9. Architecture of I8087 Floating-Point Unit. Data Formats. FPU instructions. Flags.

10. Types of memory. Memory structure, pin configuration and operations. Bus timing.

Floating-Point Unit I8087

The family of coprocessors, which are labeled the 80X87, is able to multiply, divide, add, subtract, find the square root, partial tangent, partial arctangent, and logarithms. Data types include 16-, 32-, and 64-bit signed integers; 18-digit BCD data; and 32-, 64-, and 80-bit floating-point numbers. The operations performed by the 80X87 generally execute many times faster than equivalent operations written with the most efficient programs using the microprocessor's normal instruction set. With the improved Pentium coprocessor, operations execute at about 5 times faster than those performed by the 80486 microprocessor with an equal clock frequency. Note that the Pentium can often execute a coprocessor instruction and two integer instructions simultaneously.

Data formats for the arithmetic coprocessor

Signed Integers

The signed integers used with the coprocessor are basically the same as those described in Chapter 1. When used with the arithmetic coprocessor, signed integers are 16- (word) , 32- (short integer), or 64-bits (long integer) wide. Conversion between decimal and signed-integer format is handled in exactly the same manner as for the signed integers described in Chapter 1. As you will recall, positive numbers are stored in true form with a leftmost sign-bit of 0, and negative numbers are stored in two's complement form with a leftmost sign-bit of 1.

The word integers range in value from –32,768 to +32,767, the short integer range is $\pm 2 \times 10^{+9}$, and the long integer range is $\pm 9 \times 10^{+18}$. Integer data types are found in some applications that use the arithmetic coprocessor. Refer to Figure 13–1, which shows these three forms of signed-integer data.

Data are stored in memory using the same assembler directives described and used in earlier chapters. The DW directive defines words, DD defines short integers, and DQ defines long-integers. Example 13–1 shows how several different sizes of signed-integers are defined for use by the assembler and arithmetic coprocessor.

EXAMPLE 13–1

```
0000  0002               DATA1   DW   +2          ;16-bit integer
0002  FFDE               DATA2   DW   -34         ;16-bit integer
0004  000004D2           DATA3   DD   +1234       ;short integer
0008  FFFFFF9C           DATA4   DD   -100        ;short integer
000C  0000000000005BA0   DATA5   DQ   +23456      ;long integer
0014  FFFFFFFFFFFFFF86   DATA6   DQ   -122        ;long integer
```

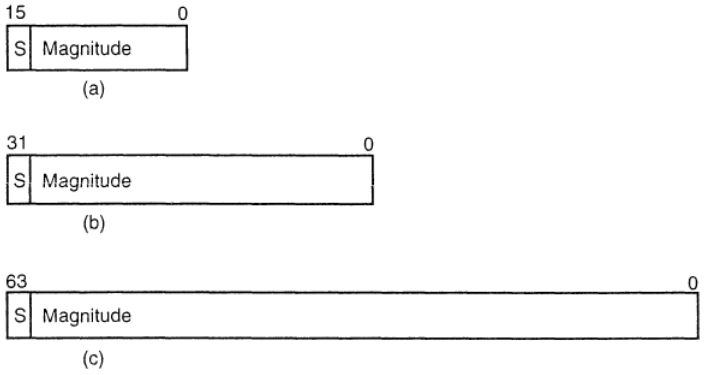


FIGURE 13–1 Integer formats for the 80X87 family of arithmetic coprocessors: (a) word, (b) short, and (c) long

Binary-Coded Decimal (BCD)

The binary-coded decimal (BCD) form requires 80-bits of memory. Each number is stored as an 18-digit packed integer in nine bytes of memory as two digits per byte. The tenth byte contains only a sign-bit for the 18-digit signed BCD number. Figure 13–2 shows the format of the BCD number used with the arithmetic coprocessor. Note that both positive and negative numbers are stored in true form and never in 10's complement form. The DT directive stores BCD data in the memory as illustrated in Example 13–2.

EXAMPLE 13–2

```
0000                DATA1   DT      200             ;200 decimal stored as BCD
     00000000000000000200
000A                DATA2   DT      -10             ;-10 decimal stored as BCD
     80000000000000000010
0014                DATA3   DT      10020           ;10,020 decimal stored as BCD
     00000000000000010020
```

Floating-Point

Floating-point numbers are often called *real numbers* because they hold signed integers, fractions, and mixed numbers. A floating-point number has three parts: a sign-bit, a biased exponent, and a significand. Floating-point numbers are written in scientific binary notation. The Intel family of arithmetic coprocessors supports three types of floating-point numbers: short (32-bits), long (64-bits), and temporary (80-bits). Refer to Figure 13–3 for the three forms of the floating-point number. Please note that the short form is also called a *single-precision number* and the long form is called a *double-precision number*. Sometimes the 80-bit temporary form is called an *extended-precision number*. The floating-point numbers, and the operations performed by the arithmetic coprocessor, conform to the IEEE-754 standard as adopted by all major personal computer software producers. This includes Microsoft, which has recently stopped supporting the Microsoft floating-point format and also the ANSI floating-point standard popular in mainframe computer systems.

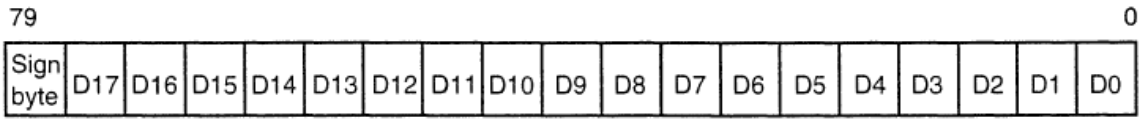
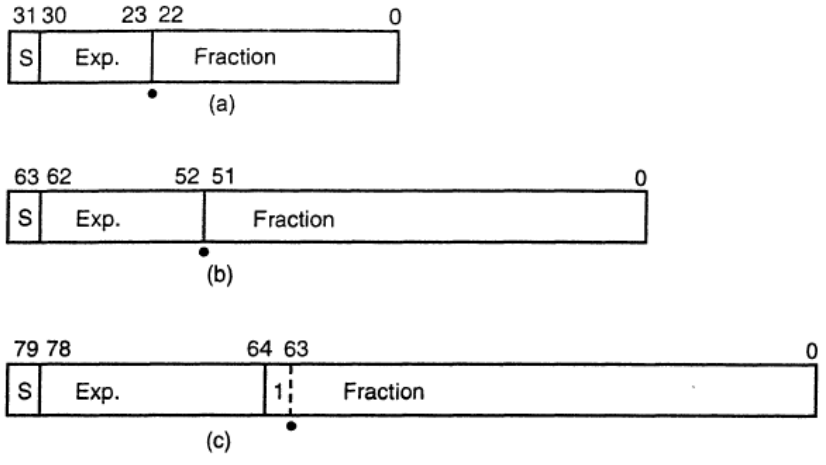


