cycle Timing Diagram

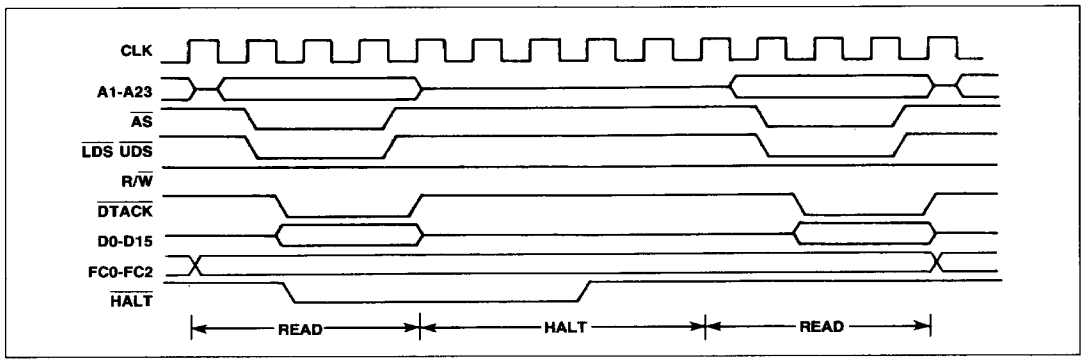


Figure 22. Halt Signal Timing Waveforms

Note that a bus cycle which is re-run does not constitute a bus error exception, and does not contribute to a double bus fault. This means that as long as the external hardware requests it, the processor will continue to re-run the same bus cycle.

The bus error (BERR) pin also has an effect on processor operation after the processor receives an external reset input. The processor reads the vector table after a reset to determine the address to start program execution. If a bus error occurs while reading the vector table (or at any time before the first instruction is executed), the processor reacts as if a double bus fault has occurred and it halts. Only an external reset will start a halted processor.

RESET OPERATION. The reset signal is a bidirectional signal that allows either the processor or an external signal to reset the system. Figure 23 is a timing diagram for reset operations. Both HALT and RESET must be applied to ensure total reset of the processor.

When the RESET and HALT are driven by an external device the entire system, including the processor, is reset. The processor responds by reading the reset vector table entry (vector number zero, address $000000) and loading into the supervisor stack pointer (SSP). Vector table entry number one at address $000004 is read next and loaded into the program counter. The processor initializes the status register to an interrupt level of seven, with no other register being affected.

Execution of the RESET instruction drives the reset pin low for 124 clock periods. In this case, the processor is trying to reset the rest of the system. The internal state of the processor, including the processor's internal registers and the status register, is unaffected by the execution of a RESET instruction. All external devices connected to the reset line will be reset at the completion of the RESET instruction.

Asserting RESET and HALT for 10 clock cycles will cause a processor reset, except when Vcc is initially applied to the processor. In this case, an external reset must be applied for 100 milliseconds.

THE RELATIONSHIP OF DTACK, BERR, AND HALT

In order to properly control termination of a bus cycle for a re-run or a bus error condition, DTACK, BERR, and HALT should be asserted and negated on the rising edge of R68000 clock. This will assure that when two signals are asserted simultaneously, the required setup time (#47) for both of them will be met during the same bus state.

This, or some equivalent precaution, should be designed external to the R68000. Parameter #48 is intended to ensure this operation in a totally asynchronous system, and may be ignored if the above conditions are met.

The preferred bus cycle terminations may be summarized as follows (case numbers refer to Table 4):

Normal Termination: DTACK occurs first (case 1).

Halt Termination: HALT is asserted at same time, or precedes DTACK (no BERR) cases 2 and 3.

Bus Error Termination: BERR is asserted in lieu of, at same time, or preceding DTACK (case 4); BERR negated at same time, or after DTACK.

Re-Run Termination: HALT and BERR asserted in lieu of, at the same time, or before DTACK (cases 6 and 7); HALT must be negated at least one cycle after BERR. (Case 5 indicates BERR may precede HALT which allows fully asynchronous assertion).

Table 4 details the resulting bus cycle termination under various combinations of control signal sequences. The negation of these same control signals under several conditions is shown in Table 5. (DTACK is assumed to be negated normally in all cases; for best results, both DTACK and BERR should be negated when address strobe is negated).

Example A: A system uses a watch-dog timer to terminate accesses to unpopulated address space. The timer asserts DTACK and BERR simultaneously after timeout (case 4).

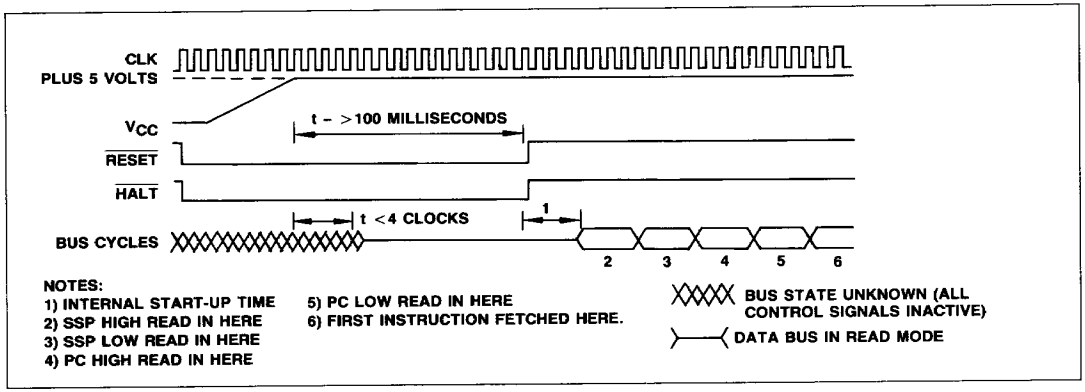


Figure 23. Reset Operation Timing Diagram

Example B: A system uses error detection on RAM contents. Designer may (a) delay DTACK until data verified, and return BERR and HALT simultaneously to re-run error cycle (case 6), or if valid, return DTACK (case 1); (b) delay DTACK until data verified and return BERR at same time as DTACK if data in error (case 4).

ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION

Asynchronous Operation

To achieve clock frequency independence at a system level, the R68000 can be used in an asynchronous manner. This entails using only the bus handshake lines (AS, UDS, LDS, DTACK, BERR, HALT, and VPA) to control the data transfer. Using this method, AS signals the start of a bus cycle and the data strobes are used as a condition for valid data on a write cycle. The slave device (memory or peripheral) then responds by placing the requested data on the data bus for a read cycle or latching data on a write cycle and asserting the data transfer acknowledge signal (DTACK) to terminate the bus cycle. If no slave reponds or the access is invalid, external control logic asserts the BERR, or BERR and HALT, signal to abort or rerun the bus cycle.

The DTACK signal is allowed to be asserted before the data from a slave device is valid on a read cycle. The length of time that DTACK may precede data is given as parameter #31 (See Figure 45) and it must be met in any asynchronous system to insure that valid data is latched into the processor. Notice that there is no maximum time specified from the assertion of AS to the assertion of DTACK. This is because the MPU will insert wait cycles of one clock period each until DTACK is recognized.

The BERR signal is allowed to be asserted after the DTACK signal is asserted. BERR must be asserted within the time given as parameter #48 after DTACK is asserted in any asynchronous system to insure proper operation. If this maximum delay time is violated, the processor may exhibit erratic behavior.

Synchronous Operation

To allow for those systems which use the system clock as a signal to generate DTACK and other asynchronous inputs, the asynchronous inputs setup time is given as parameter #47. If this setup is met on an input, such as DTACK, the processor is guaranteed to recognize that signal on the next falling edge of the system clock. However, the converse is not true—if the input signal does not meet the setup time it is not guaranteed not to be recognized. In addition, if DTACK is recognized on a falling edge, valid data will be latched into the processor (on a read cycle) on the next falling edge provided that the data meets the setup time given as parameter #27. Given this, parameter #31 may be ignored. Note that if DTACK is asserted, with the required setup time, before the falling edge of S4, no wait states will be incurred and the bus cycle will run at its maximum speed of four clock periods.

In order to assure proper operation in a synchronous system when BERR is asserted after DTACK, the following conditions must be met. Within one clock cycle after DTACK was recognized, BERR must meet the setup time parameter #27A prior to the falling edge of the next clock. The setup time is critical to proper operation, and the R68000 may exhibit erratic behavior if it is violated.

Note

During an active bus cycle, VPA and BERR are sampled on every falling edge of the clock starting with S0. DTACK is sampled on every falling edge of the clock starting with S4 and data is latched on the falling edge of S6 during a read. The bus cycle will then be terminated in S7 except when BERR is asserted in the absence of DTACK, in which case it will terminate one clock cycle later in S9.

Table 4. $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, $\overline{\text{HALT}}$ Assertion Results

Case No.	Control Signal	Asserted on Rising Edge of State		Result
		N	N + 2	
1	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	A NA NA	S X X	Normal cycle terminate and continue.
2	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	A NA A	S X S	Normal cycle terminate and halt. Continue when $\overline{\text{HALT}}$ removed.
3	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA NA A	A NA S	Normal cycle terminate and halt. Continue when $\overline{\text{HALT}}$ removed.
4	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	X A NA	X S NA	Terminate and take bus error trap.
5	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA A NA	X S A	Terminate and re-run.
6	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	X A A	X S S	Terminate and re-run when $\overline{\text{HALT}}$ removed.
7	$\overline{\text{DTACK}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA NA A	X A S	Terminate and re-run when $\overline{\text{HALT}}$ removed.

Legend:
N — the number of the current even bus state (e.g., S4, S6, etc.)
A — signal is asserted in this bus state
NA — signal is not asserted in this state
X — don't care
S — signal was asserted in previous state and remains asserted in this state

Table 5. $\overline{\text{BERR}}$ AND $\overline{\text{HALT}}$ Negation Results

Conditions of Termination in Table 4-4	Control Signal	Negated on Rising Edge of State			Results — Next Cycle
		N		N + 2	
Bus Error	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	● ●	or or	● ●	Takes bus error trap.
Re-run	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	● ●	or	●	Illegal sequence; usually traps to vector number 0.
Re-run	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	●		●	Re-runs the bus cycle.
Normal	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	● ●	or	●	May lengthen next cycle.
Normal	$\overline{\text{BERR}}$ $\overline{\text{HALT}}$	●	or	● none	If next cycle is started it will be terminated as a bus error.

● = Signal is negated in this bus state.

PROCESSING STATES

The following paragraphs describe the actions of the R68000 which are outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the processor interrupt priority mask. The sequence of memory references and actions taken by the processor on exception conditions are detailed.

The R68000 is always in one of three processing states: normal, exception, or halted. The normal processing state associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of the normal state is the stopped state which the processor enters when a STOP instruction is executed. In this state, no further references are made.

The exception processing state is associated with interrupts, trap instructions, tracing and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

The halted processing state is an indication of a catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

PRIVILEGE STATES

The processor operates in one of two states of privilege: the "user" state or the "supervisor" state. The privilege state determines legal operations. It is used to choose between the supervisor stack pointer and the user stack pointer in instruction references, and by the external memory management device to control and translate accesses.

The privilege state is a mechanism for providing security in a computer system by allowing most programs to execute in user state. In this state, the accesses are controlled, and the effects on other parts of the system are limited. Programs should access only their own code and data areas, and ought to be restricted from accessing information.

The operating system which executes in the supervisor state, has access to all resources and performs the overhead tasks for the user state programs.

SUPERVISOR STATE. The supervisor state is the higher state of privilege. For instruction execution, the supervisor state is determined by asserting (high) the S-bit of the status register. All instructions can be executed in the supervisor state. The bus cycles generated by instructions executed in the supervisor state are classified as supervisor references. While the processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the supervisor stack pointer.

All exception processing is done in the supervisor state, regardless of the setting of the S-bit. The bus cycles generated during exception processing are classified as supervisor references. All stacking operations during exception processing use the supervisor stack pointer.

USER STATE. The user state is the lower state of privilege. For instruction execution, the user state is determined by negating (low) the S-bit of the status register.

Most instructions execute the same in user state as in the supervisor state. However, some instructions which have important system effects are made privileged. User programs are not permitted to execute the STOP instruction, or the RESET instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole state register are priviled. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE to USP) and move from user stack pointer (MOVE from USP) instructions are also privileged.

The bus cycles generated by an instruction executed in user state are classified as user state references. This allows an external memory management device to translate the address and to control access to protected portions of the address space. While the processor is in user privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly, access the user stack pointer.

PRIVILEGE STATE CHANGES. Once the processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the current setting of the S-bit of the status register is saved and the S-bit is asserted, putting the processing in the supervisor state. Therefore, when instruction execution resumes to process the exception, the processor is in the supervisor privilege state.

REFERENCE CLASSIFICATION. When the processor makes a reference, it classifies the kind of reference being made by using the encoding on the three function code output lines. This allows external translation of addresses, control of access, and differentiation of special processor states, such as interrupt acknowledge. Table 6 lists the classification of references.

Table 6. Reference Classification

Function Code Output			
FC2	FC1	FC0	Reference Class
0	0	0	(Unassigned)
0	0	1	User Data
0	1	0	User Program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

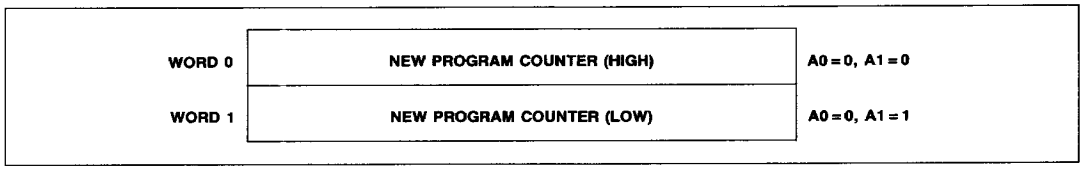


Figure 24. Exception Vector Format

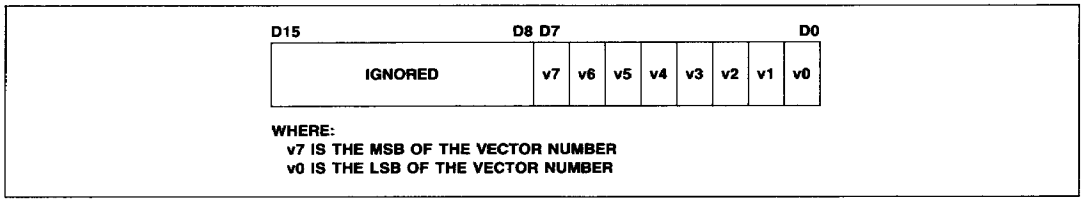


Figure 25. Peripheral Vector Number Format

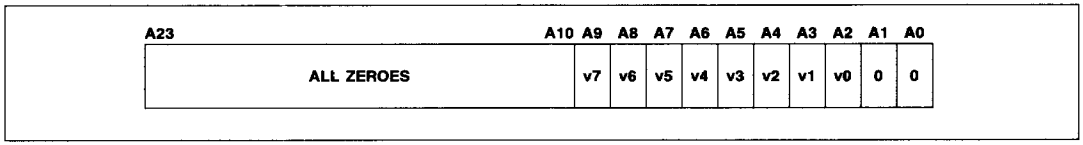


Figure 26. Address Translated From 8-Bit Vector Number

4

EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made, and the status register is set for exception processing. In the second step the exception vector is determined, and the third step is the saving of the current processor contents. In the fourth step a new context is obtained, and the processor switches to instruction processing.

