MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

MC68881

Technical Summary

HCMOS Floating-Point Coprocessor

The MC68881 floating-point coprocessor fully implements the *IEEE Standard for Binary Floating-Point Arithmetic* (754) for use with the Motorola M68000 Family of microprocessors. It is implemented using VLSI technology to give system designers the highest possible functionality in a physically small device.

Intended primarily for use as a coprocessor to the MC68020 32-bit microprocessor unit (MPU), the MC68881 provides a logical extension to the main MPU integer data processing capabilities. This extension is provided by a high-performance floating-point arithmetic unit and a set of floating-point data registers that are utilized in a manner analogous to the use of the integer data registers. The MC68881 instruction set is a natural extension of all earlier members of the M68000 Family, and it supports all addressing modes of the host MPU. Due to the flexible bus interface of the M68000 Family, the MC68881 can be used with any of the M68000 MPU devices and peripheral to non-M68000 processors.

5

This document contains information on a new product. Specifications and information herein are subject to change without notice.

The major features of the MC68881 are as follows:

- Eight general-purpose floating-point data registers, each supporting a full 80-bit extended-precision real data format (a 64-bit mantissa plus a sign bit and a 15-bit signed exponent)
- A 67-bit arithmetic unit to allow very fast calculations, with intermediate precision greater than the extended-precision format
- A 67-bit barrel shifter for high-speed shifting operations (for normalizing, etc.)
- 46 instructions, including 35 arithmetic operations
- Full conformation to the IEEE 754 standard, including all requirements and suggestions
- Support of functions not defined by the IEEE 754 standard, including a full set of trigonometric and transcendental functions
- Seven data types: byte, word, and long-word integers; single-, double-, and extended-precision real numbers; and packed binary-coded decimal (BCD) string real numbers
- 22 constants available in the on-chip ROM, including π, e, and powers of 10
- Virtual memory/machine operations
- Efficient mechanisms for procedure calls, context switches, and interrupt handling
- Fully concurrent instruction execution with the main processor
- Use with any host processor on an 8-, 16-, or 32-bit data bus

THE COPROCESSOR CONCEPT

The MC68881 functions as a coprocessor in systems where the MC68020 or MC68030 is the main processor.* It functions as a peripheral processor in systems where the main processor is the MC68000, MC68008, or MC68010.

The MC68881 utilizes the M68000 Family coprocessor interface to provide a logical extension of the MC68020 registers and instruction set in a manner transparent to the programmer. The programmer perceives the MC68020/MC68881 execution model as if both devices are implemented on one chip.

A fundamental goal of the M68000 Family coprocessor interface is to provide the programmer with an execution model based upon sequential instruction execution by the MC68020 and the MC68881. For optimum performance, however, the coprocessor interface allows concurrent operations in the MC68881 with respect to the MC68020 whenever possible. To simplify the programmer's model, the coprocessor interface is designed to emulate, as closely as possible, nonconcurrent operation between the MC68020 and the MC68882.

The MC68881 is a non-DMA type coprocessor using a subset of the general-purpose coprocessor interface supported by the MC68020. Features of the interface implemented in the MC68881 are as follows:

- The main processor(s) and MC68881 communicate via standard M68000 bus cycles.
- The main processor(s) and MC68881 communications are not dependent upon the instruction sets or internal details of the individual devices (e.g., instruction pipes or caches, addressing modes).
- The main processor(s) and MC68881 may operate at different clock speeds.
- MC68881 instructions utilize all addressing modes provided by the main processor.
- All effective addresses are calculated by the main processor at the request of the coprocessor.
- All data transfers are performed by the main processor at the request of the MC68881; thus, memory management, bus errors, address errors, and bus arbitration function as if the MC68881 instructions were executed by the main processor.
- Overlapped (concurrent) instruction execution enhances throughput while maintaining the programmer's model of sequential instruction execution.
- Coprocessor detection of exceptions requiring a trap to be taken are serviced by the main processor at the request of the MC68881; thus, exception processing functions as if the MC68881 instructions were executed by the main processor.
- Support of virtual memory/machine systems is provided via the FSAVE and FRESTORE instructions.
- Up to eight coprocessors may reside in a system simultaneously; multiple coprocessors of the same type are also allowed.
- Systems may use software emulation of the MC68881 without reassembling or relinking user software.

*Throughout this technical summary, all references to the MC68020 also apply to the MC68030.

HARDWARE OVERVIEW

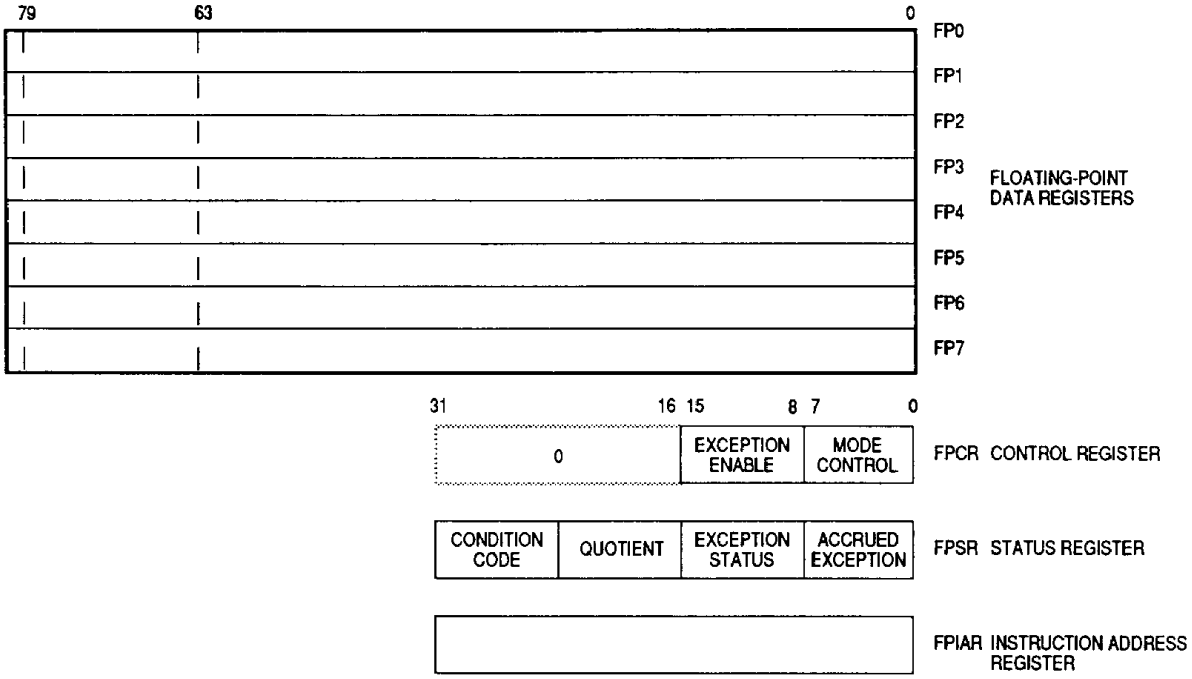
The MC68881 is a high-performance floating-point device designed to interface with the MC68020 as a coprocessor. This device fully supports the MC68020 virtual machine architecture and is implemented in HCMOS, Motorola's low-power small-geometry process. This process allows CMOS and HMOS (high-density NMOS) gates to be combined on the same device. CMOS structures are used where speed and low power are required, and HMOS structures are used where minimum silicon area is desired. Thus, HCMOS technology enables the MC68881 to be very fast, consume less power, and still have a reasonably small die size.

With some performance degradation, the MC68881 can also be used as a peripheral processor in systems where the MC68020 is not the main processor (e.g., MC68000, MC68008, MC68010). The configuration of the MC68881 as a peripheral processor or coprocessor may be completely transparent to user software (i.e., the same object code may be executed in either configuration).

The architecture of the MC68881 appears to the user as a logical extension of the M68000 Family architecture. Coupling of the coprocessor interface allows the MC68020 programmer to view the MC68881 registers as though the registers were resident in the MC68020. Thus, a MC68020/MC68881 functions as one processor that has eight integer data registers, eight address registers, and eight floating-point data registers and supports seven floating-point and integer data types.

The MC68881 programming model, shown in Figures 1–6, consists of the following features:

- Eight 80-bit floating-point data registers (FP0–FP7). These registers are analogous to the integer data registers (D0–D7) and are completely general purpose (i.e., any instruction may use any register).
- A 32-bit control register contains enable bits for each class of exception trap and mode bits for setting the user-selectable rounding and precision modes.
- A 32-bit status register contains floating-point condition codes, quotient bits, and exception status information.
- A 32-bit instruction address register contains the main processor memory address of the last floating-point instruction that was executed. This address is used in exception handling to locate the instruction that caused the exception.



5

Figure 1. Programming Model

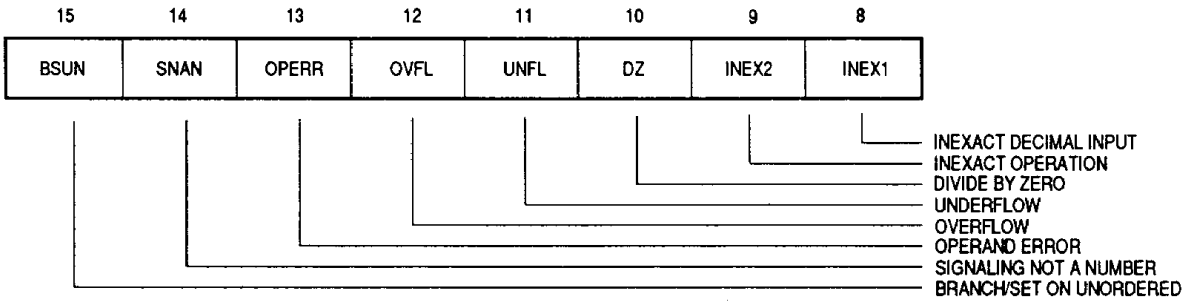


Figure 2. Exception Status/Enable Byte

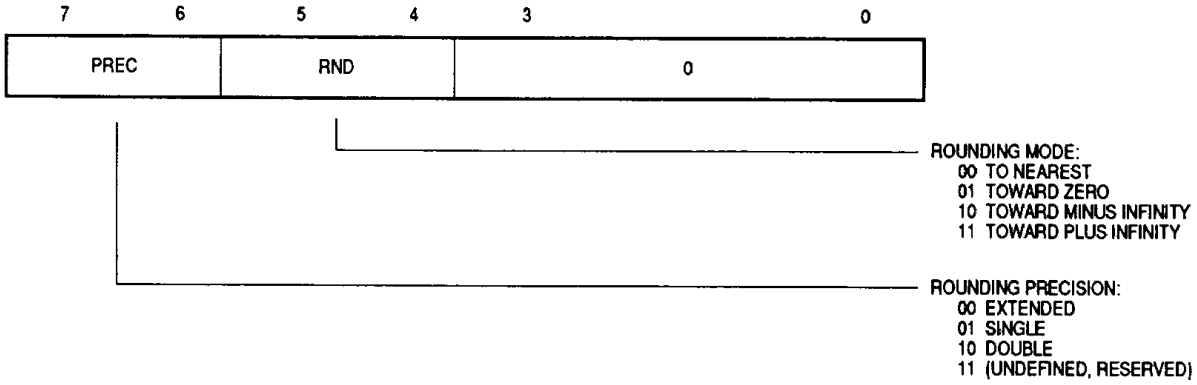
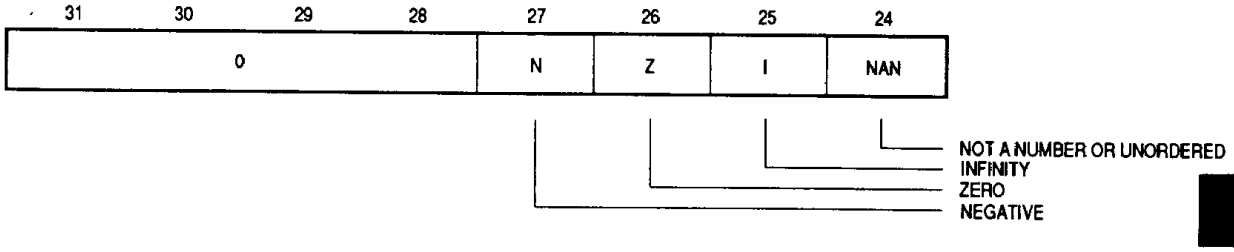


Figure 3. Mode Control Byte



5

Figure 4. Condition Code Byte

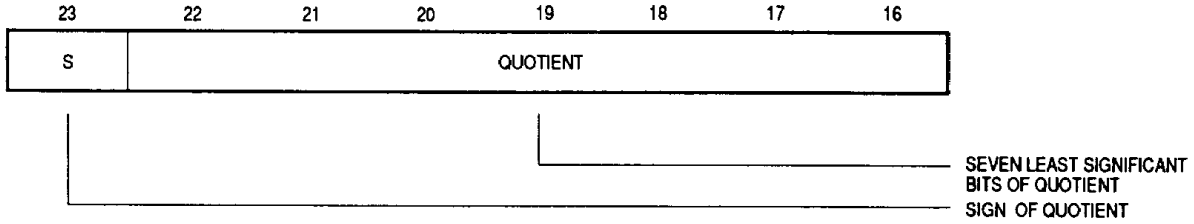


Figure 5. Quotient Byte

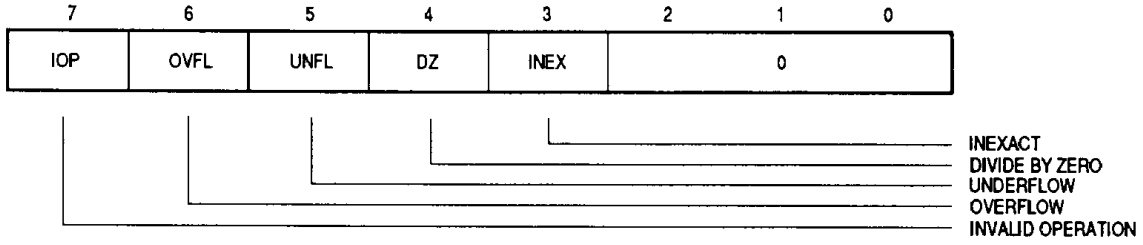


Figure 6. Accrued Exception Byte

The connection between the MC68020 and the MC68881 is a simple extension of the M68000 bus interface. The MC68881 is connected as a coprocessor to the MC68020, and the selection of the MC68881 is based upon a chip select ($\overline{CS}$), which is decoded from the MC68020 function codes and address bus. Figure 7 illustrates the MC68881/MC68020 configuration.