FIGURE 13–2 BCD data format for the 80X87 family of arithmetic coprocessors



Note: S = sign-bit and Exp. = exponent

FIGURE 13–3 Floating-point (real) format for the 80X87 family of arithmetic coprocessors. (a) Short (single-precision) with a bias of 7FH, (b) long (double-precision) with a bias of 3FFH, and (c) temporary (extended-precision) with a bias of 3FFFH

Converting to Floating-point Form. Converting from decimal to the floating-point form is a simple task that is accomplished through the following steps:

1. Convert the decimal number into binary.
2. Normalize the binary number.
3. Calculate the biased exponent.
4. Store the number in the floating-point format.

These four steps are illustrated for the decimal number 100.25_{10} in Example 13–3. Here the decimal number is converted to a single-precision (32-bit) floating-point number.

EXAMPLE 13–3

```
Step     Result

1        100.25 = 1100100.01

2        1100100.01 = 1.10010001 x 2^6

3        110 + 01111111 = 10000101

4        Sign = 0
         Exponent = 10000101
         Significand = 10010001000000000000000
```

Converting from Floating-point Form. Conversion to a decimal number from a floating-point number is summarized in the following steps:

1. Separate the sign-bit, biased exponent, and significand.
2. Convert the biased exponent into a true exponent by subtracting the bias.
3. Write the number as a normalized binary number.
4. Convert it to a de-normalized binary number.
5. Convert the de-normalized binary number to decimal.

Storing Floating-point Data in Memory. Floating-point numbers are stored with the assembler using the DD directive for single-precision, DQ for double-precision, and DT for extended-precision. Some examples of floating-point data storage are shown in Example 13–5. The author discovered that the Microsoft MACRO assembler version 6.0 contains an error that does not allow a plus-sign to be used with positive floating-point numbers. A +92.45 must be defined as 92.45 for the assembler to function correctly. Microsoft has assured the author that this error has been corrected in version 6.11 of MASM if the REAL4, REAL8, or REAL10 directives are used in place of DD, DQ, and DT to specify floating-point data.

In assembly source you must include the directive .8087, .187, .287, .387, .487 or .587 to enable generation of coprocessor instructions.

EXAMPLE 13–5

```
0000  C377999A      DATA7     DD      -247.6       ;define single-precision
0004  40000000      DATA8     DD      2.0          ;define single-precision
0008  486F4200      DATA9     PEAL4   2.45E+5      ;define single-precision
000C                DATA10    DQ      100.25       ;define double-precision
      4059100000000000
```

THE 80X87 ARCHITECTURE

The 80X87 is designed to operate concurrently with the microprocessor. Note that the 80486DX, Pentium, and Pentium Pro microprocessors contain their own internal and fully compatible versions of the 80387. With other family members, the coprocessor is an external integrated circuit that parallels most of the connections on the microprocessor. The 80X87 executes 68 different instructions. The microprocessor executes all normal instructions, and the 80X87 executes arithmetic coprocessor instructions. Both the microprocessor and coprocessor can execute their respective instructions simultaneously or concurrently. The numeric or arithmetic coprocessor is a special-purpose microprocessor that is specially designed to efficiently execute arithmetic and transcendental operations.

The microprocessor intercepts and executes the normal instruction set and the coprocessor intercepts and executes only the coprocessor instructions. Recall that the coprocessor instructions are actually escape (ESC) instructions. These instructions are used by the microprocessor to generate a memory address for the coprocessor so the coprocessor can execute a coprocessor instruction.

Internal Structure of the 80X87

Figure 13–4 shows the internal structure of the arithmetic coprocessor. Notice that this device is divided into two major sections: the control unit and the numeric execution unit.

