

CHAPTER 2

The Microprocessor and Its Architecture

INTRODUCTION

This chapter presents the microprocessor as a programmable device by first looking at its internal programming model and then how its memory space is addressed. The architecture of the family of Intel microprocessors is presented simultaneously, as are the ways that the family members address the memory system.

The addressing modes for this powerful family of microprocessors are described for the real, protected, and flat modes of operation. Real mode memory (DOS memory) exists at locations 00000H–FFFFFH, the first 1M byte of the memory system, and is present on all versions of the microprocessor. Protected mode memory (Windows memory) exists at any location in the entire protected memory system, but is available only to the 80286–Core2, not to the earlier 8086 or 8088 microprocessors. Protected mode memory for the 80286 contains 16M bytes; for the 80386–Pentium, 4G bytes; and for the Pentium Pro through the Core2, either 4G or 64G bytes. With the 64-bit extensions enabled, the Pentium 4 and Core2 address 1T byte of memory in a flat memory model. Windows Vista or Windows 64 is needed to operate the Pentium 4 or Core2 in 64-bit mode using the flat mode memory to access the entire 1T byte of memory.

CHAPTER OBJECTIVES

Upon completion of this chapter, you will be able to:

1. Describe the function and purpose of each program-visible register in the 8086–Core2 microprocessors, including the 64-bit extensions.
2. Detail the flag register and the purpose of each flag bit.
3. Describe how memory is accessed using real mode memory-addressing techniques.
4. Describe how memory is accessed using protected mode memory-addressing techniques.
5. Describe how memory is accessed using the 64-bit flat memory model.
6. Describe the program-invisible registers found within the 80286 through Core2 microprocessors.
7. Detail the operation of the memory-paging mechanism.

Before a program is written or any instruction investigated, the internal configuration of the microprocessor must be known. This section of the chapter details the program-visible internal architecture of the 8086–Core2 microprocessors. Also detailed are the function and purpose of each of these

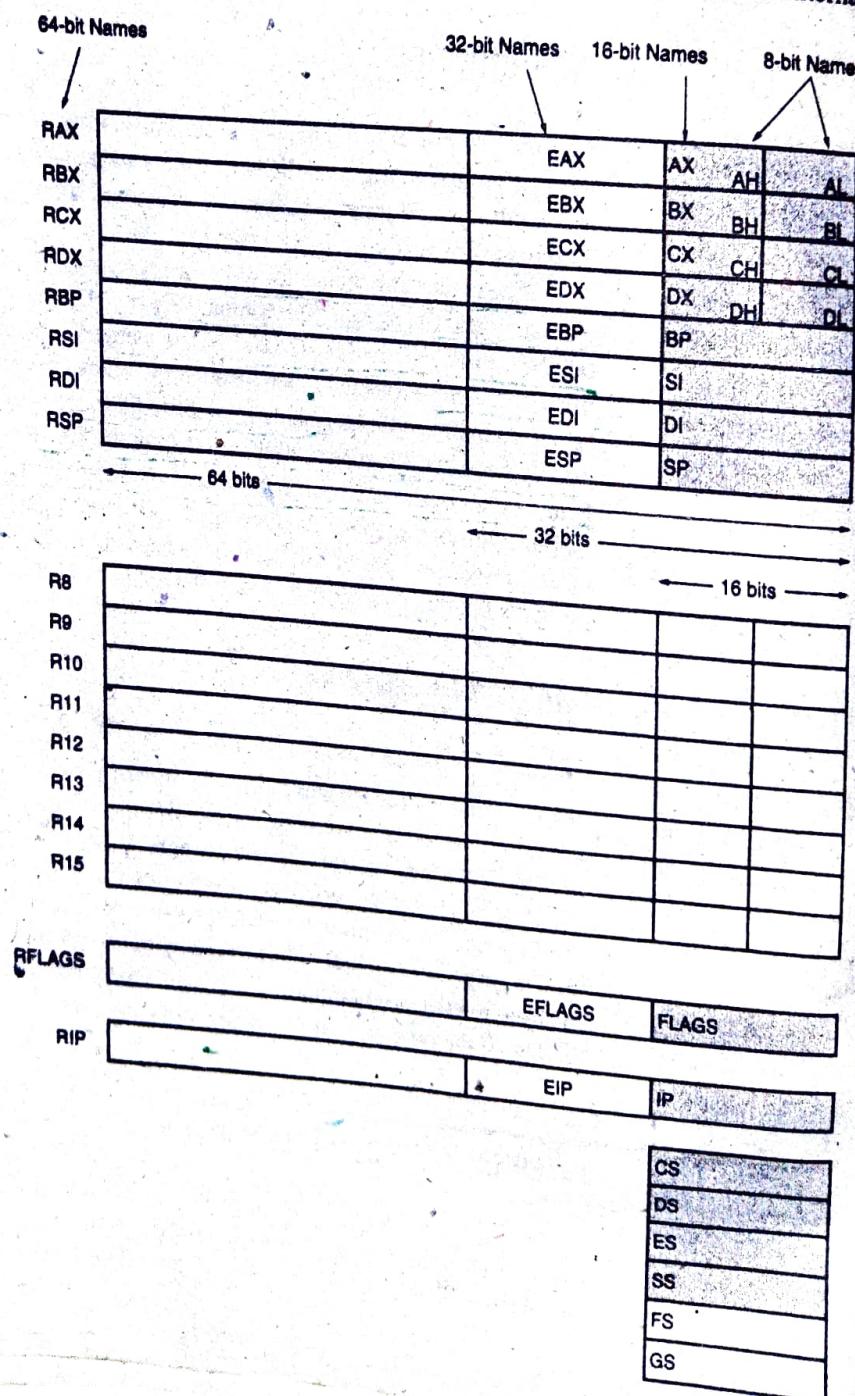
internal registers. Note that in a multiple core microprocessor each core contains the same programming model. The only difference is that each core runs a separate task or thread simultaneously.

The Programming Model

The programming model of the 8086 through the Core2 is considered to be program visible because its registers are used during application programming and are specified by the instructions. Other registers, detailed later in this chapter, are considered to be program invisible because they are not addressable directly during applications programming, but may be used indirectly during system programming. Only the 80286 and above contain the program-invisible registers used to control and operate the protected memory system and other features of the microprocessor.

Figure 2-1 illustrates the programming model of the 8086 through the Core2 microprocessor including the 64-bit extensions. The earlier 8086, 8088, and 80286 contain 16-bit internal

FIGURE 2-1 The programming model of the 8086 through the Core2 microprocessor including the 64-bit extensions.



architectures, a subset of the registers shown in Figure 2–1. The 80386 through the Core2 microprocessors contain full 32-bit internal architectures. The architectures of the earlier 8086 through the 80286 are fully upward-compatible to the 80386 through the Core2. The shaded areas in this illustration represent registers that are found in early versions of the 8086, 8088, or 80286 microprocessors and are provided on the 80386–Core2 microprocessors for compatibility to the early versions.

The programming model contains 8-, 16-, and 32-bit registers. The Pentium 4 and Core2 also contain 64-bit registers when operated in the 64-bit mode as illustrated in the programming model. The 8-bit registers are AH, AL, BH, BL, CH, CL, DH, and DL and are referred to when an instruction is formed using these two-letter designations. For example, an ADD AL,AH instruction adds the 8-bit contents of AH to AL. (Only AL changes due to this instruction.) The 16-bit registers are AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, ES, SS, FS, and GS. Note that the first 4 16 registers contain a pair of 8-bit registers. An example is AX, which contains AH and AL. The 16-bit registers are referenced with the two-letter designations such as AX. For example, an ADD DX, CX instruction adds the 16-bit contents of CX to DX. (Only DX changes due to this instruction.) The extended 32-bit registers are EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI, EIP, and EFLAGS. These 32-bit extended registers, and 16-bit registers FS and GS, are available only in the 80386 and above. The 16-bit registers are referenced by the designations FS or GS for the two new 16-bit registers, and by a three-letter designation for the 32-bit registers. For example, an ADD ECX, EBX instruction adds the 32-bit contents of EBX to ECX. (Only ECX changes due to this instruction.)