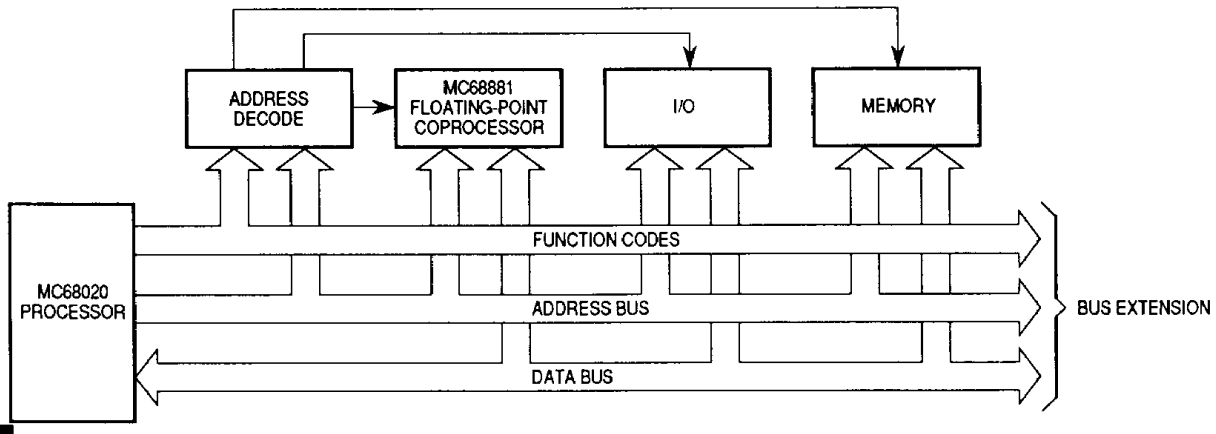


Figure 7. Typical Coprocessor Configuration

As shown in Figure 8, the MC68881 is internally divided into three processing elements; the bus interface unit (BIU), the execution control unit (ECU), and the microcode control unit (MCU). The BIU communicates with the MC68020, and the ECU and MCU execute all MC68881 instructions.

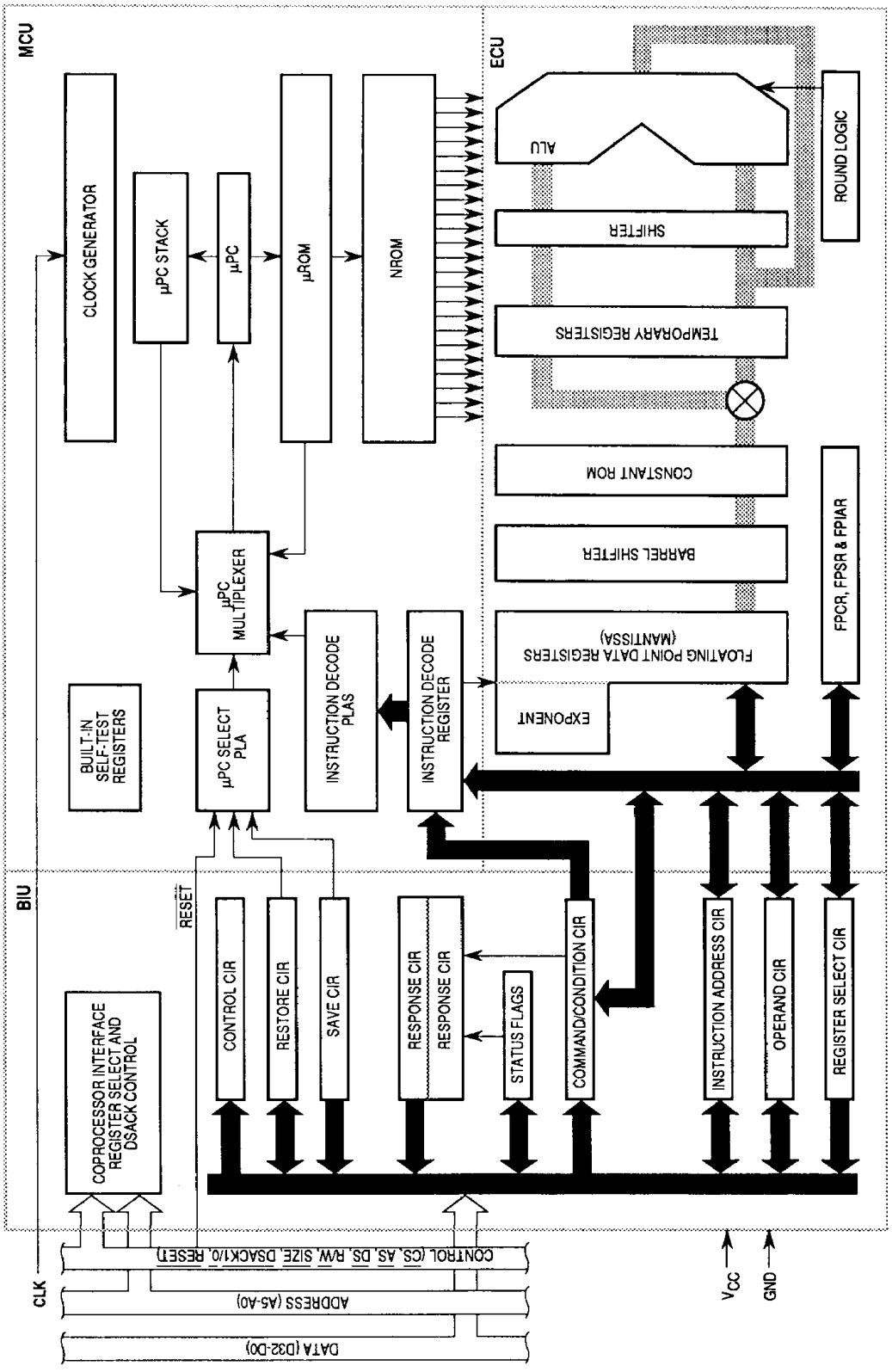


Figure 8. MC68881 Simplified Block Diagram

5

The BIU contains the coprocessor interface registers (CIRs), and the 32-bit control, status, and instruction address registers. In addition, the register select and DSACK timing control logic and the status flags used to monitor the status of communications with the main processor are contained in the BIU.

The eight, 80-bit, floating-point data registers (FP0–FP7) are located in the ECU. In addition, the ECU contains a high-speed 67-bit arithmetic unit used for both mantissa and exponent calculations, a barrel shifter that can shift from 1–67 bits in one machine cycle, and ROM constants (for use by the internal algorithms or user programs).

The MCU contains the clock generator, a two-level microcode sequencer that controls the ECU, the microcode ROM, and self-test circuitry. The built-in self-test capabilities of the MC68881 enhance reliability and ease manufacturing requirements; however, these diagnostic functions are not available to the user.

BUS INTERFACE UNIT

All communications between the MC68020 and the MC68881 occur via standard M68000 Family bus transfers. The MC68881 is designed to operate on 8-, 16-, or 32-bit data buses.

The MC68881 contains a number of CIRs that are addressed in the same manner as memory by the main processor. The M68000 Family coprocessor interface is implemented via a protocol of reading and writing to these registers by the main processor. The MC68020 implements this general-purpose coprocessor interface protocol in hardware and microcode.

When the MC68020 detects a general-type MC68881 instruction, the MC68020 writes the instruction to the memory-mapped command CIR and reads the response CIR. In this response, the BIU encodes requests for any additional action required of the MC68020 on behalf of the MC68881. For example, the response may request that the MC68020 fetch an operand from the evaluated effective address and transfer the operand to the operand CIR. Once the MC68020 fulfills the coprocessor request(s), the MC68020 is free to fetch and execute subsequent instructions.

A key concern in a coprocessor interface that allows concurrent instruction execution is synchronization during main processor and coprocessor communication. If a subsequent instruction is written to the MC68881 before the ECU has completed execution of the previous instruction, the response instructs the MC68020 to wait. Thus, the choice of concurrent or nonconcurrent instruction execution is determined on an instruction-by-instruction basis by the coprocessor.

The only difference between a coprocessor bus transfer and any other bus transfer is that the MC68020 issues a CPU address space function code during the cycle (function codes are generated by the M68000 Family processors to identify eight separate address spaces). Thus, the memory-mapped CIRs do not infringe upon instruction or data address spaces. The MC68020 places a coprocessor ID field from the coprocessor instruction onto three of the upper address lines during coprocessor accesses. This ID and the CPU address space function code are decoded to select one of eight coprocessors in the system.

Since the coprocessor interface protocol is based solely on bus transfers, the protocol is easily emulated by software when the MC68881 is used as a peripheral with any processor capable of memory-mapped I/O over an M68000-style bus. When used as a peripheral processor with the 8-bit MC68008, the 16-bit MC68000, or the MC68010, all MC68881 instructions are trapped by the main processor to an exception handler at execution time. Thus, the software emulation of the coprocessor interface protocol can be totally transparent to the user. The system can be quickly upgraded by replacing the main processor with an MC68020 without changes to the user software.

Since the bus is asynchronous, the MC68881 need not run at the same clock speed as the main processor; therefore, total system performance can be customized. For example, a system requiring very fast floating-point arithmetic with relatively slow integer arithmetic can be designed with an inexpensive main processor and a fast MC68881.

COPROCESSOR INTERFACE

The M68000 Family coprocessor interface is an integral part of the MC68881 and MC68020 designs, with the interface tasks shared between the two. The interface is fully compatible with all present and future M68000 Family products. Tasks are partitioned so that the MC68020 does not have to decode coprocessor instructions and the MC68881 does not have to duplicate main processor functions, such as effective address evaluation.

This partitioning provides an orthogonal extension of the instruction set by permitting MC68881 instructions to utilize all MC68020 addressing modes and to generate execution time exception traps. Thus, from the programmer's view, the CPU and coprocessor appear to be integrated onto a single chip. While the execution of most MC68881 instructions may be overlapped with the execution of MC68020 instructions, concurrency is completely transparent to the programmer. The MC68020 single-step and program flow (trace) modes are fully supported by the MC68881 and the M68000 Family coprocessor interface.

The M68000 Family coprocessor interface permits coprocessors to be bus masters; however, the MC68881 is never a bus master. The MC68881 requests that the MC68020 fetch all operands and store all results. In this manner, the MC68020 32-bit data bus provides high-speed transfer of floating-point operands and results while simplifying the design of the MC68881.

Since the coprocessor interface is based solely upon bus cycles and the MC68881 is never a bus master, the MC68881 can be placed on either the logical or physical side of the system memory management unit. This provides a great deal of flexibility in the system design.

The virtual machine architecture of the MC68020 is supported by the coprocessor interface and the MC68881 through the FSAVE and FRESTORE instructions. If the MC68020 detects a page fault and/or a task timeout, the MC68020 can force the MC68881 to stop whatever operation is in progress at any time and save the MC68881 internal state in memory.