EXCEPTION VECTORS. Exception vectors are memory locations from which the processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length (Figure 24), except for the reset vector, which is four words. All exception vectors lie in the supervisor data space, except for the reset vector which is in the supervisor program space. A vector number is an eight-bit number which, when multipled by four, gives the address of an exception vector. Vector numbers are generated internally or externally, depending on the cause of the exception. In the case of interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 25) to the processor on data bus lines D0 through D7. The processor translates the vector number into a full 24-bit address, as shown in Figure 26. The memory layout for exception vectors is given in Table 7.

As shown in Table 7, the memory layout is 512 words long (1024 bytes). It starts at address 0 and proceeds through address 1023. This provides 255 unique vectors; some of these are reserved for TRAPS and other system functions. Of the 255, there are 192 reserved for user interrupt vectors. However, there is no protection on the first 64 entries, so user interrupt vectors may overlap at the discretion of the systems designer.

KINDS OF EXCEPTIONS. Exceptions can be generated either internally or externally. Externally generated exceptions include interrupts ($\overline{IRQ}$), bus error ($\overline{BERR}$), and reset ($\overline{RESET}$) requests. Interrupts are requests from peripheral devices for processor action while $\overline{BERR}$ and $\overline{RESET}$ inputs are used for access control and processor restart. Internally generated exceptions come from instructions, from address errors, or from tracing. The trap (TRAP), trap on overflow (TRAPV), check register against bounds (CHK) and divide (DIV) instructions can all generate exceptions as part of their instruction execution. In addition, illegal instructions, word fetches from odd addresses and privilege violations cause exceptions. Tracing behaves like a very high priority, internally generated interrupt after each instruction execution.

EXCEPTION PROCESSING SEQUENCE. Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the S-bit is asserted, putting the processor into the supervisor privilege state. Also, the T-bit is negated which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

Table 7. Exception Vector Assignment

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
—	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14*	56	038	SD	(Unassigned, reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16-23*	64	04C	SD	(Unassigned, reserved)
	95	05F		—
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	191	0BF		—
48-63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		—
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		—

*Vector numbers 12, 13, 14, 16 through 23, and 48 through 63 are reserved for future enhancements. No user peripheral devices should be assigned these numbers.

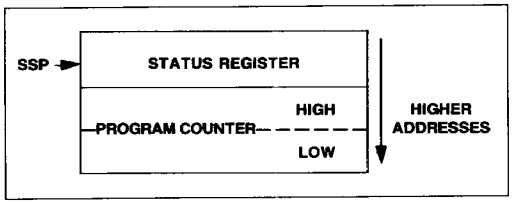


Figure 27. Exception Stack Order (Groups 1 and 2)

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch, classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

The third step is to save the current processor status except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer as shown in Figure 27. The program counter value stacked usually points to the next unexecuted instruction; however, for bus error and address error, the value stacked for the program counter is unpredictable, and may be incremented from the address of the instruction which caused the error. Additional information defining the current context is stacked for the bus error and address error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution. The instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution is started.

MULTIPLE EXCEPTIONS. These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped according to their occurrence and priority. The Group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted, and the exception processing to commence within two clock cycles. The Group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instructions. These exceptions allow the current instruction to execute to completion, but preempt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The Group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Group 0 exceptions have highest priority, while Group 2 exceptions have lowest priority. Within Group 0, reset has highest priority, followed by address error and then bus error. Within Group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, there is no priority relation within Group 2.

Table 8. Exception Grouping and Priority

Group	Exception	Processing
0	Reset Address Error Bus Error	Exception processing begins within two clock cycles.
1	Trace Interrupt Illegal Instruction Privilege Violation	Exception processing begins before the next instruction.
2	TRAP, TRAPV, CHK, Zero Divide	Exception processing is started by normal instruction execution

The priority relation between two exceptions determines which is taken first if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instruction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T-bit is asserted, the trace exception has priority, and is processed first. Before instruction processing resumes, however, the interrupt exception is also processed, and instruction processing commences finally in the interrupt handler routine. Table 8 gives a summary of exception grouping and priority.

4

EXCEPTION PROCESSING DETAILED DISCUSSION

Exceptions have a number of sources, and each exception has a unique processing sequence. The following paragraphs detail the sources of exceptions, how each arises, and how each is processed.

RESET. The reset input provides the highest exception level. The processing of the reset signal is designed for system initiation, and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state and the trace state is forced off. The processor interrupt priority mask is set at level seven. The vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is fetched as the initial supervisor stack pointer, and the address in the last two words of the reset exception vector is fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The powerup/restart code should be pointed to by the initial program counter.

The RESET instruction does not cause loading of the reset vector, but does assert the reset line to reset external devices. This allows the software to reset the system to a known state and then continue processing with the next instruction.

INTERRUPTS. Seven levels of interrupt priorities are provided. Devices may be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. Interrupt priority levels are numbered from one to seven, level seven being the highest priority. The status register contains a three-bit mask which indicates the current processor priority. Interrupts are inhibited for all priority levels less than or equal to the current processor priority.

An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines; a zero indicates no interrupt request. Interrupt requests arriving at the processor do not face immediate exception processing, but are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction and the interrupt exception processing is postponed. (The recognition of level seven is slightly different, as explained in a following paragraph.)

If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. First a copy of the status register is saved, and the privilege state is set to supervisor, then tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged. The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests an automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the program counter and status register on the supervisor stack. The saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine. A flow chart for the interrupt acknowledge sequence is given in Figure 28, a timing diagram is given in Figure 29, and the interrupt exception timing sequence is shown in Figure 30.

Priority level seven is a special case. Level seven interrupts cannot be inhibited by the interrupt priority mask, thus providing a "non-maskable interrupt" capability. An interrupt is generated each time the interrupt request level changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

UNINITIALIZED INTERRUPT. An interrupting device asserts $\overline{VPA}$ or provides an interrupt vector during an interrupt acknowledge cycle to the R68000. If the vector register has not been initialized, the responding R68000 Family peripheral will provide vector 15, the uninitialized interrupt vector. This provides a uniform way to recover from a programming error.

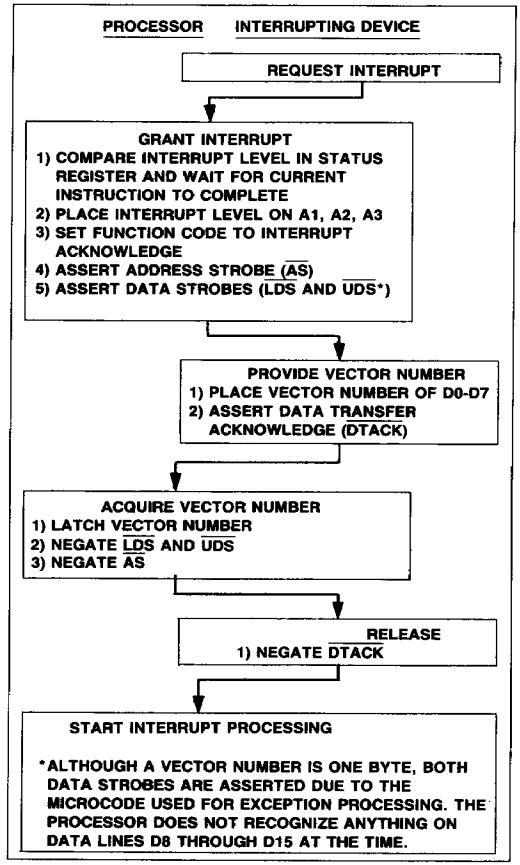


Figure 28. Interrupt Acknowledge Sequence Flow Chart

SPURIOUS INTERRUPT. If during the interrupt acknowledge cycle no device responds by asserting $\overline{DTACK}$ or $\overline{VPA}$, the bus error line should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by fetching the spurious interrupt vector instead of the bus error vector. The processor then proceeds with the usual exception processing.

INSTRUCTION TRAPS. Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

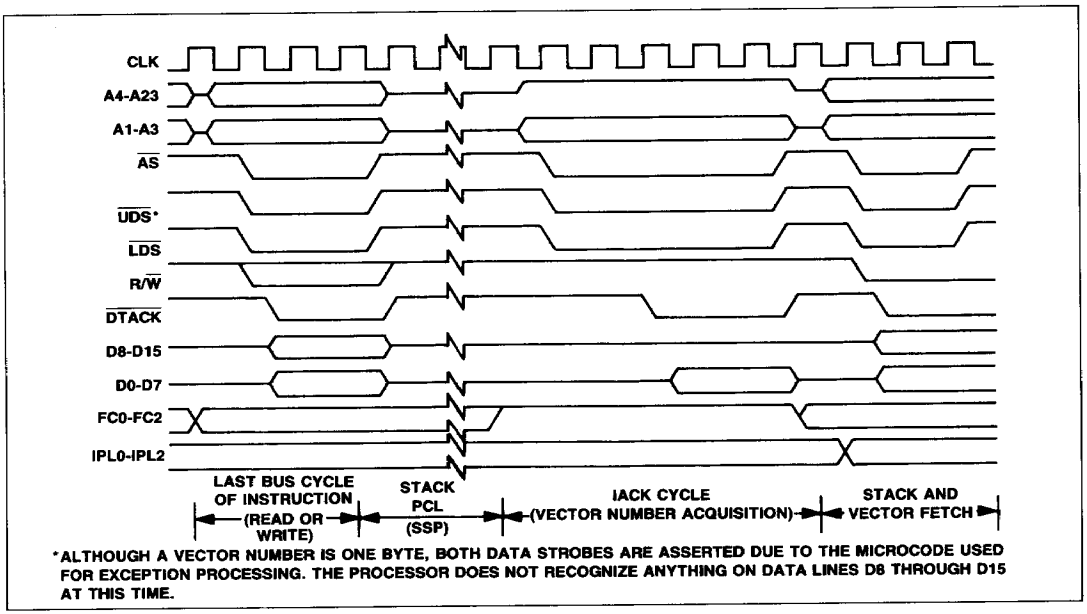


Figure 29. Interrupt Acknowledge Sequence Timing Diagram

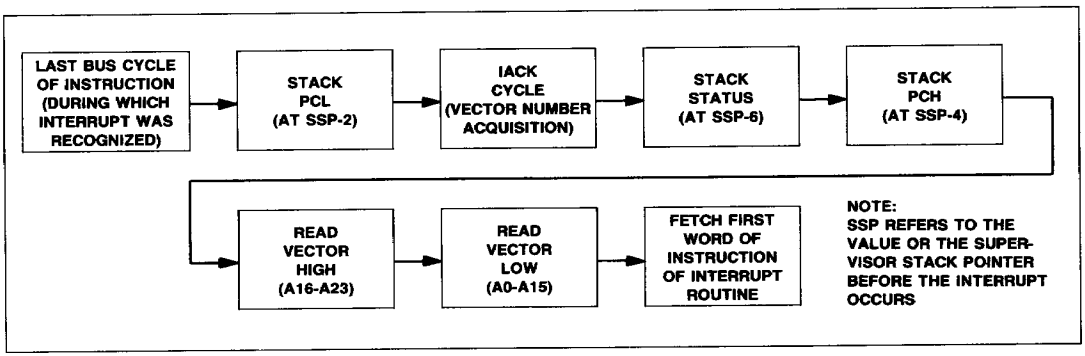


Figure 30. Interrupt Exception Timing Sequence

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception, and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a runtime error, which may be an arithmetic overflow or a subscript out of bounds.

The signed divide (DIVS) and unsigned divide (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

ILLEGAL AND UNIMPLEMENTED INSTRUCTIONS. Illegal instruction refers to any of the word bit patterns which are not the bit pattern of the first word of a legal instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs. Rockwell reserves the right to define instructions whose opcodes may be any of the illegal instructions. Three bit patterns will always force an illegal instruction trap on all R68000 Family compatible microprocessors. They are: $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for Rockwell system products. The third pattern, $4AFC, is reserved for customer use.

Word patterns with bits 15 through 12 equaling 1010 or 1111 are distinguished as unimplemented instructions and separate exception vectors are given to these patterns to permit efficient emulation. This facility allows the operating system to detect program errors, or to emulate unimplemented instructions in software.

PRIVILEGE VIOLATIONS. In order to provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user state will cause an exception. The privileged instructions are:

STOP	AND Immediate to SR
RESET	EOR Immediate to SR
RTE	OR Immediate to SR
MOVE USP	MOVE to SR

TRACING. To aid in program development, the R68000 includes a facility to allow instruction by instruction tracing. In the trace state, after each instruction is executed an exeception is forced, allowing a debugging program to monitor the execution of the program under test.

The trace facility uses the T-bit in the supervisor portion of the status register. If the T-bit is negated (off), tracing is disabled, and instruction execution proceeds from instruction to instruction as normal. If the T-bit is asserted (on) at the beginning of the execution of an instruction, a trace exception will be generated after the execution of that instruction is completed. If the instruction is not executed, either because an interrupt is taken, or the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus error, or address error exception. If the instruction is indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If, during the execution of the instruction, an exception is forced by that instruction, the forced exception is processed before the trace exception.

As an extreme illustration of the above rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.

BUS ERROR. Bus error exceptions occur when the external logic requests that a bus error be processed by an exception. The current bus cycle which the processor is making is then aborted. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing.

Exception processing for bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus error exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. The value saved for the program counter is advanced by some amount, two to ten bytes beyond the address of the first word of the instruction which made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed, and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved: whether it was a read or a write, whether the processor was processing an instruction or not, and the classification displayed on the function code outputs when the bus error occurred. The processor is processing an instruction if in the normal state or processing a Group 2 exception; the processor is not processing an instruction when processing a Group 0 or a Group 1 exception. Figure 31 illustates how the information is organized on the supervisor stack. Although this information is not sufficient to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, address error, or reset, the processor is halted, and all processing ceases. This simplifies the detection of catastrophic system failure, since the processor removes itself from the system rather than destroy all memory contents. Only the $\overline{\text{RESET}}$ pin can restart a halted processor.

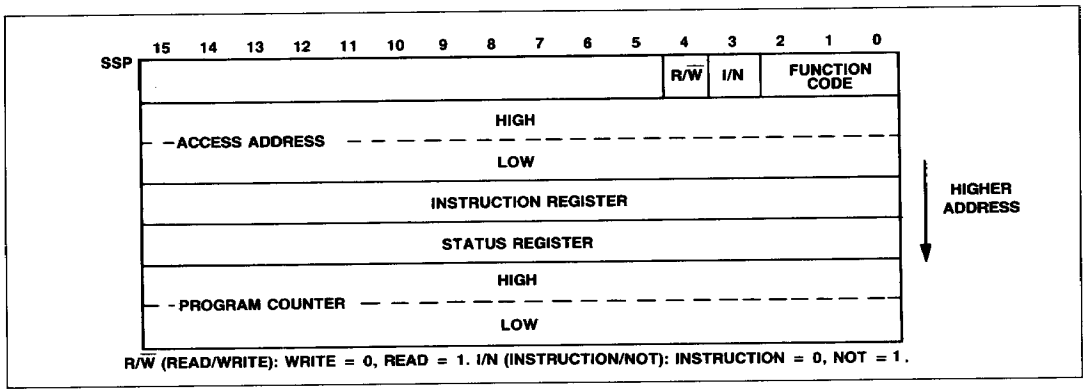


Figure 31. Supervisor Stack Order (Group 0)

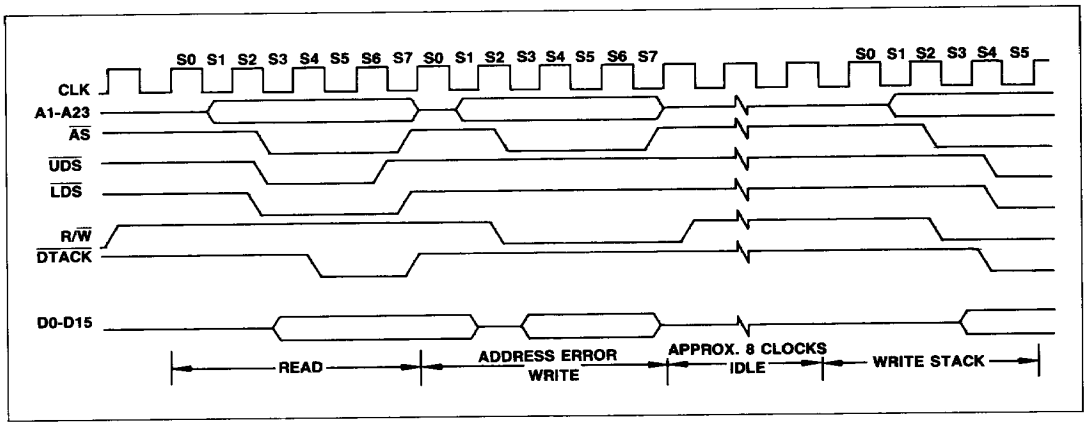


Figure 32. Address Error Timing

ADDRESS ERROR. Address error exceptions occur when the processor attempts to access a word or a long word operand or an instruction at an odd address. The effect is much like an internally generated bus error, so that the bus cycle is aborted, and the processor ceases whatever processing it is currently doing and begins exception processing. After exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted. As shown in Figure 32, an address error will execute a short bus cycle followed by an exception processing.