Some registers are general-purpose or multipurpose registers, while some have special purposes. The multipurpose registers include EAX, EBX, ECX, EDX, EBP, EDI, and ESI. These registers hold various data sizes (bytes, words, or doublewords) and are used for almost any purpose, as dictated by a program.

The 64-bit registers are designated as RAX, RBX, and so forth. In addition to the renaming of the registers for 64-bit widths, there are also additional 64-bit registers that are called R8 through R15. The 64-bit extensions have multiplied the available register space by more than 8 times in the Pentium 4 and the Core2 when compared to the original microprocessor architecture as indicated in the shaded area in Figure 2–1. An example 64-bit instruction is ADD RCX, RBX, instruction, which adds the 64-bit contents of RBX to RCX. (Only RCX changes due to this instruction.) One difference exists: these additional 64-bit registers (R8 through R15) are addressed as a byte, word, doubleword, or quadword, but only the rightmost 8 bits is a byte. R8 through R15 have no provision for directly addressing bits 8 through 15 as a byte. In the 64-bit mode, a legacy high byte register (AH, BH, CH, or DH) cannot be addressed in the same instruction with an R8 through R15 byte. Because legacy software does not access R8 through R15, this causes no problems with existing 32-bit programs, which function without modification.

Table 2–1 shows the overrides used to access portions of a 64-bit register. To access the low-order byte of the R8 register, use R8B (where B is the low-order byte). Likewise, to access the low-order word of a numbered register, such as R10, use R10W in the instruction. The letter D is used to access a doubleword. An example instruction that copies the low-order doubleword from R8 to R11 is MOV R11D, R8D. There is no special letter for the entire 64-bit register.

TABLE 2–1 Flat mode 64-bit access to numbered registers.

Register Size	Override	Bits Accessed	Example
8 bits	B	7–0	MOV R9B, R10B
16 bits	W	15–0	MOV R10W, AX
32 bits	D	31–0	MOV R14D, F115D
64 bits	—	63–0	MOV R13, R12

Multipurpose Registers

RAX
(accumulator)

RAX is referenced as a 64-bit register (RAX), a 32-bit register (EAX), a 16-bit register (AX), or as either of two 8-bit registers (AH and AL). Note that if an 8- or 16-bit register is addressed, only that portion of the 32-bit register changes without affecting the remaining bits. The accumulator is used for instructions such as multiplication, division, and some of the adjustment instructions. For these instructions, the accumulator has a special purpose, but is generally considered to be a multipurpose register. In the 80386 and above, the EAX register may also hold the offset address of a location in the memory system. In the 64-bit Pentium 4 and Core2, RAX holds a 64-bit offset address, which allows 1T (terra) byte of memory to be accessed through a 40-bit address bus. In the future, Intel plans to expand the address bus to 52 bits to address 4P (peta) bytes of memory.

RBX
(base index)

RBX is addressable as RBX, EBX, BX, BH, or BL. The BX register sometimes holds the offset address of a location in the memory system in all versions of the microprocessor. In the 80386 and above, EBX also can address memory data. In the 64-bit Pentium 4 and Core2, RBX can also address memory data.

RCX
(count)

RCX, which is addressable as RCX, ECX, CX, CH, or CL, is a general-purpose register that also holds the count for various instructions. In the 80386 and above, the ECX register also can hold the offset address of memory data. In the 64-bit Pentium 4, RCX can also address memory data. Instructions that use a count are the repeated string instructions (REP/REPE/REPNE); and shift, rotate, and LOOP/LOOPD instructions. The shift and rotate instructions use CL as the count, the repeated string instructions use CX, and the LOOP/LOOPD instructions use either CX or ECX. If operated in the 64-bit mode, LOOP uses the 64-bit RCX register for the loop counter.

RDX
(data)

RDX, which is addressable as RDX, EDX, DX, DH, or DL, is a general-purpose register that holds a part of the result from a multiplication or part of the dividend before a division. In the 80386 and above, this register can also address memory data.

RBP
(base pointer)

RBP, which is addressable as RBP, EBP, or BP, points to a memory location in all versions of the microprocessor for memory data transfers.

RDI
(destination index)

RDI, which is addressable as RDI, EDI, or DI, often addresses string destination data for the string instructions.

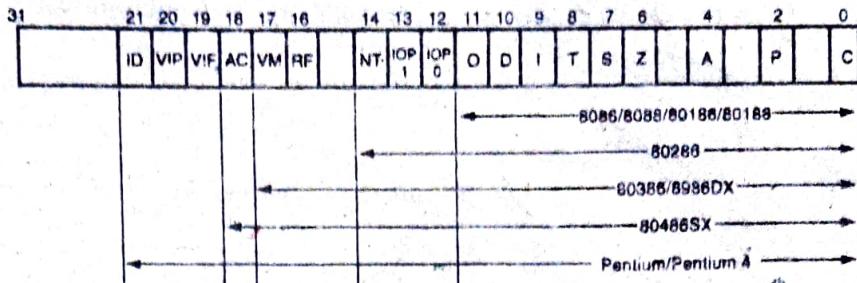
RSI
(source index)

RSI is used as RSI, ESI, or SI. The source index register often addresses source string data for the string instructions. Like RDI, RSI also functions as a general-purpose register. As a 16-bit register, it is addressed as SI; as a 32-bit register, it is addressed as ESI; and as a 64-bit register, it is addressed as RSI.

R8 through R15

These registers are only found in the Pentium 4 and Core2 if 64-bit extensions are enabled. As mentioned, data in these registers are addressed as 64-, 32-, 16-, or 8-bit sizes and are of general purpose. Most applications will not use these registers until 64-bit processors are common. Please note that the 8-bit portion is the rightmost 8-bit only; bits 8 to 15 are not directly addressable as a byte.

FIGURE 2-2 The EFLAG and FLAG register counts for the early 8086 and Pentium microprocessor family.



Special-Purpose Registers. The special-purpose registers include RIP, RSP, and RFLAGS; and the segment registers include CS, DS, ES, SS, FS, and GS.

RIP

(instruction pointer)

* next instruction pointer

RIP addresses the next instruction in a section of memory defined as a code segment. This register is IP (16 bits) when the microprocessor operates in the real mode and EIP (32 bits) when the 80386 and above operate in the protected mode. Note that the 8086, 8088, and 80286 do not contain an EIP register and only the 80286 and above operate in the protected mode. The instruction pointer, which points to the next instruction in a program, is used by the microprocessor to find the next sequential instruction in a program located within the code segment. The instruction pointer can be modified with a jump or a call instruction. In the 64-bit mode, RIP contains a 40-bit address at present to address a 1T flat address space.

RSP

(stack pointer)

RSP addresses an area of memory called the stack. The stack memory stores data through this pointer and is explained later in the text with the instructions that address stack data. This register is referred to as SP if used as a 16-bit register and ESP if referred to as a 32-bit register.

RFLAGS

Later

RFLAGS indicate the condition of the microprocessor and control its operation. Figure 2-2 shows the flag registers of all versions of the microprocessor. (Note the flags are upward-compatible from the 8086/8088 through the Core2 microprocessors.) The 8086–80286 contain a FLAG register (16 bits) and the 80386 and above contain an EFLAG register (32-bit extended flag register). The 64-bit RFLAGS contain the EFLAG register, which is unchanged in the 64-bit version.

The rightmost five flag bits and the overflow flag change after many arithmetic and logic instructions execute. The flags never change for any data transfer or program control operation. Some of the flags are also used to control features found in the microprocessor. Following is a list of each flag bit, with a brief description of their function. As instructions are introduced in subsequent chapters, additional detail on the flag bits is provided. The rightmost five flags and the overflow flag are changed by most arithmetic and logic operations, although data transfers do not affect them.

C (carry)

Carry holds the carry after addition or the borrow after subtraction. The carry flag also indicates error conditions, as dictated by some programs and procedures. This is especially true of the DOS function calls.