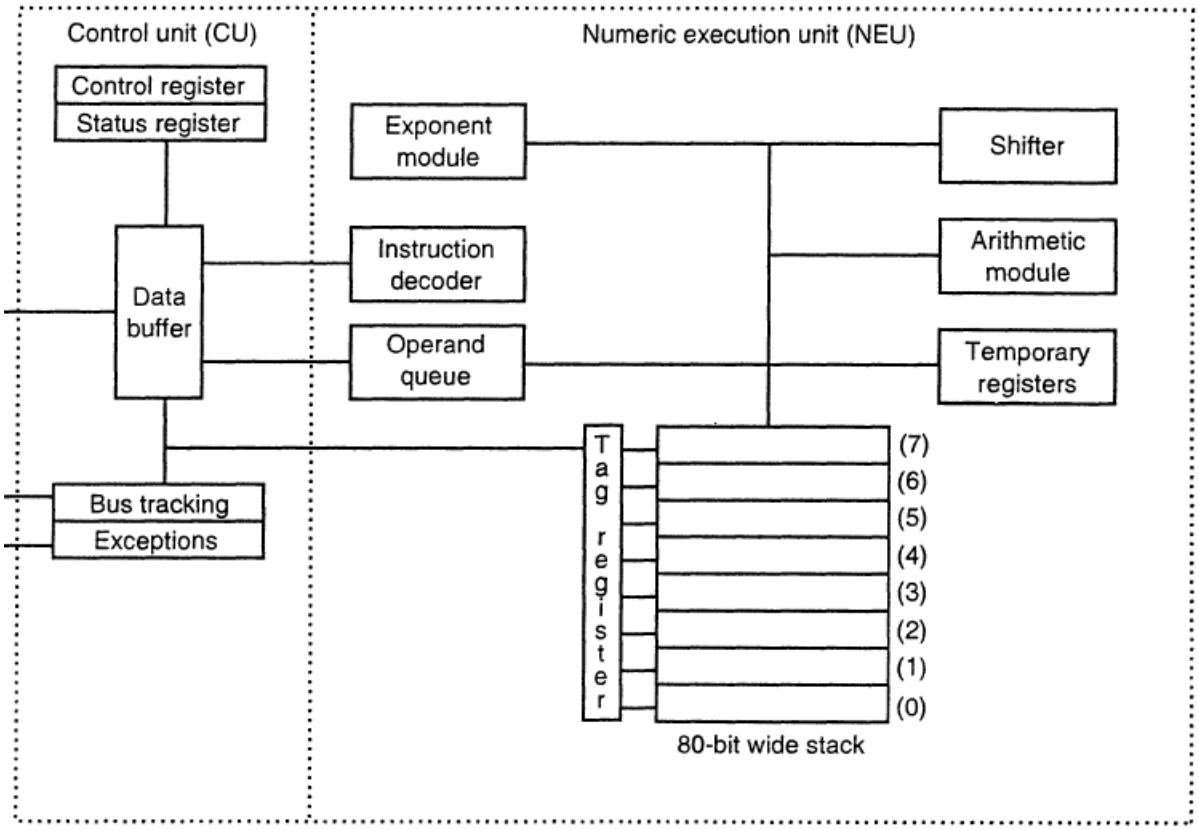


FIGURE 13–4 The internal structure of the 80X87 arithmetic coprocessor

The **control unit** interfaces the coprocessor to the microprocessor system data bus. Both of the devices monitor the instruction stream. If the instruction is an ESCape (coprocessor) instruction, the coprocessor executes it; if not, the microprocessor executes it.

The **numeric execution unit** (NEU) is responsible for executing all coprocessor instructions. The NEU has an eight-register stack that holds operands for arithmetic instructions and the results of arithmetic instructions. Instructions either address data in specific stack data registers or use a push and pop mechanism to store and retrieve data on the top of the stack. Other registers in the NEU are status, control, tag, and exception pointers. A few instructions transfer data between the coprocessor and the AX register in the microprocessor. The FSTSW AX instruction is the only instruction available to the coprocessor that allows direct communications to the microprocessor through the AX register. Note that the 8087 does not contain the FSTSW AX instruction.

The stack within the coprocessor contains eight registers that are each 80-bits wide. These stack registers always contain an 80-bit extended precision floating-point number. The only time data appear as any other form is when they reside in the memory system. The coprocessor converts from signed integer, BCD, single-precision, or double-precision form as the data are moved between the memory and the coprocessor register stack.

Status Register. The **status register** (see Figure 13–5) reflects the overall operation of the coprocessor. The status register is accessed by executing the instruction (FSTSW), which stores the contents of the status register into a word of memory. The FSTSW AX instruction copies the status register directly to the microprocessor's AX register on the 80287 or above coprocessor. Once status is stored in memory or the AX register, the bit positions of the status register can be examined by normal software. The coprocessor/microprocessor communications are carried out through the I/O ports 00FAH–00FFH on the 80287 and I/O ports 800000FAH–800000FFH on the 80386 through the Pentium Pro. Never use these I/O ports for interfacing I/O devices to the microprocessor.

The newer coprocessors (80287 and above) use status bit position 6 (SF) to indicate a stack overflow or underflow error. Following is a list of the status bits, except for SF, and their applications:

B The **busy bit** indicates that the coprocessor is busy executing a task. Busy can be tested by examining the status register or by using the FWAIT instruction. Newer coprocessors automatically synchronize with the microprocessor, so the busy flag need not be tested before performing additional coprocessor tasks.

C3–C0 The **condition code bits** indicate conditions about the coprocessor (refer to Table 13–2 for a complete listing of each combination of these bits and their functions). Note that these bits have different meanings for different instructions, as indicated in the table. The top of the stack is denoted as ST in this table.

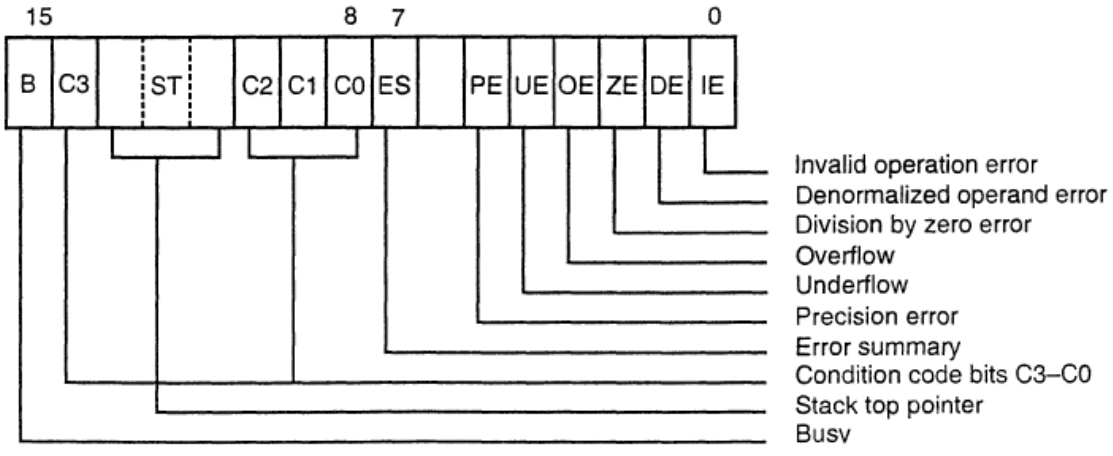


FIGURE 13–5 The 80X87 arithmetic coprocessor status register

TOP The **top-of-stack (ST) bit** indicates the current register addressed as the top-of-the-stack (ST). This is normally register 0.

ES The **error summary** bit is set if any unmasked error bit (PE, UE, OE, ZE, DE, or IE) is set. In the 8087 coprocessor, the error summary also caused a coprocessor interrupt. Since the 80287, the coprocessor interrupt has been absent from the family.

PE The **precision error** indicates that the result or operands exceed selected precision.

UE An **under-flow error** indicates a non-zero result that is too small to represent with the current precision selected by the control word.

OE An **overflow error** indicates a result that is too large to be represented. If this error is masked, the coprocessor generates infinity for an overflow error.

ZE A **zero error** indicates the divisor was zero while the dividend is a non-infinity or non-zero number.

DE A **denormalized error** indicates at least one of the operands is denormalized.