INTERFACE WITH R6500 PERIPHERALS

Rockwell's line of R6500 peripherals are directly compatible with the R68000. Some of these devices that are particularly useful are:

- R6520 Peripheral Interface Adapter (PIA)
- R6522 Versatile Interface Adapter (VIA)
- R6545 CRT Controller (CRTC)
- R6551 Asynchronous Communication Interface Adapter (ACIA)

To interface the synchronous R6500 peripherals with the asynchronous R68000, the processor modifies its bus cycle to meet the R6500 cycle requirements whenever an R6500 device address is detected. This is possible since both processors use memory mapped I/O. Figure 33 is a flow chart of the interface operation between the processor and R6500 devices. 6800 peripherals are also compatible with the R68000 processor.

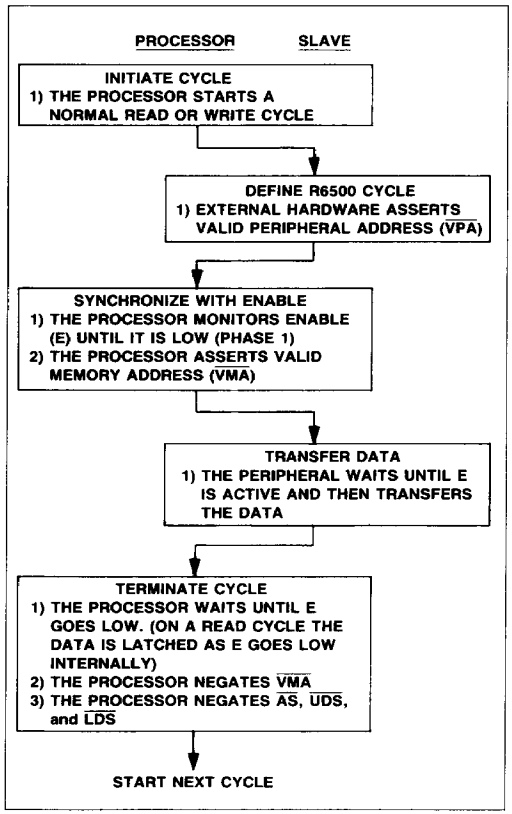


Figure 33. R6500 Interfacing Flow Chart

DATA TRANSFER OPERATION

Three signals on the processor provide the R6500 interface. They are: enable (E), valid memory address (VMA), and valid Peripheral address (VPA). Enable corresponds to the E or Ø2 signal in existing R6500 systems. The bus frequency is one tenth of the incoming R68000 clock frequency. The timing of E allows 1 MHz peripherals to be used with an 8 MHz R68000. Enable has a 60/40 duty cycle; that is, it is low for six input clocks and high for four input clocks. This duty cycle allows the processor to do successive VPA accesses on successive E pulses.

Figures 34 and 35 give a general R6500 to R68000 interface timing, while Figures 36 and 37 detail the specific timing parameters involved in the interface. At state zero (S0) in the cycle, the address bus is in the high-impedance state. A function code is asserted on the function code output lines. One-half clock later, in state 1, the address bus is released from the high-impedance state.

During state 2, the address strobe (AS) is asserted to indicate that there is a valid address on the address bus. If the bus cycle is a read cycle, the upper and/or lower data strobes are also asserted in state 2. If the bus cycle is a write cycle, the read/write (R/W) signal is switched to a low (write) during state 2. One-half clock later, in state 3, the write data is placed on the data bus, and in state 4 the data strobes are issued to indicate valid data on the data bus. The processor now inserts wait states until it recognizes the assertion of VPA.

The VPA input signals the processor that the address on the bus is the address of an R6500 device (or an area reserved for R6500 devices) and that the bus should conform to the Ø2 transfer characteristics of the R6500 bus. Valid peripheral address (VPA) is derived by decoding the address bus, conditioned by address strobe (AS). Chip select for the R6500 peripherals should be derived by decoding the address bus conditioned by VMA.

After the recognition of VPA, the processor assures that the Enable (E) is low, by waiting if necessary, and subsequently asserts VMA. Valid memory address is then used as part of the chip select equation of the peripheral. This ensures that the R6500 peripherals are selected and deselected at the correct time. The peripheral now runs its cycle during the high portion of the E signal. Figures 34 and 35 depict the best and worst case R6500 cycle timing. This cycle length is dependent strictly upon when VPA is asserted in relationship the E clock.

If we assume that external circuitry asserts VPA as soon as possible after the assertion of AS, then VPA will be recognized as being asserted on the falling edge of S4. In this case, no "extra" wait cycles will be inserted prior to the recognition of VPA assertion and only the wait cycles inserted to synchronize with the E clock will determine the total length of the cycle. In any case, the synchronization delay will be some integral number of clock cycles within the following two extremes:

1. Best Case—VPA is recognized as being asserted on the falling edge three clock cycles before E rises (or three clock cycles after E falls).

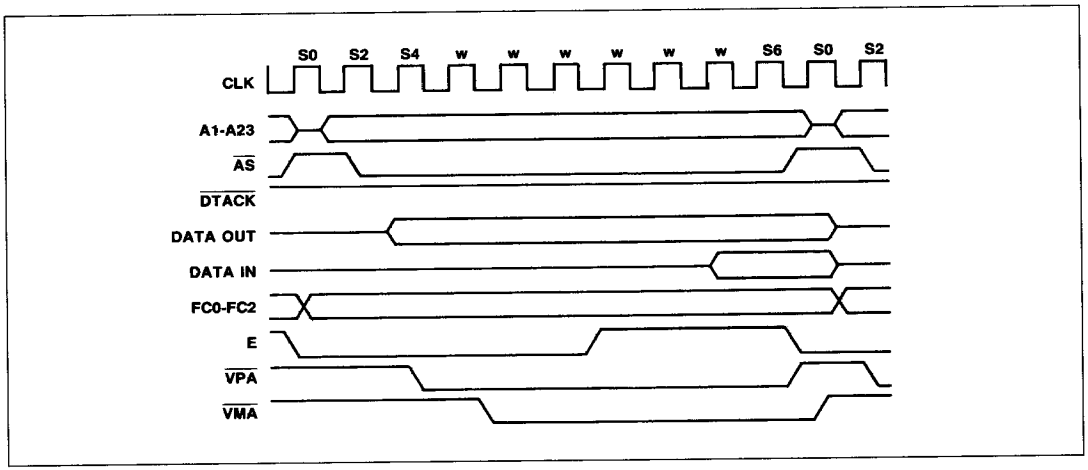


Figure 34. R68000 to R6500 Peripheral Timing—Best Case

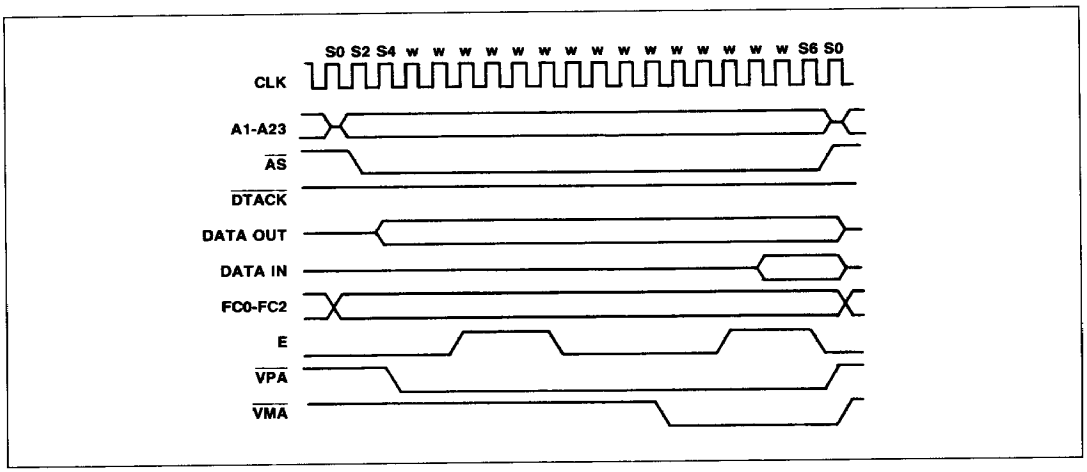


Figure 35. R68000 to R6500 Peripheral Timing—Worst Case

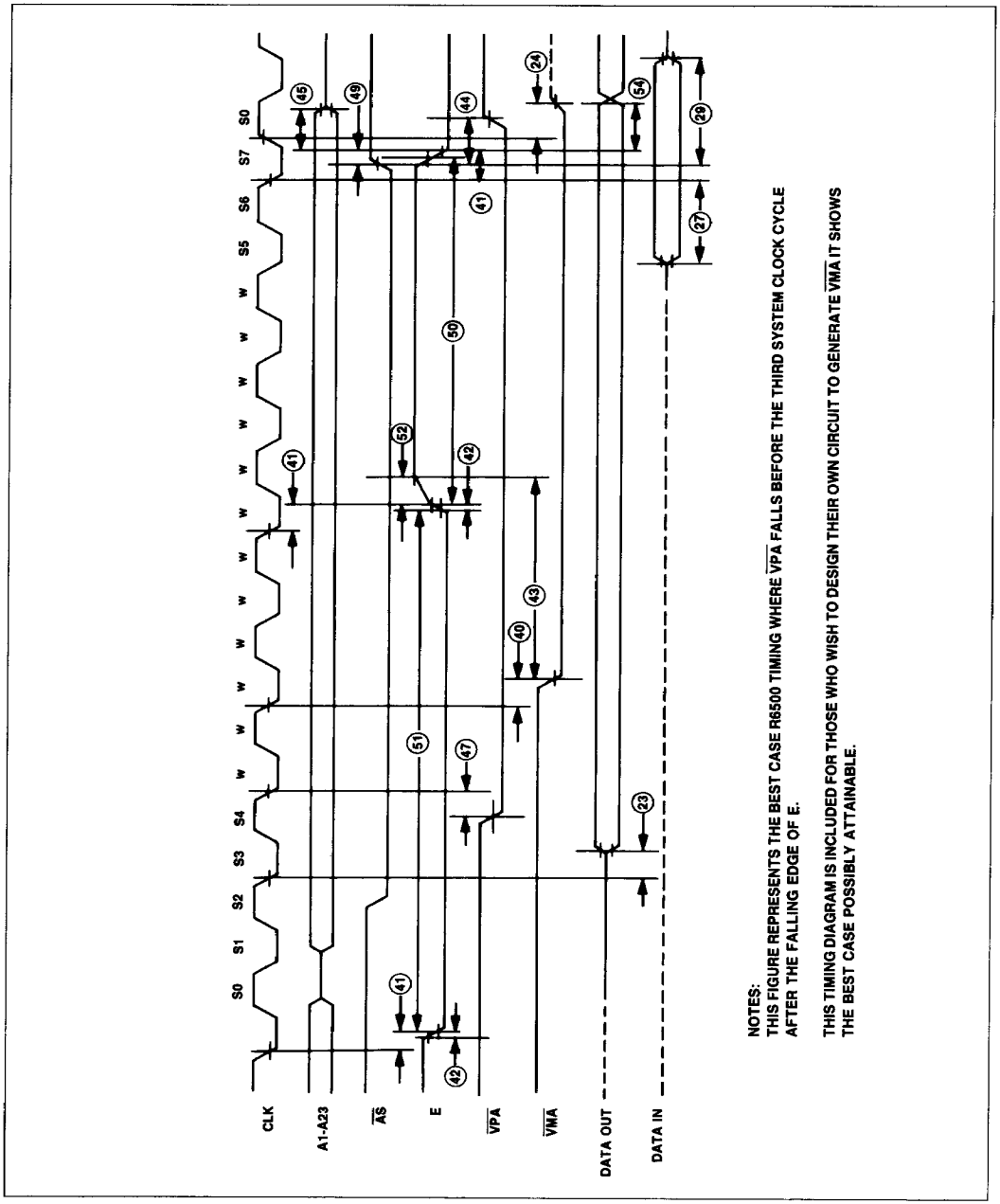


Figure 36. R6500 Timing—Best Case

2. Worst Case—$\overline{VPA}$ is recognized as being asserted on the falling edge two clock cycles before E rises (or four clock cycles after E falls).

Near the end of a read cycle, the processor latches the peripheral's data in state 6. For all cycles, the processor negates the address and data strobes one half clock cycle later in state 7, and the Enable signal goes low at this time. Another half clock later, the address bus is put in the high-impedance state. Upon write cycle completion, the data bus is put in the high-impedance state and the read/write signal is switched high. The peripheral logic must remove $\overline{VPA}$ within one clock after address strobe is negated.

$\overline{DTACK}$ should not be asserted while $\overline{VPA}$ is asserted. Note that the R68000 $\overline{VMA}$ is active low. This allows the processor to put its buses in the high-impedance state on DMA requests without inadvertently selecting peripherals.

INTERRUPT OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector, if $\overline{VPA}$ is asserted, the R68000 will assert $\overline{VMA}$ and complete a normal R6500 read cycle as shown in Figure 38. The processor will then use an internally generated vector, called an autovector, that is a function of the interrupt being served. The seven autovectors are vector numbers 25 through 31 (decimal).

Autovectors operate in the same fashion as (but are not restricted to) the R6500 interrupt sequence. The basic difference is that there are six normal interrupt vectors and one NMI type vector. As with both the R6500 and the R68000's normal vectored interrupt, the interrupt service routine can be located anywhere in the address space. This is due to the fact that while the vector numbers are fixed, the contents of the vector table entries are assigned by the user.

Since $\overline{VMA}$ is asserted during autovectoring, the R6500 peripheral address decoding should prevent unintended accesses.

DATA TYPES AND ADDRESSING MODES

Five basic data types are supported. These data types are:

Bits
BCD Digits (4-bits)
Bytes (8-bits)
Word (16-bits)
Long Words (32-bits)

In addition, operations on other data types such as memory addresses, status word data, etc., are provided for in the instruction set.

The 14 addressing modes, shown in Table 9, include six basic types:

Register Direct	Program Counter Relative
Register Indirect	Implied
Absolute	Immediate

Included in the register indirect addressing modes is the capability to do postincrementing, predecrementing, offsetting and indexing. Program counter relative mode can also be modified via indexing and offsetting.