P (parity)

Parity is a logic 0 for odd parity and a logic 1 for even parity. Parity is the count of ones in a number expressed as even or odd. For example, if a number contains three binary one bits, it has odd parity. If a number contains no one bits, it has even parity. The parity flag finds little application in modern programming and was implemented in early Intel microprocessors for checking data in data communications environments. Today parity checking is often accomplished by the data communications equipment instead of the microprocessor.

A (auxiliary carry)

The auxiliary carry holds the carry (half-carry) after borrow after subtraction between bit positions 3 and 4. This highly specialized flag bit is tested by the DAA and DS instructions to adjust the value of AL after a BCD addition or subtraction. Otherwise, the A flag bit is not used by the microprocessor or any other instructions.

Z (zero)

The zero flag shows that the result of an arithmetic or logic operation is zero. If Z = 1, the result is zero; if Z = 0, the result is not zero. This may be confusing, but that is how Intel decided to name this flag.

S (sign)

The sign flag holds the arithmetic sign of the result after an arithmetic or logic instruction executes. If S = 1, the sign bit (leftmost bit of a number) is set or negative; if S = 0, the sign bit is cleared or positive.

T (trap)

The trap flag enables trapping through an on-chip debugging feature. (A program is debugged to find an error or bug.) If the T flag is enabled (1), the microprocessor interrupts the flow of the program on conditions as indicated by the debug registers and control registers. If the T flag is a logic 0, the trapping (debugging) feature is disabled. The Visual C++ debugging tool uses the trap feature and debug registers to debug faulty software.

I (interrupt)

The interrupt flag controls the operation of the INTR (interrupt request) input pin. If I = 1, the INTR pin is enabled; if I = 0, the INTR pin is disabled. The state of the I flag bit is controlled by the STI (set I flag) and CLI (clear I flag) instructions.

D (direction)

The direction flag selects either the increment or decrement mode for the DI and/or SI registers during string instructions. If D = 1, the registers are automatically decremented; if D = 0, the registers are automatically incremented. The D flag is set with the STD (set direction) and cleared with the CLD (clear direction) instructions.

O (overflow)

Overflows occur when signed numbers are added or subtracted. An overflow indicates that the result has exceeded the capacity of the machine. For example, if 7FH (+127) is added—using an 8-bit addition—to 01H (+1), the result is 80H (-128). This result represents an overflow condition indicated by the overflow flag for signed addition. For unsigned operations, the overflow flag is ignored.

**IOPL
(I/O privilege level)**

IOPL is used in protected mode operation to select the privilege level for I/O devices. If the current privilege level is higher or more trusted than the IOPL, I/O executes without hindrance. If the IOPL is lower than the current privilege level, an interrupt occurs, causing execution to suspend. Note that an IOPL of 00 is the highest or most trusted and an IOPL of 11 is the lowest or least trusted.

NT (nested task)

The nested task flag indicates that the current task is nested within another task in protected mode operation. This flag is set when the task is nested by software.

RF (resume)

The resume flag is used with debugging to control the resumption of execution after the next instruction.

VM (virtual mode)

The VM flag bit selects virtual mode operation in a protected mode system. A virtual mode system allows multiple DOS memory

AC
(alignment check)

partitions that are 1M byte in length to coexist in the memory system. Essentially, this allows the system program to execute multiple DOS programs. VM is used to simulate DOS in the modern Windows environment.

VIF
(virtual interrupt)

The alignment check flag bit activates if a word or doubleword is addressed on a non-word or non-doubleword boundary. Only the 80486SX microprocessor contains the alignment check bit that is primarily used by its companion numeric coprocessor, the 80487SX, for synchronization.

VIP
(virtual interrupt pending)

The VIF is a copy of the interrupt flag bit available to the Pentium–Pentium 4 microprocessors.

ID (identification)

VIP provides information about a virtual mode interrupt for the Pentium–Pentium 4 microprocessors. This is used in multitasking environments to provide the operating system with virtual interrupt flags and interrupt pending information.

The ID flag indicates that the Pentium–Pentium 4 microprocessors support the CPUID instruction. The CPUID instruction provides the system with information about the Pentium microprocessor, such as its version number and manufacturer.

Segment Registers. Additional registers, called segment registers, generate memory addresses when combined with other registers in the microprocessor. There are either four or six segment registers in various versions of the microprocessor. A segment register functions differently in the real mode when compared to the protected mode operation of the microprocessor. Details on their function in real and protected mode are provided later in this chapter. In the 64-bit flat model, segment registers have little use in a program except for the code segment register. Following is a list of each segment register, along with its function in the system:

CS (code)

The code segment is a section of memory that holds the code (programs and procedures) used by the microprocessor. The code segment register defines the starting address of the section of memory holding code. In real mode operation, it defines the start of a 64K-byte section of memory; in protected mode, it selects a descriptor that describes the starting address and length of a section of memory holding code. The code segment is limited to 64K bytes in the 8086–80286, and 4G bytes in the 80386 and above when these microprocessors operate in the protected mode. In the 64-bit mode, the code segment register is still used in the flat model, but its use differs from other programming modes as explained in Section 2-5.

DS (data)

The data segment is a section of memory that contains most data used by a program. Data are accessed in the data segment by an offset address or the contents of other registers that hold the offset address. As with the code segment and other segments, the length is limited to 64K bytes in the 8086–80286, and 4G bytes in the 80386 and above.

ES (extra)

The extra segment is an additional data segment that is used by some of the string instructions to hold destination data.

SS (stack)

The stack segment defines the area of memory used for the stack. The stack entry point is determined by the stack segment and stack pointer registers. The BP register also addresses data within the stack segment.

FS and GS

The FS and GS segments are supplemental segment registers available in the 80386–Core2 microprocessors to allow two additional memory segments for access by programs. Windows uses these segments for internal operations, but no definition of their usage is available.

2-2

REAL MODE MEMORY ADDRESSING

The 80286 and above operate in either the real or protected mode. Only the 8086 and 8088 operate exclusively in the real mode. In the 64-bit operation mode of the Pentium 4 and Core2, there is no real mode operation. This section of the text details the operation of the microprocessor in the real mode. Real mode operation allows the microprocessor to address only the first 1M byte of memory space—even if it is the Pentium 4 or Core2 microprocessor. Note that the first 1M byte of memory is called the real memory, conventional memory, or DOS memory system. The DOS operating system requires that the microprocessor operates in the real mode. Windows does not use the real mode. Real mode operation allows application software written for the 8086/8088, which only contains 1M byte of memory, to function in the 80286 and above without changing the software. The upward compatibility of software is partially responsible for the continuing success of the Intel family of microprocessors. In all cases, each of these microprocessors begins operation in the real mode by default whenever power is applied or the microprocessor is reset. Note that if the Pentium 4 or Core2 operate in the 64-bit mode, it cannot execute real mode applications; hence, DOS applications will not execute in the 64-bit mode unless a program that emulates DOS is written for the 64-bit mode.

Segments and Offsets

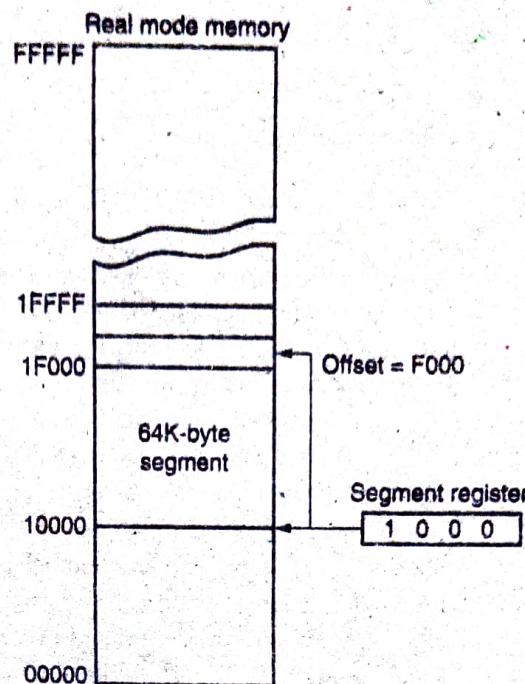
A combination of a segment address and an offset address accesses a memory location in the real mode. All real mode memory addresses must consist of a segment address plus an offset address. The **segment address**, located within one of the segment registers, defines the beginning address of any 64K-byte memory segment. The **offset address** selects any location within the 64K byte memory segment. Segments in the real mode always have a length of 64K bytes. Figure 2-3 shows how the segment plus offset addressing scheme selects a memory location. This illustration shows a memory segment that begins at location 10000H and ends at location FFFFH—64K bytes in length. It also shows how an offset address, sometimes called a **displacement**, of F000H selects location 1F000H in the memory system. Note that the offset or displacement is the distance above the start of the segment, as shown in Figure 2-3.