IE An **invalid error** indicates a stack overflow or underflow, indeterminate form (0/0, o, –o, etc), or the use of a NAN as an operand. This flag indicates errors such as those produced by taking the square root of a negative number, etc.

Control Register. The **control register** is pictured in Figure 13–6. The control register selects precision, rounding control, and infinity control. It also masks and unmasks the exception bits that correspond to the rightmost six bits of the status register. The FLDCW instruction is used to load a value into the control register.

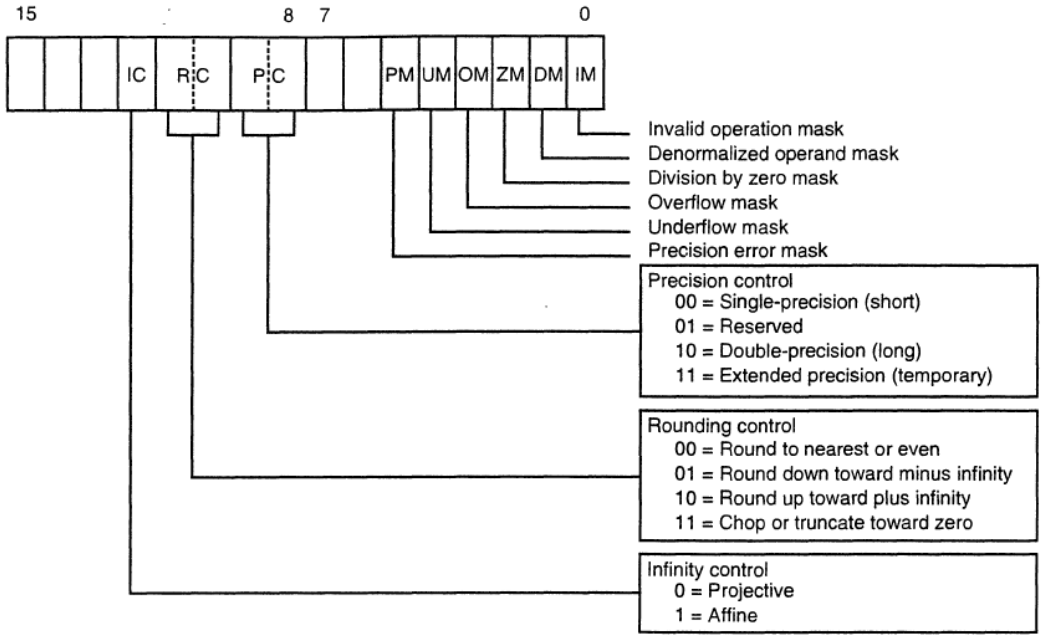


FIGURE 13–6 The 80X87 arithmetic coprocessor control register

IC **Infinity control** selects either affine or projective infinity. Affine allows positive and negative infinity, while projective assumes infinity is unsigned.

RC **Rounding control** determines the type of rounding as defined in Figure 13–6.

PC The **precision control** sets the precision of the result as defined in Figure 13–6.

Exception Masks Determine whether the error indicated by the exception affects the error bit in the status register. If a logic 1 is placed in one of the exception control bits, the corresponding status register bit is masked off.

Tag Register. The **tag register** indicates the contents of each location in the coprocessor stack. Figure 13–7 illustrates the tag register and the status indicated by each tag. The tag indicates whether a register is valid, zero, invalid or infinity, or empty. The only way that a program can view the tag register is by storing the coprocessor environment using the FSTENV, FSAVE, or FRSTOR instructions. Each of these instructions stores the tag register along with other coprocessor data.

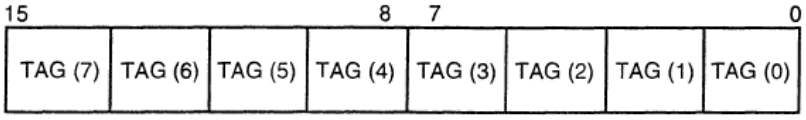


FIGURE 13–7 The 80X87 arithmetic coprocessor tag register

INSTRUCTION SET

The arithmetic coprocessor executes over 68 different instructions. Whenever a coprocessor instruction references memory, the microprocessor automatically generates the memory address for the instruction. The coprocessor uses the data bus for data transfers during coprocessor instructions, and the microprocessor uses it during normal instructions. Also note that the 80287 uses the Intel reserved I/O ports 00F8H–00FFH for communications between the coprocessor and the microprocessor (even though the coprocessor only uses ports 00FCH–00FFH). These ports are used mainly for the FSTSW AX instruction. The 80387–Pentium Pro use I/O ports 800000F8H–800000FFH for this communications.

Data Transfer Instructions

There are three basic data transfers: floating-point, signed-integer, and BCD. The only time that data ever appear in the signed-integer or BCD form is in the memory. Inside the coprocessor, data are always stored as an 80-bit extended-precision floating-point number.

Arithmetic Instructions

Arithmetic instructions for the coprocessor include addition, subtraction, multiplication, division, and square root. The arithmetic-related instructions are scaling, rounding, absolute value, and changing the sign.

The classic stack form of addressing operand data (stack addressing) uses the top of the stack as the source operand and the next to the top of the stack as the destination operand. Afterwards, a pop removes the source datum from the stack; only the result in the destination register remains at the top of the stack. To use this addressing mode, the instruction is placed in the program without any operands such as FADD or FSUB. The FADD instruction adds ST to ST(1) and stores the answer at the top of the stack; it also removes the original two datum from the stack by popping. Note carefully that FSUB subtracts ST from ST(1) and leaves the difference at ST. Therefore, a reverse subtraction (FSUBR) subtracts ST(1) from ST and leaves the difference at ST. (Note that an error exists in Intel documentation, including the Pentium data book, that describes the operation of some reverse instructions.) Another use for reverse operations is finding a reciprocal (1/X). This is accomplished, if X is at the top of the stack, by loading a 1.0 to ST (FLD1), followed by the FDIVR instruction. The FDIVR instruction divides ST(1) into ST, or X into 1, and leaves the reciprocal (1/X) at ST.

The register-addressing mode uses ST for the top of the stack and ST(n) for another location where n is the register number. With this form, one operand must be ST and the other is ST(n). Note that to double the top of the stack, the FADD ST,ST(0) instruction is used where ST(0) also addresses the top of the stack. One of the two operands in the register-addressing

mode must be ST, while the other must be in the form ST(n), where n is a stack register 0–7. For many instructions, either ST or ST(n) can be the destination. It is fairly important that the top of the stack be ST(0). This is accomplished by resetting or initializing the coprocessor before using it in a program. Another example of register-addressing is FADD ST(1),ST where the contents of ST are added to ST(1) and the result is placed into ST(1).

