TABLE 17-2 Power supply voltages that must be applied to Vcc as requested by the VID pins.

VID4	*VID3*	*VID2*	*VID1*	*VID0*	*Vcc*
0	0	0	0	0	2.05 V
0	0	0	0	1	2.00 V
0	0	0	1	0	1.95 V
0	0	0	1	1	1.90 V
0	0	1	0	0	1.85 V
0	0	1	0	1	1.80 V
0	0	1	1	0	—
0	0	1	1	1	—
0	1	0	0	0	—
0	1	0	0	1	—
0	1	0	1	0	—
0	1	0	1	1	—
0	1	1	0	0	—
0	1	1	0	1	—
0	1	1	1	0	—
0	1	1	1	1	—
1	0	0	0	0	3.5 V
1	0	0	0	1	3.4 V
1	0	0	1	0	3.3 V
1	0	0	1	1	3.2 V
1	0	1	0	0	3.1 V
1	0	1	0	1	3.0 V
1	0	1	1	0	2.9 V
1	0	1	1	1	2.8 V
1	1	0	0	0	2.7 V
1	1	0	0	1	2.6 V
1	1	0	1	0	2.5 V
1	1	0	1	1	2.4 V
1	1	1	0	0	2.3 V
1	1	1	0	1	2.2 V
1	1	1	1	0	2.1 V
1	1	1	1	1	—

THERMTRIP	**Thermal sensor trip** is an output that that becomes a zero when the temperature of the Pentium II exceeds 130°C.
TMS	The **test mode select** input controls the operation of the Pentium in test mode.
TRDY	**Target ready** is an input that is used to cause the Pentium II to perform a write-back operation.
VID4–VID0	**Voltage data** output pins are either open or grounded signals that indicate what supply voltage is currently required by the Pentium II. The power supply must apply the request voltage to the Pentium II, as listed in Table 17–2.

The Memory System

The memory system for the Pentium II microprocessor is 64G bytes in size, just like the Pentium Pro microprocessor. Both microprocessors address a memory system that is 64 bits wide with an address bus that is 36 bits wide. Most systems use SDRAM operating at 66 MHz or 100 MHz. The SDRAM for the 66 MHz system has an access time of 10 ns and the SDRAM for the 100 MHz system has an access time of 8 ns. The memory system, which connects to the chipset, is not illustrated in this chapter. Refer to Chapter 18 to see the organization of a 64-bit wide memory system without ECC.

The Pentium II memory system is divided into eight or nine banks that each store a byte of data. If the ninth byte is present, it stores an error checking code (ECC). The Pentium II, like the 80486–Pentium Pro, employs internal parity generation and checking logic for the memory system's data bus information. (Note that most Pentium II systems do not use parity checks, but it is available.) If parity checks are employed, each memory bank contains a ninth bit. The 64-bit wide memory is important to double-precision floating-point data. Recall that a double-precision floating-point number is 64 bits wide. As with the Pentium Pro, the memory system is numbered in bytes from byte 000000000H to byte FFFFFFFFFH. Please note that none of the current chip sets support more than 1G bytes of system memory, so the additional address connections are for future expansion. Figure 17–3 illustrates the basic memory map of the Pentium II system, using the AGP for the video card.

The memory map for the Pentium II system is similar to the map illustrated in earlier chapters, except that an area of the memory is used for the AGP area. The AGP area allows the video card and Windows to access the video information in a linear address space. This is unlike the 128K-byte window in the DOS area for a standard VGA video card. The benefit is much faster video updates because the video card does not need to page through the 128K-byte DOS video memory.

Transfers between the Pentium II and the memory system are controlled by the 440LX or 440 BX chipset. Data transfers between the Pentium II and the chipset are eight bytes wide. The chipset communicates to the microprocessor through the five REQ signals, as listed in Table 17–3. In essence, the chipset controls the Pentium II, which is a departure from the traditional method of connecting a microprocessor to the system directly to the memory.

The Pentium II connects only directly to the cache, which is on the Pentium II cartridge. As mentioned, the Pentium II cache operates at one-half the clock frequency of the microprocessor. Therefore, a 400 MHz Pentium

TABLE 17–3 The request (REQ) signals to the Pentium II.

REQ4–REQ0	*Name*	*Comment*
00000	Deferred reply	Deferred replies are issued for previously deferred transactions
00001	Reserved	Future use
00010	Memory read & invalidate	Memory read from DRAM or PCI write to DRAM from PCI
00011	Reserved	Future use
00100	Memory code read	Memory read
00101	Memory write	Memory write-back cycle
00110	Memory data read	Memory read
00111	Memory write	Normal memory write
01000	Interrupt acknowledge or special cycle	Interrupt acknowledge cycle for PCI bus
01001	Reserved	Future use
10000	I/O read	I/O read operation
10001	I/O write	I/O write operation
1100x	Reserved	Future use

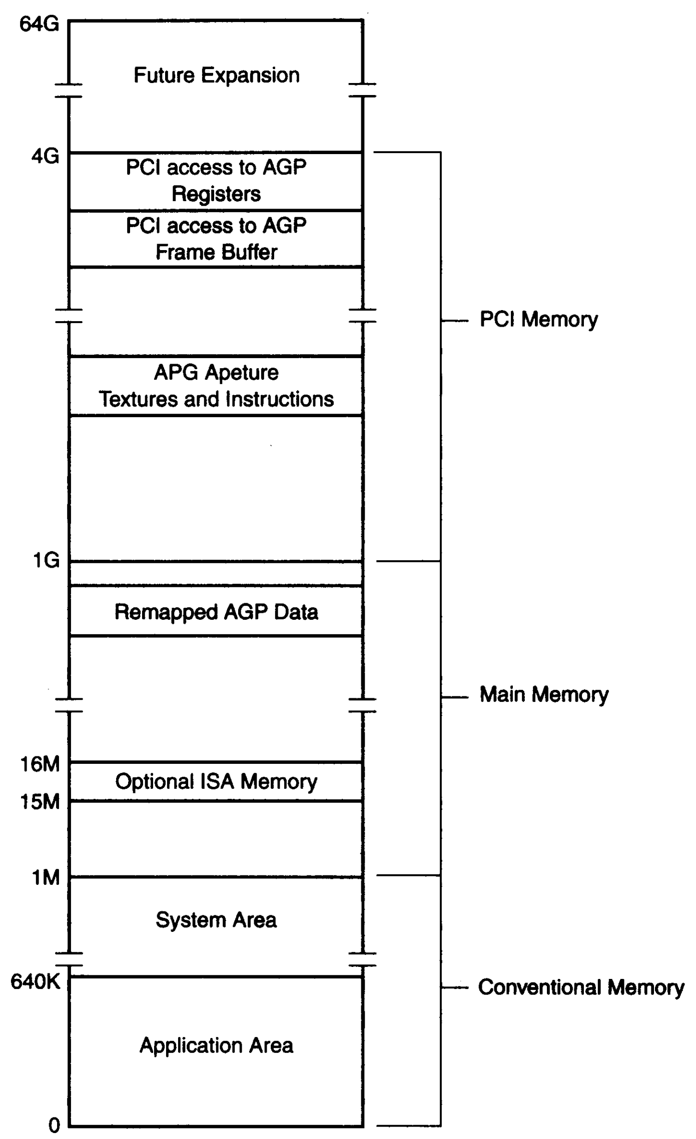


FIGURE 17–3 The memory map of a Pentium II-based computer system.

II cache operates at 200 MHz. The Pentium II Xeon cache operates at the same frequency as the microprocessor, which means that the Xeon, with its 512K, 1M, or 2M cache, outperforms the standard Pentium II.

Input/Output System

The input/output system of the Pentium II is completely compatible with earlier Intel microprocessors. The I/O port number appears on address lines A15–A3 with the bank-enable signals used to select the actual memory banks used for the I/O transfer. Transfers are controlled by the chipset, which is a departure from the standard microprocessor architecture before the Pentium II.

Beginning with the 80386 microprocessor, I/O privilege information is added to the TSS segment when the Pentium II is operated in the protected mode. Recall that this allows I/O ports to be selectively inhibited. If the blocked I/O location is accessed, the Pentium II generates a type 13 interrupt to signal an I/O privilege violation.

System Timing

As with any microprocessor, the system timing signals must be understood in order to interface the microprocessor, or so it was at one time. Because the Pentium II is designed to be controlled by the chipset, the timing signals between the microprocessor and chipset have become proprietary Intel information and are not released to the public.

17–2 PENTIUM II SOFTWARE CHANGES

The Pentium II microprocessor core is a Pentium Pro. This means that the Pentium II and the Pentium Pro are essentially the same device for software. This section of the text lists the changes to the CPUID instruction; and the SYSENTER, SYSEXIT, FXSAVE, and FXRSTORE instructions (the only modifications to the software).

CPUID Instruction

Table 17–4 lists the values passed between the Pentium II and the CPUID instruction. These are changed from earlier versions of the Pentium microprocessor.

The version information returned after executing the CPUID instruction with a logic 0 in EAX is returned in EAX. The family ID is returned in bits 8 to 11; the model ID is returned in bits 4 to 7. The stepping ID is returned in bits 0 to 3. For the Pentium II, the model number is 6 and the family ID is a 3. The stepping number refers to an update number. The higher the stepping number, the newer the version.

The features are indicated in the EDX register after executing the CPUID instruction with a zero in EAX. Only two new features are returned in EDX for the Pentium II. Bit position 11 indicates whether the microprocessor supports the two new fast call instructions SYSENTER and SYSEXIT. Bit position 23 indicates whether the microprocessor supports the MMX instruction set. The remaining bits are identical to earlier versions of the microprocessor and are not described. Bit 16 indicates whether the microprocessor supports the page attribute table or PAT. Bit 17 indicates whether the microprocessor supports the page size extension found with the Pentium Pro and Pentium II microprocessors. The page size extension allows memory above 4G through 64G to be addressed. Finally, bit 24 indicates whether the fast floating-point save and restore instructions are implemented.

TABLE 17–4 CPUID instruction.

Input EAX	*Output Register*	*Contents*
0	EAX	Maximum value for input to EAX for CPUID instruction
0	EBX	'uneG'
0	ECX	'Inei'
0	EDX	'letn'
1	EAX	Version number
1	EBX	—
1	ECX	—
1	EDX	Feature information
2	EAX	Cache data
2	EBX	Cache data
2	ECX	Cache data
2	EDX	Cache data

TABLE 17–5 The model- specific registers used with SYSENTER and SYSEXIT.

Name	*Number*	*Function*
SYSENTER_CS	174H	SYSENTER target code segment
SYSENTER_ESP	175H	SYSENTER target stack segment
SYSENTER_EIP	176H	SYSENTER target instruction pointer

SYSENTER and SYSEXIT Instructions

The SYSENTER and SYSEXIT instructions use the fast call facility introduced in the Pentium II microprocessor. Please note that these instructions function only in ring zero (privilege level 0) in protected mode. Windows operates in ring 0, but does not allow applications access to ring 0. These new instructions are meant for operating system software.

The SYSENTER instruction uses some of the model-specific registers to store CS, EIP, and ESP to execute a fast call to a procedure defined by the model-specific register. The fast call is different from a regular call because it does not push the return address onto the stack as a regular call. Table 17–5 illustrates the model-specific register used with SYSENTER and SYSEXIT. Note that the model-specific registers are read with the RDMSR instruction and written with the WRMSR instruction.

To use the RDMSR or WRMSR instructions, place the register number in the ECX register. If the WRMSR is used, place the new data for the register in EDS:EAX. For the SYSENTER instruction, you need use only the EAX register, but place a zero into EDX. If the RDMSR instruction is used, the data are returned in the EDX:EAX register pair. Example 17–1 illustrates a macro sequence that can be used to add the RDTSC, CPUID, RDMSR, WRMSR, SYSENTER, and SYSEXIT instructions to a program. Some of these macro definitions can also be obtained from Microsoft's Internet Web site as an update to the MASM assembler program.

EXAMPLE 17–1

```
;macro sequences to add RDTSC, CPUID, RDMSR, WRMSR, SYSENTER, and SYSEXIT
;
RDTSC     MACRO
          DB     0FH, 31H
          ENDM
;
CPUID     MACRO
          DB     0FH, 0A2H
          ENDM
;
RDMSR     MACRO
          DB     0FH, 32H
          ENDM
;
WRMSR     MACRO
          DB     0FH, 30H
          ENDM
;
SYSENTER  MACRO
          DB     0FH, 34H
          ENDM
;
SYSEXIT   MACRO
          DB     0FH, 35H
          ENDM
```

TABLE 17–6 Selectors addressed by the SYSENTER_CS select value.

SYSENTER_CS MSR	*Function*
SYSENTER_CS value	SYSENTER code segment selector
SYSENTER_CS value + 8	SYSENTER stack segment selector
SYSENTER_CS value + 16	SYSEXIT code segment selector
SYSENTER_CS value + 24	SYSEXIT stack segment selector

To use the SYSENTER instruction, you must first load the model-specific registers with the address of the system entrance point into the SYSENTER_CS and SYSENTER_EIP registers. This would normally be the address of the operating system such as Windows or Windows NT. Note that this instruction is meant as a system instruction to access code or software in ring 0. The stack segment register is loaded with the value placed into SYSENTER_CS plus 8. In other words, the selector pair addressed by SYSENTER_CS selector value are loaded into CS and SS. The value of the stack offset is loaded into SYSENTER_ESP.

The SYSEXIT instruction loads CS and SS with the selector pair addressed by SYSENTER_CS plus 16 and 24. Table 17–6 illustrates the selectors from the global selector table, as addressed by SYSENTER_CS. In addition to the code and stack segment selector and the memory segments that they represent, the SYSEXIT instruction passes the value in EDX to the EIP register and the value in ECX to the ESP register. The SYSEXIT instruction returns control back to application ring 3. As mentioned, these instructions appear to have been designed for quick entrance and return from the Windows or Windows NT operating systems on the personal computer.

To use SYSENTER and SYSEXIT, the SYSENTER instruction must pass the return address to the system. This is accomplished by loading the EDX register with the return offset and by placing the segment address in the global descriptor table at location SYSENTER_CS+16. The stack segment is transferred by loading the stack segment selector into SYSENTER_CS+24 and the ESP into the ECX.