The segment register in Figure 2-3 contains 1000H, yet it addresses a starting segment at location 10000H. In the real mode, each segment register is internally appended with a 0H on its rightmost end. This forms a 20-bit memory address, allowing it to access the start of a segment. The microprocessor must generate a 20-bit memory address to access a location within the first 1M of memory. For example, when a segment register contains 1200H, it addresses a 64K-byte memory segment beginning at location 12000H. Likewise, if a segment register contains 1201H, it addresses a memory segment beginning at location 12010H. Because of the internally appended 0H, real mode segments can begin only at a 16-byte boundary in the memory system. This 16-byte boundary is often called a **paragraph**.

Because a real mode segment of memory is 64K in length, once the beginning address is known, the **ending address** is found by adding FFFFH. For example, if a segment register contains 3000H, the first address of the segment is 30000H, and the last address is 30000H + FFFFH or 3FFFFH. Table 2-2 shows several examples of segment register contents and the starting and ending addresses of the memory segments selected by each segment address.

The offset address, which is a part of the address, is added to the start of the segment to address a memory location within the memory segment. For example, if the segment address is

FIGURE 2-3 The real mode memory-addressing scheme, using a segment address plus an offset.



1000H and the offset address is 2000H, the microprocessor addresses memory location 12000H. The segment address is always added to the starting address of the segment to locate the data. The segment and offset address is sometimes written as 1000:2000 for a segment address of 1000H with an offset of 2000H.

In the 80286 (with special external circuitry) and the 80386 through the Pentium 4, an extra 64K minus 16 bytes of memory is addressable when the segment address is FFFFH and the HIMEM.SYS driver for DOS is installed in the system. This area of memory (0FFFF0H–10FFEFH) is referred to as **high memory**. When an address is generated using a segment address of FFFFH, the A20 address pin is enabled (if supported in older systems) when an offset is added. For example, if the segment address is FFFFH and the offset address is 4000H, the machine addresses memory location FFFF0H + 4000H or 103FF0H. Notice that the A20 address line is the one in address 103FF0H. If A20 is not supported, the address is generated as 03FF0H because A20 remains a logic zero.

Some addressing modes combine more than one register and an offset value to form an offset address. When this occurs, the sum of these values may exceed FFFFH. For example, the address accessed in a segment whose segment address is 4000H and whose offset address is specified as the sum of F000H plus 3000H will access memory location 42000H instead of location 52000H. When the F000H and 3000H are added, they form a 16-bit (**modulo 16**) sum of 2000H used as the offset address; not 12000H, the true sum. Note that the carry of 1 (F000H + 3000H = 12000H) is dropped for this addition to form the offset address of 2000H. The address is generated as 4000:2000 or 42000H.

TABLE 2-2 Example of real mode segment addresses.

Segment Register	Starting Address	Ending Address
2000H	20000H	2FFFFH
2001H	20010H	3000FH
2100H	21000H	30FFFH
AB00H	AB000H	BFFFFH
1234H	12340H	2233FH

Default Segment and Offset Registers

The microprocessor has a set of rules that apply to segments whenever memory is addressed. These rules, which apply in the real and protected mode, define the segment register and offset register combination. For example, the code segment register is always used with the instruction pointer to address the next instruction in a program. This combination is CS:IP or CS:EIP, depending upon the microprocessor's mode of operation. The code segment register defines the start of the code segment and the **Instruction pointer** locates the next instruction within the code segment. This combination (CS:IP or CS:EIP) locates the next instruction executed by the microprocessor. For example, if CS = 1400H and IP/EIP = 1200H, the microprocessor fetches its next instruction from memory location 14000H + 1200H or 15200H.

Another of the default combinations is the **stack**. Stack data are referenced through the stack segment at the memory location addressed by either the stack pointer (SP/ESP) or the pointer (BP/EBP). These combinations are referred to as SS:SP (SS:ESP), or SS:BP (SS:EBP). For example, if SS = 2000H and BP = 3000H, the microprocessor addresses memory location 23000H for the stack segment memory location. Note that in real mode, only the rightmost 16 bits of the extended register address a location within the memory segment. In the 80386–Pentium 4, never place a number larger than FFFFH into an offset register if the microprocessor is operated in the real mode. This causes the system to halt and indicate an addressing error.

Other defaults are shown in Table 2–3 for addressing memory using any Intel microprocessor with 16-bit registers. Table 2–4 shows the defaults assumed in the 80386 and above using 32-bit registers. Note that the 80386 and above have a greater selection of segment/offset address combinations than do the 8086 through the 80286 microprocessors.

The 8086–80286 microprocessors allow four memory segments and the 80386–Core2 microprocessors allow six memory segments. Figure 2–4 shows a system that contains four memory segments. Note that a memory segment can touch or even overlap if 64K bytes of memory are not required for a segment. Think of segments as windows that can be moved over any area of memory to access data or code. Also note that a program can have more than four or six segments, but only access four or six segments at a time.

Suppose that an application program requires 1000H bytes of memory for its code, 190H bytes of memory for its data, and 200H bytes of memory for its stack. This application does not require an extra segment. When this program is placed in the memory system by DOS, it is loaded in the TPA at the first available area of memory above the drivers and other TPA program. This area is indicated by a **free-pointer** that is maintained by DOS. Program loading is handled automatically by the **program loader** located within DOS. Figure 2–5 shows how an application is stored in the memory system. The segments show an overlap because the amount of data in them does not require segments can be moved over any area of memory by changing the segment starting address. Fortunately, the DOS program loader calculates and assigns segment starting addresses.

Segment and Offset Addressing Scheme Allows Relocation

The segment and offset addressing scheme seems unduly complicated. It is complicated, but it also affords an advantage to the system. This complicated scheme of segment plus offset addressing

TABLE 2–3 Default 16-bit segment and offset combinations.

Segment	Offset	Special Purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8- or 16-bit number	Data address
ES	DI for string instructions	String destination address

TABLE 2-4 Default 32-bit segment and offset combinations.

Segment	Offset	Special Purpose
CS	EIP	Instruction address
SS	ESP or EBP	Stack address
DS	EAX, EBX, ECX, EDX, ESI, EDI, an 8- or 32-bit number	Data address
ES	EDI for string instructions	String destination address
FS	No default	General address
GS	No default	General address

allows DOS programs to be relocated in the memory system. It also allows programs written to function in the real mode to operate in a protected mode system. A **relocatable program** is one that can be placed into any area of memory and executed without change. Relocatable data are data that can be placed in any area of memory and used without any change to

FIGURE 2-4 A memory system showing the placement of four memory segments.