The size of the saved internal state of the MC68881 is dependent upon the ECU state at the time FSAVE is executed. If the MC68881 is in the reset state when FSAVE is received, only one word of state is transferred to memory, which can be examined by the operating system to determine that the coprocessor programmer's model is empty. If the coprocessor is idle when FSAVE is received, only a few words of internal state are transferred to memory. If the MC68881 is in the middle of executing an instruction, it may be necessary to save the entire internal state of the machine. Instructions that can complete execution in less time than it would take to save the larger state in mid-instruction are allowed to complete execution and then save the idle state. Thus, the size of the saved internal state is kept to a minimum. The ability to utilize several internal state sizes greatly reduces the average context switching time.

The FRESTORE instruction permits reloading of an internal state that was saved earlier and continues any operation that was previously suspended. Restoring of the reset internal state functions just like a hardware reset to the MC68881 in that defaults are re-established.

OPERAND DATA FORMATS

The MC68881 supports the following data formats:

Byte Integer (B)
Word Integer (W)
Long-Word Integer (L)
Single-Precision Real (S)
Double-Precision Real (D)
Extended-Precision Real (X)
Packed BCD String Real (P)

The capital letters contained in parentheses denote suffixes added to instructions in the assembly language source that specify the data format to be used.

INTEGER DATA FORMATS

The three integer data formats (byte, word, and long word) are the standard twos-complement data formats supported in the M68000 Family architecture. Whenever an integer is used in a floating-point operation, the integer is automatically converted by the MC68881 to an extended-precision floating-point number before being used. For example, to add an integer constant of five to the number contained in floating-point data register 3 (FP3), the following instruction can be used:


```
FADD.W #5,FP3
```

(The Motorola assembler syntax "#" is used to denote immediate addressing.)

The ability to effectively use integers in floating-point operations saves user memory since an integer representation of a number, if representable, is usually smaller than the equivalent floating-point representation.

FLOATING-POINT DATA FORMATS

The floating-point data formats, single precision (32 bits) and double precision (64 bits), are defined by the IEEE 754 standard. These main floating-point formats should be used for most calculations involving real numbers. Table 1 lists the exponent and mantissa size for single, double, and extended precision. The exponent is biased, and the mantissa is in sign and magnitude form. Since single and double precision require normalized numbers, the most significant bit of the mantissa is implied as a one and is not included, thus giving one extra bit of precision.

Table 1. Exponent and Mantissa Sizes

Data Format	Exponent Bits	Mantissa Bits	Bias
Single	8	23(+1)	127
Double	11	52(+1)	1023
Extended	15	64	16383

The extended-precision data format is also in conformance with the IEEE 754 standard, but the standard does not specify this format to the bit level as it does for single and double precision. The memory format on the MC68881 consists of 96 bits (three long words). Only 80 bits are actually used, the other 16 bits are for future expandability and for long-word alignment of floating-point data structures. Extended format has a 15-bit exponent, a 64-bit mantissa, and a 1-bit mantissa sign.

Extended-precision numbers are intended for use as temporary variables, intermediate values, or in places where extra precision is needed. For example, a compiler might select extended-precision arithmetic for evaluation of the right side of an equation with mixed-sized data and then convert the answer to the data type on the left side of the equation. Extended-precision data will not be stored in large arrays due to the amount of memory required by each number.

PACKED BCD STRING DATA FORMAT

The packed decimal data format allows packed BCD strings to be transferred to and from the MC68881. The strings consist of a 3-digit base 10 exponent and a 17-digit base 10 mantissa. Both the exponent and mantissa have a separate sign bit. All digits are packed BCD so an entire string fits in 96 bits (three long words). Like all data formats, when packed BCD strings are input to the MC68881, the strings are automatically converted to extended-precision real values, allowing packed BCD numbers to be used as inputs to any operation. For example,

```
FADD.P   #-6.023E+24,FP5
```

BCD numbers can be output from the MC68881 in a format readily used for printing by a program generated by a high-level language compiler. For example,

```
FMOVE.P  FP3,BUFFER{#-5}
```

instructs the MC68881 to convert the floating-point data register 3 (FP3) contents into a packed BCD string with five digits to the right of the decimal point (FORTRAN F format).

DATA FORMAT SUMMARY

All data formats previously described are supported orthogonally by all arithmetic and transcendental operations and by all appropriate MC68020 addressing modes. For example, the following instructions are legal:

```
FADD.B   #3,FP0
FADD.W   D2,FP3
FADD.L   BIGINT,FP7
FADD.S   #3.14159,FP5
FADD.D   (SP)+,FP6
FADD.X   [(TEMP_PTR,A7)],FP3
FADD.P   #1.23E25,FP0
```

On-chip calculations are performed to extended-precision format, and the eight floating-point data registers always contain extended-precision values. All data used in an operation is converted to extended precision by the MC68881 before the specific operation is performed, and all results are in extended precision. Use of extended precision ensures maximum accuracy without sacrificing performance.

Refer to Figure 9 for a summary of the memory formats for the seven data formats supported by the MC68881.

INSTRUCTION SET

The MC68881 instruction set is organized into six major classes:

1. Moves between the MC68881 and memory or the MC68020 (in and out)
2. Move multiple registers (in and out)
3. Monadic operations
4. Dyadic operations
5. Branch, set, or trap conditionally
6. Miscellaneous

MOVES

All moves from memory (or from an MC68020 data register) to the MC68881 cause data conversion from the source data format to the internal extended-precision format. All moves from the MC68881 to memory (or to an MC68020 data register) cause data conversion from the internal extended-precision format to the destination data format. Note that data movement instructions perform arithmetic operations, since the result is always rounded to the precision selected in the floating-point control register mode control byte. The result is rounded using the selected rounding mode and is checked for overflow and underflow.

The syntax for the move is as follows:

```
FMOVE.<fmt> <ea>,FPn    Move to MC68881
FMOVE.<fmt> FPm,<ea>    Move from MC68881
FMOVE.X     FPm,FPn     Move within MC68881
```

where <ea> is an MC68020 effective address operand, .<fmt> is the data format size, and FPm and FPn are floating-point data registers.

5

MOVE MULTIPLE REGISTERS

The floating-point move multiple instructions on the MC68881 are much like the integer counterparts on the M68000 Family processors. Any set of the floating-point registers FP0–FP7 can be moved to or from memory with one instruction. These registers are always moved as 96-bit extended data with no conversion (no possibility of conversion errors). Some move multiple instruction examples are as follows:

```
FMOVEM  <ea>,FP0–FP3/FP7
FMOVEM  FP2/FP4/FP6,<ea>
```

Move multiple instructions are useful during context switches and interrupts to save or restore the state of a program. These moves are also useful at the start and end of a procedure to save and restore the calling routine's register set. To reduce procedure call overhead, the list of registers to be saved or restored can be contained in a data register, enabling run-time optimization by allowing a called routine to save as few registers as possible. Note that no rounding or overflow/underflow checking is performed by these operations.

MONADIC OPERATIONS

Monadic operations have one operand. This operand may be in a floating-point data register, memory, or in an MC68020 data register. The result is always stored in a floating-point data register. For example, the syntax for square root is one of the following:

```
FSQRT.<fmt>   <ea>,FPn
FSQRT.X       FPm,FPn
FSQRT.X       FPn
```

The available MC68881 monadic operations are as follows:

FABS	Absolute Value
FACOS	Arc Cosine
FASIN	Arc Sine
FATAN	Arc Tangent
FATANH	Hyperbolic Arc Tangent
FCOS	Cosine
FCOSH	Hyperbolic Cosine
FETOX	e to the x Power
FETOXM1	e to the x Power −1
FGETEXP	Get Exponent
FGETMAN	Get Mantissa
FINT	Integer Part
FINTRZ	Integer Part (Truncated)
FLOG10	Log Base 10
FLOG2	Log Base 2
FLOGN	Log Base e
FLOGNP1	Log Base e of (x+1)
FNEG	Negate
FSIN	Sine
FSINCOS	Simultaneous Sine and Cosine
FSINH	Hyperbolic Sine
FSQRT	Square Root
FTAN	Tangent
FTANH	Hyperbolic Tangent
FTENTOX	10 to the x Power
FTST	Test
FTWOTOX	2 to the x Power

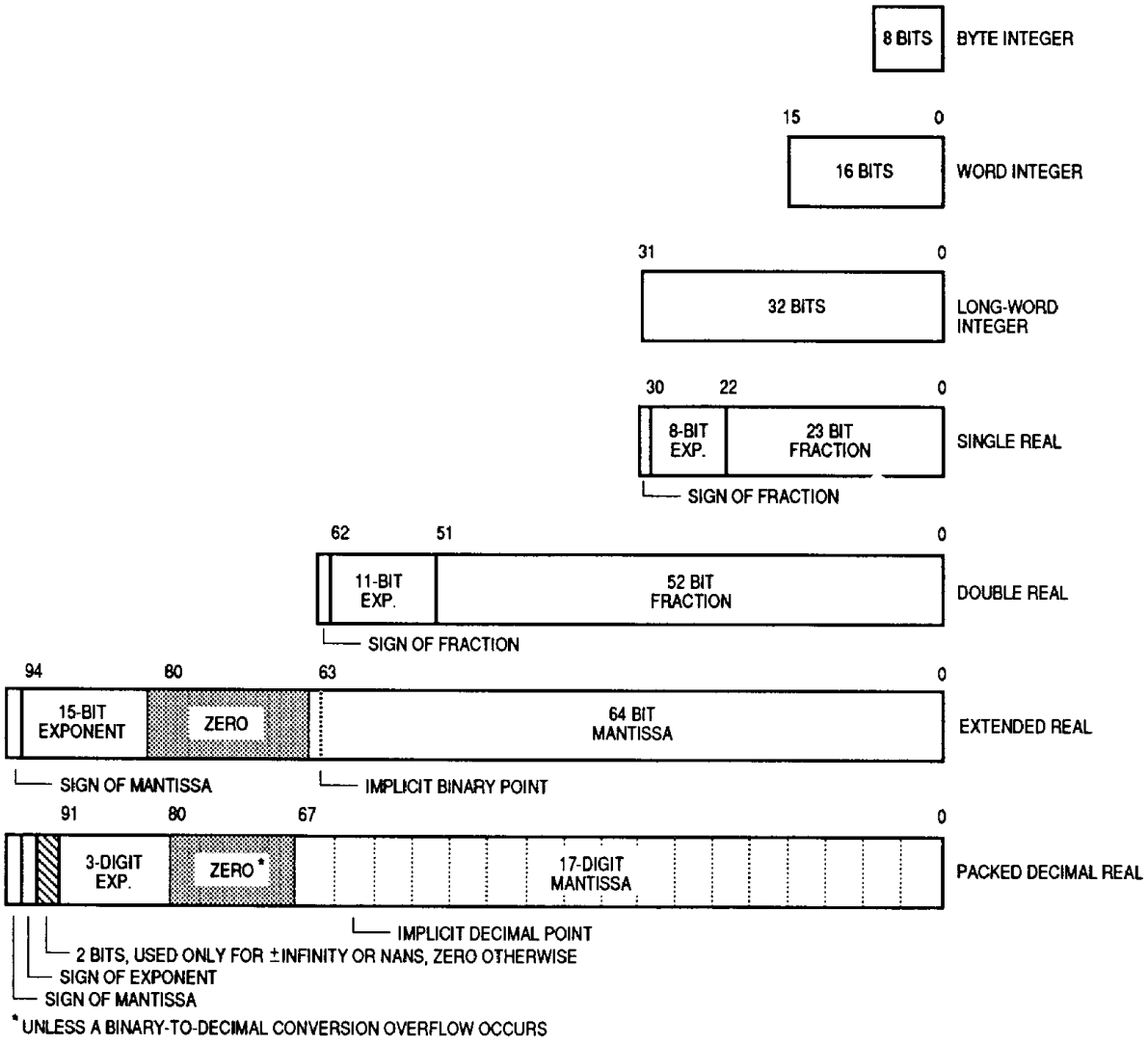


Figure 9. Data Format Summary

DYADIC OPERATIONS

Dyadic operations have two operands each. The first input operand comes from a floating-point data register, memory, or an MC68020 data register. The second input operand comes from a floating-point data register. The destination is the same floating-point data register used for the second input. For example, the syntax for floating-point add is one of the following:

```
FADD.<fmt>   <ea>,FPn
FADD.X       FPm,FPn
```

The available MC68881 dyadic operations are as follows:

FADD	Add
FCMP	Compare
FDIV	Divide
FMOD	Modulo Remainder
FMUL	Multiply
FREM	IEEE Remainder
FSCALE	Scale Exponent
FSGLDIV	Single-Precision Divide
FSGLMUL	Single-Precision Multiply
FSUB	Subtract

BRANCH, SET, AND TRAP ON CONDITION

The floating-point branch, set, and trap on condition instructions implemented by the MC68881 are similar to the equivalent integer instructions of the M68000 Family processors, except that more conditions exist due to the special values in IEEE floating-point arithmetic. When a conditional instruction is executed, the MC68881 performs the necessary condition checking and tells the MC68020 whether the condition is true or false; the MC68020 then takes the appropriate action. Since the MC68881 and MC68020 are closely coupled, the floating-point branch operations executed by the pair are very fast.