FXSAVE and FXRSTOR Instructions

The last two new instructions added to the Pentium II microprocessor are the FXSAVE and FXRSTOR instructions, which are almost identical to the FSAVE and FRSTOR instructions. The main difference is that the FXSAVE instruction is designed to properly store the state of the MMX machine, while the FSAVE properly stores the state of the floating-point coprocessor. The FSAVE instruction stores the entire tag field, while the FXSAVE instruction only stores the valid bits of the tag field. The valid tag field is used to reconstruct the restore tag field when the FXRSTOR instruction executes. This means that if the MMX state of the machine is saved, use the FXSAVE instruction; if the floating-point state of the machine is saved, use the FSAVE instruction. For new applications, it is recommended that the FXSAVE and FXRSTOR instructions should be used to save the MMX state and floating-point state of the machine. Do not use the FSAVE and FRSTOR instructions in new applications. See Example 17–2 for macros that allow FXSAVE and FXRSTOR to be used in a program. This example assumes direct memory addressing, only using 32-bit offset addresses.

EXAMPLE 17–2

```
;FXSAVE and FXRSTOR macros
;
FXSAVE    MACRO  Addr
          DB     0FH, 0AEH, 6
          DD     offset Addr
          ENDM
;
FXRSTOR   MACRO  Addr
          DB     0FH, 0AEH, 0EH
```

```
        DD      offset Addr
        ENDM
```

17–3 THE PENTIUM III

The Pentium III microprocessor is an improved version of the Pentium II microprocessor. Even though it is newer than the Pentium II, it is still based on the Pentium Pro architecture.

There are two versions of the Pentium III. One version is available with a non-blocking 512K-byte cache and packaged in the slot 1 cartridge, and the other version is available with a 256K-byte advanced transfer cache and packaged in an integrated circuit. The slot 1-version cache runs at half the processor speed, and the integrated-cache version runs at the processor clock frequency. As shown in most benchmarks of cache performance, increasing the cache size from 256K bytes to 512K bytes only improves performance by a few percent.

Chip Sets

The chip set for the Pentium III is different from the Pentium II. The Pentium III uses an Intel 810, 815, or 820 chipset. The 815 is most commonly found in newer systems that use the Pentium III. A few other vendor chip sets are available, but problems with drivers for new peripherals, such as the video cards, have been reported. An 840 chip set also was developed for the Pentium III, but Intel does not make it available.

Bus

The Coppermine version of the Pentium III increases the bus speed to either 100 MHz or 133 MHz. The faster version allows transfers between the microprocessor and the memory at higher speeds.

Suppose that you have a 1-GHz microprocessor that uses a 133-MHz memory bus. You might think that the memory bus speed could be faster to improve performance, and we agree. However, the connections between the microprocessor and the memory preclude using a higher speed for the memory. If we decided to use a 200-MHz bus speed, we must recognize that a wavelength at 200 MHz is 300,000,000/200,000,000 or 3/2 meter. An antenna is 1/4 of a wavelength. At 200 MHz, an antenna is 14.8 inches. We do not want to radiate energy at 200 MHz, so we need to keep the printed circuit board connections shorter than 1/4-wavelength. In practice, we would keep the connections to no more than 1/10 of 1/4-wavelength. This means that the connections in a 200 MHz system should be no longer than 1.48 inches. This size would present the main board manufacturer with a problem when placing the sockets for a 200 MHz memory system.

Will it be possible to approach or even exceed the 200 MHz memory system? Yes, if we develop a new technology for interconnecting the microprocessor, chipset, and memory. At present the memory functions in bursts of four 64-bit numbers each time we read the main memory. This burst of 32 bytes is read into the cache. The main memory requires 3 wait states at 100 MHz to access the first 64-bit number and then zero wait states for each of the three remaining 64-bit wide numbers for a total of seven 100 MHz bus clocks. This means we are reading data at 70 ns / 32 = 2.1875 ns per byte, which is a bus speed of 457M bytes per second. This is slower than the clock on a 1GHz microprocessor, but because most programs are cyclic and the instructions are stored in an internal cache, we can and often do approach the operating frequency of the microprocessor.

Pin-out

Figure 17–4 shows the pin-out of the socket 370 version of the Pentium III microprocessor. This integrated circuit is packaged in a 370-pin, pin grid array (PGA) socket. It is designed to function with one of the chipsets available from Intel. In addition to the full version of the Pentium III, the Celeron, which uses a 66 MHz memory bus speed, is available. The Pentium III Xeon, also manufactured by Intel, allows larger cache sizes for server applications.

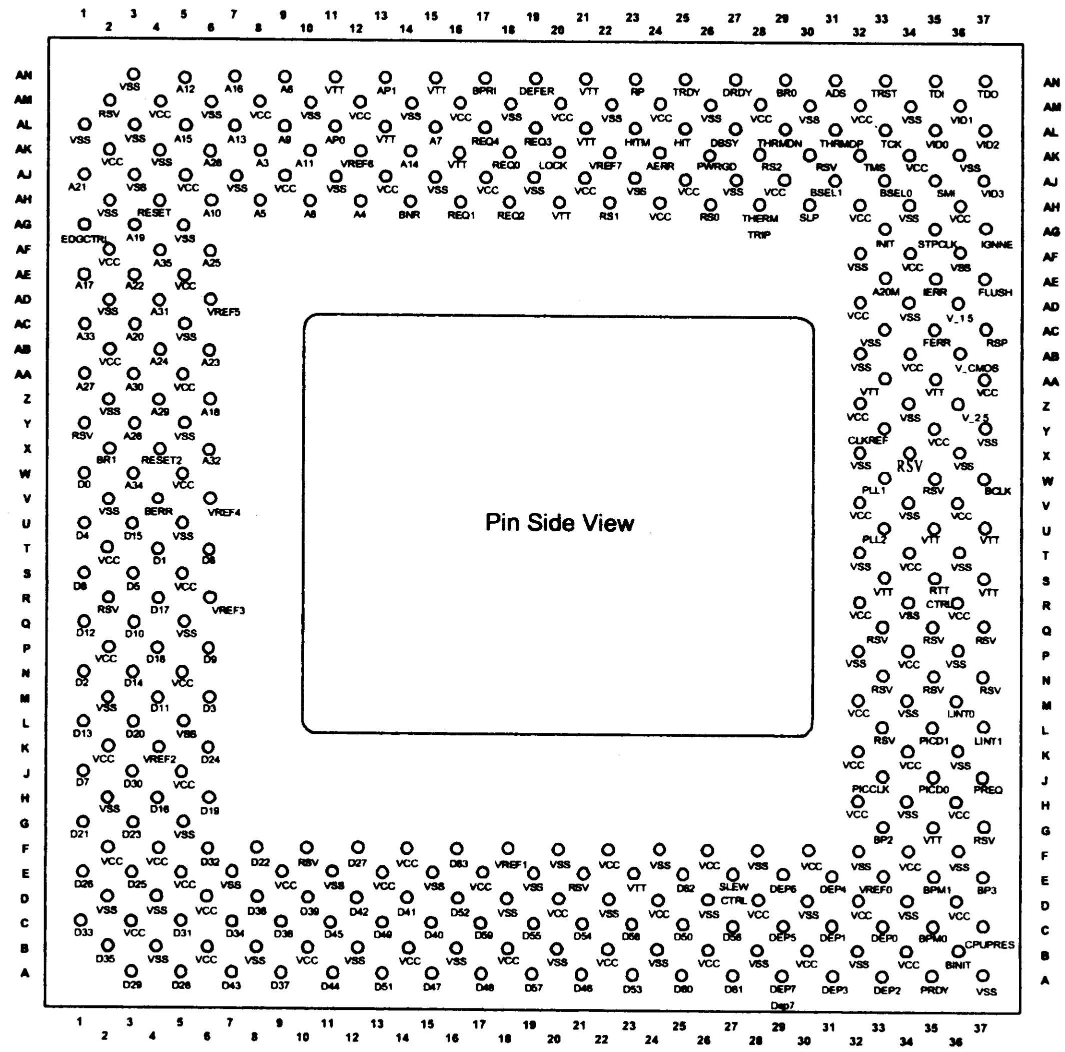


FIGURE 17–4 The pin-out of the socket 370 version of the Pentium III microprocessor. (Courtesy of Intel Corporation.)

17–4 THE PENTIUM 4

The most recent version of the Pentium Pro architecture microprocessor is the Pentium 4 microprocessor from Intel. The Pentium 4 was released initially in November 2000 with a speed of 1.3 GHz. It is currently available in speeds up to 2.0 GHz. There are two packages available for this integrated microprocessor, the 423-pin PGA and the 478-pin FC-PGA2. Both versions use the 1.8 micron technology for fabrication. As with earlier versions of the Pentium, the Pentium 4 uses a 100-MHz memory bus speed, but because it is quad pumped, the bus speed can approach 400 MHz. Figure 17–5 illustrates the pin-out of the 432-pin PGA of the Pentium 4 microprocessor.

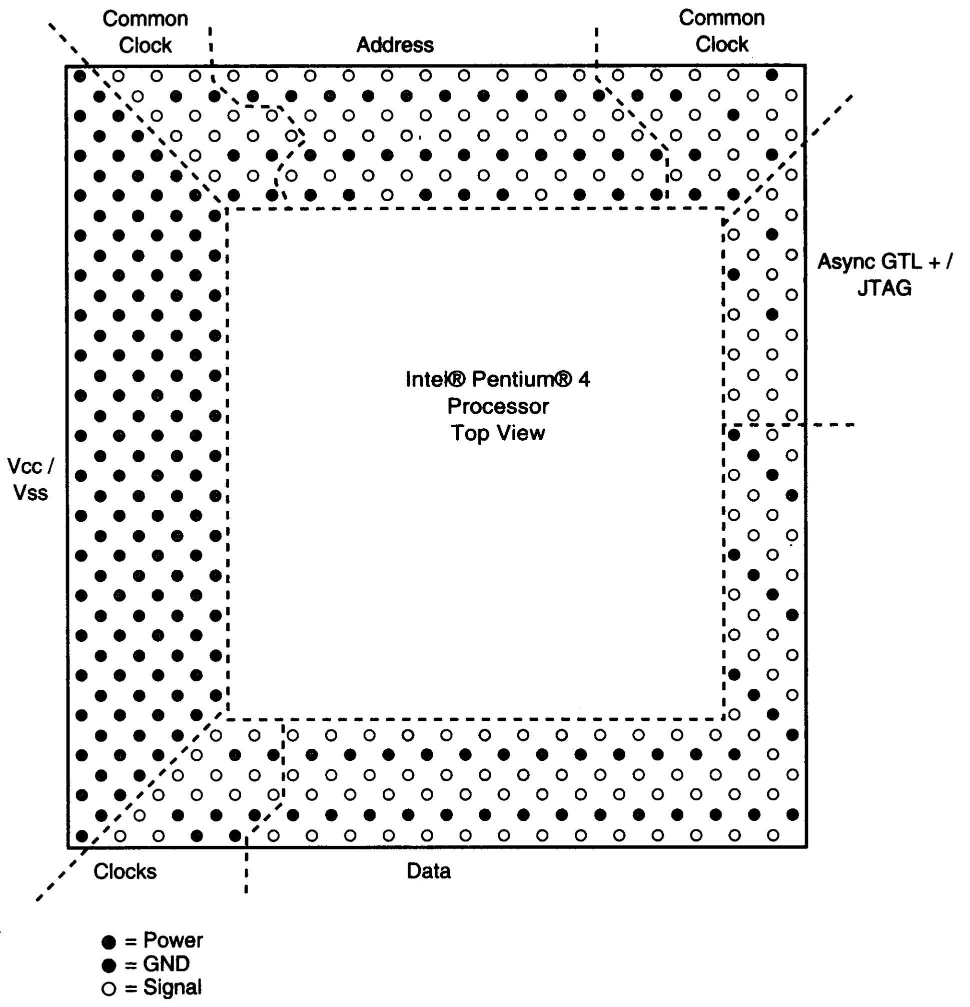


FIGURE 17–5 The pin-out of the Pentium 4, 423 PGA. (Courtesy of Intel Corporation.)

Memory Interface

The memory interface to the Pentium 4 typically uses the Intel 850 chipset. The 850 provides a dual-pipe memory bus to the microprocessor with each pipe interfaced to a 32-bit wide section of the memory. The two pipes function together to comprise the 64-bit wide data path to the microprocessor. Because of the dual pipe arrangement, the memory must be populated with pairs of RDRAM memory devices operating at either 600 MHz or 800 MHz. According to Intel this arrangement provides a 300% increase in speed over a memory populated with PC-100 memory.

Register Set

The Pentium 4 register set is nearly identical to all other versions of the Pentium except that the MMX registers are separate entities from the floating-point registers. In addition, eight 128-bit wide XMM registers are added for use with the SIMD (single instruction multiple data) instructions and the extended 128-bit packed doubled floating-point numbers.

You might think of the XMM registers as double wide MMX registers that can hold a pair of 64-bit double-precision floating-point numbers or four single-precision floating-point numbers. Likewise they can also hold 16 byte-wide numbers as the MMX registers hold 8 byte-wide numbers. The XMM registers are double width MMX registers.

If the new patch for MASM 6.14 is downloaded from Microsoft,[4] programs can be assembled using both the MMX and XMM instructions. To assemble programs that include MMX instructions, use the .MMX switch. For programs that include the SIMD instructions, use the .XMM switch. Example 17-3 illustrates a very simple program that uses the MMX instructions to add two eight-byte-wide numbers together. Notice how the .MMX switch is used to select the MMX instruction set. The MOVQ instructions transfer numbers between memory and the MMX registers. The MMX registers are numbered from MM0 to MM7. You can also use the MMX and SIMD instructions in Microsoft Visual C using the inline assembler if you download the latest patch from Microsoft for Visual Studio version 6.0.

EXAMPLE 17–3

```
                                  .MODEL TINY
                                  .MMX
0000                              .DATA

0000                              DATA1 DQ 1ffh
     00000000000001FF
0008                              DATA2 DQ 101h
     0000000000000101
0010                              DATA3 DQ ?
     0000000000000000
0000                              .CODE
                                  .STARTUP
0100 9B 0F 6F 06 0000 R              MOVQ  MM0,DATA1
0106 9B 0F 6F 0E 0008 R              MOVQ  MM1,DATA2
010C 9B 0F FC C1                     PADDB MM0,MM1
0110 9B 0F 7F 06 0010 R              MOVQ  DATA3,MM0

                                  .EXIT
                                  END
```

Similarly, the XMM software can be used in a program with the .XMM switch. Most modern programs use the XXM registers and the XXM instruction set to accomplish multimedia and other high speed operations. Example 17–4 shows a short program that illustrates the use of a few XMM instructions. This program multiplies two sets of four single-precision numbers and stores the four products into the four double words at ANS. In order to enable access to octal words (128-byte wide numbers) we use the OWORD PTR directive. Also notice that the FLAT model is used with the C profile. Since the SIMD instructions only function in protected mode (WIN32 model) we define the program in the FLAT model format. This means that the .686 and .XMM switches must precede the MODEL directive.