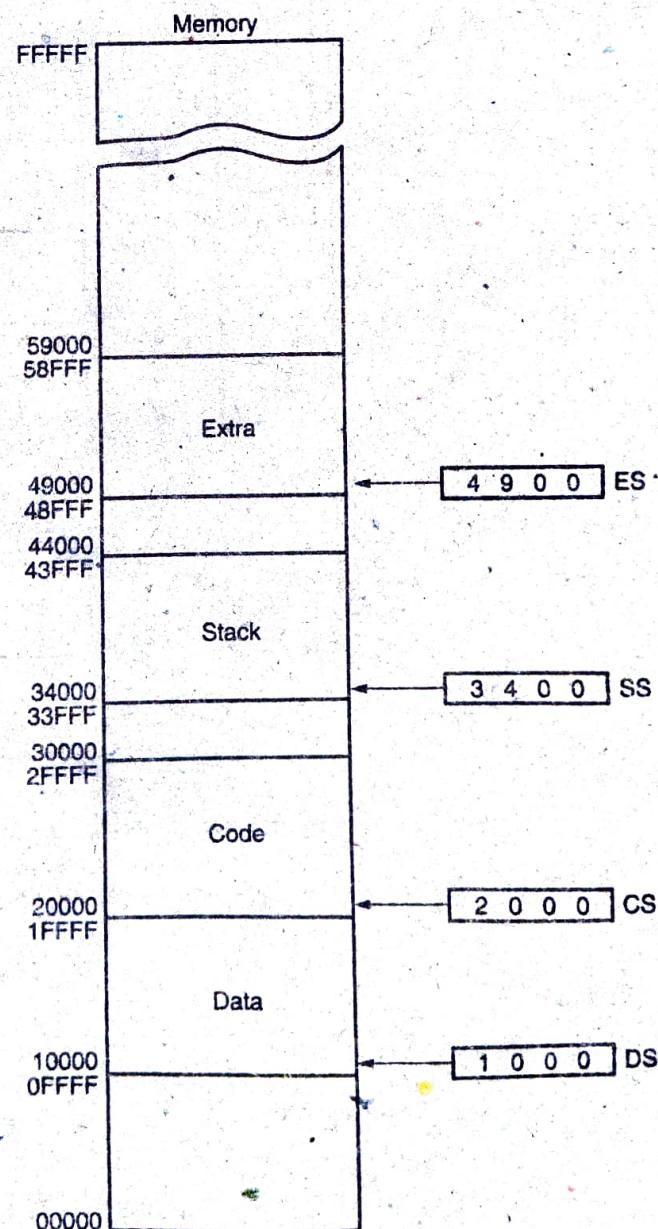
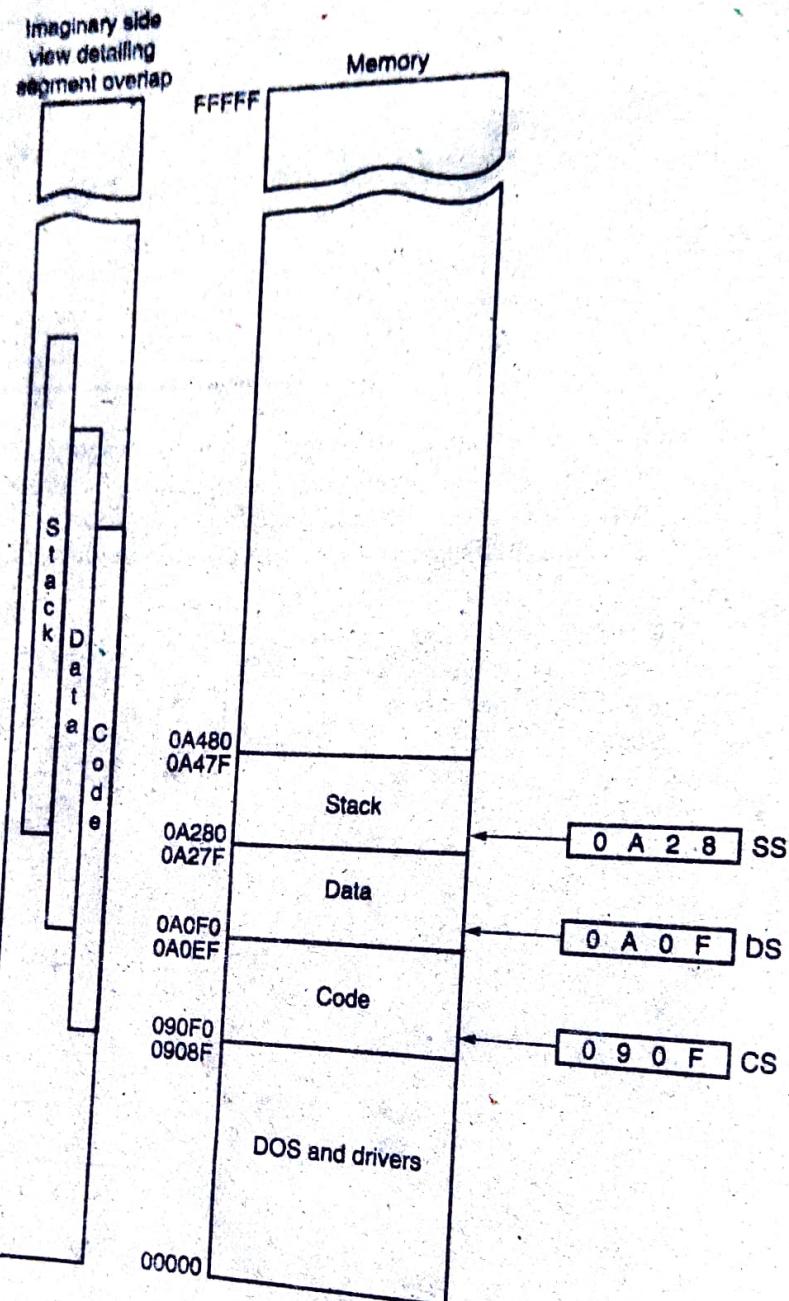


FIGURE 2-6 An application program containing a code, data, and stack segment loaded into a DOS system memory.



the program. The segment and offset addressing scheme allows both programs and data to be reloaded without changing a thing in a program or data. This is ideal for use in a general-purpose computer system in which not all machines contain the same memory areas. The personal computer memory structure is different from machine to machine, requiring relocatable software and data.

Because memory is addressed within a segment by an offset address, the memory segment can be moved to any place in the memory system without changing any of the offset addresses. This is accomplished by moving the entire program, as a block, to a new area and then changing only the contents of the segment registers. If an instruction is 4 bytes above the start of the segment, its offset address is 4. If the entire program is moved to a new area of memory, this offset address of 4 still points to 4 bytes above the start of the segment. Only the contents of the segment register must be changed to address the program in the new area of memory. Without this feature, a program would have to be extensively rewritten or altered before it is moved. This would require additional time or many versions of a program for the many different configurations of computer systems. This concept also applies to programs written to

execute in the protected mode for Windows. In the Windows environment all programs are written assuming that the first 2G of memory are available for code and data. When the program is loaded, it is placed in the actual memory, which may be anywhere and a portion may be located on the disk in the form of a swap file.

2-3

INTRODUCTION TO PROTECTED MODE MEMORY ADDRESSING

Protected mode memory addressing (80286 and above) allows access to data and programs located above the first 1M byte of memory, as well as within the first 1M byte of memory. Protected mode is where Windows operates. Addressing this extended section of the memory system requires a change to the segment plus an offset addressing scheme used with real mode memory addressing. When data and programs are addressed in extended memory, the offset address is still used to access information located within the memory segment. One difference is that the segment address, as discussed with real mode memory addressing, is no longer present in the protected mode. In place of the segment address, the segment register contains a selector that selects a descriptor from a descriptor table. The descriptor describes the memory segment's location, length, and access rights. Because the segment register and offset address still access memory, protected mode instructions are identical to real mode instructions. In fact, most programs written to function in the real mode will function without change in the protected mode. The difference between modes is in the way that the segment register is interpreted by the microprocessor to access the memory segment. Another difference, in the 80386 and above, is that the offset address can be a 32-bit number instead of a 16-bit number in the protected mode. A 32-bit offset address allows the microprocessor to access data within a segment that can be up to 4G bytes in length. Programs that are written for the 32-bit protected mode execute in the 64-bit mode of the Pentium 4.