The MC68881 conditional operations are as follows:

FBcc	Branch
FDBcc	Decrement and Branch
FScc	Set Byte According to Condition
FTRAPcc	Trap on Condition (with an Optional Parameter)

where cc is one of the 32 floating-point conditional test specifiers listed in Table 2.

MISCELLANEOUS INSTRUCTIONS

Miscellaneous instructions include moves to and from the status, control, and instruction address registers. Also included are the virtual memory/machine FSAVE and FRESTORE instructions that save and restore the internal state of the MC68881:

FMOVE	<ea>,FPcr	Move to Control Register(s)
FMOVE	FPcr,<ea>	Move from Control Register(s)
FSAVE	<ea>	Virtual Machine State Save
FRESTORE	<ea>	Virtual Machine State Restore

ADDRESSING MODES

The MC68881 does not perform address calculations. This satisfies the criterion that an M68000 Family coprocessor must not depend on certain features or capabilities that may or may not be implemented by a given main processor. Thus, when the MC68881 instructs the MC68020 to transfer an operand via the coprocessor interface, the MC68020 performs the addressing mode calculations requested in the instruction. In this case, the instruction is encoded specifically for the MC68020, and the execution of the MC68881 is not dependent on that encoding but only on the value of the command word written to the MC68881 by the main processor.

This interface is flexible and allows any addressing mode to be used with floating-point instructions. For the M68000 Family, these addressing modes include immediate, postincrement, predecrement, data or address register direct, and the indexed/indirect addressing modes of the MC68020. Some addressing modes are restricted for instructions in compliance with the M68000 Family architectural definitions (e.g., program counter relative addressing is not allowed for a destination operand).

The orthogonal instruction set of the MC68881, along with the flexible branches and addressing modes, allows a programmer writing assembly-language code or a compiler writer generating object or source code for the MC68020/MC68881 pair to think of the MC68881 as though the MC68881 were part of the MC68020. No special restrictions are imposed by the coprocessor interface, and floating-point arithmetic is coded exactly like integer arithmetic.

Table 2. Floating-Point Conditional Test Specifiers

Mnemonic	Definition
NOTE The following conditional tests do not set the BSUN bit in the status register exception byte under any circumstances.	
F	False
EQ	Equal
OGT	Ordered Greater Than
OGE	Ordered Greater Than or Equal
OLT	Ordered Less Than
OLE	Ordered Less Than or Equal
OGL	Ordered Greater or Less Than
OR	Ordered
UN	Unordered
UEQ	Unordered or Equal
UGT	Unordered or Greater Than
UGE	Unordered or Greater or Equal
ULT	Unordered or Less Than
ULE	Unordered or Less or Equal
NE	Not Equal
T	True
NOTE All the conditional tests below set the BSUN bit in the status register exception byte if the NAN condition code bit is set when a conditional instruction is executed.	
SF	Signaling False
SEQ	Signaling Equal
GT	Greater Than
GE	Greater Than or Equal
LT	Less Than
LE	Less Than or Equal
GL	Greater or Less Than
GLE	Greater, Less, or Equal
NGLE	Not (Greater, Less, or Equal)
NGL	Not (Greater or Less)
NLE	Not (Less or Equal)
NLT	Not (Less Than)
NGE	Not (Greater or Equal)
NGT	Not (Greater Than)
SNE	Signaling Not Equal
ST	Signaling True

TIMING TABLES FOR TYPICAL EXECUTION

This set of tables allows a quick determination of the typical execution time for any MC68881 instruction when the MC68020 is used as the main processor. Table 3 is used for effective address calculations performed by the MC68020. Entries from this table are added to entries in the other tables, if necessary, to obtain the total number of clock cycles for an operation. Assumptions for Tables 3–7 are as follows:

- The main processor is an MC68020 and operates on the same clock as the MC68881. Instruction prefetches do not hit in the MC68020 cache (or it is disabled), and the instruction is aligned such that a prefetch occurs before the command CIR is written by the MC68020.
- A 32-bit memory interface is used, and memory accesses occur with zero wait states. All memory operands, as well as the stack pointers, are long-word aligned.
- Accesses to the MC68881 require three clock cycles, with the exception of read accesses to the response and save CIRs, which require five clock cycles.
- Since no instruction overlap is utilized, the coprocessor interface overhead is 11 clocks. This can be reduced to two clock cycles if optimized code sequences are used or can be 11 clock cycles if overlap is attempted and a synchronization delay is required.
- Typical operand conversion and calculation times are used (i.e., input operands are assumed to be normalized numbers in the legal range for a given function).
- No exceptions are enabled or occur, and the default rounding mode and precision of round-to-nearest extended precision is used.

EFFECTIVE ADDRESS CALCULATIONS

For any instruction that requires an operand external to the MC68881, an evaluate effective address and transfer data response primitive is issued by the MC68881 during the dialog for that instruction. The time required for the MC68020 to calculate the effective address while processing this primitive for each addressing mode, excluding the transfer of the data to the MC68881, is listed in Table 3.

Table 3. Effective Address Calculations Execution Timing

Addressing Mode	Best Case	Cache Case	Worst Case
Dn or An	0	0	0
(An)	0	2	2
(An)+	3	6	6
−(An)	3	6	6
(d_{16},An) or (d_{16},PC)	0	2	3
(xxx.W)	0	2	3
(xxx).L	1	4	5
#⟨data⟩	0	0	0
(d_8,An,Xn) or (d_8,PC,Xn)	1	4	5
(d_{16},An,Xn) or (d_{16},PC,Xn)	3	6	7
(B)	3	6	7
(d_{16},B)	5	8	9
(d_{32},B)	11	14	16
([B],I)	8	11	12
([B],I,d_{16})	8	11	12
([B],I,d_{32})	10	13	15
([d_{16},B],I)	10	13	14
([d_{16},B],I,d_{16})	10	13	15
([d_{16},B],I,d_{32})	12	15	17
([d_{32},B],I)	16	19	21
([d_{32},B],I,d_{16})	16	19	21
([d_{32},B],I,d_{32})	18	21	24

B = Base address; 0, An, PC, Xn, An + Xn, PC + Xn. Form does not affect timing.
I = Index; 0 or Xn.

Note that Xn cannot be in B and I at the same time. Scaling and size of Sn does not affect timing.

ARITHMETIC OPERATIONS

Table 4 gives the typical instruction execution time for each arithmetic instruction. This group of instructions includes the majority of the MC68881 operations such as FADD, FSUB, etc. In addition to the instructions that perform arithmetic calculations as part of their function, the FCMP, FMOVE, and FTST instructions are also included since an implicit conversion is performed by those operations. For memory operands, the timing for the appropriate effective addressing mode must be added to the numbers in this table to determine the overall instruction execution times.

Table 4. Arithmetic Instructions Execution Timing

Instruction	FPm Source	Memory Source or Destination Operand Format*				
		Integer**	Single**	Double	Extended	Packed
FABS	35	62	54	60	58	872
FACOS	652	625	644	650	648	1462
FADD	51	80	72	78	76	888
FASIN	581	608	600	606	604	1418
FATAN	403	430	422	428	426	1240
FATANH	693	720	712	718	716	1530
FCMP	33	62	54	60	58	870
FCOS	391	418	410	416	414	1228
FCOSH	607	634	626	632	630	1444
FDIV	103	132	124	130	128	940
FETOX	497	524	516	522	520	1334
FETOXM1	545	572	564	570	568	1382
FGETEXP	45	72	64	70	68	882
FGETMAN	31	58	50	56	54	868
FINT	55	82	78	80	78	892
FINTRZ	55	82	78	80	74	892
FLOGN	525	552	544	550	548	1362
FLOGNP1	571	598	590	596	594	1408
FLOG10	581	608	600	606	604	1418
FLOG2	581	608	600	606	604	1418
FMOD	67	94	86	92	90	902
FMOVE to FPn	33	60	52	58	56	870
FMOVE to MEMORY***	—	100	80	86	72	1996
FMOVECR****	29	—	—	—	—	—
FMUL	71	100	92	98	96	908
FNEG	35	62	54	60	58	872
FREM	67	94	86	92	90	902
FSCALE	41	70	62	68	66	878
FSGLDIV	69	98	90	96	94	906
FSGLMUL	59	88	80	86	84	896
FSIN	391	418	410	416	414	1228
FSINCOS	451	478	470	476	474	1288
FSINH	687	714	706	712	710	1524
FSQRT	107	134	126	132	130	944
FSUB	51	80	72	78	76	888
FTAN	473	500	492	498	496	1310
FTANH	661	688	680	686	684	1498
FTENTOX	567	594	586	592	590	1404

— CONTINUED —

Table 4. Arithmetic Instructions Execution Timing

Instruction	FPm Source	Memory Source or Destination Operand Format*				
		Integer**	Single**	Double	Extended	Packed
FTST	33	60	52	58	56	870
FTWOTOX	567	594	586	592	590	1404

*Add the appropriate effective address calculation time.
**If the source or destination is an MC68020 data register, subtract 5 or 2 clock cycles, respectively.
***Assume a static K-factor is used if the destination data format is packed decimal. Add 14 clock cycles if a dynamic K-factor is used.
****The source operand is from the constant ROM rather than a floating-point data register.

MOVE CONTROL REGISTER AND MOVE MULTIPLE OPERATIONS

Table 5 gives the execution times for the FMOVE FPcr and FMOVEM instructions. The timing for the appropriate effective addressing mode must be added to the numbers in this table to determine the overall instruction execution times.

Table 5. FMOVE and FMOVEM Instructions Execution Timing

Instruction*		Best Case	Cache Case	Worst Case
FMOVE	FPcr,Rn	29	31	34
	FPcr,⟨ea⟩	31	33	36
	Rn,FPcr	26	28	31
	⟨ea⟩,FPcr	31	33	36
	#⟨data⟩,FPcr	30	30	31
FMOVEM	FPcr_list,⟨ea⟩	25 + 6n**	27 + 6n	30 + 6n
	⟨ea⟩,FPcr_list	25 + 6n	27 + 6n	30 + 6n
	#⟨data⟩,FPcr_list	24 + 6n	25 + 6n	29 + 6n
FMOVEM	FPdr_list,⟨ea⟩	35 + 25n	37 + 25n	40 + 25n
	⟨ea⟩,FPdr_list	33 + 23n	35 + 23n	38 + 23n
	Dn,⟨ea⟩	49 + 25n	51 + 25n	54 + 25n
	⟨ea⟩,Dn	47 + 23n	49 + 23n	52 + 23n

*Add the appropriate effective address calculation time.
**n is the number of registers transferred.

CONDITIONAL OPERATIONS

Table 6 lists the execution times for the MC68881 conditional instructions. Each entry in this table, except those for the FScc instruction, is complete and does not require the addition of values from any other table. For the FScc instruction, the only additional factor that must be included is the calculate effective address time for the operand to be modified.

Table 6. Conditional Instructions Execution Timing

Instruction	Comments	Best Case	Cache Case	Worst Case
FBcc.W	Branch Taken	18	20	23
	Branch Not Taken	16	18	19
FBcc.L	Branch Taken	18	20	23
	Branch Not Taken	16	18	21
FDBcc	True, Not Taken	18	20	24
	False, Not Taken	22	24	32
	False, Taken	18	20	26
FNOP	No Operation	16	18	19
FScc	Dn	16	18	21
	(An) + or − (An)*	18	22	25
	Memory**	16	20	23
FTRAPcc	Trap Taken	36	39	47
	Trap Not Taken	16	18	22
FTRAPcc.W	Trap Taken	38	41	45
	Trap Not Taken	18	20	23
FTRAPcc.L	Trap Taken	40	43	52
	Trap Not Taken	20	22	27

*For condition true; subtract one clock for condition false.
**Add the appropriate effective address calculation time.

5

FSAVE AND FRESTORE INSTRUCTIONS

The time required for a context save or restore operation is given in Table 7. The appropriate calculate effective address times must be added to the values in this table to obtain the total execution time for these operations. For the FSAVE instruction, the MC68881 may use the not-ready format code to force the MC68020 to wait while internal operations are completed to reduce the size of the saved state frame or to reach a point where a save operation can be performed. Minimum idle time occurs if the MC68881 is in the idle phase when the save CIR is written. A time between the minimum idle and maximum idle occurs if an instruction is in the end phase when the save CIR is read. A minimum busy time occurs if the MC68881 is in the initial phase or at a save boundary in the middle phase when the save CIR is read. Finally, a maximum busy time occurs if the MC68881 has just passed a save boundary in the middle phase when the save CIR is read.