EXAMPLE 17–4

```
                              .686
                              .XMM
                              .MODEL FLAT,C
00000000                      .DATA

00000000 3F800000             DATA1 DD 1.0
00000004 40000000                   DD 2.0
00000008 40400000                   DD 3.0
```

[4] The update to any version of MASM 6.XX can be downloaded from Microsoft at www.microsoft.com.

```
0000000C 40800000                DD 4.0
00000010 40C9999A          DATA2 DD 6.3
00000014 40933333                DD 4.6
00000018 40900000                DD 4.5
0000001C C0133333                DD -2.3
00000020 00000004 [        ANS   DD 4 DUP(?)
      00000000
     ]
00000000                   .CODE
00000000 0F 28 05                MOVAPS   XMM0,OWORD PTR DATA1
      00000000 R
00000007 0F 28 0D                MOVAPS   XMM1,OWORD PTR DATA2
      00000010 R
0000000E 0F 59 C1                MULPS    XMM0,XMM1
00000011 0F 29 05                MOVAPS   OWORD PTR ANS, XMM0
      00000020 R
                           END
```

Hyper Pipelined Technology

The Pentium 4 incorporates a deeper pipelined architecture than prior versions of the Pentium microprocessor. Not only does it queue instructions for execution, but it also queues microinstruction for execution in a special cache for the microprocessor core. This special microinstruction cache is 12K bytes deep. This technology excludes the execution unit from the main cache path to the microinstruction stream to increase performance.

CPUID

Like earlier versions of the Pentium, the CPUID instruction returns the standard vendor ID information if executed with a zero in EAX. The most significant part of the CPU serial number is returned if the EAX register is a 1 before execution of the CPUID instruction. The middle and least significant parts of serial number are returned in the EDX and ECX, middle and least respectively, after a second execution of the CPUID instruction with a 3 in EAX. The CPUID instruction is displayed as a hexadecimal value as XXXX-XXXX-XXXX-XXXX-XXXX-XXXX. Example 17–5 shows a sample of code that extracts the serial number from the microprocessor and stores it in three double words of memory. This functions in both the real mode and protected modes of operation.

EXAMPLE 17–5

```
               .MODEL SMALL
               .686
0000           .DATA
0000 00000000        MOST  DD ?
0004 00000000        MID   DD ?
0008 00000000        LEAST DD ?
0000           .CODE
               .STARTUP    ;read CPU serial number
0010 66| B8 00000001 MOV   EAX,1
0016 0F A2           CPUID
0018 66| A3 0000 R   MOV   MOST,EAX
001C 66| B8 00000003 MOV   EAX,3
0022 0F A2           CPUID
0024 66| 89 16 0004 R MOV  MID,EDX
0029 66| 89 0E 0008 R MOV  LEAST,ECX
               .EXIT
               END
```

Pentium 4 Mechanicals

The Pentium 4 microprocessor uses the ATX architecture, but there are some changes that should be noted. The power supply for the Pentium 4 is different from other ATX power supplies. The Pentium 4 power supply contains the standard ATX connector, a 12 V connector, and an auxiliary connector that looks similar to the AT power supply connector. All three connectors must be plugged into the Pentium 4 main board for proper operation.

Another change to the Pentium 4 system is the case. The case for the Pentium 4 main board must have four additional standoffs to support the microprocessor. Without the additional standoffs you cannot use the Pentium 4 main board because without them the heat-sink for the microprocessor cannot be attached to the main board.

The power supply for the microprocessor should also be at least 300 W to handle the additional 60 to 70 W of power required by the microprocessor. This microprocessor runs a bit hotter than prior versions of the Pentium. The system usually reports a temperature of 120°F, an increase of 25° over the normal Pentium III temperature.

17–5 SUMMARY

1. The Pentium II differs from earlier microprocessors because instead of being offered as an integrated circuit, the Pentium II is available on a plug-in cartridge or printed circuit board.
2. The level 2 cache for the Pentium II is mounted inside of the cartridge, except for the Celeron, which has no level 2 cache. The cache speed is one-half the Pentium II clock speed, except in the Xeon, where it is at the same speed as the Pentium II. All versions of the Pentium II contain an internal level 1 cache that stores 32K bytes of data.
3. The Pentium II is the first Intel microprocessor that is controlled from an external bus controller. Unlike earlier versions of the microprocessor, which issued read and write signals, the Pentium II is ordered to read or write information by an external bus controller.
4. The Pentium II operates at clock frequencies from 233 MHz to 450 MHz with bus speeds of 66 MHz or 100 MHz. The level 2 cache can be 512K-, 1M-, or 2M-bytes in size. The Pentium II contains a 64-bit data bus and a 36-bit address bus that allow up to 64G bytes of memory to be accessed.
5. The new instructions added to the Pentium II are SYSENTER, SYSEXIT, FXSAVE, and FXRSTOR.
6. The SYSENTER and SYSEXIT commands are optimized to access the operating system in privilege level 0 from a privilege level 3 access. These instructions operate at a much higher speed than a task switch or even a call and return combination.
7. The FXSAVE and FXRSTOR instructions are optimized to properly store the state of both the MMX technology unit and the floating-point coprocessor.
8. The Pentium III microprocessor is an extension of the Pentium Pro architecture with the addition of the SIMD instruction set that uses the XXM registers.
9. The Pentium 4 microprocessor is an extension of the Pentium Pro architecture, which includes enhancements that allow it to operate at higher clock frequencies than previously possible because of the 1.8 micron fabrication technology.
10. The Pentium 4 microprocessor requires a modified ATX power supply and case to function properly in a system.
11. Version 6.14 of the MASM program and Visual Studio version 6 now support the new MMX and SIMD instructions using the .686 switch with the .MMX and .XXM switches.

17–6 QUESTIONS AND PROBLEMS

1. What is the size of the level 1 cache in the Pentium II microprocessor?
2. What sizes are available for the level 2 cache in the Pentium II microprocessor? (List all versions.)
3. What is the difference between the level 2 cache on the Pentium-based system and the Pentium II-based system?

4. What is the difference between the level 2 cache in the Pentium Pro and the Pentium II?
5. The speed of the Pentium II Xeon level 2 cache is ______ times faster than the cache in the Pentium II (excluding the Celeron).
6. How much memory can be addressed by the Pentium II?
7. Is the Pentium II available in integrated circuit form?
8. How many pin connections are found on the Pentium II cartridge?
9. What is the purpose of the PID control signals?
10. What happened to the read and write pins on the Pentium II?
11. At what bus speeds does the Pentium II operate?
12. How fast is the SDRAM connected to the Pentium II system for a 100 MHz bus speed version?
13. How wide is the Pentium II memory if ECC is employed?
14. What new model-specific registers (MSR) have been added to the Pentium II microprocessor?
15. What new CPUID identification information has been added to the Pentium II microprocessor?
16. How is a model-specific register addressed and what instruction is used to read it?
17. Write software that stores a 12H into model-specific register 175H.
18. Write a short procedure that determines whether the microprocessor contains the SYSENTER and SYSEXIT instructions. Your procedure must return carry set if the instructions are present, and return carry cleared if not present.
19. How is the return address transferred to the system when using the SYSENTER instruction?
20. How is the return address retrieved when using the SYSEXIT instruction to return to the application?
21. The SYSENTER instruction transfers control to software at what privilege level?
22. The SYSEXIT instruction transfers control to software at what privilege level?
23. What is the difference between the FSAVE and the FXSAVE instructions?
24. The Pentium III is an extension of the ________________architecture.
25. What new instructions appear in the Pentium III microprocessor that do not appear in the Pentium Pro microprocessor?
26. What changes to the power supply does the Pentium 4 microprocessor require?
27. Write a short program that reads and displays the serial number of the PIII microprocessor on the video screen.
28. Develop a procedure that multiplies 256 double-precision floating-point numbers and stores the products in a memory array called ANSWER.

APPENDIX A

INTEL 8085

Intel's 8 bit processor i8080 which was widely used needed more than one voltage to be supplied for it to operate. This device also needed a bus controller, System Clock generator and a few more devices for it to function. This was apart from the usual memory and I/O devices. Intel decided to bring out a microprocessor which could require only ROM, RAM and I/O devices to make a functional microcomputer. The voltages required by such a processor was to be cut down to one and TTL compatible. They brought out the microprocessor in 1976 and called it the 'i8085'. This device had a built in oscillator, system timing controller and needed just a crystal to be connected to it. The device was to be packaged in a 40 pin dip (most popular at that time). The memory address was to be same as i8080. Further there was addition of two signals for serial data transmission and reception for use in the audio cassette interface for program storage. All this and the 40 pin dip package needed the device to have some pins to be multiplexed. Intel therefore opted to multiplex the 8 bit data bus with the lower address lines. Thus cutting down the pins needed by 16 bit address and 8 bit data down to 16 from the usual 24 pins. There were no memory devices, at that time, which worked with multiplexed address and data. Intel also made the RAM and ROM devices which functioned with multiplexed address and data. These devices were called i8155 and i8355 respectively. Intel went a step further and put some I/O in these devices. Thus the three chip microcomputer objective of Intel was achieved-8085 +8155 +8355, a few pull up and a FSK decoder/encoder for the serial lines and one could build a microcomputer with a off line program storage on a audio cassette recorder. This was the evolution of 8085 and the multiplexed address and data bus. The next part of this chapter will describes the 8085 architecture.

The MCS 85 family included the 8085 as the microprocessor, the 8355 containing the 2kX8 ROM and two 8 bit parallel ports, the 8155 containing the lkX8 RAM and one 8 bit parallel port. All these three devices had multiplexed address and data buses. The 8085 microprocessor has 8 bit data bus and 16 bit address bus. It has two address spaces-one for memory and the other for Input/Output (I/O). The selection of the address space is made using a signal IO/M. The 8085 is a 40 pin device and needs a single 5 volt supply for its operation. It operates from 3Mhz, 5Mhz, 6Mhz and has a built in oscillator. Thus a crystal, RC or LC network of the desired frequency needs to be connected. The system controller is built-in, four vectored interrupts available, Serial input and output port, arithmetic instructions for decimal, binary and double precision are available. Addresses 64K bytes of memory and 256 bytes of I/O space. The 8085 pin assignment is shown at Figure A-1 followed by the functions of the signals from the pins. There are six groups of 8085 signals and they are listed below:

The Address Bus Group

This is the group of signals which represent the address output by the 8085 as AD0....AD7 and the A8....A15. The AD0...AD7 signals are the multiplexed lower 8 bit Address along with the 8 bit data bus. In the figure shown, these are output from the pins 12 to 19 for the multiplexed lower 8 bit address/Data and pins 21 to 28 for A8 to A15.

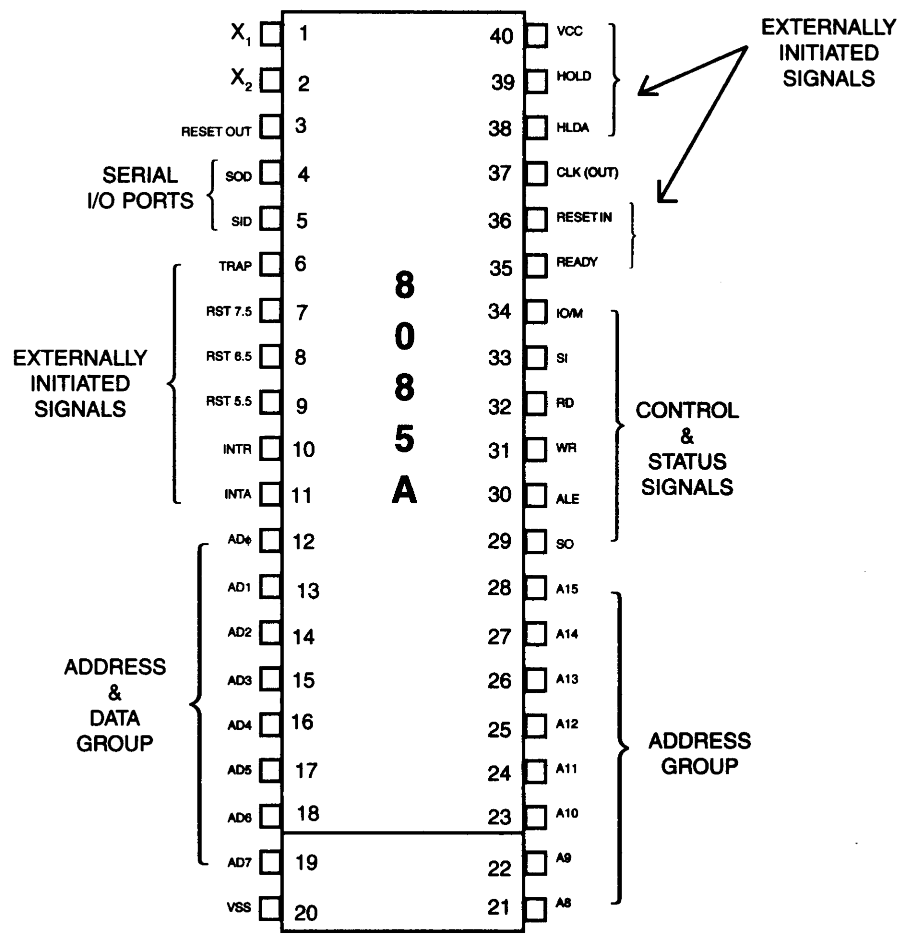


FIGURE A–1 Pin diagram of 8085

The Data Group

These signals are D0 to D7 and are bi-directional. These are multiplexed with address and are available as AD0....AD7 on pins 12 to 19. The presence of Address on this multiplexed bus is indicated by the ALE (Address Latch Enable pin 30) signal which is used for latching the address A0)...A7 from the multiplexed address and Data bus AD0..AD7.