The top of the stack is always used as the destination for the memory-addressing mode because the coprocessor is a stack-oriented machine. For example, the FADD DATA instruction adds the real-number contents of memory location DATA to the top of the stack.

MEMORY DEVICES

Before attempting to interface memory to the microprocessor, it is essential to completely understand the operation of memory components. In this section, we explain the function of the four common types of memory: **read-only memory** (ROM), **flash memory** (EEPROM), **static random access memory** (SRAM), and **dynamic random access memory** (DRAM).

Memory Pin Connections

Pin connections common to all memory devices are the address inputs, data outputs or input/outputs, some type of selection input, and at least one control input used to select a read or write operation. See Figure 9–1 for ROM and RAM generic-memory devices.

Address Connections. All memory devices have address inputs that select a memory location within the memory device. Address inputs are almost always labeled from A_0, the least significant address input, to A_n, where subscript $_n$ can be any value but is always labeled as one less than the total number of address pins. For example, a memory device that has 10 address pins has its address pins labeled from A_0 to A_9. The number of address pins found on a memory device is determined by the number of memory locations found within it.

Today, the more common memory devices have between 1K (1,024) to 16M (16,777,216) memory locations, with 256M memory location devices on the horizon. A 1K memory device has 10 address pins (A_0–A_9); therefore, 10 address inputs are required to select any of its 1,024 memory locations. It takes a 10-bit binary number (1,024 different combinations) to select any single location on a 1,024-location device. If a memory device has 11 address connections (A_0–A_{10}), it has 2,048 (2K) internal memory locations. The number of memory locations can thus be extrapolated from the number of address pins. For example, a 4K memory device has 12 address connections, an 8K device has 13, and so forth. A device that contains 1M locations requires a 20-bit address (A_0–A_{19}).

Data Connections. All memory devices have a set of data outputs or input/outputs. The device illustrated in Figure 9–1 has a common set of input/output (I/O) connections. Today, many memory devices have bi-directional common I/O pins.

The data connections are the points at which data are entered for storage or extracted for reading. Data pins on memory devices are almost always labeled D_0 through D_7 for an 8-bit-wide

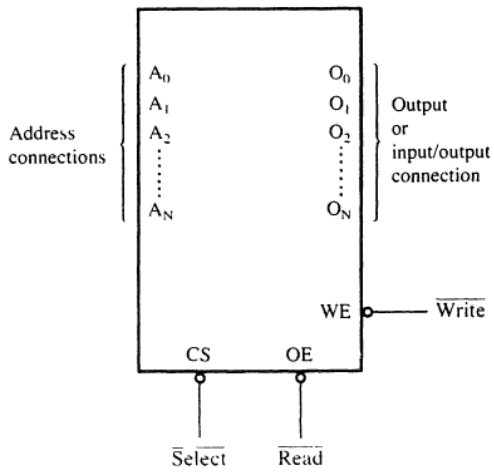


FIGURE 9–1 A pseudo-memory component illustrating the address, data, and control connections

memory device. In this sample memory device, there are eight I/O connections, which means that the memory device stores 8-bits of data in each of its memory locations. An 8-bit-wide memory device is often called a **byte-wide** memory. Although most devices are currently 8-bits wide, not all memory devices are 8-bits wide. Some devices are 16-bits, 4-bits, or just 1-bit wide.

Selection Connections. Each memory device has an input—sometimes more than one—that selects or enables the memory device. This kind of input is most often called a **chip select** ($\overline{CS}$), **chip enable** ($\overline{CE}$), or simply **select** ($\overline{S}$) input. RAM memory generally has at least one $\overline{CS}$ or $\overline{S}$ input, and ROM at least one $\overline{CE}$. If the $\overline{CE}$, $\overline{CS}$, or $\overline{S}$ input is active (a logic 0 in this case, becaus of the over-bar), the memory device performs a read or a write; if it is inactive (a logic 1 in this case), the memory device cannot do a read or a write because it is turned off or disabled. If more than one $\overline{CS}$ connection is present, all must be activated to read or write data.

Control Connections. All memory devices have some form of control input or inputs. A ROM usually has only one control input, while a RAM often has one or two control inputs.

The control input most often found on a ROM is the **output enable** ($\overline{OE}$) or **gate** ($\overline{G}$) connection, which allows data to flow out of the output data pins of the ROM. If $\overline{OE}$ and the selection input are both active, then the output is enabled; if $\overline{OE}$ is inactive, the output is disabled at its high-impedance state. The $\overline{OE}$ connection enables and disables a set of three-state buffers located within the memory device and must be active to read data.

A RAM memory device has either one or two control inputs. If there is one control input, it is often called R/$\overline{W}$. This pin selects a read operation or a write operation only if the device is selected by the selection input ($\overline{CS}$). If the RAM has two control inputs, they are usually labeled $\overline{WE}$ (or $\overline{W}$) and $\overline{OE}$ (or $\overline{G}$). Here, $\overline{WE}$ **(write enable)** must be active to perform a memory write operation, and OE must be active to perform a memory read operation. When these two controls ($\overline{WE}$) and $\overline{OE}$) are present, they must never both be active at the same time. If both control inputs are inactive (logic 1's), then data are neither written nor read and the data connections are at their high-impedance state.

ROM Memory

The read-only memory (ROM) permanently stores programs and data that are resident to the system and must not change when power is disconnected. The ROM is permanently programmed so data are always present, even when power is disconnected. This type of memory is often called *nonvolatile memory.*

The ROM is available in many forms today. A device we call a ROM is purchased in mass quantities from a manufacturer and programmed during its fabrication at the factory. The EPROM **(erasable programmable read-only memory),** a type of ROM, is more commonly used when software must be changed often or when too limited a number are in demand to make the ROM economical. For a ROM to be practical, we usually must purchase at least 10,000 devices. An EPROM is programmed in the field on a device called an *EPROM programmer.* The EPROM is also erasable if exposed to high-intensity ultraviolet light for about 20 minutes or less, depending on the type of EPROM.