Table 7. FSAVE and FRESTORE Instructions Execution Timing

Instruction	State Frame	Best Case	Cache Case	Worst Case
FRESTORE	Null	19	21	22
	Idle	55	57	58
	Busy	312	314	315
FSAVE	Null	14	16	18
	Idle (Minimum)	50	52	54
	Idle (Maximum)	286	218	290
	Busy (Minimum)	316	318	320
	Busy (Maximum)	552	554	556

*Add the appropriate effective address calculation time.

FUNCTIONAL SIGNAL DESCRIPTIONS

The following paragraphs contain a brief description of the input and output signals for the MC68881 floating-point coprocessor. The signals are functionally organized into groups as shown in Figure 10.

NOTE

The terms **assertion** and **negation** are used extensively to avoid confusion when describing active-low and active-high signals. The term **assert** or **assertion** is used to indicate that a signal is active or true, independent of whether that level is represented by a high or low voltage. The term **negate** or **negation** is used to indicate that a signal is inactive or false.

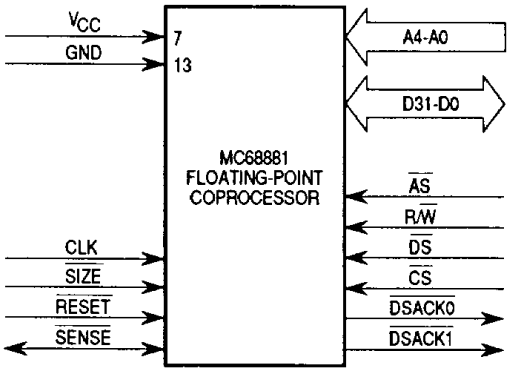


Figure 10. Functional Signal Groups

ADDRESS BUS (A0–A4)

These active-high inputs are used by the main processor to select the CIR locations in the CPU address space. These lines control the register selection (see Table 8).

Table 8. Coprocessor Interface Register Selection

A0–A4	Offset	Width	Type	Register
0000x	$00	16	Read	Response
0001x	$02	16	Write	Control
0010x	$04	16	Read	Save
0011x	$06	16	R/$\overline{W}$	Restore
0100x	$08	16	—	(Reserved)
0101x	$0A	16	Write	Command
0110x	$0C	16	—	(Reserved)
0111x	$0E	16	Write	Condition
100xx	$10	32	R/$\overline{W}$	Operand
1010x	$14	16	Read	Register Select
1011x	$16	16	—	(Reserved)
110xx	$18	32	Read	Instruction Address
111xx	$1C	32	R/$\overline{W}$	Operand Address

5

When the MC68881 is configured to operate over an 8-bit data bus, the A0 pin is used as an address signal for byte accesses of the CIR. When the MC68881 is configured to operate over a 16- or 32-bit data bus, both the A0 and the $\overline{SIZE}$ pins are strapped high and/or low as listed in Table 9.

Table 9. Data Bus Size Configuration

A0	$\overline{SIZE}$	Data Bus
—	Low	8-Bit
Low	High	16-Bit
High	High	32-Bit

DATA BUS (D0–D31)

This 32-bit, bidirectional, three-state bus serves as the general-purpose data path between the MC68020 and the MC68881. Regardless of whether the MC68881 is operated as a coprocessor or a peripheral processor, all interprocessor transfers of instruction information, operand data, status information, and requests for service occur as standard M68000 bus cycles.

The MC68881 will operate over an 8-, 16-, or 32-bit data bus. Depending upon the system data bus configuration, both the A0 and $\overline{SIZE}$ pins are configured specifically for the applicable bus configuration. (Refer to **ADDRESS BUS (A0–A4)** and **SIZE ($\overline{SIZE}$)** for further details).

SIZE ($\overline{SIZE}$)

This active-low input is used in conjunction with the A0 pin to configure the MC68881 for operation over an 8-, 16-, or 32-bit data bus. When the MC68881 is configured to operate over a 16- or 32-bit data bus, both $\overline{SIZE}$ and A0 are strapped high and/or low as listed in Table 9.

ADDRESS STROBE ($\overline{AS}$)

This active-low input indicates a valid address on the address bus and valid signals for both the chip select ($\overline{CS}$) and read/write (R/$\overline{W}$).

CHIP SELECT ($\overline{CS}$)

This active-low input enables the main processor access to the MC68881 CIRs. When operating the MC68881 as a peripheral processor, the chip-select decode is system dependent (i.e., like the chip select on any peripheral). $\overline{CS}$ must be valid (either asserted or negated) when $\overline{AS}$ is asserted.

READ/WRITE (R/$\overline{W}$)

This input indicates the direction of a bus transaction (read/write) by the main processor. A logic high (1) indicates a read from the MC68881, and a logic low (0) indicates a write to the MC68881. The R/$\overline{W}$ signal must be valid when $\overline{AS}$ is asserted.

DATA STROBE ($\overline{DS}$)

This active-low input indicates valid data on the data bus during a write bus cycle.

DATA TRANSFER AND SIZE ACKNOWLEDGE ($\overline{DSACK0}$, $\overline{DSACK1}$)

These active-low, three-state outputs indicate the completion of a bus cycle to the main processor. The MC68881 asserts both $\overline{DSACK0}$ and $\overline{DSACK1}$ upon assertion of $\overline{CS}$.

If the bus cycle is a main processor read, the MC68881 asserts $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$ to indicate information on the data bus is valid. (Both $\overline{\text{DSACKx}}$ signals may be asserted in advance of the valid data being placed on the bus.) If the bus cycle is a main processor write to the MC68881, $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$ are used to acknowledge acceptance of the data by the MC68881.

The MC68881 also uses $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$ to dynamically indicate to the MC68020 the port size (data bus width) on a cycle-by-cycle basis. Depending upon which of the $\overline{\text{DSACKx}}$ pins is asserted in a given bus cycle, the MC68020 assumes data has been transferred to/from an 8-, 16-, or 32-bit data port. Table 10 lists $\overline{\text{DSACKx}}$ assertions used by the MC68881 for the various bus cycles over the various data bus configurations.

Table 10. $\overline{\text{DSACKx}}$ Assertions

Data Bus	A4	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	Comments
32-Bit	1	Low	Low	Valid Data on D31–D0
32-Bit	0	Low	High	Valid Data on D31–D16
16-Bit	x	Low	High	Valid Data on D31–D16 or D15–D0
8-Bit	x	High	Low	Valid Data on D31–D24, D23–D16, D15–D8, or D7–D0
All	x	High	High	Insert Wait States in Current Bus Cycle

Table 10 indicates that all accesses over a 32-bit bus where A4 equals zero are to 16-bit registers. The MC68881 implements all 16-bit CIRs on data lines D16–D31 (to eliminate the need for on-chip multiplexers); however, the MC68020 expects 16-bit registers that are located in a 32-bit port at odd-word addresses (A1 = 1) to be implemented on data lines D0–D15. For accesses to these registers when configured for 32-bit bus operation, the MC68881 generates $\overline{\text{DSACKx}}$ signals as listed in Table 10 to inform the MC68020 of valid data on D16–D31 instead of D0–D15.

An external holding resistor is required to maintain both $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$ high between bus cycles. To reduce the signal rise time, $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$ are actively pulled up (negated) by the MC68881 following the rising edge of $\overline{\text{AS}}$ or $\overline{\text{DS}}$, and both $\overline{\text{DSACKx}}$ lines are then three-stated (placed in the high-impedance state) to avoid interference with the next bus cycle.

RESET ($\overline{\text{RESET}}$)

This active-low input causes the MC68881 to initialize the floating-point data registers to non-signaling not-a-numbers (NANs) and clears the floating-point control, status, and instruction address registers.

When performing a power-up reset, external circuitry should keep $\overline{\text{RESET}}$ asserted for a minimum of four clock cycles after V_{CC} is within tolerance, assuring correct initialization of the MC68881 when power is applied. For compatibility with all M68000 Family devices, 100 ms should be used as the minimum.

When performing a reset of the MC68881 after V_{CC} has been within tolerance for more than the initial power-up time, the $\overline{\text{RESET}}$ line must have an asserted pulse width greater than two clock cycles. For compatibility with all M68000 Family devices, 10 clock cycles should be used as the minimum.

CLOCK (CLK)

The MC68881 clock input is a TTL-compatible signal that is internally buffered for development of the internal clock signals. The clock input should be a constant-frequency square wave with no stretching or shaping techniques required. The clock should not be gated off at any time and must conform to minimum and maximum period and pulse-width times.

SENSE DEVICE ($\overline{\text{SENSE}}$)

This pin may be used optionally as an additional GND pin or as as indicator to external hardware that the MC68881 is present in the system. This signal is internally connected to the GND of the die, but it is not necessary to connect it to the external ground for correct device operation. If a pullup resistor (which should be larger than 10 Ω) is connected to this pin location, external hardware may sense the presence of the MC68881 in a system.

POWER (V_{CC} and GND)

These pins provide the supply voltage and system reference level for the internal circuitry of the MC68881. Care should be taken to reduce the noise level on these pins with appropriate capacitive decoupling.

NO CONNECT (NC)

One pin of the MC68881 package is designated as a no connect (NC). This pin, reserved for future use by Motorola, should not be used for signal routing nor connected to V_{CC} or GND.

SIGNAL SUMMARY

Table 11 provides a summary of all MC68881 signals described in the previous paragraphs.

Table 11. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Three State
Address Bus	A0–A4	Input	High	—
Data Bus	D0–D13	Input/Output	High	Yes
Size	$\overline{SIZE}$	Input	Low	—
Address Strobe	$\overline{AS}$	Input	Low	—
Chip Select	$\overline{CS}$	Input	Low	—
Read/Write	R/$\overline{W}$	Input	High/Low	—
Data Strobe	$\overline{DS}$	Input	Low	—
Data Transfer and Size Acknowledge	$\overline{DSACK0}$, $\overline{DSACK1}$	Output	Low	Yes
Reset	$\overline{RESET}$	Input	Low	—
Clock	CLK	Input	—	—
Sense Device	$\overline{SENSE}$	Input/Output	Low	No
Power Input	V_{CC}	Input	—	—
Ground	GND	Input	—	—

5

INTERFACING METHODS

The following paragraphs describe how to connect an MC68882 to an M68000 Family processor.

MC68881 TO MC68020 INTERFACING

The following paragraphs describe how to connect the MC68881 to an MC68020 for coprocessor operation via an 8-, 16-, or 32-bit data bus.

32-Bit Data Bus Coprocessor Connection

Figure 11 illustrates the coprocessor interface connection of an MC68881 to an MC68020 via a 32-bit data bus. The MC68881 is configured to operate over a 32-bit data bus when both A0 and $\overline{SIZE}$ are connected to V_{CC}.

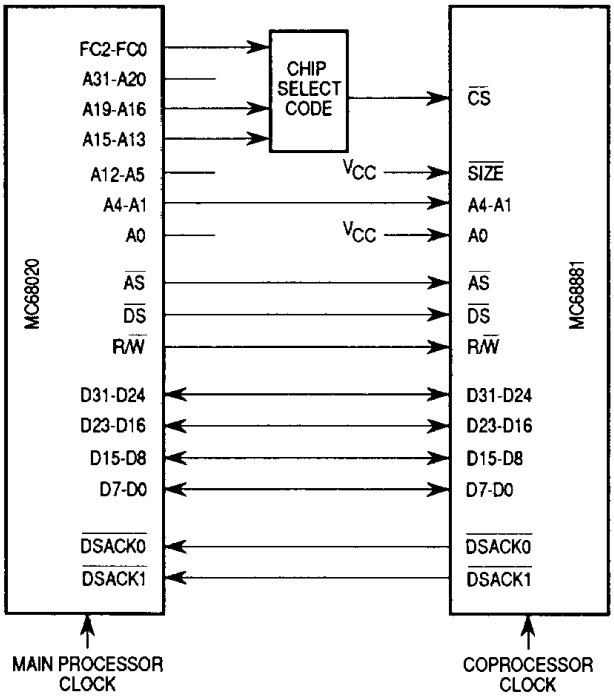


Figure 11. 32-Bit Data Bus Coprocessor Connection

5

16-Bit Data Bus Coprocessor Connection

Figure 12 illustrates the coprocessor interface connection of an MC68881 to an MC68020 via a 16-bit data bus. The MC68881 is configured to operate over a 16-bit data bus when $\overline{SIZE}$ is connected to V_{CC} and A0 is connected to GND. The 16 least significant data pins (D0–D15) must be connected to the 16 most significant data pins (D16–D31) when the MC68881 is configured to operate over a 16-bit data bus (i.e., connect D0 to D16, D1 to D17, . . . and D15 to D31). The $\overline{DSACKx}$ pins of the two devices are directly connected although it is not necessary to connect $\overline{DSACK0}$ since the MC68881 never asserts it in this configuration.