The Control Signals Group

This group consists of six signals of which two are encoded. These signals are as follows:

ALE	Address Latch Enable available at pin 30—this signal goes true (High) whenever lower 8 bit address A0...A7 are available on the multiplexed address and data bus lines AD0...AD7. This is always true at the beginning of the 8085 machine cycle.
RD	This signal goes true (Low) whenever the 8085 is reading data from memory or IO. This is available at pin 32.
WR	This signal goes true (Low) whenever the 8085 is writing data to memory or IO. This is available at pin 31.

IO/M This signal is used for selecting the address space being accessed by the 8085 through its Address lines. If this signal is High it indicates that the 80085 is accessing IO address space and when this signal is of Low it is accessing Memory Address space. This signal is available at pin 34.

SI & SO These are encoded signals and used to identify the machine cycle being executed by the 8085. The following is the representation of these signals and they are used in combination with IO/M,RD,WR

IO/M=0	RD=0	WR=1	SI=1	S0=1	Opcode Fetch
IO/M=0	RD=0	WR=1	SI=1	SO=0	Memory Read
IO/M=0	RD=1	WR=0	SI=0	SO=1	Memory Write Fetch
IO/M=0	RD=0	WR=1	SI=1	SO=1	Opcode Fetch
IO/M=1	RD=1	WR=0	SI=1	SO=1	IO Write
IO/M=1	RD=0	WR=1	SI=1	SO=0	IO Write
IO/M=1	RD=1	WR=1	SI=1	SO=1	Interrupt Acknowledge
IO/M=Z	RD=Z	WR=Z	SI=0	SO=0	HALT
IO/M=Z	RD=Z	WR=Z	SI=X	SO=X	HOLD
IO/M=Z	RD=Z	WR=Z	SI=X	SO=X	RESET

The Power Supply And Clock Group

V_{cc} The positive Voltage is the only supply needed and it is +5 Volts. This is to be provided at pin 40.

GND This is the return line of the Vcc and is to be connected to pin 20.

X1 & X2 are the oscillator inputs where the crystal of desired frequency is connected. The oscillator is in built and following it is a divide by two circuit from which the CPU gets the clock. A typical value of 6 Mhz.

Interrupts and Externally Initiated Signal

INTR Interrupt Request is a general purpose interrupt which is checked by the 8085 at the end of an instruction cycle (just after the last clock cycle of an instruction). During HOLD and HALT states also it is checked. After it is found to have been sensed, the Program Counter is inhibited from incrementing and an INTA interrupt acknowledge is issued. When INTA is issued, a restart address or a call address is inserted which will be executed by the 8085 as an Interrupt service routine. Before executing any further instructions the Program counter and the PSW are saved on the stack. The Program counter is now loaded with the address inserted after the INTA is issued.

INTA This is an interrupt request INTR acknowledge. This is used for activating an interrupt port or the 8259 priority interrupt controller

RST 5.5, RST 6.5, This causes a RESTART from a specific address to take place, the addresses are 2CH, 34H,

RST 7.5 3CH, and the Trigger Type is High Level, High Level and Rising Edge respectively. They have the same timing as INTR but have a higher priority than INTR and are maskable by using the SIM (Set Interrupt Mask) register. Their priorities are in increasing order with RST 7.5 as highest

TRAP This is a non maskable interrupt and has higher priority than the RST 7.5. The address where the Interrupt service routine for trap is expected is at 24H.

HOLD This input is used by a device which wants to be a bus master. Such a device will assert this pin and the 8085 will release the bus at the end of the current machine cycle by tristating the address, data busses, IO/M line, RD and WR signals. After this it issues a

	HLDA (Hold Acknowledge) and the new bus master may take over. The 8085 will takeover the tri-stated busses after the HOLD line is un-asserted.
HLDA	This single is asserted half a clock cycle after the HOLD single has been accepted by the 8085. This is a acknowledgment to the requesting device that the 8085 has relinquished the bus.
READY	A low on this line indicates to the 8085 that a memory or IO device is not ready for the data transfer operations. The 8085 will wait for an integral number of clock cycles till the READY line is asserted.
RESETIN	This is a shcmitt trigger asynchronous input. The Data, Address and control busses are tri-stated, the program counter is set to 0000H address, the Interrupt Enable and HOLD flip Flops (Flags) are reset. For its proper resetting operations this pin should be held low for atleast 3 clock cycles after the Vcc has stabilized. As long as this pin is asserted the 8085 busses will be in tri-state condition.
RESETOUT	This is an Output indicating that the 8085 has been reset and it is synchronised with the clock and lasts an integral number of clock cycles till the reset is un-asserted. This is used for resetting the peripheral chips of the 8085.

Serial IO Pins

SID	This is an input bit and is loaded into bit 7 of the accumulator whenever the RIM (Read Interrupt Mask) instruction is executed. This and the SOD where 8085's approach to serial communications.
SOD	This is an output bit and sets or resets as specified by the SIM (Set Interrupt Mark) instruction is executed. This and the SID where 8085's approach to serial communications.

Functional Description

The block diagram (see Figure A-2) depicts the internal building blocks of the 8085 whose functions are described here:

Timing and Control

This block has a clock generator which oscillates at the frequency of the csytal connected to X1 & X2 input. The output of the clock generator is provided at the Clock Output pin for use of the peripheral devices. The control signal RD and WR are generated and output is given at the respective pins as the status signal ALE, IO/M, SO, SI; the DMA signals HOLD, HLDA and the reset signal RESETIN, RESETOUT

Arithmetic Logic Unit

As the name indicates, this block performs the arithmetic and logic operations. It uses the Accumulator and the Temp register for operands and for holding data during the ALU's operation. The Flag Flip Flops are the bits which represent the condition of the ALU after the current operation. These blocks are shown in the block diagram just above the ALU. The Accumulator register and the general purpose register 'A' are referred almost as one. The flag Flip Flops are called Status Register or Processor Status Word. The bits and their position is shown below.

D7	D6	D5	D4	D3	D2	D1	D0
S Sign Flag	Z Zero Flag		AC Auxiliary Carry Flag		P Parity Flag		C Carry Flag

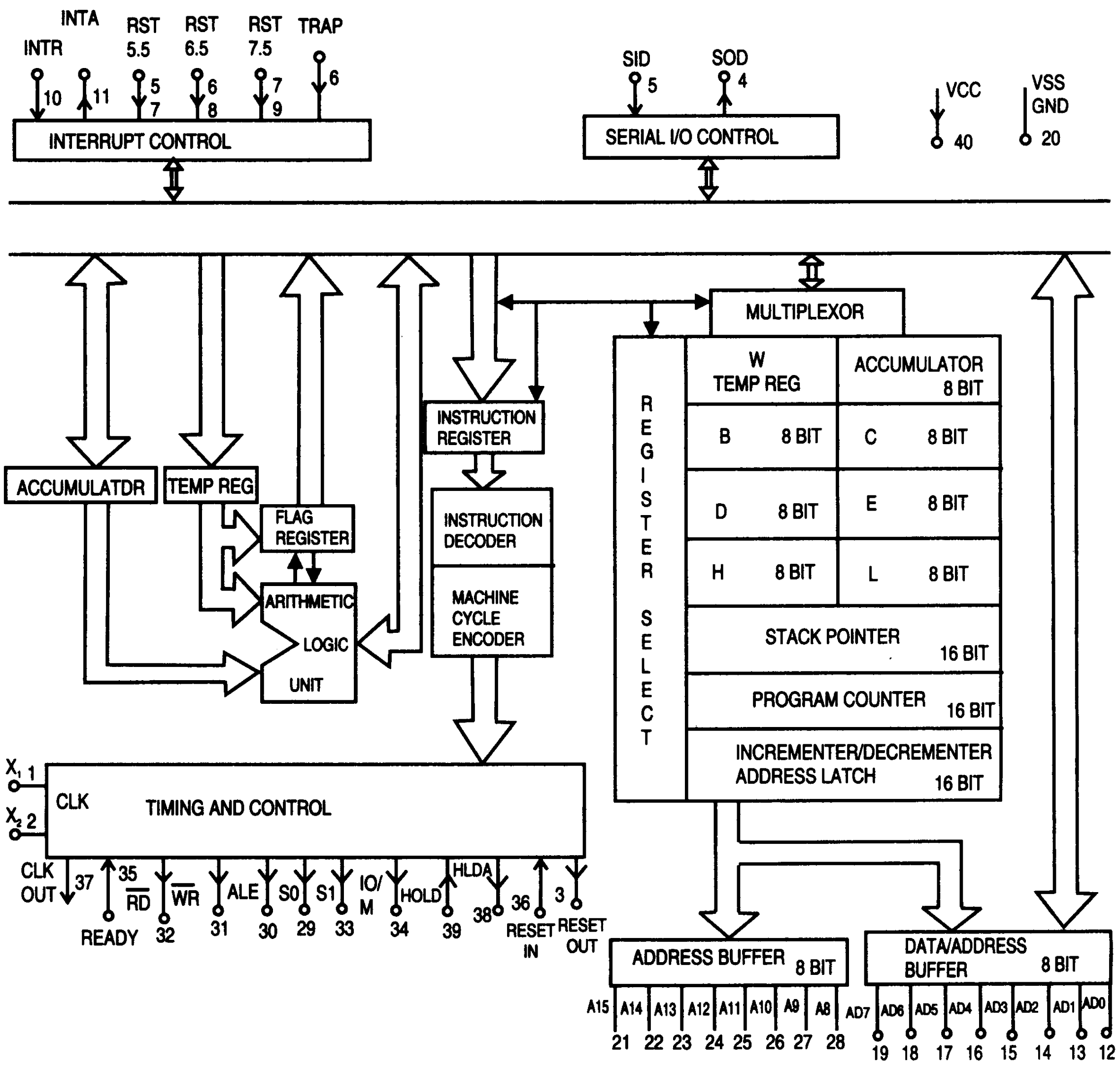


FIGURE A–2 Functional block diagram of intel 8085 with pin numbers

Carry Flag This flag is designated the label 'C'. This flag is affected by an arithmetic operation whenever a carry is generated. During a subtract operation which has a negative result the flag is set, also when a subtract with a borrow is being performed by the programmer this flag may be set and referred as a borrow flag. It is also settable and resetable by instructions.

Parity Flag This flag is referred as 'P'. Whenever the accumulator contains even number of 1's this flag is set indicating an even parity. Similarly for odd parity it is cleared when there is a odd number of 1's.

Auxiliary Carry AC is the reference name given to it and is generated when the lower nybble of the accumulator is greater than '9'. This flag causes the addition of six to the lower nybble when it is set and the Decimal Adjust

Accumulator instruction is used. The result is stored in the lower nybble. Further this nybble is added to the upper nybble and if the upper nybble exceeds 9 then there is the addition of six to the upper nybble; if a carry comes out of this operation on the upper nybble the Carry Flag C is set. This is useful for performing BCD arithmetic.

Zero Flag Whenever an instruction has caused the accumulator contents to become 0 this flag is set. This is also true for the other registers which become zero. The flag is cleared if the instruction did not result in a 0 result in the register.

Sign Flag The sign bit of an 8 bit number is the D7 bit and if this bit becomes 1 after an instruction then this flag is set. Conditional transfer control instructions like the Jump on condition..label, the subroutine calls on condition also use the flag register for their performance.

The Instruction Decoder

When an instruction is fetched from the program memory it is loaded in the instruction register. The operation code part of the instruction in the instruction register is used by this decoder to generate a sequence of machine code to execute the instruction also using the operands to do so in case they are present.

Register Array, Stack Pointer, Program Counter Block

Register Array

This block is accessed via a multiplexor which multiplexes the internal data bus and the instruction decoder. There is a register select decoder which selects the register that is to be active. W register is an 8 bit temporary register for doing intermediate operations. The other registers in this block are:

The **A** register or accumulator is an 8 bit register which is closely associated with the ALU and the Flag Register.

The register **B, C, D, E, H, L,** are 8 bit general purpose registers and can be used as 6 individual 8 bit registers or as 3 paired 16 bit registers. The pairs are restricted only as follows B & C, D & E, H & L. These are used for counters or for Double Precision Arithmetic. H & L are associated with single byte memory transfer where they would contain the memory address and the single byte instruction would specify the register to be used for the transfer.

The **Stack Pointer** is a 16 bit register and does the classical operation of stack management for the 8085. This facilitates the Last In First Out operations needed for stack management.

The Program Counter

The PC as it is referred is a 16 bit register which contains the address of the next instruction to be fetched from the program memory and lodged in the instruction register. The upper 8 bits of this register are put in the address register and the lower part is put in the data/address buffer register. These two buffers are output to the pins whenever the timing control block issues the Address Latch Enable (ALE) signal. The upper part is put on the pins carrying A* to A15 signals and the lower part is put on the pins outputing AD0 to AD7. This register Program Counter is forced to the reset Vector address of 8085 on receiving the Reset Signal. The reset Vector address of the 8085 is 0000H.

Interrupt Control

This block takes inputs from the pins representing INTR, RST 7.5, RST 6.5, RST 5.5 and TRAP. It interacts with the register array block and other blocks through the internal data bus. It generates the INTA Interrupt Acknowledge accordingly to the respective 8085 pin.

Serial Input & Output

This block operates the SID and SOD pins whenever the specific instructions like RIM & SIM are used. It uses the accumulator bit contents for operating.

Timing Diagram

The Write Cycle The first clock during a machine cycle is the T1 state, and 8085 machine cycle has 4T states T1, T2, T3, T4. The following is the description state by state.

Fetch opcode

T1 state-

1. The Program Counter lower 8 bit addresses are put on AD0. AD7 and upper* bits are put on AD8 to AD15
2. The ALE signal is asserted-made high-indicating address is present on AD0 to AD7 lines.
3. IO/M pin is made low indicating that the data is from memory
4. RD is T3, T2, stack not active that is low. As this is memory read for fetching Op Code.
5. WR is not active that is high. For a memory read RD.