PROM memory devices are also available, but they are not as common today. The PROM **(programmable read-only memory)** is also programmed in the field by burning open tiny Nichrome or silicon oxide fuses, but once programmed it cannot be erased.

A.C. Characteristics

T_A = 0°C to 70°C, V_{CC}[1] = +5V ±5%, V_{PP}[2] = V_{CC} ±0.6V[3]

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.[4]	Max.		
t_{ACC1}	Address to Output Delay		250	450	ns	PD/PGM = $\overline{CS}$ = V_{IL}
t_{ACC2}	PD/PGM to Output Delay		280	450	ns	$\overline{CS}$ = V_{IL}
t_{CO}	Chip Select to Output Delay			120	ns	PD/PGM = V_{IL}
t_{PF}	PD/PGM to Output Float	0		100	ns	$\overline{CS}$ = V_{IL}
t_{DF}	Chip Deselect to Output Float	0		100	ns	PD/PGM = V_{IL}
t_{OH}	Address to Output Hold	0			ns	PD/PGM = $\overline{CS}$ = V_{IL}

Capacitance[5] T_A = 25°C, f = 1 MHz

Symbol	Parameter	Typ.	Max.	Unit	Conditions
C_{IN}	Input Capacitance	4	6	pF	V_{IN} = 0V
C_{OUT}	Output Capacitance	8	12	pF	V_{OUT} = 0V

A.C. Test Conditions:

Output Load: 1 TTL gate and C_L = 100 pF
Input Rise and Fall Times: ≤20 ns
Input Pulse Levels: 0.8V to 2.2V
Timing Measurement Reference Level:
Inputs 1V and 2V
Outputs 0.8V and 2V

WAVEFORMS

A. Read Mode
PD/PGM = V_{IL}

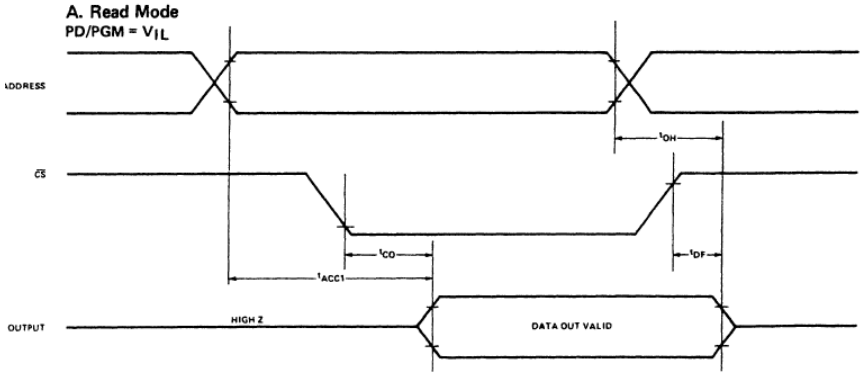


FIGURE 9–3 The timing diagram of AC characteristics of the 2716 EPROM (Courtesy of Intel Corporation)

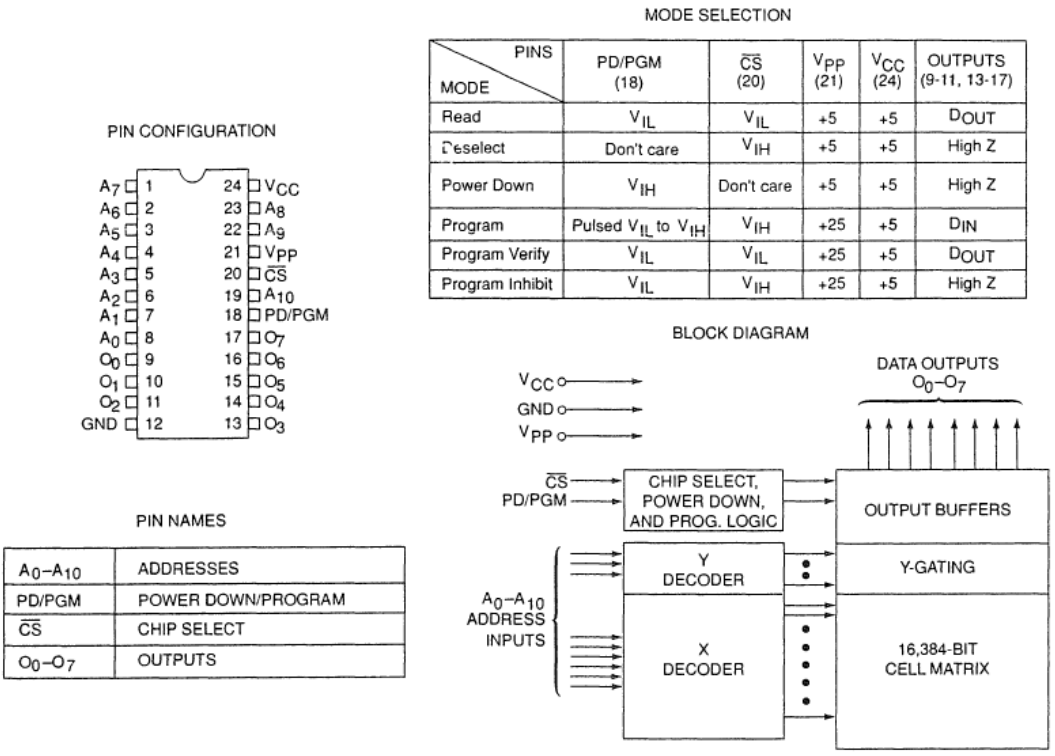


FIGURE 9–2 The pin-out of the 2716, 2K × 8 EPROM (Courtesy of Intel Corporation)

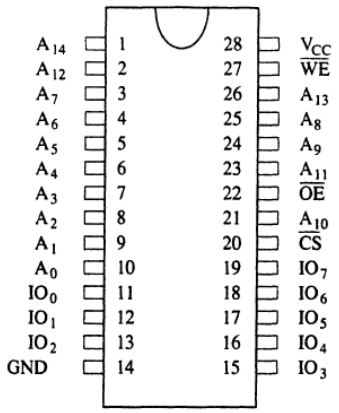
Figure 9–3 illustrates the timing diagram for the 2716 EPROM. Data only appear on the output connections after a logic 0 is placed on both the $\overline{CE}$ and $\overline{OE}$ pin connections. If $\overline{CE}$ and $\overline{OE}$ are not both logic 0's, the data output connections remain at their high-impedance or off states. Note that the V_{PP} pin must be placed at a logic 1 for data to be read from the EPROM. In some cases, the V_{PP} pin is in the same position as the $\overline{WE}$ pin on the SRAM. This can allow a single socket to hold either an EPROM or an SRAM. Examples are the 2716 EPROM and the 6116 SRAM, both 2K × 8 devices that have the same pin-out, except for V_{PP} on the EPROM and $\overline{WE}$ on the SRAM.