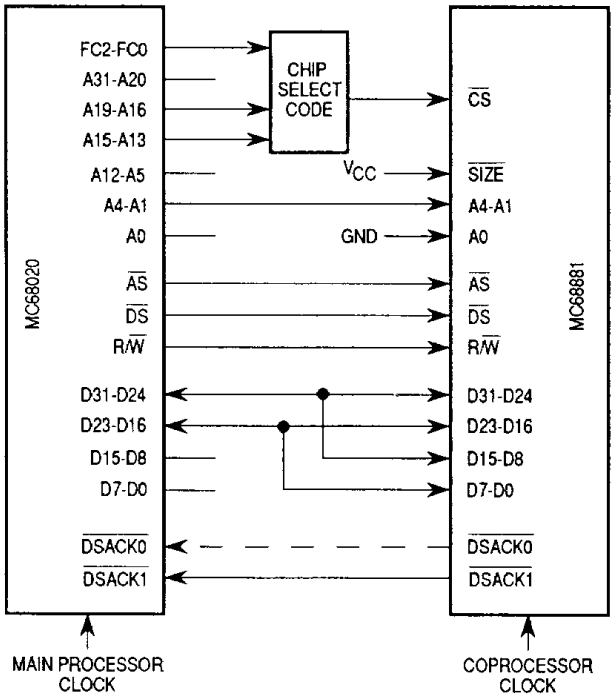


Figure 12. 16-Bit Data Bus Coprocessor Connection

8-Bit Data Bus Coprocessor Connection

Figure 13 illustrates the connection of an MC68881 to an MC68020 as a coprocessor over an 8-bit data bus. The MC68881 is configured to operate over an 8-bit data bus when $\overline{SIZE}$ is connected to GND. The 24 least significant data pins (D0–D23) must be connected to the eight most significant data pins (D24–D31) when the MC68881 is configured to operate over an 8-bit data bus (i.e., connect D0 to D8, D16, and D24; D1 to D9, D17, and D25; . . . and D7 to D15, D23, and D31). The $\overline{DSACKx}$ pins of the two devices are directly connected although it is not necessary to connect $\overline{DSACK1}$ since the MC68881 never asserts it in this configuration.

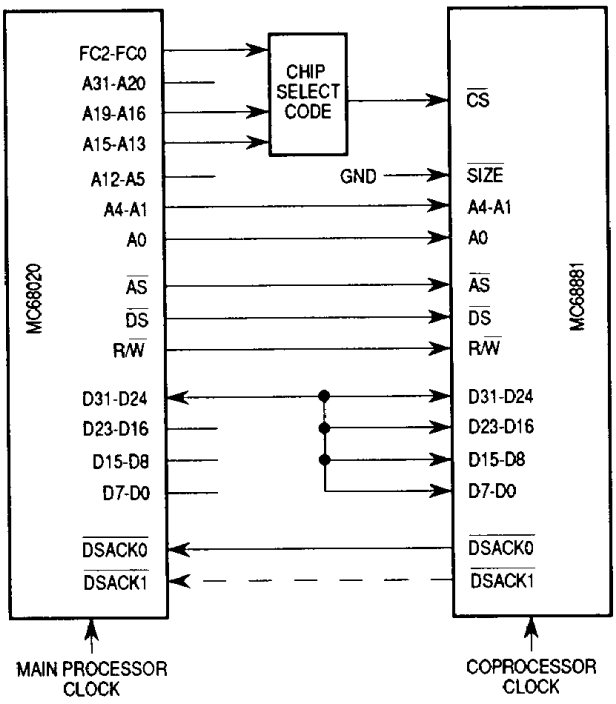


Figure 13. 8-Bit Data Bus Coprocessor Connection

5

MC68881 TO MC68000/MC68008/MC68010 INTERFACING

The following paragraphs describe how to connect the MC68881 to an MC68000/MC68008/MC68010 processor for operation as a peripheral via an 8- or 16-bit data bus.

16-Bit Data Bus Peripheral Processor Connection

Figure 14 illustrates the connection of an MC68881 to an MC68000 or MC68010 as a peripheral processor over a 16-bit data bus. The MC68881 is configured to operate over a 16-bit data bus when $\overline{SIZE}$ is connected to V_{CC} and A0 is connected to GND. The 16 least significant data pins (D0–D15) must be connected to the 16 most significant data pins (D16–D31) when the MC68881 is configured to operate over a 16-bit data bus (i.e., connect D0 to D16, D1 to D17, . . . and D15 to D31). The $\overline{DSACK1}$ pin of the MC68881 is connected to the $\overline{DTACK}$ pin of the main processor; $\overline{DSACK0}$ is not used.

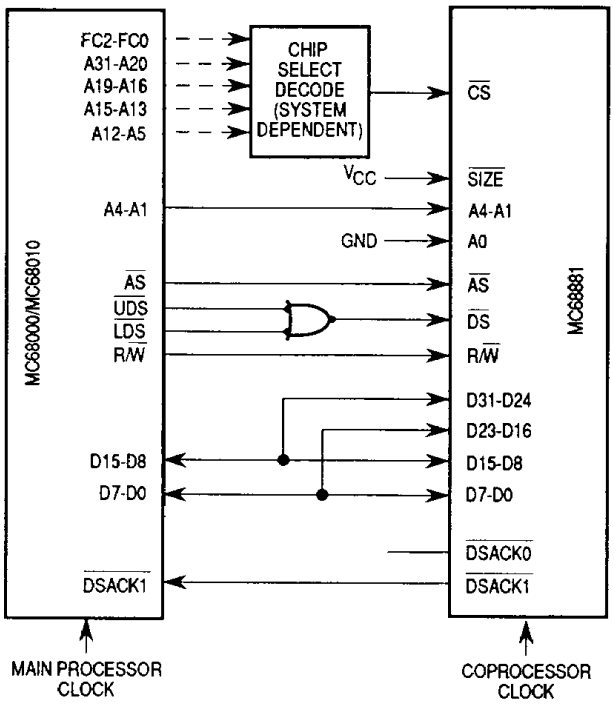


Figure 14. 16-Bit Data Bus Peripheral Processor Connection

When connected as a peripheral processor, the MC68881 chip-select decode is system dependent. If the MC68000 is used as the main processor, the MC68881 $\overline{CS}$ must be decoded in the supervisor or user data spaces. However, if the MC68010 is used for the main processor, the MOVES instruction may be used to emulate any CPU space access that the MC68020 generates for coprocessor communications. Thus, the $\overline{CS}$ decode logic for such systems may be the same as in an MC68020 system, so that the MC68881 will not use any part of the data address spaces.

8-Bit Data Bus Peripheral Processor Connection

Figure 15 illustrates the connection of an MC68881 to an MC68008 as a peripheral processor over an 8-bit data bus. The MC68881 is configured to operate over an 8-bit data bus when $\overline{SIZE}$ is connected to GND. The eight least significant data pins (D0–D7) must be connected to the 24 most significant pins (D8–D31) when the MC68881 is configured to operate over an 8-bit data bus (i.e., connect D0 to D8, D16, and D24; D1 to D9, D17, and D25; . . . and D7 to D15, D23, and D31). The $\overline{DSACK0}$ pin of the MC68881 is connected to the $\overline{DTACK}$ pin of the MC68008; $\overline{DSACK1}$ is not used.

When connected as a peripheral processor, the MC68881 chip-select decode is system dependent, and $\overline{CS}$ must be decoded in the supervisor or user data spaces.

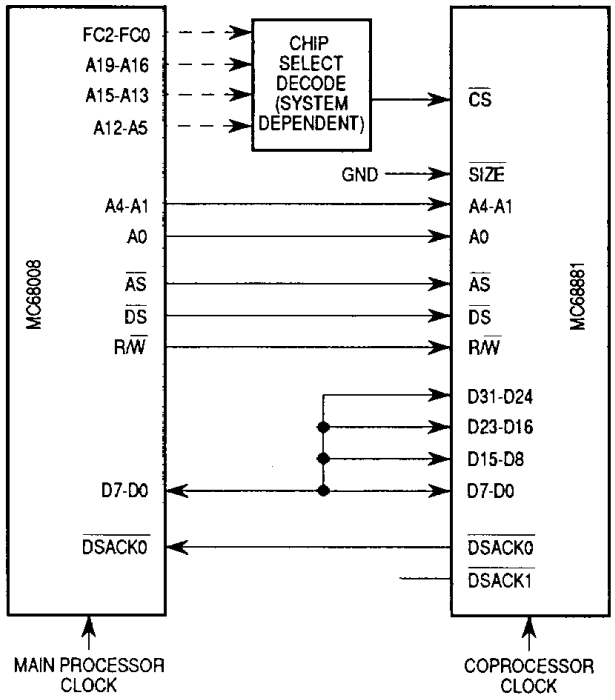


Figure 15. 8-Bit Data Bus Peripheral Processor Connection

5

ELECTRICAL SPECIFICATIONS

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	T_A	0 to 70	°C
Storage Temperature	T_{stg}	−55 to +150	°C

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 GND or V_{CC}).

THERMAL CHARACTERISTICS — PGA PACKAGE

Characteristic	Symbol	Value	Rating
Thermal Resistance — Ceramic			°C/W
Junction to Ambient	θ_{JA}	33	
Junction to Case	θ_{JC}	15	
Thermal Resistance — PLCC			
Junction to Ambient	θ_{JA}	45	
Junction to Case	θ_{JA}	TBD	

5

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 = $P_{INT} + P_{I/O}$
P_{INT} = $I_{CC} \times V_{CC}$, Watts — Chip Internal Power
$P_{I/O}$ = Power Dissipation on Input and Output Pins — User Determined

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

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

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

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.

The total thermal resistance of a package (θ_{JA}) can be separated into two components, θ_{JC} and θ_{CA}, representing the barrier to heat flow from the semiconductor junction to the package (case) surface (θ_{JC}) and from the case to the outside ambient (θ_{CA}). These terms are related by the equation:

$$\theta_{JA} = \theta_{JC} + \theta_{CA} \quad (4)$$

θ_{JC} is device related and cannot be influenced by the user. However, θ_{CA} is user dependent and can be minimized by such thermal management techniques as heat sinks, ambient air cooling, and thermal convection. Thus, good thermal management on the part of the user can significantly reduce θ_{CA} so that θ_{JA} approximately equals θ_{JC}. Substitution of θ_{JC} for θ_{JA} in Equation (1) will result in a lower semiconductor junction temperature.

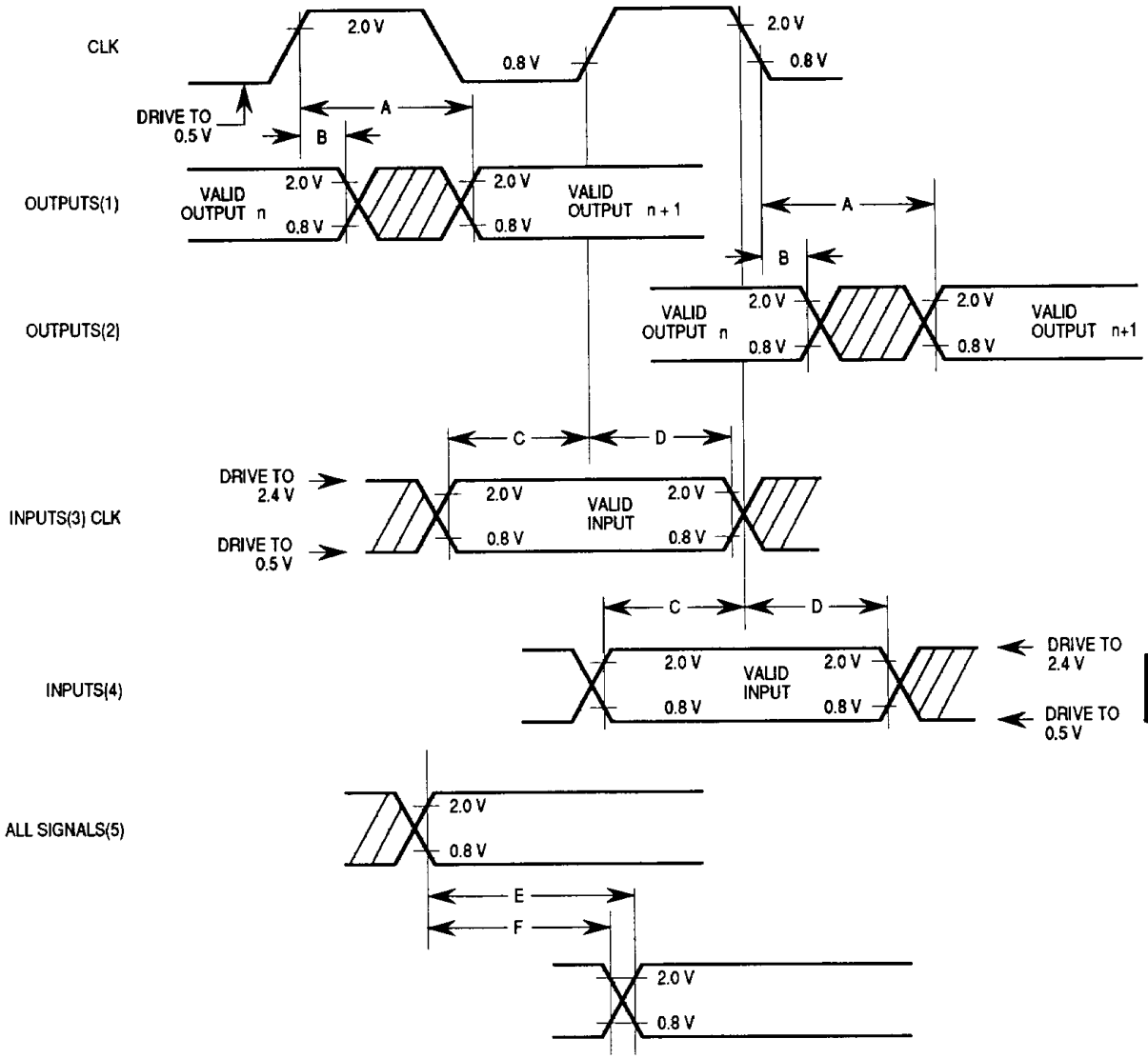
Values for thermal resistance presented in this document, unless estimated, were derived using the procedure described in Motorola Reliability Report 7843, "Thermal Resistance Measurement Method for MC68XX Microcomponent Devices," and are provided for design purposes only. Thermal measurements are complex and dependent on procedure and setup. User-derived values for thermal resistance may differ.