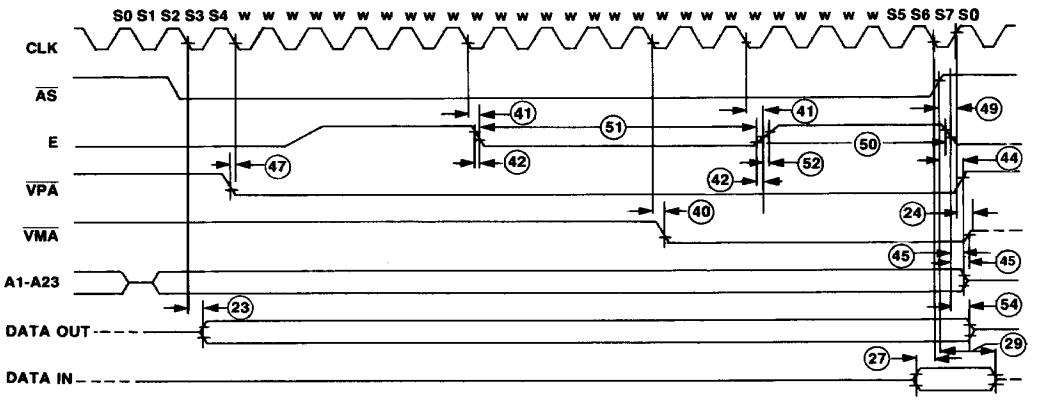
Table 9. Addressing Modes

Mode	Generation
Register Direct Addressing	
Data Register Direct	EA = Dn
Address Register Direct	EA = An
Absolute Data Addressing	
Absolute Short	EA = (Next Word)
Absolute Long	EA = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	EA = (PC) + d_{16}
Relative with Index and Offset	EA = (PC) + (Xn) + d_8
Register Indirect Addressing	
Register Indirect	EA = (An)
Postincrement Register Indirect	EA = (An), An ← An + N
Predecrement Register Indirect	An ← An − N, EA = (An)
Register Indirect with Offset	EA = (An) + d_{16}
Indexed Register Indirect with Offset	EA = (An) + (Xn) + d_8
Immediate Data Addressing	
Immediate	DATA = Next Word(s)
Quick Immediate	Inherent Data
Implied Addressing	
Implied Register	EA = SR, USP, SP, PC

NOTES:
EA = Effective Address
An = Address Register
Dn = Data Register
Xn = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
( ) = Contents of
d_8 = Eight-bit Offset (displacement)
d_{16} = Sixteen-bit Offset (displacement)
N = 1 for Byte, 2 for Words and 4 for Long Word. If An is the stack pointer and the operand size is byte, N = 2 to keep the stack pointer on a word boundry.
← = Replaces

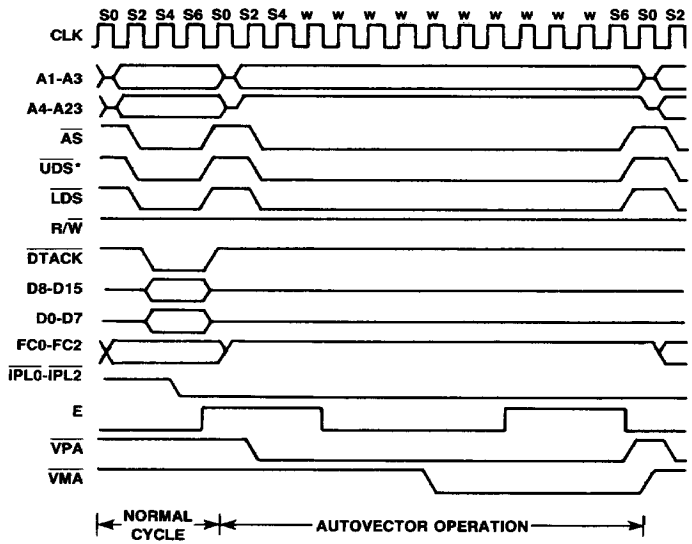
INSTRUCTION SET OVERVIEW

The R68000 instruction set is shown in Table 10. Some additional instructions are variations, or subsets, of these and they appear in Table 11. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned multiply and divide, "quick" arithmetic operations. BCD arithmetic and expanded operations (through traps).



NOTE: THIS TIMING DIAGRAM IS INCLUDED FOR THOSE WHO WISH TO DESIGN THEIR OWN CIRCUIT TO GENERATE VMA. IT SHOWS THE WORST CASE POSSIBLY ATTAINABLE.

Figure 37. RC68000 to R6500 Peripheral Timing Diagram — Worst Case



*ALTHOUGH A VECTOR NUMBER IS ONE BYTE, BOTH DATA STROBES ARE ASSERTED DUE TO THE MICROCODE USED FOR EXCEPTION PROCESSING. THE PROCESSOR DOES NOT RECOGNIZE ANYTHING ON DATA LINES D8 THROUGH D15 AT THIS TIME.

Figure 38. Autovector Operation Timing Diagram

Table 10. Instruction Set Summary

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
ADBC	Add Decimal with Extend	EOR	Exclusive Or	PEA	Push Effective Address
ADD	Add	EXG	Exchange Registers		
AND	Logical And	EXT	Sign Extend	RESET	Reset External Devices
ASL	Arithmetic Shift Left			ROL	Rotate Left without Extend
ASR	Arithmetic Shift Right	JMP	Jump	ROR	Rotate Right without Extend
		JSR	Jump to Subroutine	ROXL	Rotate Left with Extend
B_{CC}	Branch Conditionally			ROXR	Rotate Right with Extend
BCHG	Bit Test and Change	LEA	Load Effective Address	RTE	Return from Exception
BCLR	Bit Test and Clear	LINK	Link Stack	RTR	Return and Restore
BRA	Branch Always	LSL	Logical Shift Left	RTS	Return from Subroutine
BSET	Bit Test and Set	LSR	Logical Shift Right		
BSR	Branch to Subroutine			SBCD	Subtract Decimal with Extend
BTST	Bit Test	MOVE	Move	S_{CC}	Set Conditional
		MULS	Signed Multiply	STOP	Stop
CHK	Check Register Against Bounds	MULU	Unsigned Multiply	SUB	Subtract
CLR	Clear Operand			SWAP	Swap Data Register Halves
CMP	Compare	NBCD	Negate Decimal with Extend		
		NEG	Negate	TAS	Test and Set Operand
DB_{CC}	Test Condition, Decrement and Branch	NOP	No Operation	TRAP	Trap
		NOT	One's Complement	TRAPV	Trap on Overflow
DIVS	Signed Divide			TST	Test
DIVU	Unsigned Divide	OR	Logical Or		
				UNLK	Unlink

4

Table 11. Variations of Instruction Types

Instruction Type	Variation	Description	Instruction Type	Variation	Description
ADD	**ADD**	Add	**MOVE**	**MOVE**	Move
	ADDA	Add Address		MOVEA	Move Address
	ADDQ	Add Quick		MOVEM	Move Multiple Registers
	ADDI	Add Immediate		MOVEP	Move Peripheral Data
	ADDX	Add with Extend		MOVEQ	Move Quick
				MOVE from SR	Move from Status Register
AND	**AND**	Logical And		MOVE to SR	Move to Status Register
	ANDI	And Immediate		MOVE to CCR	Move to Condition Codes
	ANDI to CCR	And Immediate to Condition Codes		MOVE USP	Move User Stack Pointer
	ANDI to SR	And Immediate to Status Register	**NEG**	**NEG**	Negate
				NEGX	Negate with Extend
CMP	**CMP**	Compare	**OR**	**OR**	Logical Or
	CMPA	Compare Address		ORI	Or Immediate
	CMPM	Compare Memory		ORI to CCR	Or Immediate to Condition Codes
	CMPI	Compare Immediate		ORI to SR	Or Immediate to Status Register
EOR	**EOR**	Exclusive Or			
	EORI	Exclusive Or Immediate	**SUB**	**SUB**	Subtract
	EORI to CCR	Exclusive Or Immediate to Condition Codes		SUBA	Subtract Address
				SUBI	Subtract Immediate
	EORI to SR	Exclusive Or Immediate to Status Register		SUBQ	Subtract Quick
				SUBX	Subtract with Extend

The following paragraphs contain an overview of the form and structure of the R68000 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

Data Movement
Integer Arithmetic
Logical
Shift and Rotate
Bit Manipulation
Binary Coded Decimal
Program Control
System Control

The complete range of instruction capabilities combined with the flexible addressing modes described previously provide a very flexible base for program development.

ADDRESSING

Instructions for the R68000 contain two kinds of information: the type of function to be performed, and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

Register Specification — the number of the register is given in the register field of the instruction.

Effective Address — use of the different effective address modes.

Implicit Reference — the definition of certain instructions implies the use of specific registers.

DATA MOVEMENT OPERATIONS

The move (MOVE) instruction provides a means for data acquisition (transfer and storage). The move instruction and the effective addressing modes allow both address and data manipulation. Data move instructions allow byte, word, and long word operands to be transferred from memory to memory, memory to register, register to memory, and register to register. Address move instructions allow word and long word operand transfers and ensure that only legal address manipulations are executed. In addition to the general move instruction there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), unlink stack (UNLK), and move quick (MOVEQ). Table 12 summarizes the data movement operations.

INTEGER ARITHMETIC OPERATIONS

The arithmetic operators include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP), clear (CLR), and negate (NEG). The add and subtract instructions are available for both address and data operations, and with data operations accepting all operand sizes. Address operations are limited to legal address size operands (16 or 32 bits). Data, address, and memory compare operations are also available. The clear and negate instructions may be used on all sizes of data operands.

Table 12. Data Movement Operations

Instruction	Operand Size	Operation
EXG	32	Rx↔Ry
LEA	32	EA→An
LINK	—	An →–(SP) SP →An SP + displacement →SP
MOVE	8, 16, 32	s →d
MOVEM	16, 32	(EA) →An, Dn An, Dn →EA
MOVEP	16, 32	(EA) →Dn Dn →(EA)
MOVEQ	8	#xxx →Dn
PEA	32	EA →–(SP)
SWAP	32	Dn[31:16]↔Dn[15:0]
UNLK	—	An →Sp (SP) + →An

NOTES:
s = source –( ) = indirect with predecrement
d = destination ( )+ = indirect with postdecrement
[ ] = bit number # = immediate data

The multiply and divide operations are available for signed and unsigned operands using word multiply to produce a long word product, and a long word dividend with word divisor to produce a word quotient with a word remainder.

Multiprecision and mixed size arithmetic can be accomplished using a set of extended instructions. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

A text operand (TST) instruction that sets the condition codes as a result of a compare of the operand with zero is available. Test and set (TAS) is a synchronization instruction useful in multiprocessor systems. Table 13 summrizes the integer arithmetic operations.

INSTRUCTION FORMAT

Instructions, as shown in Figure 39, vary from one to five words in length. The first word of the instruction, called the operation word, specifies the length of the instruction and the operation to be performed. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective address mode specified in the operation word.

Table 13. Integer Arithmetic Operations

Instruction	Operand Size	Operation
ADD	8, 16, 32 16, 32	Dn + (EA) → Dn (EA) + Dn → (EA) (EA) + #xxx → (EA) An + (EA) → An
ADDX	8, 16, 32 16, 32	Dx + Dy + X → Dx −(Ax) + −(Ay) + X → (Ax)
CLR	8, 16, 32	0 → EA
CMP	8, 16, 32 16, 32	Dn − (EA) (EA) − #xxx (Ax)+ − (Ay)− An − (EA)
DIVS	32 ÷ 16	Dn ÷ (EA) → Dn
DIVU	32 ÷ 16	Dn ÷ (EA) → Dn
EXT	8 → 16 16 → 32	$(Dn)_8 \rightarrow Dn_{16}$ $(Dn)_{16} \rightarrow Dn_{32}$
MULS	16 × 16 → 32	Dn × (EA) → Dn
MULU	16 × 16 → 32	Dn × (EA) → Dn
NEG	8, 16, 32	0 − (EA) → (EA)
NEGX	8, 16, 32	0 − (EA) − X → (EA)
SUB	8, 16, 32 16, 32	Dn − (EA) → Dn (EA) − Dn → (EA) (EA) − #xxx → (EA) An − (EA) → An
SUBX	8, 16, 32	Dx − Dy − X → Dx −(Ax) − −(Ay) − X → (Ax)
TAS	8	[EA] − 0, 1 → EA[7]
TST	8, 16, 32	(EA) − 0

NOTES:
[ ] = bit number
−( ) = indirect with predecrement
( )+ = indirect with postdecrement
= immediate data

PROGRAM/DATA REFERENCES

The R68000 separates memory references into two classes: program references, and data references. Program references reference that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Operand reads are from the data space, except in the case of the program counter relative addressing mode. All operand writes are to the data space.

REGISTER SPECIFICATION

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 40 shows the general format of the single effective address instruction operation word. The effective address is composed of two 3-bit fields: the mode field, and the register field. The value in the mode field selects the different address modes. The register field contains the number of a register.

The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in the following word or words and is considered part of the instruction, as shown in Figure 39. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

REGISTER DIRECT MODES. These effective addressing modes specify that the operand is in one of the 16 multifunction registers.

Data Register Direct. The operand is in the data register specified by the effective address register field.

Address Register Direct. The operand is in the address register specified by the effective address register field.

MEMORY ADDRESS MODES. These effective addressing modes specify that the operand is in memory and provide the specific address of that operand.

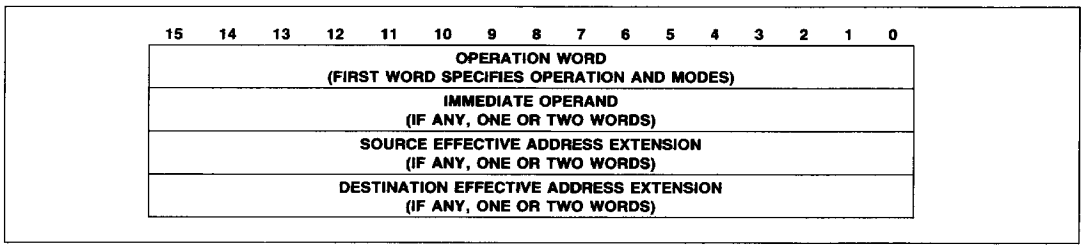


Figure 39. Instruction Format

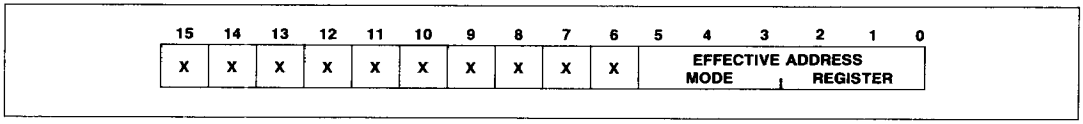


Figure 40. Single-Effective-Address Instruction Operation Word General Format

Address Register Indirect. The address of the operand is in the address register specified by the register field. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

Address Register Indirect With Postincrement. The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two or four depending upon whether the size of the operand is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer on a word boundary. The reference classifies as a data reference.

Address Register Indirect With Predecrement. The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four depending upon whether the operand size is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

Address Register Indirect with Displacement. This address mode requires one word of extension. The address of the operand is the sum of the address in the address register and the sign-extended 16-bit displacement integer in the extension word. The reference classifies as a data reference with the exception of the jump to subroutine instructions.

Address Register Indirect With Index. This address mode requires one word of extension. The address of the operand sums the addresses in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

SPECIAL ADDRESS MODE. The special address modes use the effective address register field to specify the special addressing mode instead of a register number.