Selectors and Descriptors

The selector, located in the segment register, selects one of 8192 descriptors from one of two tables of descriptors. The descriptor describes the location, length, and access rights of the segment of memory. Indirectly, the segment register still selects a memory segment, but not directly as in the real mode. For example, in the real mode, if CS = 0008H, the code segment begins at location 00080H. In the protected mode, this segment number can address any memory location in the entire system for the code segment, as explained shortly.

There are two descriptor tables used with the segment registers: one contains global descriptors and the other contains local descriptors. The global descriptors contain segment definitions that apply to all programs, whereas the local descriptors are usually unique to an application. You might call a global descriptor a system descriptor and call a local descriptor an application descriptor. Each descriptor table contains 8192 descriptors, so a total of 16,384 total descriptors are available to an application at any time. Because the descriptor describes a memory segment, this allows up to 16,384 memory segments to be described for each application. Since a memory segment can be up to 4G bytes in length, this means that an application could have access to $4G \times 16,384$ bytes of memory or 64T bytes.

Figure 2-6 shows the format of a descriptor for the 80286 through the Core2. Note that each descriptor is 8 bytes in length, so the global and local descriptor tables are each a maximum of 64K bytes in length. Descriptors for the 80286 and the 80386-Core2 differ slightly, but the 80286 descriptor is upward-compatible.

The base address portion of the descriptor indicates the starting location of the memory segment. For the 80286 microprocessor, the base address is a 24-bit address, so segments begin at any location in its 16M bytes of memory. Note that the paragraph boundary limitation is

The diagram illustrates the evolution of 64-bit descriptors across three generations of Intel microprocessors: 80286, 80386-P4, and 64-bit P4.

80286: This row shows a 32-bit descriptor. It consists of a 16-bit Base field (B15-B0), a 16-bit Limit field (L15-L0), and an Access Rights field. The offset is explicitly labeled as 0.

80286							
0000 0000 0000 0000				Access Rights		Base	
				B23		B16	
Base				B0 L15		Limit	
B15				B0		L0	
Offset 0							

80386-P4: This row shows a 32-bit descriptor. It includes a 16-bit Base field (B24-B15), a 16-bit Limit field (L16-L0), and an Access Rights field. The offset is explicitly labeled as 0.

80386-P4							
Base				G D O A V		Limit	
B24				L19		L16	
B31				Access Rights		Base	
				B23		B16	
Base				B0 L15		Limit	
B15				B0		L0	
Offset 0							

64-bit P4: This row shows a 64-bit descriptor. It includes a 32-bit Base field (B31-B0), a 32-bit Limit field (L31-L0), and an Access Rights field. The offset is explicitly labeled as 0.

64-bit P4							
0000 0000				G D L A V		0000	
0000 0000 0000 0000				Access Rights		0000 0000	
				0000 0000 0000 0000		0000 0000 0000 0000	
Offset 0							

FIGURE 2–6 The 80286 through Core2 64-bit descriptors.

removed in these microprocessors when operated in the protected mode so segments may begin at any address. The 80386 and above use a 32-bit base address that allows segments to begin at any location in its 4G bytes of memory. Notice how the 80286 descriptor's base address is upward-compatible to the 80386 through the Pentium 4 descriptor because its most-significant 16 bits are 0000H. Refer to Chapters 18 and 19 for additional detail on the 64G memory space provided by the Pentium Pro through the Core2.

The segment limit contains the last offset address found in a segment. For example, if a segment begins at memory location F00000H and ends at location F000FFH, the base address is F00000H and the limit is FFH. For the 80286 microprocessor, the base address is F00000H and the limit is 00FFH. For the 80386 and above, the base address is 00F00000H and the limit is 000FFH. Notice that the 80286 has a 16-bit limit and the 80386 through the Pentium 4 have a 20-bit limit. An 80286 can access memory segments that are between 1 and 64K bytes in length. The 80386 and above access memory segments that are between 1 and 1M byte, or 4K and 4G bytes in length.

There is another feature found in the 80386 through the Pentium 4 descriptor that is not found in the 80286 descriptor: the G bit, or **granularity** bit. If G = 0, the limit specifies a segment limit of 00000H to FFFFFH. If G = 1, the value of the limit is multiplied by 4K bytes (appended with FFFH). The limit is then 00000FFFFH to FFFFFFFFH, if G = 1. This allows a segment length of 4K to 4G bytes in steps of 4K bytes. The reason that the segment length is 64K bytes in the 80286 is that the offset address is always 16 bits because of its 16-bit internal architecture. The 80386 and above use a 32-bit architecture that allows an offset address, in the protected mode operation, of the 32 bits. This 32-bit offset address allows segment lengths of 4G bytes and the 16-bit offset address allows segment lengths of 64K bytes. Operating systems operate in a 16- or 32-bit environment. For example, DOS uses a 16-bit environment, while most Windows applications use a 32-bit environment called WIN32.

In the 64-bit descriptor, the L bit (probably means large, but Intel calls it the 64-bit) selects 64-bit addresses in a Pentium 4 or Core2 with 64-bit extensions when L = 1 and 32-bit

compatibility mode when L = 0. In 64-bit protected operation, the code segment register is still used to select a section of code from the memory. Notice that the 64-bit descriptor has no limit or base address. It only contains an access rights byte and the control bits. In the 64-bit mode, there is no segment or limit in the descriptor and the base address of the segment, although not placed in the descriptor, is 00 0000 0000H. This means that all code segments start at address zero for 64-bit operation. There are no limit checks for a 64-bit code segment.

Example 2-1 shows the segment start and end if the base address is 10000000H, the limit is 001FFH, and the G bit = 0.

EXAMPLE 2-1

```
Base = Start = 10000000H
G = 0
End = Base + Limit = 10000000H + 001FFH = 100001FFH
```

Example 2-2 uses the same data as Example 2-1, except that the G bit = 1. Notice that the limit is appended with FFFF to determine the ending segment address.

EXAMPLE 2-2

```
Base = Start = 10000000H
G = 1
End = Base + Limit = 10000000H + 001FFFFH = 101FFFFFFH
```

The AV bit, in the 80386 and above descriptor, is used by some operating systems to indicate that the segment is available (AV = 1) or not available (AV = 0). The D bit indicates how the 80386 through the Core2 instructions access register and memory data in the protected or real mode. If D = 0, the instructions are 16-bit instructions, compatible with the 8086–80286 microprocessors. This means that the instructions use 16-bit offset addresses and 16-bit register by default. This mode is often called the 16-bit instruction mode or DOS mode. If D = 1, the instructions are 32-bit instructions. By default, the 32-bit instruction mode assumes that all offset addresses and all registers are 32 bits. Note that the default for register size and offset address is overridden in both the 16- and 32-bit instruction modes. Both the MSDOS and PCDOS operating systems require that the instructions are always used in the 16-bit instruction mode. Windows 3.1, and any application that was written for it, also requires that the 16-bit instruction mode is selected. Note that the instruction mode is accessible only in a protected mode system such as Windows Vista. More detail on these modes and their application to the instruction set appears in Chapters 3 and 4.

The **access rights byte** (see Figure 2-7) controls access to the protected mode segment. This byte describes how the segment functions in the system. The access rights byte allows complete control over the segment. If the segment is a data segment, the direction of growth is specified. If the segment grows beyond its limit, the microprocessor's operating system program is interrupted, indicating a general protection fault. You can even specify whether a data segment can be written or is write-protected. The code segment is also controlled in a similar fashion and can have reading inhibited to protect software. Again, note that in 64-bit mode there is only a code segment and no other segment descriptor types. A 64-bit flat model program contains its data and stacks in the code segment.