One important piece of information provided by the timing diagram and data sheet is the memory access time—the time that it takes the memory to read information. As Figure 9–3 illustrates, memory access time (T_{ACC}) is measured from the appearance of the address at the address inputs until the appearance of the data at the output connections. This is based on the assumption that the $\overline{CE}$ input goes low at the same time that the address inputs become stable. Also, $\overline{OE}$ must be a logic 0 for the output connections to become active. The basic speed of th EPROM is 450 ns. (Recall from Chapter 7 that the 8086/8088 operated with a 5 MHz clock allowed memory 460 ns to access data.) This type of memory component requires wait states to operate properly with the 8086/8088 microprocessors because of its rather long access time. If wait states are not desired, higher speed versions of the EPROM are available at an additional cost. Today, EPROM memory is available with access times of as little as 100 ns.

Static Ram (SRAM) Devices

Static RAM memory devices retain data for as long as DC power is applied. Because no special action (except power) is required to retain stored data, these devices are called *static memory*. They are also called *volatile memory* because they will not retain data without power. The main difference between a ROM and a RAM is that a RAM is written under normal operation, while a ROM is programmed outside the computer and is only normally read. The SRAM stores temporary data

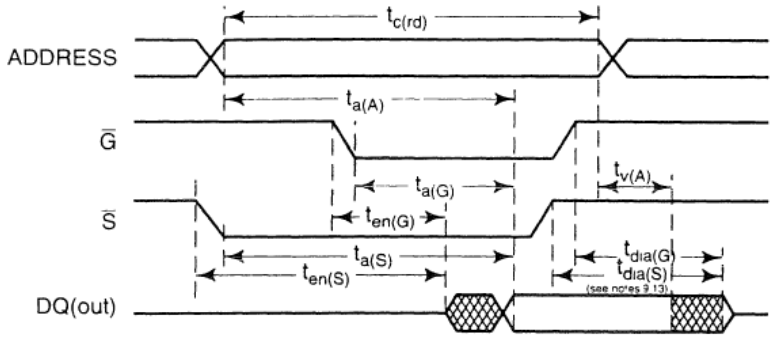
FIGURE 9–6 Pin diagram of the 62256, 32K × 8 static RAM



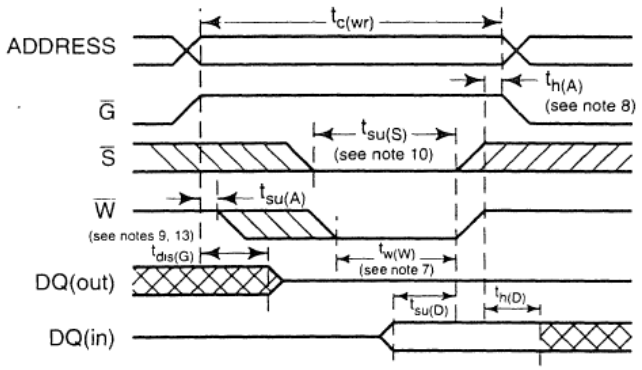
PIN FUNCTION	
A_0 - A_{14}	Addresses
IO_0 - IO_7	Data connections
$\overline{CS}$	Chip select
$\overline{OE}$	Output enable
$\overline{WE}$	Write enable
V_{CC}	+5V Supply
GND	Ground

NOTES. 5. $\overline{W}$ is high Read Cycle.
6. $\overline{W}$ must be high during all address transitions.
7. A write occurs during the overlap of a low $\overline{S}$ and a low $\overline{W}$.
8. $t_{h(A)}$ is measured from the earlier of $\overline{S}$ or $\overline{W}$ going high to the end of the write cycle.
9. During this period, I/O pins are in the output state so that the input signals of opposite phase to the outputs must not be applied.
10 If the Slow transition occurs simultaneously with the $\overline{W}$ low transitions or after the $\overline{W}$ transition, output remains in a high impedance state
11. G is continuously low (G = V_{IL}).
12. If $\overline{S}$ is low during this period, I/O pins are in the output state. Data input signals of opposite phase to the outputs must not be applied.
13. Transition is measured ± 200 mV from steady-state voltage.
14 If the $\overline{S}$ low transition occurs before the W low transition, then the data input signals of opposite phase to the outputs must not be applied for the duration of $t_{dis(W)}$ after the $\overline{W}$ low transition.

timing waveform of read cycle (see note 5)



timing waveform of write cycle no. 1 (see note 6)



Dynamic Ram (DRAM) Memory

About the largest static RAM available today is a 128K × 8. Dynamic RAMs, on the other hand, are available in much lager sizes: up to 16M × 1. In all other respects, DRAM is essentially the same as SRAM, except that it retains data for only 2 or 4 ms on an integrated capacitor. After 2 or 4 ms, the contents of the DRAM must be completely rewritten *(refreshed)* because the capacitors, which store a logic 1 or logic 0, lose their charges.

Instead of requiring the almost impossible task of reading the contents of each memory location with a program and then rewriting them, the manufacturer has internally constructed the DRAM so that, in the 64K × 1 version, the entire contents of the memory is refreshed with 256 reads in a 4 ms interval. Refreshing also occurs during a write, a read, or during a special refresh cycle. Much more information on refreshing DRAMs is provided in Section 9–6.

Another disadvantage of DRAM memory is that it requires so many address pins that the manufacturers have multiplexed the address inputs. Figure 9–7 illustrates a 64K × 4 DRAM, the TMS4464, which stores 256K bits of data. Notice that it contains only 8 address inputs where it should contain 16—the number required to address 64K memory locations. The only way that 16 address bits can be forced into 8 address pins is in two 8-bit increments. This operation requires two special pins called **column address strobe** ($\overline{CAS}$) and **row address strobe** ($\overline{RAS}$). First, A_0–A_7 are placed on the address pins and strobed into an internal row latch by $\overline{RAS}$ as the row address. Next, the address bits A_8–A_{15} are placed on the same eight address inputs and strobed into an internal column latch by $\overline{CAS}$ as the column address (see Figure 9–8 for this timing). The 16-bit address held in these internal latches addresses the contents of one of the 4-bit memory locations. Note that $\overline{CAS}$ also performs the function of the chip selection input to the DRAM.