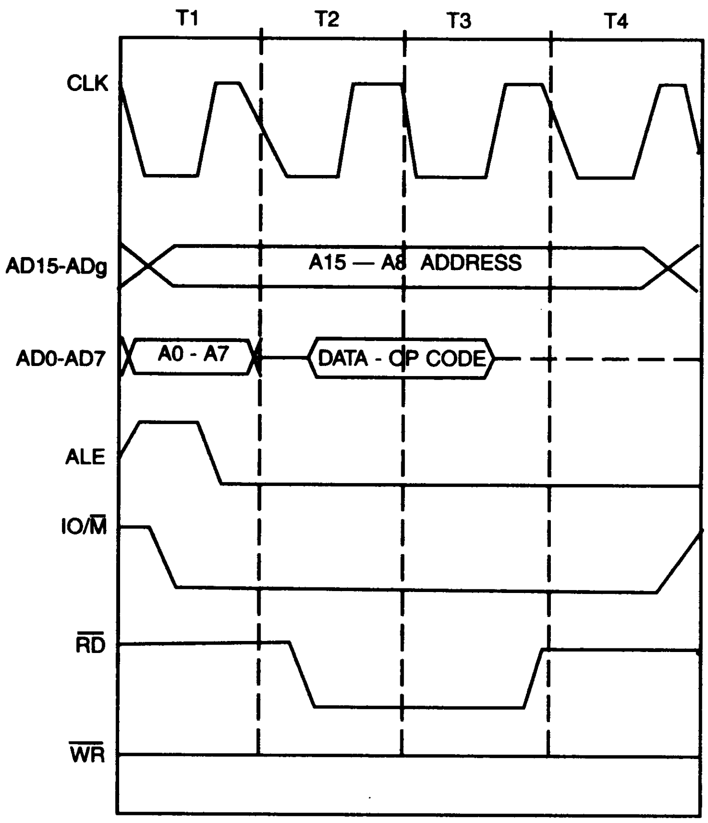


FIGURE A–3 OP code fetch machine cycles

Instruction Format

The 8085 has one, two and three byte instructions and their byte ordering is little endian where the operand one (second byte) will be the destination and operand two (third byte) will be the source. The instruction format is as follows-

One Byte

Operation Code only

Two Byte

Op Code	Data or I/O address

Three Byte

Op Code	Lower address	Higher Address

The destination and source registers are specified in the opcode itsel and the format is given below

0	1	D	D	D	S	S	S

The 01 is the opcode for register/memory move from/to memory/register, DDD is the destination Register, SSS is the Source register. The Register coding is

DDD/SSS	reg
000	B
001	C
010	D
011	E
100	H
101	L
110	m (memory)
111	A

For example if the data from register C is to be moved to memory (pointer to be the H&L register). The Source is C so SSS=001 and the destination is memory DDD=110 so the instruction is 01 110 001 = 71H. This instruction is expressed in mnemonic form as

MOV m,C	;	MOV is opcode, m is memory (operand 1), C is the Operand 2
MOV B,C	;	SSS=001, DDD=000, so MOV B,C 01 000 001
MOV L,m	;	SSS=110, DDD=101 so MOV L,m 01 101 110

RST instructions type is represented in the Destination field which are the bits D6, D5, D4 and the code is AAA

11 AAA 111 is the RST AAA.

The other instructions have fixed opcodes which are not having fields within them for changing the destination and/or source as there is no need

Address Space and Addressing Modes

Address Space

The 8085 has two address space—Memory and Input/Output. The address space is understood as those unique isolated regions or spaces which are accessible using the same address bus. So the I/O devices which are connected to the 8085 have their own addresses which may be the same as the memory addresses but are isolated by a signal

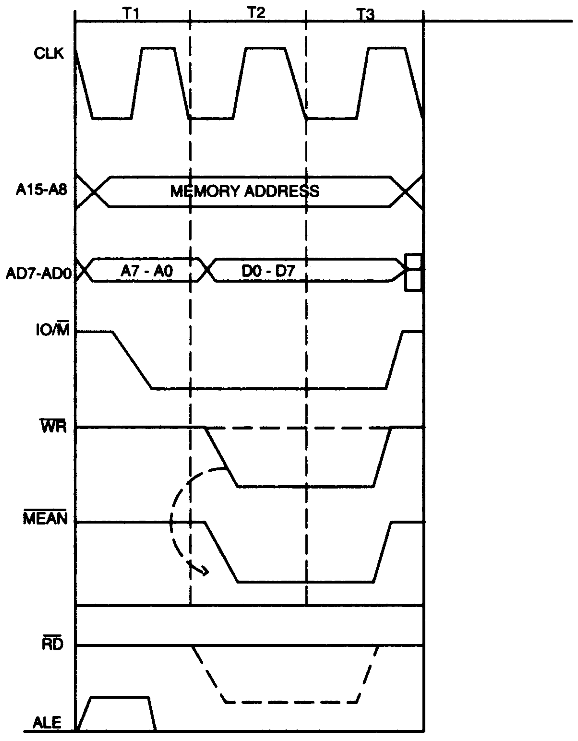


FIGURE A–4 Memory write cycle.

specifying whether the address is of a I/O or Memory—the signal IO/M does this in 8085. This method is also called isolated IO & Memory.

Addressing Modes

These have been explained in the earlier chapters and should be referred if more clarity is needed. The following are the addressing modes of 8085.

Implied Addressing	The instruction is directly decoded to a 8085 control signal as the operation is an implied one having no need for data, that is DAA adjusts the accumulator as two nybbles for the purpose of BCD arithmetic.
Immediate Addressing	The data is available in operand two of the instruction and no other reference need be done.
Direct Addressing	The address of the data is in operands two & three. Computation for effective address is not needed as it is directly available eg. LDA 16 bit address, STA 16 bit address
Register Direct Addressing	The 16 bit address is contained in the H and L register pairs. Computation for effective address is not needed as it is directly available.
Indirect Addressing	Here the Operands two and three contain the address of a memory location which contains the address where the data may be located.

Instruction Set

Data Transfer Group

MVI Rddd 8 bit data	The data is the second operand 8 bit, it is copied to the Accumulator
MOV Rddd, m	The contents of memory location whose address is in H & L register pair are copied to register Rddd.
LDA 16 bit Address	Loads the accumulator directly from the 16 bit address specified in operand 2 & 3 of the instruction.
STA 16 bit Address	The contents of accumulator are copied to the 16 bit memory location whose address specified is operand 2 & 3 of the instruction.
LDAX B	Loads the accumulator from the address specified in register B&C. This is register direct addressing
LDAX D	Loads the accumulator from the address specified in register D&E. This is register direct addressing
STAX B	Stores accumulator contents to the address specified in register D&E. This is register direct addressing
LXI H 16 bit data	The H & L register pair is loaded with data in operand 2 & 3. Immediate instruction.
LXI B 16 bit data	The B & C register pair is loaded with data in operand 2 & 3. Immediate instruction.
LXI D 16 bit data	The D & E register pair is loaded with data in operand 2 & 3. Immediate instruction.
XCHG	Exchanges D&E, H&L registers

Stack Operations

PUSH B	Pushes the register pair B & C to stack
PUSH D	Pushes the register pair D & E to stack
PUSH H	Pushes the register pair H & L to stack
PUSH PSW	Pushes the Accumulator A register and flag register to stack
POP B	Pops the stack to the register pair B&C
POP D	Pops the stack to the register pair D&E
POP H	Pops the stack to the register pair H&L
POP PSW	Pops the accumulator A register and the flag register from stack
XTHL	exchanges top of stack and H&L registers
LXI SP, 16 bit data	Loads the SP with location in memory, it is to point
INX SP	the SP is incremented by two
DCX SP	the SP is decremented by two

Arithmetic Instructions

ADI 8 bit data	8 bit data in operand one is added to A register
ACI 8 bit data	8 bit in operand one is added to A register along with Carry
ADD Rsss	the register Rsss is added to Accumulator
ADC Rsss	the register Rsss with Carry is added to Accumulator
ADD m	single bye instruction to add contents of memory location pointed by H&L pair to the accumulator

ADC m	single bye instruction to add contents of memory location pointed by H&L pair to the accumulator using the Carry
DAD D	Double Precision add D&E are Added to H&L
DAD H	Double Precision add H&L are Added to H&L
DAD SP	Double Precision add SP is Added to H&L
SUB Rsss	The register Rsss is subtracted from A register with borrow
SUI 8 bit Data	8 bit data in operand one is subtracted from A register
SBI 8 bit Data	8 bit data in operand one is subtracted from A register with Borrow
SUB m	Subtract contents of memory pointed by H&L registers from A register
SBB m	Subtract contents of memory pointed by H&L registers from A register with borrow
INR Rsss	Increment register Rsss
DCR Rsss	Decrement register Rsss
INR m	Increment memory
DCR m	Decrement memory
INX B	Increment B&C register pairs
INX D	Increment D&E register pairs
INX H	Increment H&L register pairs
DCX B	Decrement B&C register pairs
DCX D	Decrement D&E register pairs
DCX H	Decrement H&L register pairs

Logic Instructions

ANA Rsss	And register Rsss with accumulator
XRA Rsss	XOR register Rsss with accumulator
ORA Rsss	OR register Rsss with accumulator
CMP Rsss	Compare register Rsss with accumulator
ANA m	And register memory with accumulator
XRA m	XOR register memory with accumulator
ORA m	OR register memory with accumulator
CMP m	Compare register memory with accumulator
ANI 8 bit data	AND 8 bit data in operand one with accumulator
XRI 8 bit data	XOR immediate 8 bit data in operand one with accumulator
ORI 8 bit data	OR immediate 8 bit data in operand one with accumulator
CPI 8 bit data	Compare immediate 8 bit data in operand one with accumulator
RLC	Rotate Left Accumulator one bit
RRC	Rotate Right Accumulator one bit
RAL	Rotate Left Accumulator one bit through Carry
RAR	Rotate Right Accumulator one bit through Carry

Transfer Control/Branch Instructions

JMP 16 bit addr	Jump Unconditional
JC 16 bit addr	Jump on Carry

JNC 16 bit addr	Jump on No Carry
JZ 16 bit addr	Jump on Zero
JNZ 16 bit addr	Jump on Not Zero
JP 16 bit addr	Jump on Positive
JM 16 bit addr	Jump on Minus
JPE 16 bit addr	Jump on Parity Even
JPO 16 bit addr	Jump on Parity Odd
PCHL	H&L to Program Counter

Subroutine Calls

CALL 16 bit address	Call unconditional
CC 16 bit address	Call on Carry
CNC 16 bit address	Call on No Carry
CZ 16 bit address	Call on Zero
CNZ 16 bit address	Call on Not Zero
CP 16 bit address	Call on Positive
CM 16 bit address	Call on Minus
CPE 16 bit address	Call on Parity Even
CPO 16 bit address	Call on Parity Odd

Subroutine Returns

RET	Return
RC	Return on Carry
RNC	Return on No Carry
RZ	Return on ZERO

Implied

CMA	Compliment Accumulator
STC	Set Carry
CMC	Compliment Carry
DAA	Decimal Adjust Accumulator
RIM	Set Interrupt Mask
SIM	Reset Interrupt Mask

Machine Control Instructions

EI	Enable Interrupts
DI	Disable Interrupts
NOP	No-Operation
HLT	HAlt
RST X.Y	Restart from RST X.y

Interrupt Structure and Restart Operations

8085 has a five interrupts namely INTR, RST 5.5, RST 6.5 RST 7.5 and TRAP. TRAP is the Non Maskable Interrupt of 8085. INTR, RST 5.5, RST 6.5, RST 7.5 are maskable interrupts. The RST 5.5, RST 6.5, RST 7.5 and TRAP are the restart interrupts with each of the RST types having independent masking bits whereas TRAP is non maskable. The RST interrupts cause a RESTART (saving the program counter on the stack and branching to the RESTART address. DRST 5.5, RST 6.5, RST, 7.5 RESTART addresses are 2CH, 34H, 3CH and for the TRAP it is 24H. The TRAP causes internal operation of a restart irrespective of the state of the interrupt enable or masks. The status of the RST interrupt masks can be affected by the SIM instruction and the RESETIN.

The priorities of these interrupts are fixed—with the TRAP having the Highest priority followed by RST 7.5, RST 6.5, RST 5.5, INTR being the lowest priority. If a lower priority interrupt occurs when a higher priority of interrupt is being executed the lower priority interrupt is performed even if the higher priority interrupt has enabled the interrupt masks. The TRAP input is used for disaster management like power failure or brown out, or bus errors.

The Three Chip Design

The system interface of the 8085 for a minimal system is straight forward. This is possible because the MCS85 family has some devices which have multiplexed address & Data bus. These devices are i8355 & i8155. The i8355 contains 2k Bytes of ROM and two 8 bit I/O ports. The i8155 has 256 bytes of RAM, two 8 bit I/O ports, one 6 bit I/O port and a programmable timer in a chip. This facilitates making a minimum 3 chip 8085 application with all the peripherals needed to make a simple computer. The specification of such a microprocessor system is

8 bit CPU
2 Kilo Bytes ROM
256 Bytes RAM
1 Timer/Counter
4 eight bit Ports
1 six bit Port
4 interrupt levels
Serial In/Serial Out

The circuit diagram of such a minimal system is shown in Figure A-5 notice that the device select lines, CE pins of 8355 and 8155 are going into the middle of the Address bus-this indicates that these lines will be going to an address line. Further the CE pin of 8155 is HIGH true and the CE pin of 8355 is LOW true. If A15 is connected to i8355 ROM CE pin and also to i8155 RAM chip then ROM would be selected whenever the A15 line went low, that is in the address range 0000H-7FFFH, the ROM would be selected which is suitable as the reset vector of 8085 is 0000H and the ROM should be located there. Similarly when A15 goes HIGH the i8156 is selected which is also suitable. This is for a minimum system so that this type of loose decoding (wrap around repeated address range) of the device will not affect the function.