Absolute Short Address. This address mode requires one word of extension. The address of the operand is the extension word. The 16-bit address is sign extended before it is used. The reference classifies as a data reference with the exception of the jump and jump to subroutine instructions.

Absolute Long Address. This address mode requires two words of extension. The address of the operand is developed by the concatenation of the extension words. The high-order part of the address is the first extension word; the low-order part of the address is the second extension word. The reference classifies as a data reference with the exception of the jump and jump to subroutine instructions.

Program Counter With Displacement. This address mode requires one word of extension. The address of the operand sums the addresses in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference classifies as a program reference.

Program Counter With Index. This address mode requires one word of extension. This address sums the addresses in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The value in the program counter is the address of the extension word. This reference classifies as a program reference.

Immediate Data. This address mode requires either one or two words of extension depending on the size of the operation.

Byte Operation	— operand is low order byte of extension word
Word Operation	— operand is extension word
Long Word Operation	— operand is in the two extension words, high-order 16 bits are in the first extension word, low-order 16 bits are in the second extension word.

IMPLICIT REFERENCE

Some instructions make implicit reference to the program counter (PC), the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), or the status register (SR).

A selected set of instructions may reference the status register by means of the effective address field. These are:

ANDI to CCR
ANDI to SR
EORI to CCR
EORI to SR
MOVE to CCR
MOVE to SR
MOVE from SR
ORI to CCR
ORI to SR

EFFECTIVE ADDRESS ENCODING SUMMARY

Table 14 summarizes the effective addressing modes discussed in the previous paragraphs.

Table 14. Effective Address Encoding Summary

Addressing Mode	Mode	Register
Data Register Direct	000	register number
Address Register Direct	001	register number
Address Register Indirect	010	register number
Address Register Indirect with Postincrement	011	register number
Address Register Indirect with Predecrement	100	register number
Address Register Indirect with Displacement	101	register number
Address Register Indirect with Index	110	register number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate	111	100

SYSTEM STACK. The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes. Address register seven (A7) is the system stack pointer (SP). The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S-bit in the status register. If the S-bit indicates supervisor state (High), SSP is the active system stack pointer, and the USP cannot be referenced as an address register. If the S-bit indicates user state (Low), the USP is the active system stack pointer, and the SSP cannot be referenced. Each system stack fills from high memory to low memory.

LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are available for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 15 summarizes the logical operations.

SHIFT AND ROTATE OPERATIONS

Shift operations in both directions are provided by arithmetic instructions ASR and ASL and logical shift instructions LSR and LSL. The rotate instructions (with and without extend) available are ROXR, ROXL, ROR, and ROL. All shift and rotate operations can be performed in either registers or memory. Register shifts and rotates support all operand sizes and allow a shift count specified in a data register.

Table 15. Logical Operations

Instruction	Operand Size	Operation
AND	8, 16, 32	Dn∧(EA) →Dn (EA)∧Dn →(EA) (EA)∧#xxx →(EA)
OR	8, 16, 32	Dn ∨ (EA) →Dn (EA) ∨ Dn →(EA) (EA) ∨ #xxx →(EA)
EOR	8, 16, 32	(EA) ⊕ Dy →(EA) (EA) ⊕ #xxx →(EA)
NOT	8, 16, 32	~ (EA) →(EA)

NOTES:
~ = invert
# = immediate data
∧ = logical AND
∨ = logical OR
⊕ = logical exclusive OR

Memory shifts and rotates are for word operands only and allow only single-bit shifts or rotates.

Table 16 summarizes the shift and rotate operations.

BIT MANIPULATION OPERATIONS

The following instructions provide bit manipulation operations: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). Table 17 is a summary of the bit manipulation operations. (Bit 2 of the status register is Z.)

Table 16. Shift and Rotate Operations

Instruction	Operand Size	Operation
ASL	8, 16, 32	X/C ← 0
ASR	8, 16, 32	→ X/C
LSL	8, 16, 32	X/C ← 0
LSR	8, 16, 32	0 → X/C
ROL	8, 16, 32	C ←
ROR	8, 16, 32	→ C
ROXL	8, 16, 32	C ← X
ROXR	8, 16, 32	X → C

Table 17. Bit Manipulation Operations

Instruction	Operand Size	Operation
BTST	8, 32	~ bit of (EA) →Z
BSET	8, 32	~ bit of (EA) →Z 1 →bit of EA
BCLR	8, 32	~ bit of (EA) →Z 0 →bit of EA
BCHG	8, 32	~ bit of (EA) →Z ~ bit of (EA) →bit of EA
NOTE: ~ = invert		

BINARY CODED DECIMAL OPERATIONS

The following instructions accomplish multiprecision arithmetic operations on binary coded decimal numbers: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). Table 18 summarizes the binary coded decimal operations.

PROGRAM CONTROL OPERATIONS

Program control operations implementation requires a series of conditional and unconditional branch instructions and return instructions. These instructions are summarized in Table 19.

The conditional instructions provide setting and branching for the following conditions:

CC — carry clear
CS — carry set
EQ — equal
F — never true
GE — greater or equal
GT — greater than
HI — high
LE — less or equal
LS — low or same
LT — less than
MI — minus
NE — not equal
PL — plus
T — always true
VC — no overflow
VS — overflow

Table 18. Binary Coded Decimal Operations

Instruction	Operand Size	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $-(Ax)_{10} + -(Ay)_{10} + x \rightarrow (Ax)$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $-(Ax)_{10} - -(Ay)_{10} - X \rightarrow (Ax)$
NBCD	8	$0 - (EA)_{10} - X \rightarrow (EA)$
NOTE: −( ) = indirect with predecrement		

Table 19. Program Control Operations

Instruction	Operation
Conditional	
B_{CC}	Branch conditionally (14 conditions) 8- and 16-bit displacement
DB_{CC}	Test condition, decrement, branch 16-bit displacement
S_{CC}	Set byte conditionally (16 conditions)
Unconditional	
BRA	Branch always 8- and 16-bit displacement
BSR	Branch to subroutine 8- and 16-bit displacement
JMP	Jump
JSR	Jump to subroutine
Returns	
RTR	Return and restore condition codes
RTS	Return from subroutine

SYSTEM CONTROL OPERATIONS

System control operations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the status register. These instructions are summarized in Table 20.

INSTRUCTION SET

The following paragraphs provide information about the addressing categories and instruction set of the R68000.

ADDRESSING CATEGORIES

Effective address modes may be categorized by the ways in which they may be used. The following classifications will be used in the instructions definitions.

Data — If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.

Memory — If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.

Alterable — If an effective address mode may be used to refer to alterable (writeable) operands, it is considered an alterable addressing effective address mode.

Control — If an effective address mode may be used to refer to memory operands without an associated size, it is considered control addressing effective address mode.

Table 21 shows the various categories to which each of the effective address modes belong. Table 22 is the instruction set summary.

Table 20. System Control Operations

Instruction	Operation
Privileged	
ANDI to SR	Logical AND to Status Register
EORI to SR	Logical EOR to Status Register
MOVE EA to SR	Load New Status Register
MOVE USP	Move User Stack Pointer
ORI to SR	Logical OR to Status Register
RESET	Reset External Devices
RTE	Return from Exception
STOP	Stop Program Execution
Trap Generating	
CHK	Check Data Register Against Upper Bounds
TRAP	Trap
TRAPV	Trap on Overflow
Status Register	
ANDI to CCR	Logical AND to Condition Codes
EORI to CCR	Logical EOR to Condition Codes
MOVE EA to CCR	Load New Condition Codes
MOVE SR to EA	Store Status Register
ORI to CCR	Logical OR to Condition Codes

The status register addressing mode is not permitted unless it is explicitly mentioned as a legal addressing mode.

These categories may be combined, so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable.

INSTRUCTION PREFETCH

The R68000 uses a two-word tightly-coupled instruction prefetch mechanism to enhance performance. This mechanism is described in terms of the microcode operations involved. If the execution of an instruction is defined to begin when the microroutine for that instruction is entered, some features of the prefetch mechanism can be described.

1) When execution of an instruction begins, the operation word and the word following have already been fetched. The operation word is in the instruction decoder.
2) In the case of multi-word instructions, as each additional word of the instruction is used internally, a fetch is made to the instruction stream to replace it.
3) The last fetch from the instruction stream is made when the operation word is discarded and decoding is started on the next instruction.
4) If the instruction is a single-word instruction causing a branch, the second word is not used. But because this word is fetched by the preceding instruction, it is impossible to avoid this superfluous fetch.
5) In the case of an interrupt or trace exception, both words are not used.
6) The program counter usually points to the last word fetched from the instruction stream.

INSTRUCTION EXECUTION TIMES

The following paragraphs contain listings of the instruction execution times in terms of external clock (CLK) periods. In this timing data, it is assumed that both memory read and write cycle times are four clock periods. Any wait states caused by a longer memory cycle must be added to the total instruction time. The number of bus read and write cycles for each instruction is enclosed in parenthesis following the execution periods and is shown as (r/w) where r is the number of read cycles and w is the number of write cycles.

Table 21. Effective Addressing Mode Categories

Effective Address Modes	Mode	Register	Addressing Categories			
			Data	Memory	Control	Alterable
Dn	000	Register Number	X	—	—	X
An	001	Register Number	—	—	—	X
(An)	010	Register Number	X	X	X	X
(An)+	011	Register Number	X	X	—	X
−(An)	100	Register Number	X	X	—	X
d(An)	101	Register Number	X	X	X	X
d(An, ix)	110	Register Number	X	X	X	X
xxx.W	111	000	X	X	X	X
xxx.L	111	001	X	X	X	X
d(PC)	111	010	X	X	X	—
d(PC, ix)	111	011	X	X	X	—
#xxx	111	X	X	X	—	—

Table 22. Instruction Set

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
ABCD	Add Decimal with Extend	(Destination)$_{10}$ + (Source)$_{10}$ + X → Destination	*	U	*	U	*
ADD	Add Binary	(Destination) + (Source) → Destination	*	*	*	*	*
ADDA	Add Address	(Destination) + (Source) → Destination	—	—	—	—	—
ADDI	Add Immediate	(Destination) + Immediate Data → Destination	*	*	*	*	*
ADDQ	Add Quick	(Destination) + Immediate Data → Destination	*	*	*	*	*
ADDX	Add Extended	(Destination) + (Source) + X → Destination	*	*	*	*	*
AND	AND Logical	(Destination) Λ (Source) → Destination	—	*	*	0	0
ANDI	AND Immediate	(Destination) Λ Immediate Data → Destination	—	*	*	0	0
ANDI to CCR	AND Immediate to Condition Codes	(Source) Λ CCR → CCR	*	*	*	*	*
ANDI to SR	AND Immediate to Status Register	(Source) Λ SR → SR	*	*	*	*	*
ASL, ASR	Arithmetic Shift	(Destination) Shifted by <count> → Destination	*	*	*	*	*
B$_{CC}$	Branch Conditionally	If $_{CC}$ then PC + d → PC	—	—	—	—	—
BCHG	Test a Bit and Change	~ (<bit number>) OF Destination → Z ~ (<bit number>) OF Destination → <bit number> OF Destination	—	—	*	—	—
BCLR	Test a Bit and Clear	~ (<bit number>) OF Destination → Z 0 → <bit number> → OF Destination	—	—	*	—	—
BRA	Branch Always	PC + d → PC	—	—	—	—	—
BSET	Test a Bit and Set	~ (<bit number>) OF Destination → Z 1 → <bit number> OF Destination	—	—	*	—	—
BSR	Branch to Subroutine	PC → (SP); PC + d → PC	—	—	—	—	—
BTST	Test a Bit	~ (<bit number>) OF Destination → Z	—	—	*	—	—
CHK	Check Register Against Bounds	If Dn <0 or Dn > (<ea>) then TRAP	—	*	U	U	U
CLR	Clear and Operand	0 → Destination	—	0	1	0	0
CMP	Compare	(Destination) – (Source)	—	*	*	*	*
CMPA	Compare Address	(Destination) – (Source)	—	*	*	*	*
CMPI	Compare Immediate	(Destination) – Immediate Data	—	*	*	*	*
CMPM	Compare Memory	(Destination) – (Source)	—	*	*	*	*
DB$_{CC}$	Test Condition, Decrement and Branch	If ~ $_{CC}$ then Dn – 1 → Dn; if Dn ≠ – 1 then PC + d → PC	—	—	—	—	—
DIVS	Signed Divide	(Destination)/(Source) → Destination	—	*	*	*	0
DIVU	Unsigned Divide	(Destination)/(Source) → Destination	—	*	*	*	0
EOR	Exclusive OR Logical	(Destination) ⊕ (Source) → Destination	—	*	*	0	0
EORI	Exclusive OR Immediate	(Destination) ⊕ Immediate Data → Destination	—	*	*	0	0
EORI to CCR	Exclusive OR Immediate to Condition Codes	(Source) ⊕ CCR → CCR	*	*	*	*	*

NOTES:

Λ = logical AND
v = logical OR
⊕ = logical exclusive OR
~ = logical complement

* = affected
— = unaffected
0 = cleared
1 = set
U = undefined

Table 22. Instruction Set (Continued)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
EORI to SR	Exclusive OR Immediate to Status Register	(Source) ⊕ SR → SR	*	*	*	*	*
EXG	Exchange Register	Rx ↔ Ry	—	—	—	—	—
EXT	Sign Extend	(Destination) Sign-Extended → Destination	—	*	*	0	0
JMP	Jump	Destination → PC	—	—	—	—	—
JSR	Jump to Subroutine	PC → −(SP); Destination → PC	—	—	—	—	—
LEA	Load Effective Address	<ea> → An	—	—	—	—	—
LINK	Link and Allocate	An → (SP); SP → An; SP + Displacement → SP	—	—	—	—	—
LSL, LSR	Logical Shift	(Destination) Shifted by <count> → Destination	*	*	*	0	*
MOVE	Move Data from Source to Destination	(Source) → Destination	—	*	*	0	0
MOVE to CCR	Move to Condition Code	(Source) → CCR	*	*	*	*	*
MOVE to SR	Move to the Status Register	(Source) → SR	*	*	*	*	*
MOVE from SR	Move from the Status Register	SR → Destination	—	—	—	—	—
MOVE USP	Move User Stack Pointer	USP → An; An → USP	—	—	—	—	—
MOVEA	Move Address	(Source) → Destination	—	—	—	—	—
MOVEM	Move Multiple Registers	Register → Destination (Source) → Registers	—	—	—	—	—
MOVEP	Move Peripheral Data	(Source) → Destination	—	—	—	—	—
MOVEQ	Move Quick	Immediate Data → Destination	—	*	*	0	0
MULS	Signed Multiply	(Destination)X(Source) → Destination	—	*	*	0	0
MULU	Unsigned Multiply	(Destination)X(Source) → Destination	—	*	*	0	0
NBCD	Negate Decimal with Extend	0 − (Destination)$_{10}$ − X → Destination	*	U	*	U	*
NEG	Negate	0 − (Destination) → Destination	*	*	*	*	*
NEGX	Negate with Extend	0 − (Destination) − X → Destination	*	*	*	*	*
NOP	No Operation	—	—	—	—	—	—
NOT	Logical Complement	~ (Destination) → Destination	—	*	*	0	0
OR	Inclusive OR Logical	(Destination) ∨ (Source) → Destination	—	*	*	0	0
ORI	Inclusive OR Immediate	(Destination) ∨ Immediate Data → Destination	—	*	*	0	0
ORI to CCR	Inclusive OR Immediate to Condition Codes	(Source) ∨ CCR → CCR	*	*	*	*	*
ORI to SR	Inclusive OR Immediate to Status Register	(Source) ∨ SR → SR	*	*	*	*	*
PEA	Push Effective Address	<ea> → −(SP)	—	—	—	—	—
RESET	Reset External Device	—	—	—	—	—	—
ROL, ROR	Rotate (Without Extend)	(Destination) Rotated by <count> → Destination	—	*	*	0	*