AC ELECTRICAL SPECIFICATIONS DEFINITIONS

The AC specifications presented consist of output delays, input setup and hold times, and signal skew times. All signals are specified relative to an appropriate edge of the clock input and, possibly, relative to one or more other signals.

The measurement of the AC specifications is defined by the waveforms shown in Figure 16. To test the parameters guaranteed by Motorola, inputs must be driven to the voltage levels specified in Figure 16. Outputs are specified with minimum and/or maximum limits, as appropriate, and are measured as shown. Inputs are specified with minimum and, as appropriate, maximum setup and hold times, and are measured as shown. Finally, the measurements for signal-to-signal specifications are also shown.

Note that the testing levels used to verify conformance to the AC specifications does not affect the guaranteed DC operation of the device as specified in the DC electrical characteristics.



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

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

Figure 16. Drive Levels and Test Points for AC Specifications

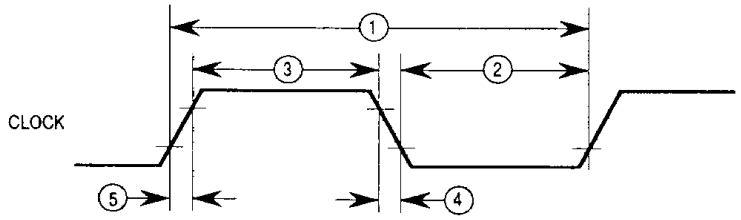
DC ELECTRICAL SPECIFICATIONS (V_{CC} = 5.0 Vdc ± 5%; GND = 0 Vdc; T_A = 0°C to 70°C)

Characteristic		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	GND −0.5	0.8	V
Input Leakage Current @ 5.25 V	CLK, $\overline{RESET}$, R/$\overline{W}$, A0–A4, $\overline{CS}$, $\overline{DS}$, $\overline{AS}$, $\overline{SIZE}$	I_{in}	—	10	μA
Hi-Z (Off State) Input Current @ 2.4 V/0.4 V	$\overline{DSACK0}$, $\overline{DSACK1}$, D0–D31	I_{TSI}	—	20	μA
Output High Voltage (I_{OH} = −400 μA)	$\overline{DSACK0}$, $\overline{DSACK1}$, D0–D31	V_{OH}	2.4	—	V
Output Low Voltage (I_{OL} = 5.3 mA)	$\overline{DSACK0}$, $\overline{DSACK1}$, D0–D31	V_{OL}	—	0.5	V
Power Dissipation		P_D	—	0.75	W
Input Capacitance* (V_{in} = 0, T_A = 25°C, f = 1 MHz)		C_{in}	—	20	pF
Output Load Capacitance		C_L	—	130	pF

*Capacitance is periodically sampled rather than 100% tested.

AC ELECTRICAL SPECIFICATIONS — CLOCK INPUT (V_{CC} = 5.0 Vdc ± 5%; GND = 0 Vdc; T_A = 0 to 70°C; refer to Figure 17)

Num	Characteristic	12 MHz		16.67 MHz		20 MHz		25 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
	Frequency of Operation	8	12.5	8	16.67	12.5	20	12.5	25	MHz
1	Cycle Time Clock	80	125	60	125	50	80	40	80	ns
2,3	Clock Pulse Width	32	87	24	95	20	54	15	59	ns
4,5	Rise and Fall Times	—	5	—	5	—	5	—	4	ns



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

Figure 17. Clock Input Timing Diagram

AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES

(V_{CC} = 5.0 Vdc ± 5%; GND = 0 Vdc; T_A = 0 to 70°C; refer to Figures 18, 19, and 20)

Num	Characteristic	12 MHz		16.67 MHz		20 MHz		25 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
6[5]	Address Valid to $\overline{AS}$ Asserted	20	—	15	—	10	—	5	—	ns
6A[5]	Address Valid to $\overline{DS}$ Asserted (Read)	20	—	15	—	10	—	5	—	ns
6B[5]	Address Valid to $\overline{DS}$ Asserted (Write)	65	—	50	—	50	—	35	—	ns
7[5]	$\overline{AS}$ Negated to Address Invalid	15	—	10	—	10	—	5	—	ns
7A[6]	$\overline{DS}$ Negated to Address Invalid	15	—	10	—	10	—	5	—	ns
8[9]	$\overline{CS}$ Negated to $\overline{AS}$ Asserted	0	—	0	—	0	—	0	—	ns
8A[9]	$\overline{CS}$ Negated to $\overline{DS}$ Asserted (Read)	0	—	0	—	0	—	0	—	ns
8B	$\overline{CS}$ Asserted to $\overline{DS}$ Asserted (Write)	40	—	30	—	25	—	20	—	ns
9	$\overline{AS}$ Negated to $\overline{CS}$ Negated	10	—	10	—	10	—	5	—	ns
9A	$\overline{DS}$ Negated to $\overline{CS}$ Negated	10	—	10	—	10	—	5	—	ns
10	R/$\overline{W}$ High to $\overline{AS}$ Asserted (Read)	20	—	15	—	10	—	5	—	ns
10A	R/$\overline{W}$ High to $\overline{DS}$ Asserted (Read)	20	—	15	—	10	—	5	—	ns
10B	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	45	—	35	—	30	—	25	—	ns
11	$\overline{AS}$ Negated to R/$\overline{W}$ Low (Read) or $\overline{AS}$ Negated to R/$\overline{W}$ High (Write)	15	—	10	—	10	—	5	—	ns
11A	$\overline{DS}$ Negated to R/$\overline{W}$ Low (Read) or $\overline{DS}$ Negated to R/$\overline{W}$ High (Write)	15	—	10	—	10	—	5	—	ns

5

AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES

(Continued)

Num	Characteristic	12 MHz		16.67 MHz		20 MHz		25 MHz		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
12	$\overline{DS}$ Width Asserted (Write)	50	—	40	—	38	—	30	—	ns
13	$\overline{DS}$ Width Negated	50	—	40	—	38	—	30	—	ns
13A[4]	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	40	—	30	—	30	—	25	—	ns
14[2]	$\overline{CS}$, $\overline{DS}$ Asserted to Data-Out Valid) (Read)	—	110	—	80	—	60	—	45	ns
15	$\overline{DS}$ Negated to Data-Out Invalid (Read)	0	—	0	—	0	—	0	—	ns
16	$\overline{DS}$ Negated to Data-Out High Impedance (Read)	—	70	—	50	—	30	—	30	ns
17	Data-In Valid to $\overline{DS}$ Asserted (Write)	20	—	15	—	10	—	5	—	ns
18	$\overline{DS}$ Negated to Data-In Invalid (Write)	20	—	15	—	10	—	5	—	ns
19[2]	$\overline{START}$ True to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted	—	70	—	50	—	35	—	25	ns
19A[7]	$\overline{DSACK0}$ Asserted to $\overline{DSACK1}$ Asserted (Skew)	−20	20	−15	15	−10	10	−10	10	ns
20	$\overline{DSACK0}$ or $\overline{DSACK1}$ Asserted to Data-Out Valid	—	60	—	50	—	43	—	32	ns
21[8]	$\overline{START}$ False to $\overline{DSACK0}$ and $\overline{DSACK1}$ Negated	—	70	—	50	—	30	—	30	ns
22[8]	$\overline{START}$ False to $\overline{DSACK0}$ and $\overline{DSACK1}$ High Impedance	—	90	—	70	—	40	—	40	ns
23[3,8]	$\overline{START}$ True to Clock High (Synchronous Read)	0	—	0	—	0	—	0	—	ns
24[3]	Clock Low to Data-Out Valid (Synchronous Read)	—	140	—	105	—	80	—	60	ns
25[3,8]	$\overline{START}$ True to Data-Out Valid (Synchronous Read)	— 1.5	140 + 2.5	— 1.5	105 + 2.5	— 1.5	80 + 2.5	— 1.5	60 + 2.5	ns Clks
26[3]	Clock Low to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted (Synchronous Read)	—	100	—	75	—	55	—	45	ns
27[3,8]	$\overline{START}$ True to $\overline{DSACK0}$ and $\overline{DSACK1}$ Asserted (Synchronous Read)	— 1.5	100 + 2.5	— 1.5	75 + 2.5	— 1.5	55 + 2.5	— 1.5	45 + 2.5	ns Clks

NOTES:

1. Timing measurements are referenced to and from a low voltage of 0.8 V and a high voltage of 2.0 V, unless otherwise noted. The voltage swing through this range should start outside, and pass through, the range such that the rise or fall will be linear between 0.8 V and 2.0 V.
2. These specifications only apply if the MC68881 has completed all internal operations initiated by the termination of the previous bus cycle when $\overline{DS}$ was negated.
3. Synchronous read cycles occur *only* when the save or response CIR locations are read.
4. This specification only applies to systems in which back-to-back accesses (read-write or write-write) of the operand CIR can occur. When the MC68881 is used as a coprocessor to the MC68020/MC68030, this can occur when the addressing mode is immediate.
5. If the $\overline{SIZE}$ pin is *not* strapped to either V_{CC} or GND, it must have the same setup times as do addresses.
6. If the $\overline{SIZE}$ pin is *not* strapped to either V_{CC} or GND, it must have the same hold times as do addresses.
7. This number is reduced to 5 ns if $\overline{DSACK0}$ and $\overline{DSACK1}$ have equal loads.
8. START is not an external signal; rather, it is the logical condition that indicates the start of an access. The logical equation for this condition is $\overline{START} = \overline{CS} + \overline{AS} + (R/\overline{W} \cdot \overline{DS})$.
9. If a subsequent access is not a FPCP access, $\overline{CS}$ must be negated before the assertion of $\overline{AS}$ and/or $\overline{DS}$ on the non-FPCP access. These specifications replace the old specifications 8 and 8A. (The old specifications implied that, in all cases, transitions of $\overline{CS}$ must not occur simultaneously with transitions of $\overline{AS}$ or $\overline{DS}$. This is not a requirement of the MC68881.)