Descriptors are chosen from the descriptor table by the segment register. Figure 2-8 shows how the segment register functions in the protected mode system. The segment register contains a 13-bit selector field, a table selector bit, and a requested privilege level field. The 13-bit selector chooses one of the 8192 descriptors from the descriptor table. The TI bit selects either the global descriptor table (TI = 0) or the local descriptor table (TI = 1). The requested privilege level (RPL) requests the access privilege level of a memory segment. The highest privilege level is 00 and the lowest is 11. If the requested privilege level matches or is higher in priority than the privilege level set by the access rights byte, access is granted. For example, if the

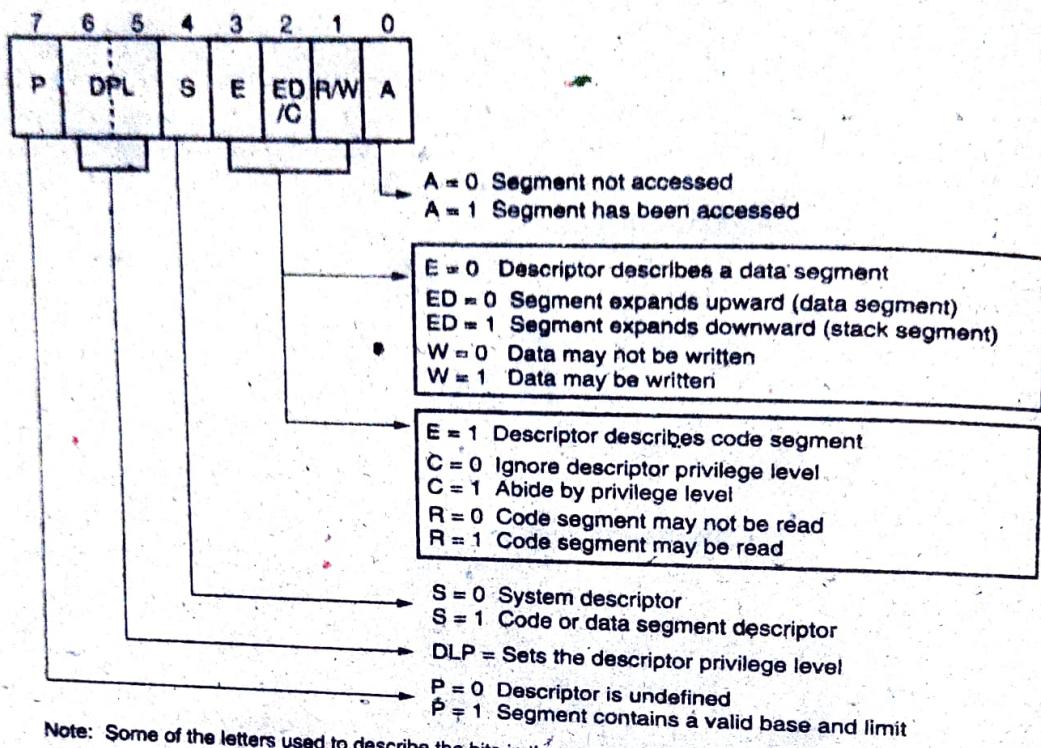


FIGURE 2-7 The access rights byte for the 80286 through Core2 descriptor.

requested privilege level is 10 and the access rights byte sets the segment privilege level at 11, access is granted because 10 is higher in priority than privilege level 11. Privilege levels are used in multiuser environments. Windows uses privilege level 00 (ring 0) for the kernel and driver programs and level 11 (ring 3) for applications. Windows does not use levels 01 or 10. If privilege levels are violated, the system normally indicates an application or privilege level violation.

Figure 2-9 shows how the segment register, containing a selector, chooses a descriptor from the global descriptor table. The entry in the global descriptor table selects a segment in the memory system. In this illustration, DS contains 0008H, which accesses the descriptor number 1 from the global descriptor table using a requested privilege level of 00. Descriptor number 1 contains a descriptor that defines the base address as 0010000H with a segment limit of 000FFH. This means that a value of 0008H loaded into DS causes the microprocessor to use memory locations 0010000H–001000FFH for the data segment with this example descriptor table. Note that descriptor zero is called the null descriptor, must contain all zeros, and may not be used for accessing memory.

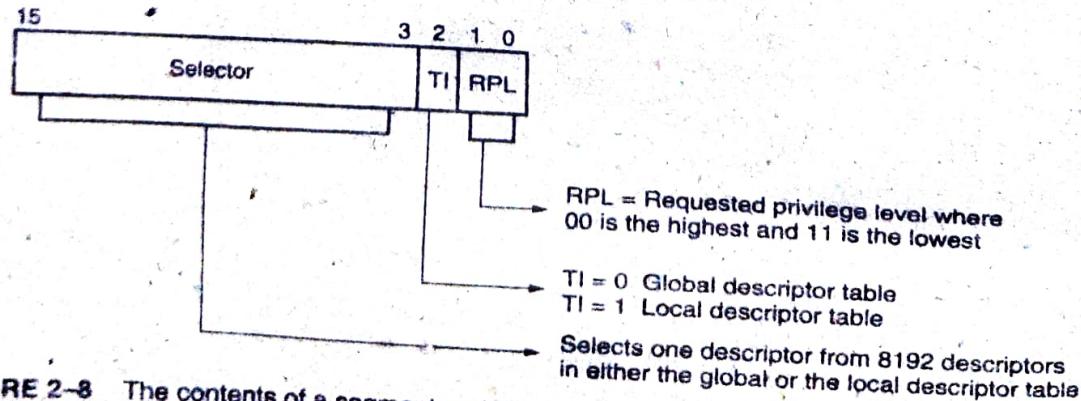


FIGURE 2-8 The contents of a segment register during protected mode operation of the 80286 through Core2 microprocessors.