NOTES:
∧ = logical AND
∨ = logical OR
⊕ = logical exclusive OR
~ = logical complement

* = affected
— = unaffected
0 = cleared
1 = set
U = undefined

Table 22. Instruction Set (Continued)

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
ROXL, ROXR	Rotate with Extend	(Destination) Rotated by <count> →Destination	*	*	*	0	*
RTE	Return from Exception	(SP) + →SR; (SP) + →PC	*	*	*	*	*
RTR	Return and Restore Condition Codes	(SP) + →CC; (SP) + →PC	*	*	*	*	*
RTS	Return from Subroutine	(SP) + →PC	—	—	—	—	—
SBCD	Subtract Decimal with Extend	$(Destination)_{10}$ − $(Source)_{10}$ − X →Destination	*	U	*	U	*
S_{CC}	Set According to Condition	If CC then 1's →Destination else 0's →Destination	—	—	—	—	—
STOP	Load Status Register and Stop	Immediate Data →SR; STOP	*	*	*	*	*
SUB	Subtract Binary	(Destination) − (Source) →Destination	*	*	*	*	*
SUBA	Subtract Address	(Destination) − (Source) →Destination	—	—	—	—	—
SUBI	Subtract Immediate	(Destination) − Immediate Data →Destination	*	*	*	*	*
SUBQ	Subtract Quick	(Destination) − Immediate Data →Destination	*	*	*	*	*
SUBX	Subtract with Extend	(Destination) − (Source) − X →Destination	*	*	*	*	*
SWAP	Swap Register Halves	Register [31:16]↔Register [15:0]	—	*	*	0	0
TAS	Test and Set an Operand	(Destination) Tested →CC; 1 →[7] OF Destination	—	*	*	0	0
TRAP	Trap	PC →− (SSP); SR →− (SSP); (Vector) →PC	—	—	—	—	—
TRAPV	Trap on Overflow	If ν then TRAP	—	—	—	—	—
TST	Test and Operand	(Destination) Tested →CC	—	*	*	0	0
UNLK	Unlink	An →SP; (SP) + →An	—	—	—	—	—

NOTES:

[ ]	= bit number	*	= affected
Λ	= logical AND	—	= unaffected
ν	= logical OR	0	= cleared
$\oplus$	= logical exclusive OR	1	= set
~	= logical complement	U	= undefined

Note

The number of periods includes instruction fetch and all applicable operand fetches and stores.

EFFECTIVE ADDRESS OPERAND CALCULATION TIMING

Table 23 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

MOVE INSTRUCTION CLOCK PERIODS

Tables 24 and 25 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

STANDARD INSTRUCTION CLOCK PERIODS

The number of clock periods shown in Table 26 delineate the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 26, the headings have the following meanings: An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand.

IMMEDIATE INSTRUCTION CLOCK PERIODS

The number of clock periods shown in Table 27 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as (r/w). The number

Table 23. Effective Address Calculation Timing

Addressing Mode		Byte, Word	Long
	Register		
Dn	Data Register Direct	0(0/0)	0(0/0)
An	Address Register Direct	0(0/0)	0(0/0)
	Memory		
(An)	Address Register Indirect	4(1/0)	8(2/0)
(An)+	Address Register Indirect with Postincrement	4(1/0)	8(2/0)
−(An)	Address Register Indirect with Predecrement	6(1/0)	10(2/0)
d(An)	Address Register Indirect with Displacement	8(2/0)	12(3/0)
d(An, ix)*	Address Register Indirect with Index	10(2/0)	14(3/0)
xxx.W	Absolute Short	8(2/0)	12(3/0)
xxx.L	Absolute Long	12(3/0)	16(4/0)
d(PC)	Program Counter with Displacement	8(2/0)	12(3/0)
d(PC, ix)*	Program Counter with Index	10(2/0)	14(3/0)
#xxx	Immediate	4(1/0)	8(2/0)

*The size of the index register (ix) does not affect execution time.

4

Table 24. Move Byte and Word Instruction Clock Periods

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
(An)	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
(An)+	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
−(An)	10(2/0)	10(2/0)	14(2/1)	14(2/1)	14(2/1)	18(3/1)	20(3/1)	18(3/1)	22(4/1)
d(An)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d(An, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
xxx.W	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
xxx.L	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	24(5/1)	26(5/1)	24(5/1)	28(6/1)
d(PC)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d(PC, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
#xxx	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)

*The size of the index register (ix) does not affect execution time.

of clock periods and the number of read and write cycles must be added respectively to those of the effective adress calculation where indicated.

In Table 27, the headings have the following meanings: # = immediate operand, Dn = data register operand, An = address register operand, M = memory operand, and SR = status register.

SINGLE OPERAND INSTRUCTION CLOCK PERIODS

Table 28 indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 25. Move Long Instruction Clock Periods

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
(An)	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
(An)+	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
−(An)	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
d(An)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
d(An, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
xxx.W	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
xxx.L	20(5/0)	20(5/0)	28(5/2)	28(5/2)	28(5/2)	32(6/2)	34(6/2)	32(6/2)	36(7/2)
d(PC)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(5/2)
d(PC, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
#xxx	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)

*The size of the index register (ix) does not affect execution time.

Table 26. Standard Instruction Clock Periods

Instruction	Size	op<ea>, An†	op<ea>, Dn	op Dn, <M>
ADD	Byte, Word	8(1/0)+	4(1/0)+	8(1/1)+
	Long	6(1/0)+**	6(1/0)+**	12(1/2)+
AND	Byte, Word	—	4(1/0)+	8(1/1)+
	Long	—	6(1/0)+**	12(1/2)+
CMP	Byte, Word	6(1/0)+	4(1/0)+	—
	Long	6(1/0)+	6(1/0)+	—
DIVS	—	—	158(1/0)+*	—
DIVU	—	—	140(1/0)+*	—
EOR	Byte, Word	—	4(1/0)***	8(1/1)+
	Long	—	8(1/0)***	12(1/2)+
MULS	—	—	70(1/0)+*	—
MULU	—	—	70(1/0)+*	—
OR	Byte, Word	—	4(1/0)+	8(1/1)+
	Long	—	6(1/0)+**	12(1/2)+
SUB	Byte, Word	8(1/0)+	4(1/0)+	8(1/1)+
	Long	6(1/0)+**	6(1/0)+**	12(1/2)+

NOTES:
+ add effective address calculation time
† word or long only
* indicates maximum value
** The base time of six clock periods is increased to eight if the effective address mode is register direct or immediate (effective address time should also be added).
*** Only available effective address mode is data register direct
DIVS, DIVU The divide algorithm used by the R68000 provides less than 10% difference between the best and worst case timings.
MULS, MULU The multiply algorithm requires 38 + 2n clocks where n is defined as:
MULU: n = the number of ones in each <ea>
MULU: n = concatanate the <ea> with a zero as the LSB; n is the resultant number of 10 or 01 patterns in the 17-bit source; i.e., worst case happens when the source is $5555.

Table 27. Immediate Instruction Clock Periods

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte, Word	8(2/0)	—	12(2/1)+
	Long	16(3/0)	—	20(3/2)+
ADDQ	Byte, Word	4(1/0)	8(1/0)*	8(1/1)+
	Long	8(1/0)	8(1/0)	12(1/2)+
ANDI	Byte, Word	8(2/0)	—	12(2/1)+
	Long	16(3/0)	—	20(3/1)+
CMPI	Byte, Word	8(2/0)	—	8(2/0)+
	Long	14(3/0)	—	12(3/0)+
EORI	Byte, Word	8(2/0)	—	12(2/1)+
	Long	16(3/0)	—	20(3/2)+
MOVEQ	Long	4(1/0)	—	—
ORI	Byte, Word	8(2/0)	—	12(2/1)+
	Long	16(3/0)	—	20(3/2)+
SUBI	Byte, Word	8(2/0)	—	12(2/1)+
	Long	16(3/0)	—	20(3/2)+
SUBQ	Byte, Word	4(1/0)	8(1/0)*	8(1/1)+
	Long	8(1/0)	8(1/0)	12(1/2)+

\+ add effective address calculation time
* word only

4

Table 28. Single Operand Instruction Clock Periods

Instruction	Size	Register	Memory
CLR	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NBCD	Byte	6(1/0)	8(1/1)+
NEG	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NEGX	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NOT	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
S_{CC}	Byte, False	4(1/0)	8(1/1)+
	Byte, True	6(1/0)	8(1/1)+
TAS	Byte	4(1/0)	10(1/1)+
TST	Byte, Word	4(1/0)	4(1/0)+
	Long	4(1/0)	4(1/0)+

\+ add effective address calculation time

SHIFT/ROTATE INSTRUCTION CLOCK PERIODS

Table 29 delineates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

BIT MANIPULATION INSTRUCTION CLOCK PERIODS

Table 30 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

CONDITIONAL INSTRUCTION CLOCK PERIODS

Table 31 delineates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

JMP, JSR, LEA, PWA, MOVEM INSTRUCTION CLOCK PERIODS

Table 32 indicates the number of clock periods required for the jump, jump to subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w).

Table 29. Shift/Rotate Instruction Clock Periods

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	—
LSR, LSL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	—
ROR, ROL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	—
ROXR, ROXL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	—

+ add effective address calculation time
n is shift or rotate count

Table 30. Bit Manipulation Instruction Clock Periods

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	—	8(1/1) +	—	12(2/1) +
	Long	8(1/0)*	—	12(2/0)*	—
BCLR	Byte	—	8(1/1) +	—	12(2/1) +
	Long	10(1/0)*	—	14(2/0)*	—
BSET	Byte	—	8(1/1) +	—	12(2/1) +
	Long	8(1/0)*	—	12(2/0)*	—
BTST	Byte	—	4(1/0) +	—	8(2/0) +
	Long	6(1/0)	—	10(2/0)	—

+ add effective address calculation time
* indicates maximum value

Table 31. Conditional Instruction Clock Periods

Instruction	Displacement	Branch Taken	Branch Not Taken
B_{CC}	Byte	10(2/0)	8(1/0)
	Word	10(2/0)	12(2/0)
BRA	Byte	10(2/0)	—
	Word	10(2/0)	—
BSR	Byte	18(2/2)	—
	Word	18(2/2)	—
DB_{CC}	CC true	—	12(2/0)
	CC false	10(2/0)	14(3/0)

Table 32. JMP, JSR, LEA, PEA, MOVEM INSTRUCTION CLOCK PERIODS

Instr	Size	(An)	(An)+	−(An)	d(An)	d(An, Ix)*+	xxx.W	xxx.L	d(PC)	d(PC, Ix)*
JMP	—	8(2/0)	—	—	10(2/0)	14(3/0)	10(2/0)	12(3/0)	10(2/0)	14(3/0)
JSR	—	16(2/2)	—	—	18(2/2)	22(2/2)	18(2/2)	20(3/2)	18(2/2)	22(2/2)
LEA	—	4(1/0)	—	—	8(2/0)	12(2/0)	8(2/0)	12(3/0)	8(2/0)	12(2/0)
PEA	—	12(1/2)	—	—	16(2/2)	20(2/2)	16(2/2)	20(3/2)	16(2/2)	20(2/2)
MOVEM M → R	Word	12 + 4n (3 + n/0)	12 + 4n (3 + n/0)	—	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)	16 + 4n (4 + n/0)	20 + 4n (5 + n/0)	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)
	Long	12 + 8n (3 + 2n/0)	12 + 8n (3 + 2n/0)	—	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)	16 + 8n (4 + 2n/0)	20 + 8n (5 + 2n/0)	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)
MOVEM R → M	Word	8 + 4n (2/n)	—	8 + 4n (2/n)	12 + 4n (3/n)	14 + 4n (3/n)	12 + 4n (3/n)	16 + 4n (4/n)	— —	— —
	Long	8 + 8n (2/2n)	— —	8 + 8n (2/2n)	12 + 8n (3/2n)	14 + 8n (3/2n)	12 + 8n (3/2n)	16 + 8n (4/2n)	— —	— —

n is the number of registers to move
* The size of the index register (ix) does not affect the instruction's execution time

MULTI-PRECISION INSTRUCTION CLOCK PERIODS

Table 33 delineates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The number of read and write cycles is shown in parenthesis as: (r/w).

In Table 33, the headings have the following meanings: Dn = data register operand and M = memory operand.

Table 33. Multi-Precision Instruction Clock Periods

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
CMPM	Byte, Word	—	12(3/0)
	Long	—	20(5/0)
SUBX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
ABCD	Byte	6(1/0)	18(3/1)
SBCD	Byte	6(1/0)	18(3/1)

MISCELLANEOUS INSTRUCTION CLOCK PERIODS

Table 34 and 35 indicate the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

EXCEPTION PROCESSING CLOCK PERIODS

Table 36 delineates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first instruction of the handler routine. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table 34. Miscellaneous Instruction Clock Periods

Instruction	Size	Register	Memory	Instruction	Size	Register	Memory
ANDI to CCR	Byte	**20**(3/0)	—	LINK	—	**16**(2/2)	—
ANDI to SR	Word	**20**(3/0)	—	MOVE from USP	—	**4**(1/0)	—
CHK	—	**10**(1/0) +	—	MOVE to USP	—	**4**(1/0)	—
EORI to CCR	Byte	**20**(3/0)	—	NOP	—	**4**(1/0)	—
EORI to SR	Word	**20**(3/0)	—	RESET	—	**132**(1/0)	—
ORI to CCR	Byte	**20**(3/0)	—	RTE	—	**20**(5/0)	—
ORI to SR	Word	**20**(3/0)	—	RTR	—	**20**(5/0)	—
MOVE from SR	—	**6**(1/0)	**8**(1/1) +	RTS	—	**16**(4/0)	—
MOVE to CCR	—	**12**(2/0)	**12**(2/0) +	STOP	—	**4**(0/0)	—
MOVE to SR	—	**12**(2/0)	**12**(2/0) +	SWAP	—	**4**(1/0)	—
EXG	—	**6**(1/0)	—	TRAPV	—	**4**(1/0)	—
EXT	Word	**4**(1/0)	—	UNLK	—	**12**(3/0)	—
	Long	**4**(1/0)	—				
+ add effective address calculation time							

Table 35. Move Peripheral Instruction Execution Times

Instruction	Size	Register→Memory	Memory→Register
MOVEP	Word	**16**(2/2)	**16**(4/0)
	Long	**24**(2/4)	**24**(6/0)

Table 36. Exception Processing Clock Periods

Exception	Periods
Address Error	**50**(4/7)
Bus Error	**50**(4/7)
CHK Instruction	**44**(5/4) +
Divide by Zero	**42**(5/4)
Illegal Instruction	**34**(4/3)
Interrupt	**44**(5/3)*
Privilege Violation	**34**(4/3)
$\overline{\text{RESET}}$**	**40**(6/0)
Trace	**34**(4/3)
TRAP Instruction	**38**(4/4)
TRAPV Instruction	**34**(4/3)

\+ add effective address calculation time
* The interrupt acknowledge cycle is assumed to take four clock periods.
** Indicates the time from when $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are first sampled as negated to when instruction execution starts.

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	−0.3 to +7.0	V
Input Voltage	V_{IN}	−0.3 to +7.0	V
Operating Temperature Range	T_A	T_L to T_H*	°C
Storage Temperature	T_{STG}	−65 to +150	°C
*See ordering information			

THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance	θ_{JA}		
64-Pin Ceramic		30	°C/W
64-Pin Plastic Dip		55 ±5	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either V_{SS} or V_{CC}.

POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad (1)$$

Where:

T_A ≡ Ambient Temperature, °C

θ_{JA} ≡ Package Thermal Resistance, Junction-to-Ambient, °C/W

$P_D \equiv P_{INT} + P_{I/O}$

$P_{INT} \equiv I_{CC} \cdot V_{CC}$, Watts—Chip Internal Power

$P_{I/O}$ ≡ Power Dissipation on Input and Output Pins—User Determined

For most applications $P_{I/O} \ll P_{INT}$ and can be neglected.

An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

$$P_D = K \div (T_J + 273°C) \quad (2)$$

Solving equations 1 and 2 for K gives:

$$K = P_D \cdot (T_A + 273°C) + \theta_{JA} \cdot P_D^2 \quad (3)$$

Where K is a constant pertaining to the particular part. K can be determined from equation 3 by measuring P_D (at equilibrium) for a known T_A. Using this value of K the values of P_D and T_J can be obtained by solving equations (1) and (2) iteratively for any value of T_A.

4

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 5.0 Vdc ±5%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H °C. See Figures 41, 42, and 43.)

Characteristic	Symbol	Min	Max	Unit	Test Conditions
Input High Voltage	V_{IH}	2.0	V_{CC}	V	
Input Low Voltage	V_{IL}	V_{SS} − 0.3	0.8	V	
Input Leakage Current	I_{IN}				
$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, CLK, $\overline{IPL0}$–$\overline{IPL2}$, $\overline{VPA}$		—	2.5	μA	V_{IN} = 5.25
$\overline{HALT}$, $\overline{RESET}$		—	20	μA	V_{CC} = OV
Three-State (Off State) Input Current	I_{TSI}				
$\overline{AS}$, A1–A23, D0–D15, FC0–FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$		—	20	μA	V_{IN} = 0.4V to 2.4V V_{CC} = 5.25V
Output High Voltage	V_{OH}				
E*		V_{CC} − 0.75	—	V	V_{CC} = 4.75V
E, $\overline{AS}$, A1–A23, $\overline{BG}$, D0–D15, FC0–FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$		2.4	—	V	I_{OH} = −400 μA
Output Low Voltage	V_{OL}				V_{CC} = 4.75V
$\overline{HALT}$		—	0.5	V	(I_{OL} = 1.6 mA)
$\overline{BG}$, FC0–FC2, A1–A23		—	0.5	V	(I_{OL} = 3.2 mA)
$\overline{RESET}$		—	0.5	V	(I_{OL} = 5.0 mA)
$\overline{AS}$, D0–D15, $\overline{LDS}$, R/$\overline{W}$ $\overline{UDS}$, $\overline{VMA}$, E		—	0.5	V	(I_{OL} = 5.3 mA)
Power Dissipation	P_D***	—	1.5	W	
Input Capacitance	C_{IN}	—	20.0	pF	V_{CC} = 5.0V, V_{IN} = OV f = 1 MHz, T_A = 25°C

*With external pullup resistor of 1.1 kΩ

**Capacitance is periodically sampled rather than 100% tested.

***During normal operation instantaneous V_{CC} current requirements may be as high as 1.5A.

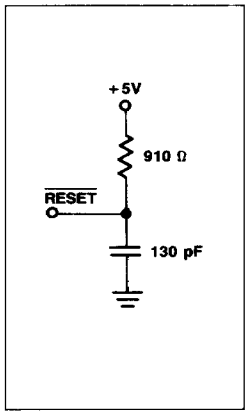


Figure 41. $\overline{\text{RESET}}$ Test Load

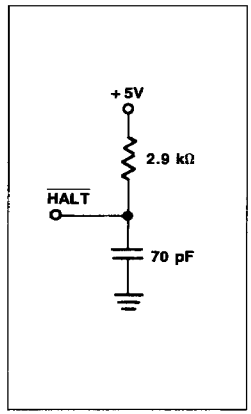


Figure 42. $\overline{\text{HALT}}$ Test Load

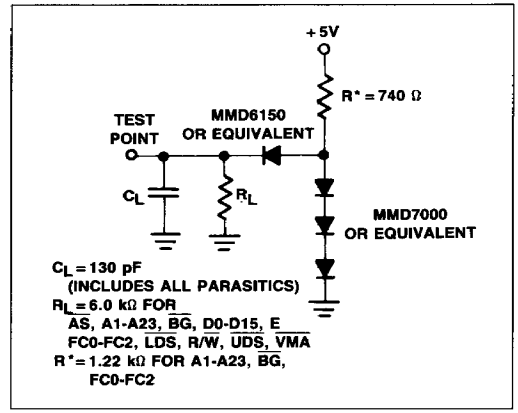


Figure 43. Test Loads

CLOCK TIMING (See Figure 44)

Characteristic	Symbol	4 MHz		6 MHz		8 MHz		10 MHz		12.5 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
Frequency of Operation	F	2.0	4.0	2.0	6.0	2.0	8.0	2.0	10.0	4.0	12.5	MHz
Cycle Time	t_{cyc}	250	500	167	500	125	500	100	500	80	250	ns
Clock Pulse Width	t_{CL} t_{CH}	115 115	250 250	75 75	250 250	55 55	250 250	45 45	250 250	35 35	125 125	ns
Rise and Fall Times	t_{Cr} t_{Cf}	— —	10 10	— —	10 10	— —	10 10	— —	10 10	— —	5 5	ns

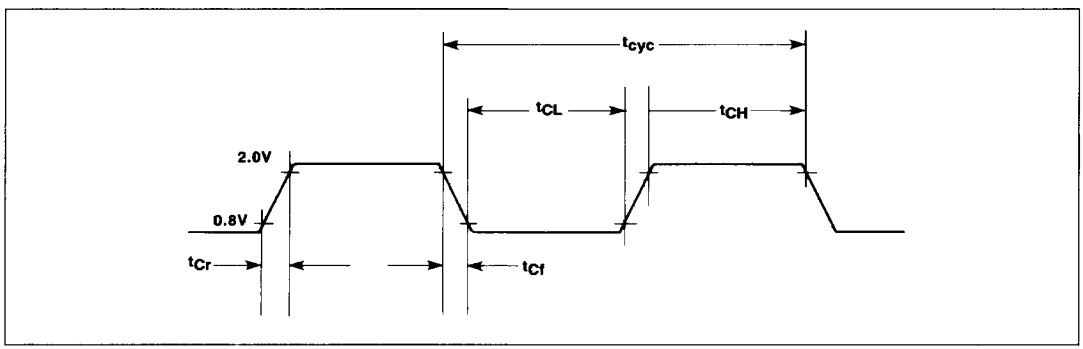


Figure 44. Input Clock Waveform

AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES

(V_{CC} = 5.0 Vdc ±5%, V_{SS} = 0 Vdc; $T_A = T_L$ to T_H, see Figures 45 and 46)

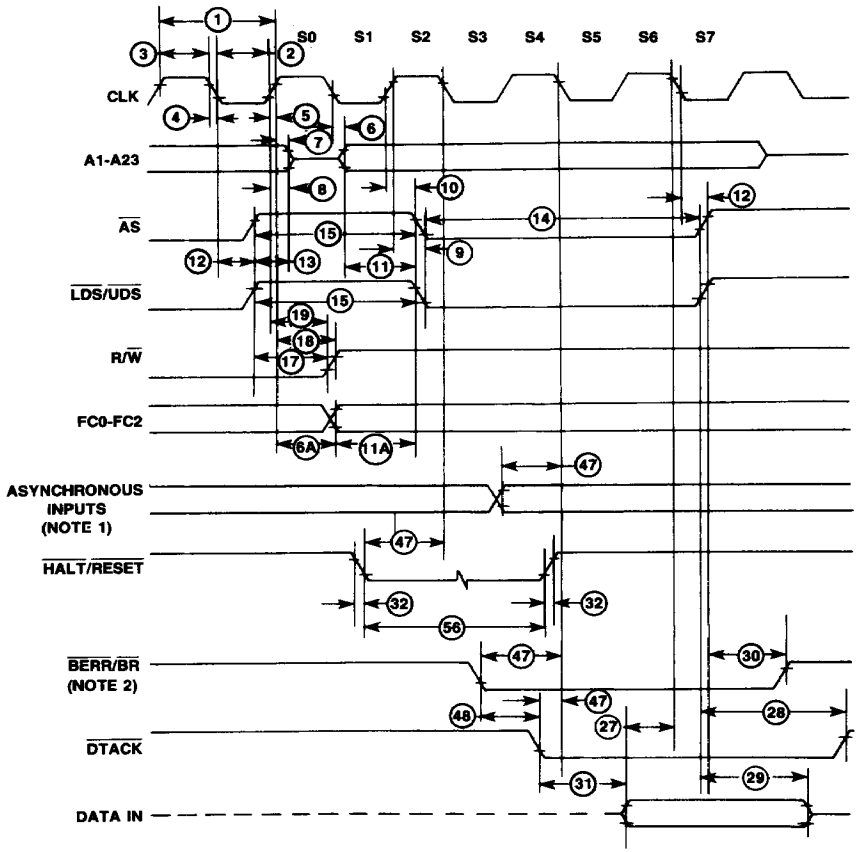
Num.	Characteristic	Symbol	4 MHz		6 MHz		8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
1	Clock Period	t_{cyc}	250	500	167	500	125	500	100	500	80	250	ns
2	Clock Width Low	t_{CL}	115	250	75	250	55	250	45	250	35	125	ns
3	Clock Width High	t_{CH}	115	250	75	250	55	250	45	250	35	125	ns
4	Clock Fall Time	t_{Cf}	—	10	—	10	—	10	—	10	—	5	ns
5	Clock Rise Time	t_{Cr}	—	10	—	10	—	10	—	10	—	5	ns
6	Clock Low to Address	t_{CLAV}	—	90	—	80	—	70	—	60	—	55	ns
6A	Clock High to FC Valid	t_{CHFCV}	—	90	—	80	—	70	—	60	—	55	ns
7	Clock High to Address Data High Impedance (Maximum)	t_{CHAZx}	—	120	—	100	—	80	—	70	—	60	ns
8	Clock High to Address/FC Invalid (Minimum)	t_{CHAZn}	0	—	0	—	0	—	0	—	0	—	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low (Maximum)	t_{CHSLx}	—	80	—	70	—	60	—	55	—	55	ns
10	Clock High to $\overline{AS}$, $\overline{DS}$ Low (Minimum)	t_{CHSLn}	0	—	0	—	0	—	0	—	0	—	ns
11[2]	Address to $\overline{AS}$, $\overline{DS}$ (Read) Low/$\overline{AS}$ Write	t_{AVSL}	55	—	35	—	30	—	20	—	0	—	ns
11A[2]	FC Valid to $\overline{AS}$, $\overline{DS}$ (Read) Low/$\overline{AS}$ Write	t_{FCVSL}	80	—	70	—	60	—	50	—	40	—	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}	—	90	—	80	—	70	—	55	—	50	ns
13[2]	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	t_{SHAZ}	60	—	40	—	30	—	20	—	10	—	ns
14[2]	$\overline{AS}$, $\overline{DS}$ Width Low (Read)/$\overline{AS}$ Write	t_{SL}	535	—	337	—	240	—	195	—	160	—	ns
14A[2]	$\overline{DS}$ Width Low (Write)	t_{DWPW}	285	—	170	—	115	—	95	—	80	—	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width High	t_{SH}	285	—	180	—	150	—	105	—	65	—	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$ High	t_{CHSZ}	—	120	—	100	—	80	—	70	—	60	ns
17[2]	$\overline{AS}$, $\overline{DS}$ High to R/$\overline{W}$ High	t_{SHRH}	60	—	50	—	40	—	20	—	10	—	ns
18[1]	Clock High to R/$\overline{W}$ High (Maximum)	t_{CHRHx}	—	90	—	80	—	70	—	60	—	60	ns
19	Clock High to R/$\overline{W}$ High (Minimum)	t_{CHRHn}	0	—	0	—	0	—	0	—	0	—	ns
20[1]	Clock High to R/$\overline{W}$ Low	t_{CHRL}	—	90	—	80	—	70	—	60	—	60	ns
20A[6]	$\overline{AS}$ Low to R/$\overline{W}$ Valid	t_{ASRV}	—	20	—	20	—	20	—	20	—	20	ns
21[2]	Address Valid to R/$\overline{W}$ Low	t_{AVRL}	45	—	25	—	20	—	0	—	0	—	ns
21A[2]	FC Valid to R/$\overline{W}$ Low	t_{FCVRL}	80	—	70	—	60	—	50	—	30	—	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Low (Write)	t_{RLSL}	200	—	140	—	80	—	50	—	30	—	ns
23	Clock Low to Data Out Valid	t_{CLDO}	—	90	—	80	—	70	—	55	—	55	ns
24	Clock High to R/$\overline{W}$, $\overline{VMA}$ High Impedance	t_{CHRZ}	—	120	—	100	—	80	—	70	—	60	ns
25[2]	$\overline{DS}$ High to Data Out Invalid	t_{SHDO}	60	—	40	—	30	—	20	—	15	—	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	t_{DOSL}	55	—	35	—	30	—	20	—	15	—	ns
27[5]	Data In to Clock Low (Setup Time)	t_{DICL}	30	—	25	—	15	—	10	—	10	—	ns
27A	Late $\overline{BERR}$ Low to Clock Low (Setup Time)	t_{BELCL}	45	—	45	—	45	—	45	—	45	—	ns
28[2]	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High	t_{SHDAH}	0	490	0	325	0	245	0	190	0	150	ns

AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES (CONTINUED)

Num.	Characteristic	Symbol	4 MHz		6 MHz		8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
29	$\overline{DS}$ High to Data Invalid (Hold Time)	t_{SHDI}	0	—	0	—	0	—	0	—	0	—	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	t_{SHBEH}	0	—	0	—	0	—	0	—	0	—	ns
31[2,5]	$\overline{DTACK}$ Low to Data In (Setup Time)	t_{DALDI}	—	180	—	120	—	90	—	65	—	50	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	$t_{RHr,f}$	0	200	0	200	0	200	0	200	0	200	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	—	90	—	80	—	70	—	60	—	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	—	90	—	80	—	70	—	60	—	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
37A	$\overline{BGACK}$ Low to $\overline{BR}$ High (to Prevent Rearbitration)	t_{BGKBR}	30	—	25	—	20	—	20	—	20	—	ns
38	$\overline{BG}$ Low to Bus High Impedance (with $\overline{AS}$ High)	t_{GLZ}	—	120	—	100	—	80	—	70	—	60	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}	—	90	—	80	—	70	—	70	—	70	ns
41	Clock Low to E Transition	t_{CLC}	—	100	—	85	—	70	—	55	—	45	ns
42	E Output Rise and Fall Time	$t_{Er,f}$	—	25	—	25	—	25	—	25	—	25	ns
43	$\overline{VMA}$ Low to E High	t_{VMLEH}	325	—	240	—	200	—	150	—	90	—	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	t_{SHVPH}	0	240	0	160	0	120	0	90	0	70	ns
45	E Low to Address/$\overline{VMA}$/FC Invalid	t_{ELAI}	55	—	35	—	30	—	10	—	10	—	ns
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.
47[5]	Asynchronous Input Setup Time	t_{ASI}	30	—	25	—	20	—	20	—	20	—	ns
48[3,5]	$\overline{BERR}$ Low to $\overline{DTACK}$ Low	t_{BELDAL}	30	—	25	—	20	—	20	—	20	—	ns
49	E Low to $\overline{AS}$, $\overline{DS}$ Invalid	t_{ELSI}	−80	—	−80	—	−80	—	−80	—	−80	—	ns
50	E Width High	t_{EH}	900	—	600	—	450	—	350	—	280	—	ns
51	E Width Low	t_{EL}	1400	—	900	—	700	—	550	—	440	—	ns
52	E Extended Rise Time	t_{CIEHX}	—	80	—	80	—	80	—	80	—	80	ns
53	Data Hold from Clock High	t_{CHDO}	0	—	0	—	0	—	0	—	0	—	ns
54	Data Hold from E Low (Write)	t_{ELDOZ}	60	—	40	—	30	—	20	—	15	—	ns
55	R/$\overline{W}$ to Data Bus Impedance Change	t_{RLDO}	55	—	35	—	30	—	20	—	10	—	ns
56[4]	$\overline{HALT}$/$\overline{RESET}$ Pulse Width	t_{HRPW}	10	—	10	—	10	—	10	—	10	—	Clk. Per.