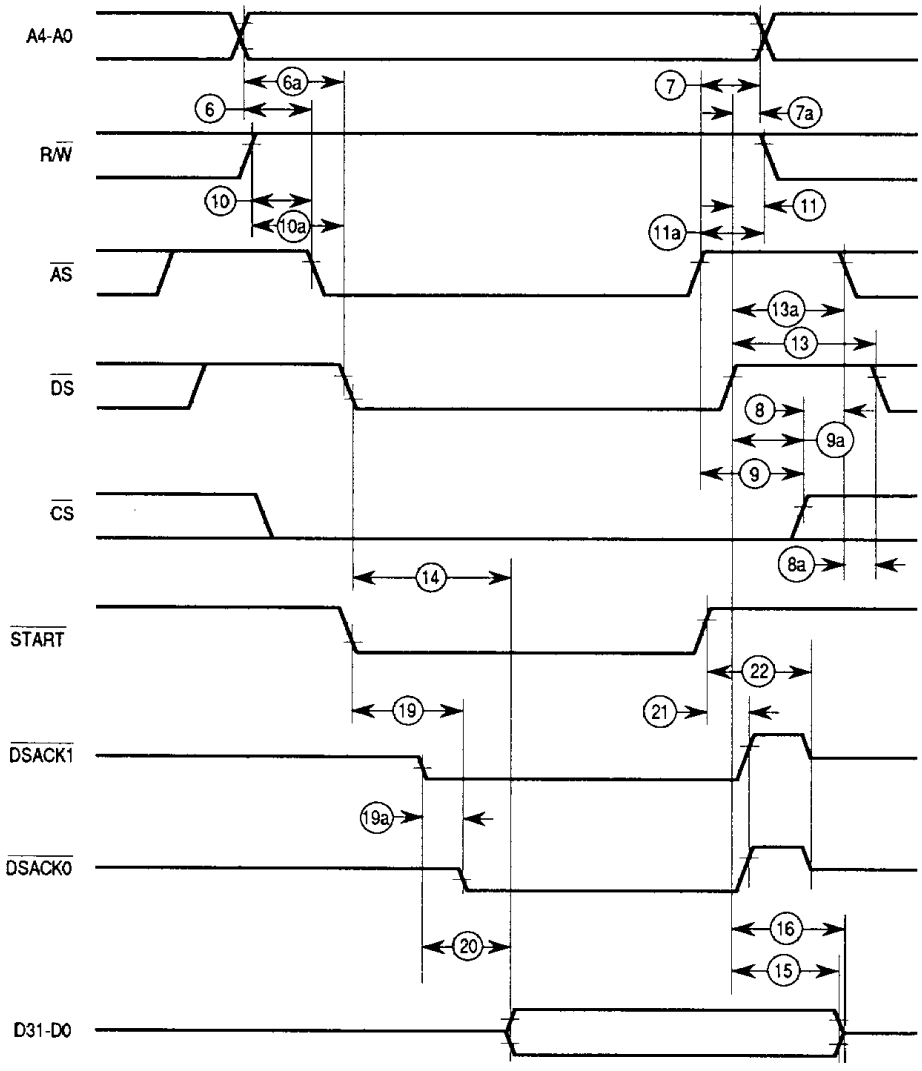


Figure 18. Asynchronous Read Cycle Timing Diagram

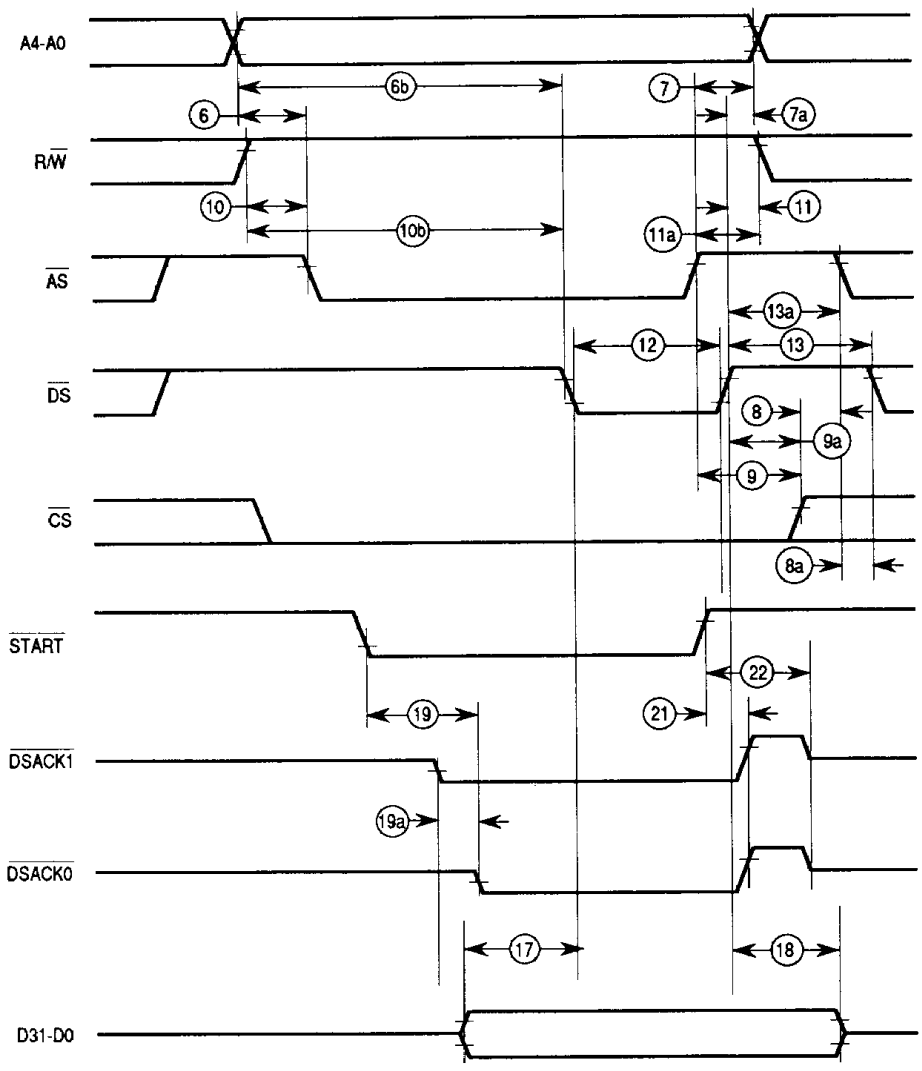


Figure 19. Asynchronous Write Cycle Timing Diagram

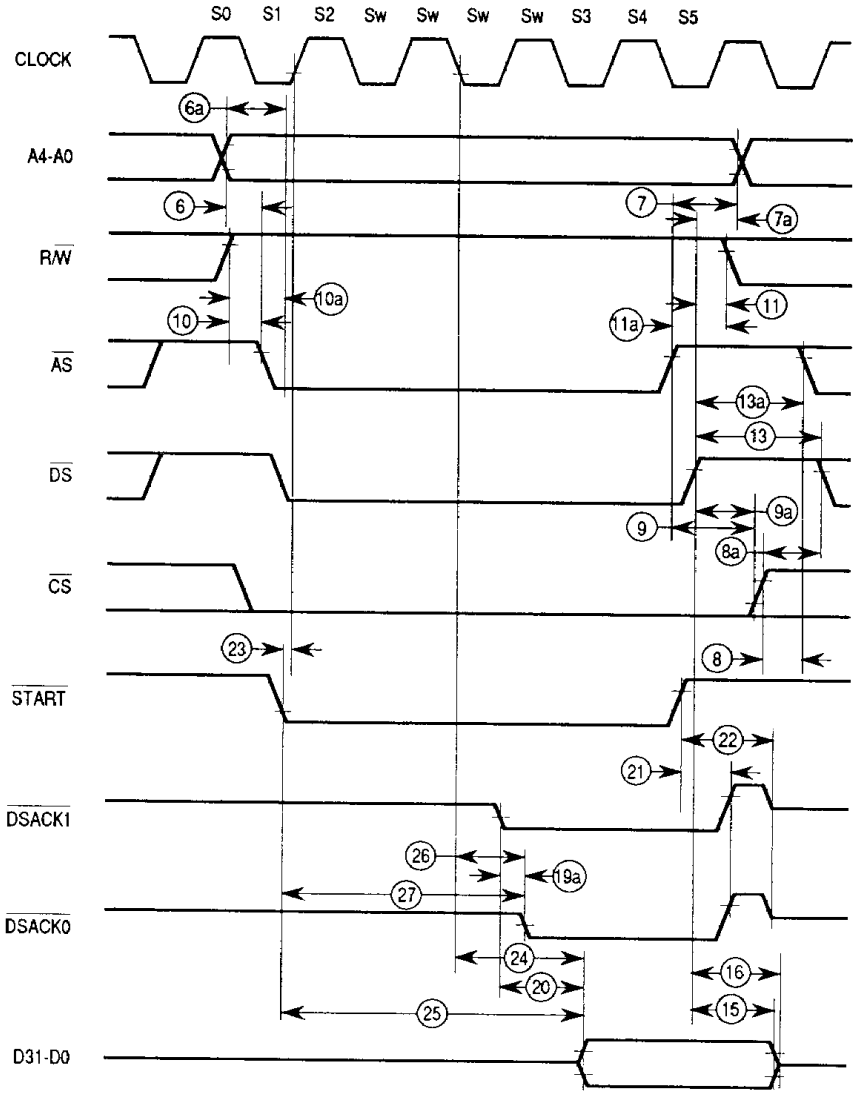
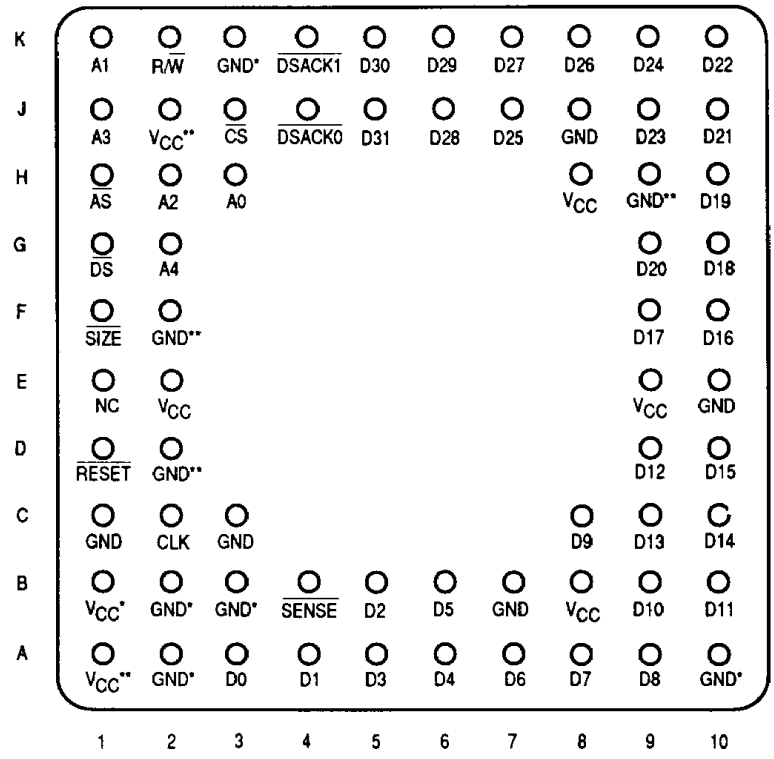


Figure 20. Synchronous Read Cycle Timing Diagram

PIN ASSIGNMENTS

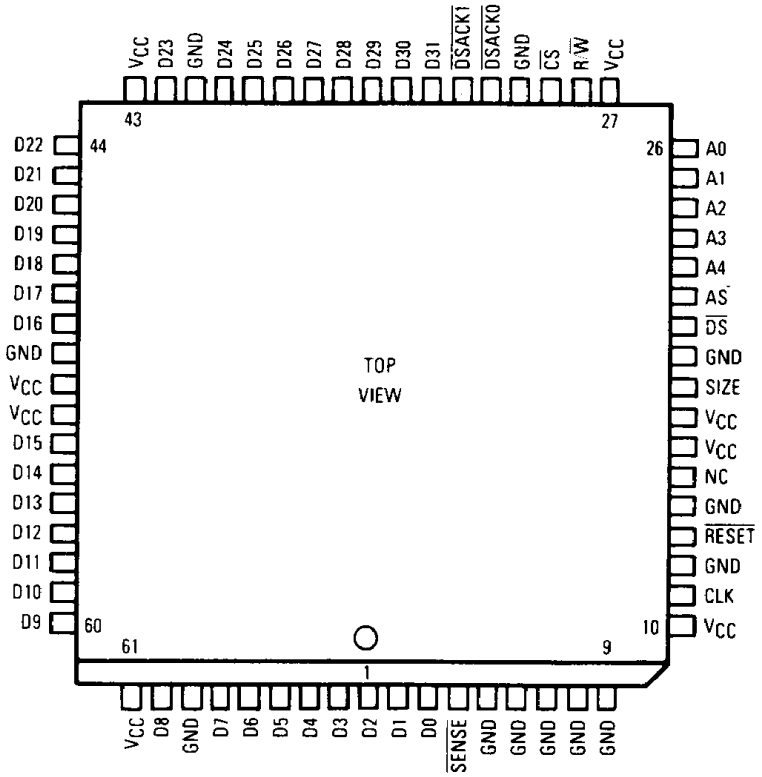
68-LEAD PIN GRID ARRAY



Pin Group	V_{CC}	GND
D31-D16	H8	J8
D15-D00	B8	B7
Internal Logic $\overline{DSACK1}$, $\overline{DSACK0}$	E2, E9	A2, B2, B3, B4*** C3, E10, K3
Separate	—	C1
Extra	A1, B1, J2	A10, D2, F2, H9

* New assignment for the A93N mask.
** Reserved for future Motorola use.
*** SENSE pin, may be used as an additional GND pin.

68-LEAD PLASTIC LEADED CHIP CARRIER



5