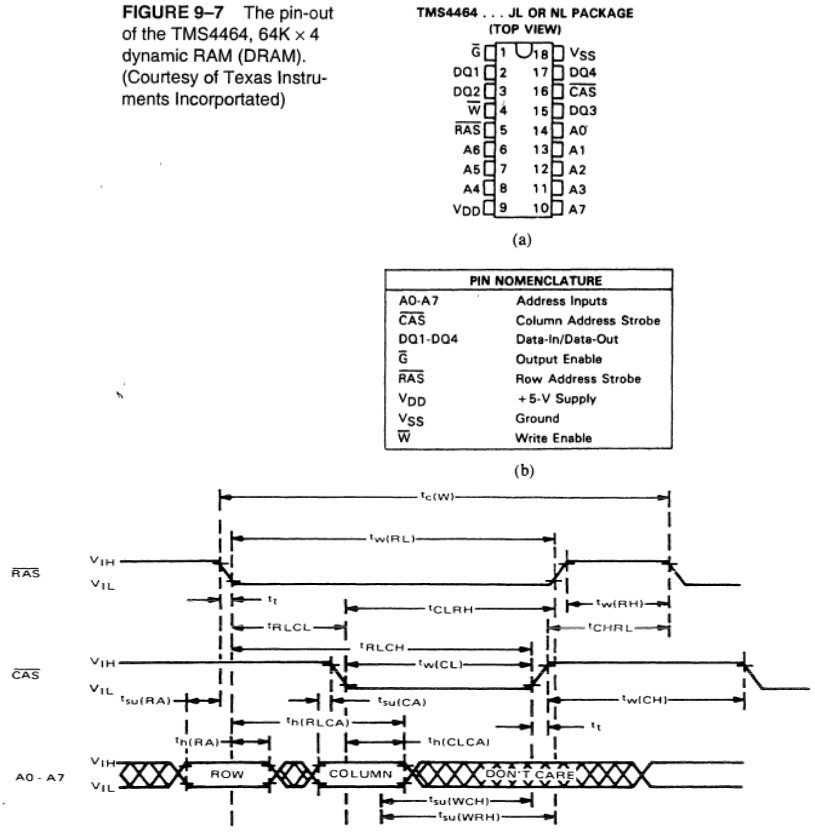


FIGURE 9–7 The pin-out of the TMS4464, 64K × 4 dynamic RAM (DRAM). (Courtesy of Texas Instruments Incorporated)

FIGURE 9–8 RAS, CAS, and address input timing for the TMS4464 DRAM (Courtesy of Texas Instruments Incorporated)

ADDRESS DECODING

In order to attach a memory device to the microprocessor, it is necessary to decode the address from the microprocessor to make the memory function at a unique section or partition of the memory map. Without an address decoder, only one memory device can be connected to a microprocessor, which would make it virtually useless. In this section, we describe a few of the more common address-decoding techniques, as well as the decoders that are found in many systems.

Why Decode Memory?

When the 8088 microprocessor is compared to the 2716 EPROM, a difference in the number of address connections surfaces—the EPROM has 11 address connections and the microprocessor has 20. This means that the microprocessor sends out a 20-bit memory address whenever it reads or writes data. Since the EPROM has only 11 address inputs, there is a mismatch that must somehow be corrected. If only 11 of the 8088's address pins are connected to the memory, then the 8088 will see only 2K bytes of memory instead of the 1M byte that it "expects" the memory to contain. The decoder corrects the mismatch by decoding the address pins that do not connect to the memory component.

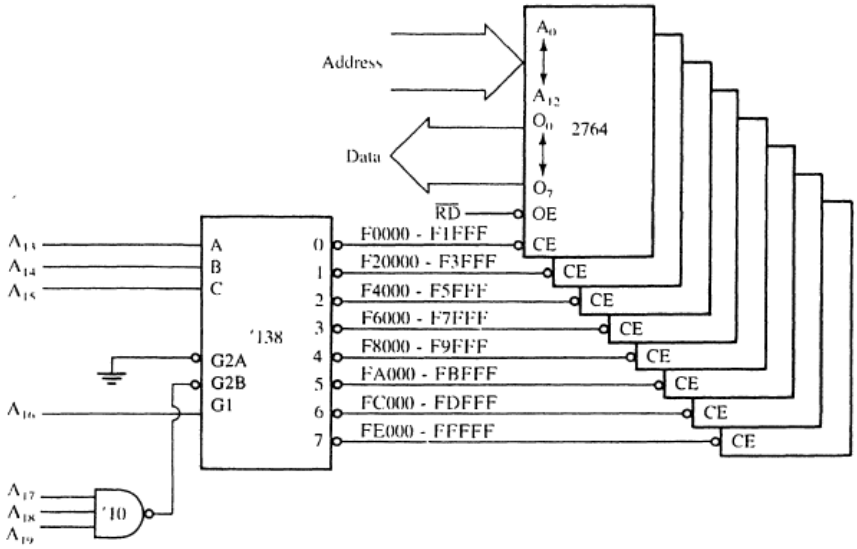


FIGURE 9–14 A circuit that uses eight 2764 EPROMs for a 64K × 8 section of memory in an 8088 microprocessor-based system. The addresses selected in this circuit are F0000H–FFFFFH.

PLD programmable decoders

Recently, the PAL has replaced PROM address decoders in the latest memory interfaces. There are three PLD devices that function in basically the same manner, but have different names: PLA **(programmable logic array),** PAL **(programmable array logic),** and GAL **(gated array logic).** Although these devices have been in existence since the mid-1970s, they have only recently appeared in memory systems and digital designs. The PAL and the PLA are fuse programmed, as is the PROM, and some PLD devices are erasable, as are EPROMs. In essence, all three devices are arrays of logic elements that are programmable.

Combinatorial Programmable Logic Arrays. One of the two basic types of PALs is the combinatorial programmable logic array. This device is internally structured as a programmable array of combinational logic circuits.