Notes:

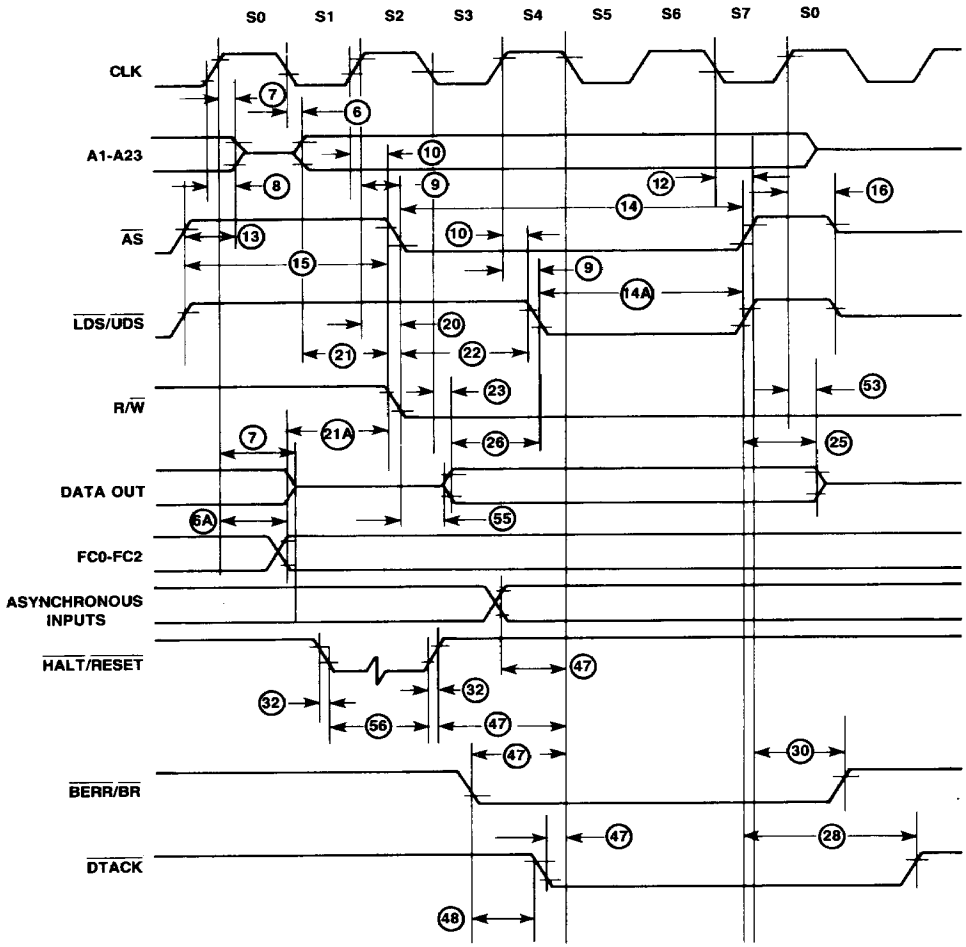
1. For a loading capacitance of less than or equal to 50 picofarads, subtract 5 nanoseconds from the value given in these columns.
2. Actual value depends on clock period.
3. If #47 is satisfied for both $\overline{DTACK}$ and $\overline{BERR}$, #48 may be 0 nanoseconds.
4. For power up, the MPU must be held in $\overline{RESET}$ state for 100 ms to stabilize all on-chip circuitry. After the system is powered up, #56 refers to the minimum pulse width required to reset the system.
5. If the asynchronous setup time (#47) requirements are satisfied the $\overline{DTACK}$ low-to-data setup time (#31) requirement can be ignored. The data must only satisfy the data-in clock-low setup time (#27) for the following cycle.
6. When $\overline{AS}$ and R/$\overline{W}$ are equally loaded (±20%), subtract 10 nanoseconds from the value given in these columns.



NOTES:

1. SETUP TIME FOR THE ASYNCHRONOUS INPUTS BGACK, IPL0-IPL2, AND VPA GUARANTEES THEIR RECOGNITION AT THE NEXT FALLING EDGE OF THE CLOCK.
2. BR NEEDS FALL AT THIS TIME ONLY IN ORDER TO INSURE BEING RECOGNIZED AT THE END OF THIS BUS CYCLE.
3. TIMING MEASUREMENTS ARE REFERENCED TO AND FROM A LOW VOLTAGE OF 0.8 VOLTS AND A HIGH VOLTAGE OF 2.0 VOLTS, UNLESS OTHERWISE NOTED.

Figure 45. Read Cycle Timing



NOTES:
1. BECAUSE OF LOADING VARIATIONS, R/$\overline{W}$ MAY BE VALID AFTER $\overline{AS}$ EVEN THOUGH BOTH ARE INITIATED BY THE RISING EDGE OF S2 (SPECIFICATION 20A).
2. TIMING MEASUREMENTS ARE REFERENCED TO AND FROM A LOW VOLTAGE OF 0.8 VOLTS AND A HIGH VOLTAGE OF 2.0 VOLTS, UNLESS OTHERWISE NOTED.

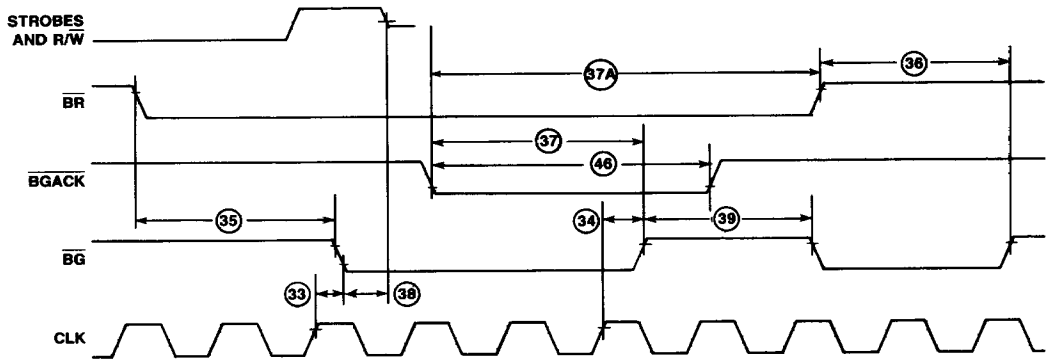
Figure 46. Write Cycle Timing

AC ELECTRICAL SPECIFICATIONS — BUS ARBITRATION

(V_{CC} = 5.0 Vdc ±5%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H °C. See Figure 47.)

Num.	Characteristic	Symbol	4 MHz		6 MHz		8 MHz		10 MHz		12.5 MHz		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	—	90	—	80	—	70	—	60	—	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	—	90	—	80	—	70	—	60	—	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
37A	$\overline{BGACK}$ Low to $\overline{BR}$ High (to Prevent Rearbitration)	t_{BGKBR}	30	—	25	—	20	—	20	—	20	—	ns
38	$\overline{BG}$ Low to Bus High Impedance (with $\overline{AS}$ High)	t_{GLZ}	—	120	—	100	—	80	—	70	—	60	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.

THESE WAVEFORMS SHOULD ONLY BE REFERENCED IN REGARD TO THE EDGE-TO-EDGE MEASUREMENT OF THE TIMING SPECIFICATIONS. THEY ARE NOT INTENDED AS A FUNCTIONAL DESCRIPTION OF THE INPUT AND OUTPUT SIGNALS. REFER TO OTHER FUNCTIONAL DESCRIPTIONS AND THEIR RELATED DIAGRAMS FOR DEVICE OPERATION.

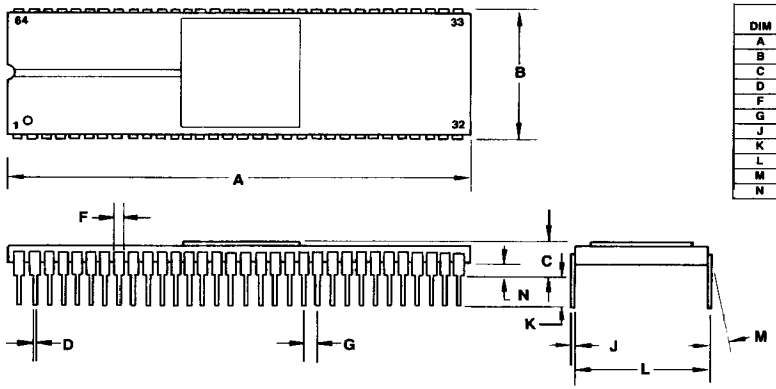


NOTES:
1. SETUP TIME FOR THE ASYNCHRONOUS INPUTS $\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{IPL0}$-$\overline{IPL2}$, AND $\overline{VPA}$ GUARANTEES THEIR RECOGNITION AT THE NEXT FALLING EDGE OF THE CLOCK.
2. WAVEFORM MEASUREMENTS FOR ALL INPUTS AND OUTPUTS ARE SPECIFIED AT: LOGIC HIGH = 2.0 VOLTS, LOGIC LOW = 0.8 VOLTS

Figure 47. AC ELECTRICAL Waveforms — Bus Arbitration

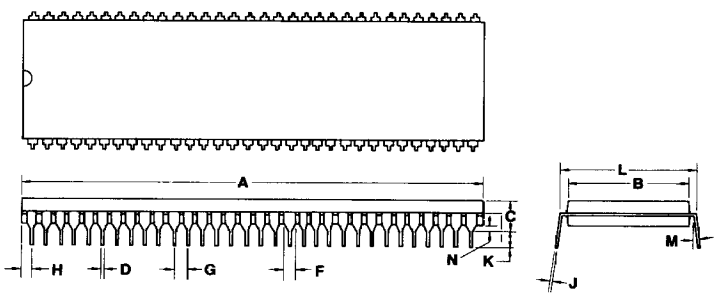
PACKAGE DIMENSIONS

64-PIN CERAMIC DUAL IN-LINE PACKAGE (DIP)



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	80.52	82.04	3.170	3.230
B	22.25	22.96	0.876	0.904
C	3.05	4.32	0.120	0.170
D	0.38	0.53	0.015	0.021
F	0.76	1.40	0.030	0.055
G	2.54 BSC		0.100 BSC	
J	0.20	0.33	0.008	0.013
K	2.54	4.19	0.100	0.165
L	22.61	23.11	0.890	0.910
M	—	10°	—	10°
N	1.02	1.52	0.040	0.060

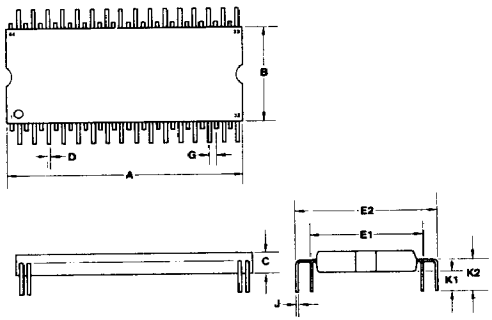
64-PIN PLASTIC DUAL IN-LINE PACKAGE (DIP)



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	81.15	81.91	3.195	3.225
B	20.07	20.57	0.790	0.810
C	4.32	4.82	0.170	0.190
D	0.38	0.58	0.015	0.023
F	1.42	1.62	0.056	0.064
G	2.54 BSC		0.100 BSC	
H	1.15	1.65	0.045	0.065
J	0.20	0.30	0.008	0.012
K	3.05	3.55	0.120	0.140
L	22.71	23.01	0.894	0.906
M	0	10	0	10
N	0.51	1.01	0.020	0.040

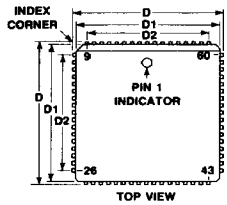
64-PIN PLASTIC QUAD IN-LINE PACKAGE (QUIP)

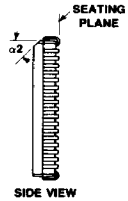
DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	41.10	41.61	1.618	1.638
B	17.02	17.23	0.670	0.690
C	3.56	4.58	0.140	0.180
D	0.48	0.56	0.018	0.022
E1	19.05 BSC		0.750 BSC	
E2	23.50 BSC		0.925 BSC	
G	1.27 BSC		0.050 BSC	
J	0.18	0.33	0.007	0.013
K1	2.92	3.18	0.115	0.125
K2	4.83	5.34	0.190	0.210

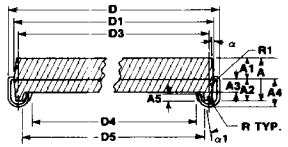


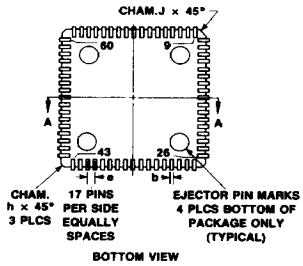
4

68-PIN PLASTIC CHIP CARRIER (PCC)



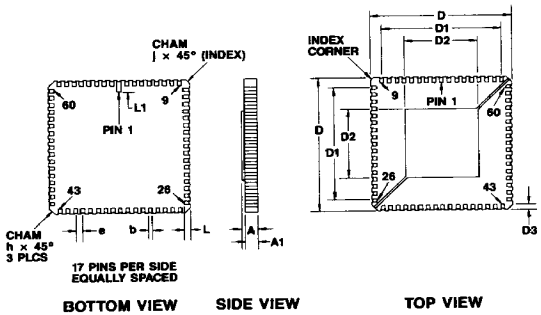






DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	3.683	3.785	.145	.149
A1	1.829	1.930	.072	.076
A2	1.803	1.905	.071	.075
A3	1.372	1.473	.054	.058
A4	2.311	2.464	.091	.097
A5	0.203	0.305	.008	.012
b	0.457 TYP		.018 TYP	
D	25.02	25.27	.985	.995
D1	24.00	24.26	.945	.955
D2	20.19	20.45	.795	.805
D3	23.24	23.50	.915	.925
D4	20.96	21.21	.825	.835
D5	22.23	22.48	.875	.885
e	1.27 BSC		.050 BSC	
h	0.254 TYP		.010 TYP	
J	1.143 TYP		.045 TYP	
α	4° TYP		4° TYP	
α1	10° TYP		10° TYP	
α2	45° TYP		45° TYP	
R	0.889 TYP		.035 TYP	
R1	0.254 TYP		.010 TYP	

68-PIN CERAMIC CHIP CARRIER (LCC)



DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	1.65	1.91	0.065	0.075
A1	1.40	1.65	0.055	0.065
b	0.51	0.76	0.020	0.030
D	23.83	24.43	0.938	0.962
D1	20.12	20.52	0.792	0.808
D2	12.45	12.70	0.490	0.500
D3	1.27 REF		0.050 REF	
e	1.27 BSC		0.050 BSC	
h	1.02 REF		0.040 REF	
J	0.51 REF		0.020 REF	
L	1.27 REF		0.050 REF	
L1	1.96	2.36	0.077	0.093

4