Program Examples

The following programs are for acquainting the reader to 8085 memory access techniques. Initially it is to be assumed that the program code may be written in memory locations 6000H to 7 FFFH

Block Transferring 'n' number of bytes from memory location 7000 to 7 FFF to 7100H to 71 FFFH where n = 0...FFH.

```
ORG 6 FFFH
n db 10 h ORG       ;  16 bytes in the value of n, so 16 byte transfer
6 000H
LXI H, 6FFFH        ;  H & L to point to number of bytes to transfer
```

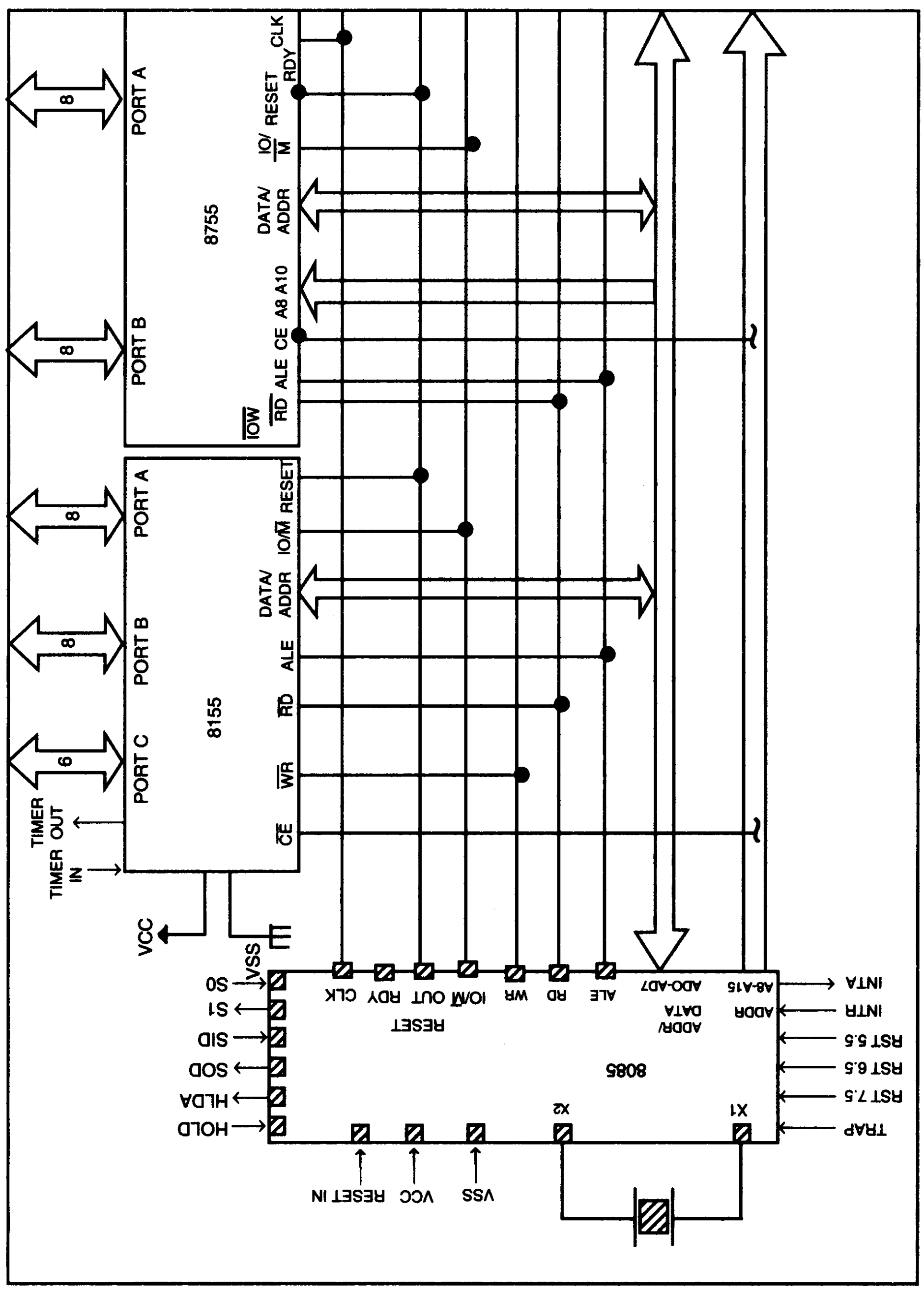


FIGURE A–5 A minimal three chip 8085 microcomputer system

```
MOV C, m             ;  C is loaded with count n, and is the loop counter
LXI D, 7100H         ;  D & E are pointing to destination memory
INX H                ;  make H & L point to the source memory
Again: MOV A, m      ;  Accumulator is read source location (H & L)
STAXD                ;  Accumulator is stored in destination location (D &  B)
INX H                ;  Point to next source location
INX D                ;  Point to next destination location
DCK C                ;  Loop count decremented
JNZ AGAIN            ;  If loop not done jump to Again
HLT                  ;  HALT
```

BCD ADDITION of 8 packed BCD numbers are located at 7010 and 7020 H. The result is stored in 7030 H.

```
                        ORG. 7010H

NUMBER 1                db 23, 45, 56, 62, 95, 42, 71, 85
                        ORG 7020H
NUMBER 2                db  55, 66, 33, 29, 87, 69, 75, 18
                        ORG 7030 H
RESULT                  db 00, 00, 00, 00,  00, 00, 00,  00,
                        ORG 6000H
    LXI SP, 7090 H  ;  initialise the stack at a location allowing room for
                    ;  stack operation
    MVI, C, 08H     ;  Loop count set to number of BCD bytes to add
    LXI H, 7030 H   ;  Initialise HL to point to result
    LXI D, 7020 H   ;  Initialise DE  to point to num 2
                    ;  note that the actual address are used in code
                    ;  though they could be referred by num 1, num 2
                    ;  this is done for associating label of value.
    PUSH H          ;  There are three points needed so save HL pointing
                    ;  to result and load HL with the address
    LXI H, 7010 H   ;  of num 1
    STC             ;  CLEAR THE CARRY, IF ANY, AS A RESULT OR
    CMC             ;  PREVIOUS CODE EXECUTION
AGAIN : LDAX D      ;  get the next num 2 in A reg
    ADC m           ;  add the next num 1 to A reg
    DAA             ;  do BCD addition (packed BCD)
    XTHL            ;  Save num 1 address and restore result address
                    ;  by exchanging the top of stack with A & L registers
    MOV m A         ;  store next result byte
    INX H           ;  H & L  pointed to the next result location
    INX D           ;  D & E pointed to the next num location
    XTHL            ;  Save next result location of load sHL with num 1 location done
    INX H           ;  Point to next num 1 location
    DCR C           ;  Loop count
    JNZ AGAIN       ;  if loop count not done repeat from Again
    HLT
```

Finding largest number in array of bytes starting at 7010 H, Array size in 6FFfH.

```
ORG 6 FFFH        DH 5
ARRAY-SIZE        ORG 6000
LXI H, ARRAY  ;   initialise loop count to Array size
SIZE
MOV C,  M     ;
DCR C         ;   As the comparision is between two bytes the
              ;   number of comparisions made is one less than array size.
XRA H         ;   First largest number in A is zero
INX A         ;   Compare A with memory having next byte
CMP m         ;
JNC H 1       ;   (A-m) if positive array is clear, so A is the greatest
MOV A, m      ;   get the large number from memory to  A register
DCR C         ;   Check whether loop count done
JNZ H 2       ;   A register now contains largest number in Array
HALT          ;
```

APPENDIX B

The Assembler, Disk Operating System, and Basic I/O System

This appendix is provided so the assembler can be understood and to also show the DOS (disk operating system), BIOS (basic I/O system). These function calls are used by assembly to control the personal computer. The function calls control everything from reading and writing disk data, to managing the keyboard and displays, to controlling the mouse. The assembler represented in this text is the Microsoft ML (Version 6.X) and MASM (version 5.10) macro assembler programs. It is fairly important that version 6.X be used instead of the dated version 5.10.

ASSEMBLER USAGE

The assembler program requires that a symbolic program first be written, using a word processor, text editor, or the workbench program provided with the assembler package. The editor provided with version 5.10 is M.EXE, and it is strictly a full-screen editor. The editor provided with version 6.X is PWB.EXE, and it is a fully integrated development system that contains extensive help. Refer to the documentation that accompanies your assembler package for details on the operation of the editor program. If at all possible, use version 6.X of the assembler because it contains a detailed help file that guides you through assembly language statements, directives, and even the DOS and BIOS interrupt function calls.

If you are using a word processor to develop your software, make sure that it is initialized to generate a pure ASCII file. The source file that you generate must use the extension .ASM, which is required for the assembler to properly identify your source program.

Once your source file is prepared, it must be assembled. If you are using the workbench provided with version 6.X, this is accomplished by selecting the compile feature with your mouse. If you are using a word processor and DOS command lines with version 6.14, see Example B–1 for the dialog for version 6.14 to assemble a file called FROG.ASM. Note that this example shows the portions typed by the user in italics.

EXAMPLE B–1

```
A>MASM

Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981, 1997. All rights reserved.

Source filename [.ASM]:FROG
Object filename [FROG.OBJ]:FROG
List filename [NUL.LST]:FROG
Cross reference [NUL.CRF]:FROG
```

Once a program is assembled, it must be linked before is can be executed. The linker converts the object file into an executable file (.EXE). Example B–2 shows the dialog required for the linker using an MASM version 5.10 object file. If the ML version 6.X assembler is in use, it automatically assembles and links a program by using the COMPILE or BUILD command from workbench. After compiling with ML, workbench allows the program to be debugged with a debugging tool called *code view*. Code view is also available with MASM, but CV must be typed at the DOS command line to access it.

EXAMPLE B–2

```
A:\>LINK

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Object modules [.OBJ]:FROG
Run file [FROG.EXE]:FROG
List file [NUL.MAP]:FROG
Libraries [.LIB]:SUBR
```

If MASM version 6.X is in use, the command line syntax differs from version 5.10. Example B–3 shows the command line syntax for ML, the assembler and linker for MASM version 6.X.

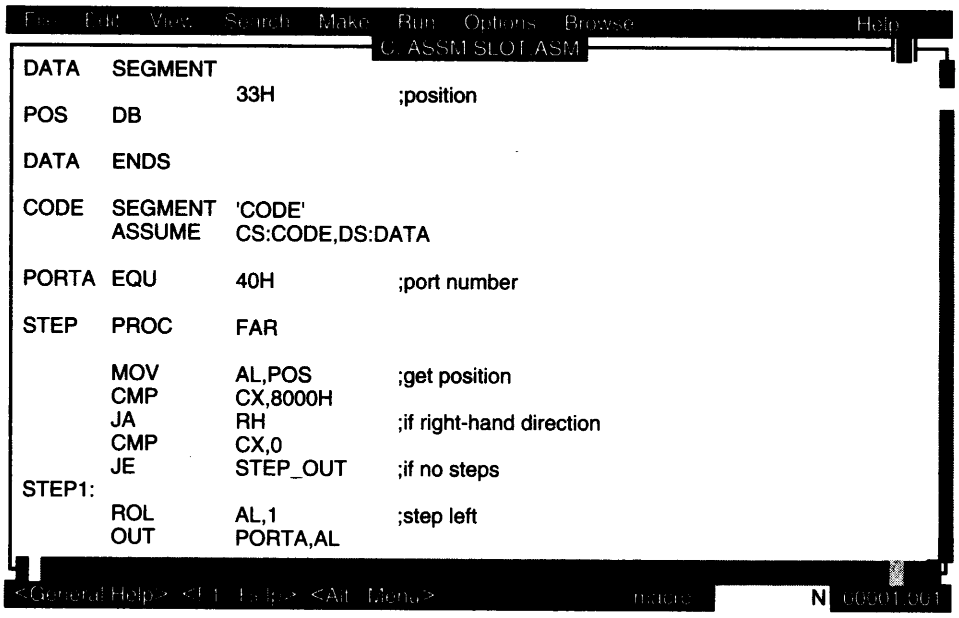
EXAMPLE B–3

```
C:\>ML /FlTEST.LST TEST.ASM

Microsoft (R) Macro Assembler Version 6.14.844
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

    Assembling: TEST.ASM

Microsoft (R) Segmented-Executable Linker Version 5.13
Copyright (C) Microsoft Corp 1984-1993. All rights reserved.

Object Modules [.OBJ]: TEST.obj/t
Run File [TEST.com]: "TEST.com"
List File [NUL.MAP]: NUL
Libraries [.LIB]:
Definitions File [NUL.DEF]: ;
```

Version 6.X of the Microsoft MASM program contains the Programmer's Workbench program. Programmer's Workbench allows an assembly language program to be developed with its full screen editor and tool bar. Figure B–1 illustrates the display found with Programmer's Workbench. To access this program, type PWB at the DOS prompt. The make option allows a program to be automatically assembled and linked, making these tasks simple in comparison to version 5.10 of the assembler.

ASSEMBLER MEMORY MODELS

Memory models and the .MODEL statement are introduced in Chapter 4 and used exclusively throughout the text. Here, we completely define the memory models available for software development. Each model defines the way that a program is stored in the memory system. Table B–1 describes the different models available with both MASM and ML.


File Edit View Search Make Run Options Browse Help
C:\ASSM\SLOT.ASM
<General Help> <F1 Help> <Alt Menu>
macro
N
00001.001

```
DATA    SEGMENT
POS     DB          33H          ;position

DATA    ENDS

CODE    SEGMENT     'CODE'
        ASSUME      CS:CODE,DS:DATA

PORTA   EQU         40H          ;port number

STEP    PROC        FAR

        MOV         AL,POS       ;get position
        CMP         CX,8000H
        JA          RH           ;if right-hand direction
        CMP         CX,0
        JE          STEP_OUT     ;if no steps
STEP1:
        ROL         AL,1         ;step left
        OUT         PORTA,AL
```

FIGURE B–1 The edit screen from Programmer's Workbench used to develop assembly language programs.

TABLE B–1 Memory models for the assembler.

Model Type	*Description*
Tiny	All data and code must fit into one segment. Tiny programs are written in .COM format, which means that the program must be originated at location 100H.
Small	This model contains two segments: one data segment of 64K bytes and one code segment of 64K bytes.
Medium	This model contains one data segment of 64K bytes and any number of code segments for large programs.
Compact	One code segment contains the program, and any number of data segments contain the data.
Large	The large model allows any number of code and data segments.
Huge	This model is the same as large, but the data segments may contain more than 64K bytes each.
Flat	Only available to MASM 6.X. The flat model uses one segment of 512K bytes to store all data and code. Note that this model is mainly used with Windows NT.

Note that the tiny model is used to create a .COM file instead of an execute file. The .COM file is different because all data and code fit into one code segment. A .COM file must use an origin of offset address 0100H as the start of the program. A .COM file loads from the disk and executes faster than the normal execute (.EXE) file. For most applications, we normally use the execute file (.EXE) and the small memory model.

TABLE B–2 Defaults for the .MODEL directive.

Model	*Directives*	*Name*	*Align*	*Combine*	*Class*	*Group*
Tiny	.CODE	_TEXT	word	PUBLIC	'CODE'	DGROUP
	.FARDATA	FAR_DATA	para	private	'FAR_DATA'	
	.FARDATA?	FAR_BSS	para	private	FAR_BSS	
	.DATA	_DATA	word	PUBLIC	DATA'	DGROUP
	.CONST	CONST	word	PUBLIC	'CONST'	DGROUP
	.DATA?	_BSS	word	PUBLIC	'BSS'	DGROUP
Small	.CODE	_TEXT	word	PUBLIC	'CODE'	
	.FARDATA	FAR_DATA	para	private	'FAR_DATA'	
	.FARDATA?	FAR_BSS	para	private	'FAR_BSS'	
	.DATA	_DATA	word	PUBLIC	'DATA'	DGROUP
	.CONST	CONST	word	PUBLIC	'CONST'	DGROUP
	.DATA?	_BSS	word	PUBLIC	'BSS'	DGROUP
	.STACK	STACK	para	STACK	'STACK'	DGROUP
Medium	CODE	name_TEXT	word	PUBLIC	'CODE'	
	.FARDATA	FAR_DATA	para	private	'FAR_DATA'	
	.FARDATA?	FAR_BSS	para	private	'FAR_BSS'	
	.DATA	_DATA	word	PUBLIC	'DATA'	DGROUP
	.CONST	CONST	word	PUBLIC	'CONST'	DGROUP
	.DATA?	_BSS	word	PUBLIC	'BSS'	DGROUP
	.STACK	STACK	para	STACK	'STACK'	DGROUP
Compact	CODE	_TEXT	word	PUBLIC	'CODE'	
	.FARDATA	FAR_DATA	para	private	'FAR_DATA'	
	.FARDATA?	FAR_BSS	para	private	'FAR_BSS'	
	.DATA	_DATA	word	PUBLIC	'DATA'	DGROUP
	.CONST	CONST	word	PUBLIC	'CONST'	DGROUP
	.DATA?	_BSS	word	PUBLIC	'BSS'	DGROUP
	.STACK	STACK	para	STACK	'STACK'	DGROUP
Large or huge	CODE	name_TEXT	word	PUBLIC	'CODE'	
	.FARDATA	FAR_DATA	para	private	'FAR_DATA'	
	.FARDATA?	FAR_BSS	para	private	'FAR_BSS'	
	.DATA	_DATA	word	PUBLIC	'DATA'	DGROUP
	.CONST	CONST	word	PUBLIC	'CONST'	DGROUP
	.DATA?	_BSS	word	PUBLIC	'BSS'	DGROUP
	.STACK	STACK	para	STACK	'STACK'	DGROUP
Flat	CODE	_TEXT	dword	PUBLIC	'CODE'	
	.FARDATA	_DATA	dword	PUBLIC	'DATA'	
	.FARDATA?	_BSS	dword	PUBLIC	'FBSS'	
	.DATA	_DATA	dword	PUBLIC	'DATA'	DGROUP
	.CONST	CONST	dword	PUBLIC	'CONST'	DGROUP
	.DATA?	_BSS	dword	PUBLIC	'BSS'	DGROUP
	.STACK	STACK	dword	STACK	'STACK'	DGROUP

When models are used to create a program, certain defaults apply, as illustrated in Table B–2. The directive in this table is used to start a particular type of segment for the models listed in the table. If the .CODE directive is placed in a program, it indicates the beginning of the code segment. Likewise, .DATA indicates the start of a data segment. The name *column* indicates the name of the segment. *Align* indicates whether the segment is aligned on a word, doubleword, or a 16-byte paragraph. *Combine* indicates the type of segment created. The *class* indicates the class of the segment, such as 'CODE' or 'DATA'. The *group* indicates the group type of the segment.

The directive from Table B–2 selects the type of information in a program. For example, .CODE is placed before the code. The name column is used if full-segment descriptions are mixed with the programming models for reference. The alignment specifies how the data in the segment are aligned. A para (paragraph) alignment starts a segment at the next paragraph, i.e., the next hexadecimal address ending in a 0H. The combine column indicates how various segment are combined and labeled (PUBLIC or private). The class is the actual segment name, and the group is the grouping of segments.

Example B–4 shows a program that uses the small model. The small model is used for programs that contain one DATA and one CODE segment. This applies to many programs that are developed. Notice that not only is the program listed, but so is all the information generated by the assembler. Here, the .DATA directive and .CODE directive indicate the start of each segment. Also notice how the DS register is loaded in this program. As presented throughout the text, the .STARTUP directive can be used to load the data segment register, set up the stack, and define the starting address of a program. In this example, an alternate method (END BEGIN) is illustrated for loading the data segment register and defining the starting address of the program.

EXAMPLE B–4

```
Microsoft (R) Macro Assembler Version 6.14.8444

                          .MODEL SMALL
                          .STACK 100H
0000                      .DATA

0000 0A            FROG   DB    10
0001 0064 [        DATA1  DB    100 DUP (2)
          02
            ]

0000                      .CODE

0000 B8 —— R     BEGIN: MOV   AX,DGROUP          ;set up DS
0003 8E D8                MOV   DS,AX
                           .
                           .
                           .
                          END   BEGIN

Segments and Groups:

          N a m e               Size    Length  Align  Combine   Class

DGROUP . . . . . . . . . . . . . GROUP
_DATA  . . . . . . . . . . . . . 16 Bit  0065    Word   Public    'DATA'
STACK  . . . . . . . . . . . . . 16 Bit  0100    Para   Stack     'STACK'
_TEXT  . . . . . . . . . . . . . 16 Bit  0005    Word   Public    'CODE'

Symbols:

          N a m e               Type    Value   Attr

@CodeSize  . . . . . . . . . . . Number 0000h
@DataSize  . . . . . . . . . . . Number 0000h
@Interface . . . . . . . . . . . Number 0000h
@Model . . . . . . . . . . . . . Number 0002h
```

```
@code . . . . . . . . . . . . . . Text            _TEXT
@data . . . . . . . . . . . . . . Text            DGROUP
@fardata? . . . . . . . . . . . . Text            FAR_BSS
@fardata . . . . . . . . . . . . .Text            FAR_DATA
@stack . . . . . . . . . . . . . .Text            DGROUP
BEGIN . . . . . . . . . . . . . . L Near 0000     _TEXT
DATA1 . . . . . . . . . . . . . . Byte   0001     _DATA
FROG . . . . . . . . . . . . . . .Byte   0000     _DATA

           0 Warnings
           0 Errors
```

Example B–5 lists a program that uses the large model. Notice how it differs from the small model program of Example B–4. Models can be very useful in developing software, but often we use full-segment descriptions, as depicted in most examples in the text.

EXAMPLE B–5

```
Microsoft (R) Macro Assembler Version 6.14.8444

                              .MODEL LARGE
                              .STACK 1000H
0000                          .FARDATA?

0000 00                FROG   DB     ?
0001 0064 [            DATA1  DW     100 DUP (?)
           0000
               ]

0000                    .CONST

0000 54 68 69 73 20 69 MES1   DB      'This is a character string'
     73 20 61 20 63 68
     61 72 61 63 74 65
     72 20 73 74 72 69
     6E 67
001A 53 6F 20 69 73 20 MES2   DB      'So is this!'
     74 68 69 73 21

0000                          .DATA

0000 000C              DATA2  DW     12
0002 00C8 [            DATA3  DB     200 DUP (1)
           01
             ]

0000                          .CODE

0000                   FUNC   PROC   FAR
                              .
                              .
                              .
                              .
0000 CB                       RET

0001                   FUNC   ENDP

                              END    FUNC

Segments and Groups:

            N a m e             Size     Length  Align Combine  Class
```

```
DGROUP . . . . . . . . . . . . . GROUP
_DATA  . . . . . . . . . . . . . 16 Bit   00CA    Word  Public    'DATA'
STACK  . . . . . . . . . . . . . 16 Bit   1000    Para  Stack     'STACK'
CONST  . . . . . . . . . . . . . 16 Bit   0025    Word  Public    'CONST' ReadOnly
EXA_TEXT . . . . . . . . . . . . 16 Bit   0001    Word  Public    'CODE'
FAR_BSS  . . . . . . . . . . . . 16 Bit   00C9    Para  Private   'FAR_BSS'
_TEXT  . . . . . . . . . . . . . 16 Bit   0000    Word  Public    'CODE'

Procedures,  parameters and locals:

             N a m e            Type   Value    Attr

FUNC . . . . . . . . . . . . . . P Far   0000    EXA_TEXT Length= 0001 Public

Symbols:

             N a m e          Type   Value    Attr

@CodeSize  . . . . . . . . . . Number 0001h
@DataSize  . . . . . . . . . . Number 0001h
@Interface . . . . . . . . . . Number 0000h
@Model . . . . . . . . . . . . Number 0005h
@code  . . . . . . . . . . . . Text          EXA_TEXT
@data  . . . . . . . . . . . . Text          DGROUP
@fardata?  . . . . . . . . . . Text          FAR_BSS
@fardata . . . . . . . . . . . Text          FAR_DATA
@stack . . . . . . . . . . . . Text          DGROUP
DATA1  . . . . . . . . . . . . Word   0001   FAR_BSS
DATA2  . . . . . . . . . . . . Word   0000   _DATA
DATA3  . . . . . . . . . . . . Byte   0002   _DATA
FROG . . . . . . . . . . . . . Byte   0000   FAR_BSS
MES1 . . . . . . . . . . . . . Byte   0000   CONST
MES2 . . . . . . . . . . . . . Byte   001A   CONST

           0 Warnings
           0 Errors
```

DOS FUNCTION CALLS

In order to use DOS function calls, always place the function number into register AH and load all other pertinent information into registers, as described in the Table as entry data. Once this is accomplished, follow with an INT 21H to execute the DOS function. Example B–6 shows how to display an ASCII A on the CRT screen at the current cursor position with a DOS function call. Table B–3 is a complete listing of the DOS function calls. Note that some function calls require a segment and offset address, indicated as DS:DI, for example. This means the data segment is the segment address and DI is the offset address. All of the function calls use INT 21H, and AH contains the function call number. Note that functions marked with an @ should not be used unless DOS version 2.XX is in use. Also note that not all function numbers are implemented. As a rule, DOS function calls save all registers not used as exit data, but in certain cases some registers may change. In order to prevent problems, it is advisable to save registers where problems occur.

EXAMPLE B–6

```
0000 B4 06            MOV   AH,6          ;load function 06H
0002 B2 41            MOV   DL,'A'        ;select letter 'A'
0004 CD 21            INT   21H           ;call DOS function
```

TABLE B–3 DOS function calls (pp. 587–608).

00H	TERMINATE A PROGRAM
Entry	AH = 00H CS = program segment prefix address
Exit	DOS is entered
01H	READ THE KEYBOARD
Entry	AH = 01H
Exit	AL = ASCII character
Notes	If AL = 00H, the function call must be invoked again to read an extended ASCII character. Refer to Chapter 7, Table 7–3 for a listing of the extended ASCII keyboard codes. This function call automatically echoes whatever is typed to the video screen.
02H	WRITE TO STANDARD OUTPUT DEVICE
Entry	AH = 02H DL = ASCII character to be displayed
Notes	This function call normally displays data on the video display.
03H	READ CHARACTER FROM COM1
Entry	AH = 03H
Exit	AL = ASCII character read from the communications port
Notes	This function call reads data from the serial communications port.
04H	WRITE TO COM1
Entry	AH = 04H DL = character to be sent out of COM1
Notes	This function transmits data through the serial communications port. The COM port assignment can be changed to use other COM ports with functions 03H and 04H by using the DOS MODE command to reassign COM1 to another COM port.

05H	WRITE TO LPT1
Entry	AH = 05H DL = ASCII character to be printed
Notes	Prints DL on the line printer attached to LPT1. Note that the line printer port can be changed with the DOS MODE command.

06H	DIRECT CONSOLE READ/WRITE

Entry	AH = 06H
	DL = 0FFH or DL = ASCII character

Exit	AL = ASCII character
Notes	If DL = 0FFH on entry, then this function reads the console. If DL = ASCII character, then this function displays the ASCII character on the console (CON) video screen. If a character is read from the console keyboard, the zero flag (ZF) indicates whether a character was typed. A zero condition indicates that no key was typed, and a not-zero condition indicates that AL contains the ASCII code of the key or a 00H. If AL = 00H, the function must again be invoked to read an extended ASCII character from the keyboard. Note that the key does not echo to the video screen.

07H	DIRECT CONSOLE INPUT WITHOUT ECHO
Entry	AH = 07H
Exit	AL = ASCII character
Notes	This functions exactly as function number 06H with DL = 0FFH, but it will not return from the function until the key is typed.

08H	READ STANDARD INPUT WITHOUT ECHO
Entry	AH = 08H
Exit	AL = ASCII character
Notes	Performs as function 07H, except that it reads the standard input device. The standard input device can be assigned as either the keyboard or the COM port. This function also responds to a control-break, where function 06H and 07H do not. A control-break causes INT 23H to execute. By default, this functions as does function 07H.

09H	**DISPLAY A CHARACTER STRING**
Entry	AH = 09H DS:DX = address of the character string
Notes	The character string must end with an ASCII $ (24H). The character string can be of any length and may contain control characters such as carriage return (0DH) and line feed (0AH).

0AH	**BUFFERED KEYBOARD INPUT**
Entry	AH = 0AH DS:DX = address of keyboard input buffer
Notes	The first byte of the buffer contains the size of the buffer (up to 255). The second byte is filled with the number of characters typed upon return. The third byte through the end of the buffer contains the character string typed, followed by a carriage return (0DH). This function continues to read the keyboard (displaying data as typed) until either the specified number of characters are typed or until the enter key is typed.

0BH	**TEST STATUS OF THE STANDARD INPUT DEVICE**
Entry	AH = 0BH
Exit	AL = status of the input device
Notes	This function tests the standard input device to determine if data are available. If AL = 00, no data are available. If AL = 0FFH, then data are available that must be input using function number 08H.

0CH	**CLEAR KEYBOARD BUFFER AND INVOKE KEYBOARD FUNCTION**

Entry	AH = 0CH
	AL = 01H, 06H, 07H, or 0AH

Exit	See exit for functions 01H, 06H, 07H, or 0AH
Notes	The keyboard buffer holds keystrokes while programs execute other tasks. This function empties or clears the buffer and then invokes the keyboard function located in register AL.

0DH	**FLUSH DISK BUFFERS**
Entry	AH = 0DH
Notes	Erases all file names stored in disk buffers. This function does not close the files specified by the disk buffers, so care must be exercised in its usage.

0EH	SELECT DEFAULT DISK DRIVE
Entry	AH = 0EH DL = desired default disk drive number
Exit	AL = the total number of drives present in the system
Notes	Drive A = 00H, drive B = 01H, drive C = 02H, and so forth.

0FH	@OPEN FILE WITH FCB
Entry	AH = 0FH DS:DX = address of the unopened file control block (FCB)
Exit	AL = 00H if file found AL = 0FFH if file not found
Notes	The file control block (FCB) is only used with early DOS software and should never be used with new programs. File control blocks do not allow path names as do the newer file access function codes presented later. Figure B–2 (p. 609) illustrates the structure of the FCB. To open a file, the file must either be present on the disk or be created with function call 16H.

10H	@CLOSE FILE WITH FCB
Entry	AH = 10H DS:DX = address of the opened file control block (FCB)

Exit	AL = 00H if file closed
	AL = 0FFH if error found

Notes	Errors that occur usually indicate either that the disk is full or the media is bad.

11H	@SEARCH FOR FIRST MATCH (FCB)
Entry	AH = 11H DS:DX = address of the file control block to be searched
Exit	AL = 00H if file found AL = 0FFH if file not found
Notes	Wild card characters (? or *) may be used to search for a file name. The ? wild card character matches any character and the * matches any name or extension.

12H	@SEARCH FOR NEXT MATCH (FCB)
Entry	AH = 12H DS:DX = address of the file control block to be searched
Exit	AL = 00H if file found AL = 0FFH if file not found
Notes	This function is used after function 11H finds the first matching file name.

13H	@DELETE FILE USING FCB
Entry	AH = 13H DS:DX = address of the file control block to be deleted

Exit	AL = 00H if file deleted
	AL = 0FFH if error occurred

Notes	Errors that most often occur are defective media errors.

14H	@SEQUENTIAL READ (FCB)
Entry	AH = 14H DS:DX = address of the file control block to be read

Exit	AL = 00H if read successful
	AL = 01H if end of file reached
	AL = 02H if DTA had a segment wrap
	AL = 03H if less than 128 bytes were read

15H	@SEQUENTIAL WRITE (FCB)
Entry	AH = 15H DS:DX = address of the file control block to be written
Exit	AL = 00H if write successful AL = 01H if disk is full AL = 02H if DTA had a segment wrap

16H	@CREATE A FILE (FCB)
Entry	AH = 16H DS:DX = address of an unopened file control block
Exit	AL = 00H if file created AL = 01H if disk is full

17H	@RENAME A FILE (FCB)
Entry	AH = 17H DS:DX = address of a modified file control block

Exit	AL = 00H if file renamed
	AL = 01H if error occurred

Notes	See Figure B–3 (p. 609) for the modified FCB used to rename a file.
19H	RETURN CURRENT DRIVE
Entry	AH = 19H
Exit	AL = current drive
Notes	AL = 00H for drive A, 01H for drive B, and so forth.
1AH	SET DISK TRANSFER AREA
Entry	AH = 1AH DS:DX = address of new DTA
Notes	The disk transfer area is normally located within the program segment prefix at offset address 80H. The DTA is used by DOS for all disk data transfers using file control blocks.
1BH	GET DEFAULT DRIVE FILE ALLOCATION TABLE (FAT)
Entry	AH = 1BH
Exit	AL = number of sectors per cluster DS:BX = address of the media–descriptor CX = size of a sector in bytes DX = number of clusters on drive
Notes	See Figure B–4 (p. 609) for the format of the media–descriptor byte. The DS register is changed by this function, so make sure to save it before using this function.
1CH	GET ANY DRIVE FILE ALLOCATION TABLE (FAT)

Entry	AH = 1CH
	DL = disk drive number

Exit	AL = number of sectors per cluster DS:BX = address of the media–descriptor CX = size of a sector in bytes DX = number of clusters on drive

21H	@RANDOM READ USING FCB
Entry	AH = 21H DS:DX = address of opened FCB

Exit	AL = 00H if read successful
	AL = 01H if end of file reached
	AL = 02H if the segment wrapped
	AL = 03H if less than 128 bytes read

22H	@RANDOM WRITE USING FCB
Entry	AH = 22H DS:DX = address of opened FCB

Exit	AL = 00H if write successful
	AL = 01H if disk full
	AL = 02H if the segment wrapped

23H	@RETURN NUMBER OF RECORDS (FCB)
Entry	AH = 23H DS:DX = address of FCB
Exit	AL = 00H number of records AL = 0FFH if file not found
24H	@SET RELATIVE RECORD SIZE (FCB)
Entry	AH = 24H DS:DX = address of FCB
Notes	Sets the record field to the value contained in the FCB.
25H	SET INTERRUPT VECTOR

Entry	AH = 25H
	AL = interrupt vector number
	DS:DX = address of new interrupt procedure

Notes	Before changing the interrupt vector, it is suggested that the current interrupt vector first be saved using DOS function 35H. This allows a back-link so the original vector can later be restored.
26H	CREATE NEW PROGRAM SEGMENT PREFIX
Entry	AH = 26H DX = segment address of new PSP
Notes	Figure B–5 (p. 609) illustrates the structure of the program segment prefix.

27H	@RANDOM FILE BLOCK READ (FCB)
Entry	AH = 27H CX = the number of records DS:DX = address of opened FCB
Exit	AL = 00H if read successful AL = 01H if end of file reached AL = 02H if the segment wrapped AL = 03H if less than 128 bytes read CX = the number of records read

28H	@RANDOM FILE BLOCK WRITE (FCB)
Entry	AH = 28H CX = the number of records DS:DX = address of opened FCB

Exit	AL = 00H if write successful
	AL = 01H if disk full
	AL = 02H if the segment wrapped
	CX = the number of records written

29H	@PARSE COMMAND LINE (FCB)

Entry	AH = 29H
	AL = parse mask
	DS:SI = address of FCB
	DS:DI = address of command line

Exit	AL = 00H if no file name characters found AL = 01H if file name characters found AL = 0FFH if drive specifier incorrect DS:SI = address of character after name DS:DI = address first byte of FCB

2AH	READ SYSTEM DATE
Entry	AH = 2AH

Exit	AL = day of the week
	CX = the year (1980–2099)
	DH = the month
	DL = day of the month

Notes	The day of the week is encoded as Sunday = 00H through Saturday = 06H. The year is a binary number equal to 1980 through 2099.

2BH	SET SYSTEM DATE

Entry	AH = 2BH
	CX = the year (1980–2099)
	DH = the month
	DL = day of the month

2CH	READ SYSTEM TIME
Entry	AH = 2CH

Exit	CH = hours (0–23)
	CL = minutes
	DH = seconds
	DL = hundredths of seconds

Notes	All times are returned in binary form, and hundredths of seconds may not be available.
2DH	SET SYSTEM TIME

Entry	AH = 2DH
	CH = hours
	CL = minutes
	DH = seconds
	DL = hundredths of seconds

2EH	DISK VERIFY WRITE
Entry	AH = 2EH AL = 00H to disable verify on write AL = 01H to enable verify on write
Notes	By default, disk verify is disabled.
2FH	READ DISK TRANSFER AREA ADDRESS
Entry	AH = 2FH
Exit	ES:BX = contains DTA address
30H	READ DOS VERSION NUMBER
Entry	AH = 30H
Exit	AH = fractional version number AL = whole number version number
Notes	For example, DOS version number 3.2 is returned as a 3 in AL and a 14H in AH.

31H	TERMINATE AND STAY RESIDENT (TSR)

Entry	AH = 31H
	AL = the DOS return code
	DX = number of paragraphs to reserve for program

Notes	A paragraph is 16 bytes, and the DOS return code is read at the batch file level with ERRORCODE.
33H	TEST CONTROL-BREAK

Entry	AH = 33H
	AL = 00H to request current control-break
	AL = 01H to change control-break
	DL = 00H to disable control-break
	DL = 01H to enable control-break

Exit	DL = current control-break state
34H	GET ADDRESS OF InDOS FLAG
Entry	AH = 34H
Exit	ES:BX = address of InDOS flag
Notes	The InDOS flag is available in DOS versions 3.2 or newer and indicates DOS activity. If InDOS = 00H, DOS is inactive; or 0FFH, if DOS is active and pursuing another operation.
35H	READ INTERRUPT VECTOR

Entry	AH = 35H
	AL = interrupt vector number

Exit	ES:BX = address stored at vector
Notes	This DOS function is used with function 25H to install/remove interrupt handlers.
36H	DETERMINE FREE DISK SPACE

Entry	AH = 36H
	DL = drive number

Exit	AX = FFFFH if drive invalid AX = number of sectors per cluster BX = number of free clusters CX = bytes per sector DX = number of clusters on drive
Notes	The default disk drive is DL = 00H, drive A = 01H, drive B = 02H, and so forth.

38H	RETURN COUNTRY CODE

Entry	AH = 38H
	AL = 00H for current country code
	BX = 16-bit country code
	DS:DX = data buffer address

Exit	AX = error code if carry set BX = counter code DS:DX = data buffer address
39H	CREATE SUBDIRECTORY
Entry	AH = 39H DS:DX = address of ASCII-Z string subdirectory name
Exit	AX = error code if carry set
Notes	The ASCII-Z string is the name of the subdirectory in ASCII code ended with a 00H instead of a carriage return/line feed.
3AH	ERASE SUBDIRECTORY
Entry	AH = 3AH DS:DX = address of ASCII-Z string subdirectory name
Exit	AX = error code if carry set
3BH	CHANGE SUBDIRECTORY
Entry	AH = 3BH DS:DX = address of new ASCII-Z string subdirectory name
Exit	AX = error code if carry set
3CH	CREATE A NEW FILE

Entry	AH = 3CH
	CX = attribute word
	DS:DX = address of ASCII-Z string file name

Exit	AX = error code if carry set AX = file handle if carry cleared
Notes	The attribute word can contain any of the following (added together): 01H read-only access, 02H = hidden file or directory, 04H = system file, 08H = volume label, 10H = subdirectory, and 20H = archive bit. In most cases, a file is created with 0000H.

3DH	OPEN A FILE

Entry	AH = 3DH
	AL = access code
	DS:DX = address of ASCII-Z string file name

Exit	AX = error code if carry set AX = file handle if carry cleared
Notes	The access code in AL = 00H for a read-only access, AL = 01H for a write-only access, and AL = 02H for a read/write access. For shared files in a network environment, bit 4 of AL = 1 will deny read/write access, bit 5 of AL = 1 will deny a write access, bits 4 and 5 of AL = 1 will deny read access, bit 6 of AL = 1 denies none, bit 7 of AL = 0 causes the file to be inherited by child; if bit 7 of AL = 1, file is restricted to current process.

3EH	CLOSE A FILE

Entry	AH = 3EH
	BX = file handle

Exit	AX = error code if carry set

3FH	READ A FILE

Entry	AH = 3FH
	BX = file handle
	CX = number of bytes to be read
	DS:DX = address of file buffer to hold data read

Exit	AX = error code if carry set AX = number of bytes read if carry cleared

40H	WRITE A FILE

Entry	AH = 40H
	BX = file handle
	CX = number of bytes to write
	DS:DX = address of file buffer that holds write data

Exit	AX = error code if carry set AX = number of bytes written if carry cleared

41H	DELETE A FILE
Entry	AH = 41H DS:DX = address of ASCII-Z string file name
Exit	AX = error code if carry set

42H	MOVE FILE POINTER

Entry	AH = 42H
	AL = move technique
	BX = file handle
	CX:DX = number of bytes pointer moved

Exit	AX = error code if carry set AX:DX = bytes pointer moved
Notes	The move technique causes the pointer to move from the start of the file if AL = 00H, from the current location if AL = 01H, and from the end of the file if AL = 02H. The count is stored so DX contains the least-significant 16-bits and either CX or AX contains the most-significant 16-bits.
43H	READ/WRITE FILE ATTRIBUTES

Entry	AH = 43H
	AL = 00H to read attributes
	AL = 01H to write attributes
	CX = attribute word (see function 3CH)
	DS:DX = address of ASCII-Z string file name

Exit	AX = error code if carry set CX = attribute word of carry cleared
44H	I/O DEVICE CONTROL (IOTCL)

Entry	AH = 44H
	AL = sub function code (see notes)

Exit	AX = error code (see function 59H) if carry set
Notes	The sub function codes found in AL are as follows:
	00H = read device status Entry: BX = file handle Exit: DX = status 01H = write device status Entry: BX = file handle, DH = 0, DL = device information Exit: AX = error code if carry set 02H = read control data from character device Entry: BX = file handle, CX = number of bytes, DS:DX = I/O buffer address Exit: AX = number of bytes read 03H = write control data to character device Entry: BX = file handle, CX = number of bytes, DS:DX = I/O buffer address

	Exit: AX = number of bytes written
	04H = read control data from block device
	Entry: BL = drive number (0 = default, 1 = A, 2 = B, etc),
	CX = number of bytes, DS:DX = I/O buffer address
	Exit: AX = number of bytes read
	05H = write control data to block device
	Entry: BL = drive number, CX = number of bytes,
	DS:DX = I/O buffer address
	Exit: AX = number of bytes written
	06H = check input status
	Entry: BX = file handle
	Exit: AL = 00H ready or FFH not ready
	07H = check output status
	Entry: BX = file handle
	Exit: AL = 00H ready or FFH not ready
	08H = removable media?
	Entry: BL = drive number
	Exit: AL = 00H removable, 01H fixed
	09H = network block device?
	Entry: BL = drive number
	Exit: bit 12 of DX set for network block device
	0AH = local or network character device?
	Entry: BX = file handle
	Exit: bit 15 of DX set for network character device
	0BH = change entry count (must have SHARE.EXE loaded)
	Entry: CX = delay loop count, DX = retry count
	Exit: AX = error code if carry set
	0CH = generic I/O control for character devices
	Entry: BX = file handle, CH = category, CL = function
	Categories: 00H = unknown, 01H = COM port, 02H =
	CON, 05H = LPT ports
	Function:
	CL = 45H; set iteration count
	CL = 4AH; select code page
	CL = 4CH; start code page preparation
	CL = 4DH; end code page preparation
	CL = 5FH; set display information
	CL = 65H; get iteration count
	CL = 6AH; query selected code page
	CL = 6BH; query preparation list
	CL = 7FH; get display information
	0DH = generic I/O control for block devices
	Entry: BL = drive number, CH = category, CL = function,
	DS:DX = address of parameter block
	Category: 08H = disk drive
	Function:
	CL = 40H; set device parameters
	CL = 41H; write track
	CL = 42H; format and verify track
	CL = 46H, set media ID code
	CL = 47H; set access flag
	CL = 60H; get device parameters
	CL = 61H; read track
	CL = 62H; verify track
	CL = 66H; get media ID code
	CL = 67H; get access code