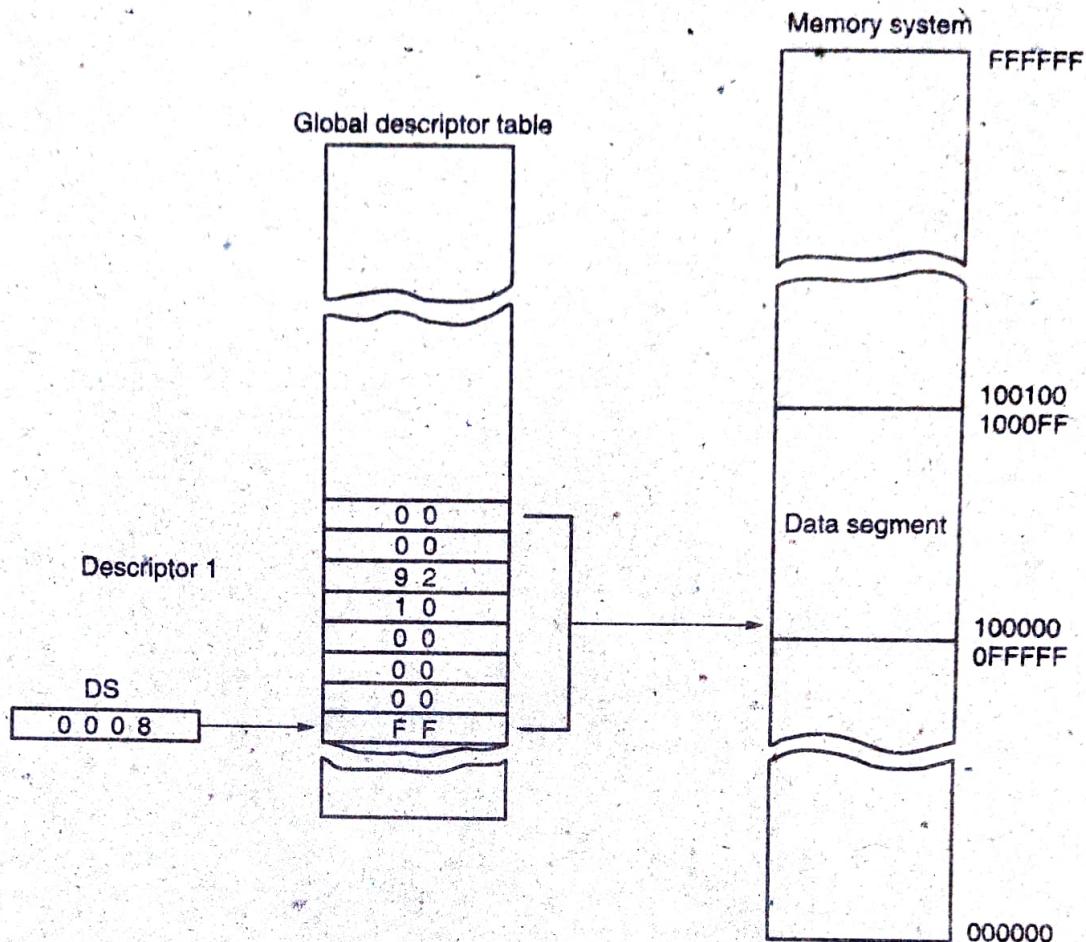


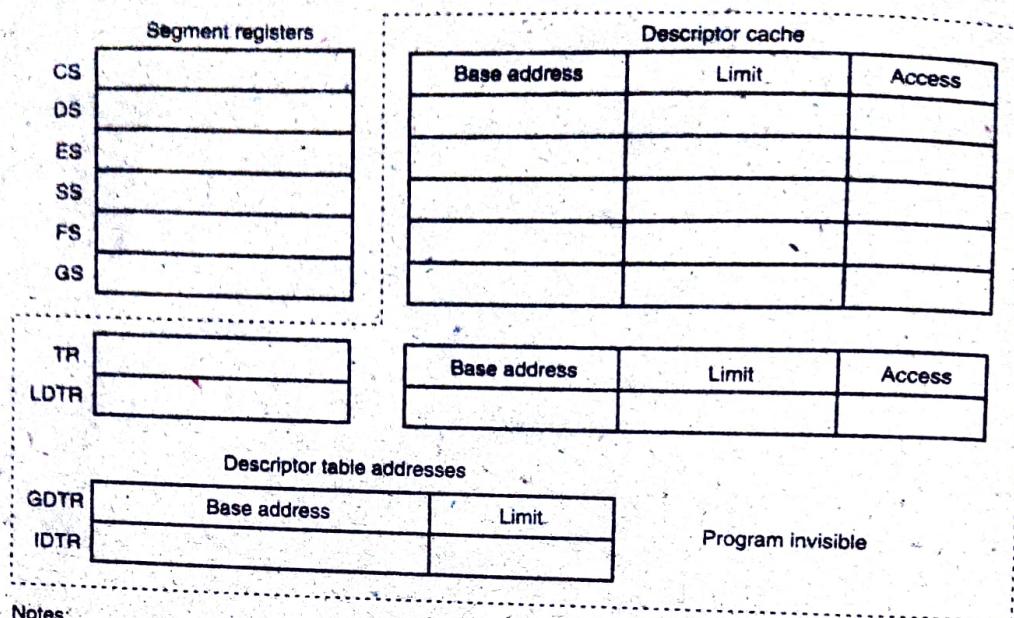
FIGURE 2–9 Using the DS register to select a description from the global descriptor table. In this example, the DS register accesses memory locations 00100000H–001000FFH as a data segment.

Program-Invisible Registers

The global and local descriptor tables are found in the memory system. In order to access and specify the address of these tables, the 80286–Core2 contain program-invisible registers. The program-invisible registers are not directly addressed by software so they are given this name (although some of these registers are accessed by the system software). Figure 2–10 illustrates the program-invisible registers as they appear in the 80286 through the Core2. These registers control the microprocessor when operated in protected mode.

Each of the segment registers contains a program-invisible portion used in the protected mode. The program-invisible portion of these registers is often called cache memory because cache is any memory that stores information. This cache is not to be confused with the level 1 or level 2 caches found with the microprocessor. The program-invisible portion of the segment register is loaded with the base address, limit, and access rights each time the number segment register is changed. When a new segment number is placed in a segment register, the microprocessor accesses a descriptor table and loads the descriptor into the program-invisible portion of the segment register. It is held there and used to access the memory segment until the segment number is again changed. This allows the microprocessor to repeatedly access a memory segment without referring to the descriptor table (hence the term *cache*).

The GDTR (global descriptor table register) and IDTR (interrupt descriptor table register) contain the base address of the descriptor table and its limit. The limit of each descriptor

**Notes:**

1. The 80286 does not contain FS and GS nor the program-invisible portions of these registers.
2. The 80286 contains a base address that is 24-bits and a limit that is 16-bits.
3. The 80386/80486/Pentium/Pentium Pro contain a base address that is 32-bits and a limit that is 20-bits.
4. The access rights are 8-bits in the 80286 and 12-bits in the 80386/80486/Pentium-Core2.

FIGURE 2–10 The program-invisible register within the 80286–Core2 microprocessors.

table is 16 bits because the maximum table length is 64K bytes. When the protected mode operation is desired, the address of the global descriptor table and its limit are loaded into the GDTR.

Before using the protected mode, the interrupt descriptor table and the IDTR must also be initialized. More detail is provided on protected mode operation later in the text. At this point, programming and additional description of these registers are impossible.

The location of the local descriptor table is selected from the global descriptor table. One of the global descriptors is set up to address the local descriptor table. To access the local descriptor table, the LDTR (local descriptor table register) is loaded with a selector, just as a segment register is loaded with a selector. This selector accesses the global descriptor table and loads the address, limit, and access rights of the local descriptor table into the cache portion of the LDTR.

The TR (task register) holds a selector, which accesses a descriptor that defines a task. A task is most often a procedure or application program. The descriptor for the procedure or application program is stored in the global descriptor table, so access can be controlled through the privilege levels. The task register allows a context or task switch in about 17 μ s. Task switching allows the microprocessor to switch between tasks in a fairly short amount of time. The task switch allows multitasking systems to switch from one task to another in a simple and orderly fashion.

2-4

MEMORY PAGING

The **memory paging mechanism** located within the 80386 and above allows any physical memory location to be assigned to any linear address. The **linear address** is defined as the address generated by a program. The **physical address** is the actual memory location accessed by a program. With the memory paging unit, the linear address is invisibly translated to any physical address, which allows an application written to function at a specific address to be relocated through the paging mechanism. It also allows memory to be placed into areas where no memory exists. An example is the upper memory blocks provided by EMM386.EXE in a DOS system.

The EMM386.EXE program reassigns extended memory, in 4K blocks, to the system memory between the video BIOS and the system BIOS ROMS for upper memory blocks. Without the paging mechanism, the use of this area of memory is impossible.

In Windows, each application is allowed a 2G linear address space from location 00000000H-7FFFFFFFH even though there may not be enough memory or memory available at these addresses. Through paging to the hard disk drive and paging to the memory through the memory paging unit, any Windows application can be executed.

Paging Registers

The paging unit is controlled by the contents of the microprocessor's control registers. See Figure 2-11 for the contents of control registers CR0 through CR4. Note that these registers are available to the 80386 through the Core2 microprocessors. Beginning with the Pentium, an additional control register labeled CR4 controls extensions to the basic architecture provided in the Pentium or newer microprocessor. One of these features is a 2M- or a 4M-byte page that is enabled by controlling CR4.

The registers important to the paging unit are CR0 and CR3. The leftmost bit (PG) position of CR0 selects paging when placed at a logic 1 level. If the PG bit is cleared (0), the linear address generated by the program becomes the physical address used to access memory. If the PG bit is set (1), the linear address is converted to a physical address through the paging mechanism. The paging mechanism functions in both the real and protected modes.

CR3 contains the page directory base or root address, and the PCD and PWT bits. The PCD and PWT bits control the operation of the PCD and PWT pins on the microprocessor. If PCD is set (1), the PCD pin becomes a logic one during bus cycles that are not paged. This allows the external hardware to control the level 2 cache memory. (Note that the level 2 cache memory is an internal [on modern versions of the Pentium] high-speed memory that functions as a buffer between the microprocessor and the main DRAM memory system.) The PWT bit also appears on the PWT pin during bus cycles that are not paged to control the write-through cache in the system. The page directory base address locates the directory for the page translation unit. Note that this address locates the page directory at any 4K boundary in the memory system because it is appended internally with 000H. The page directory contains 1024 directory entries of 4 bytes each. Each page directory entry addresses a page table that contains 1024 entries.

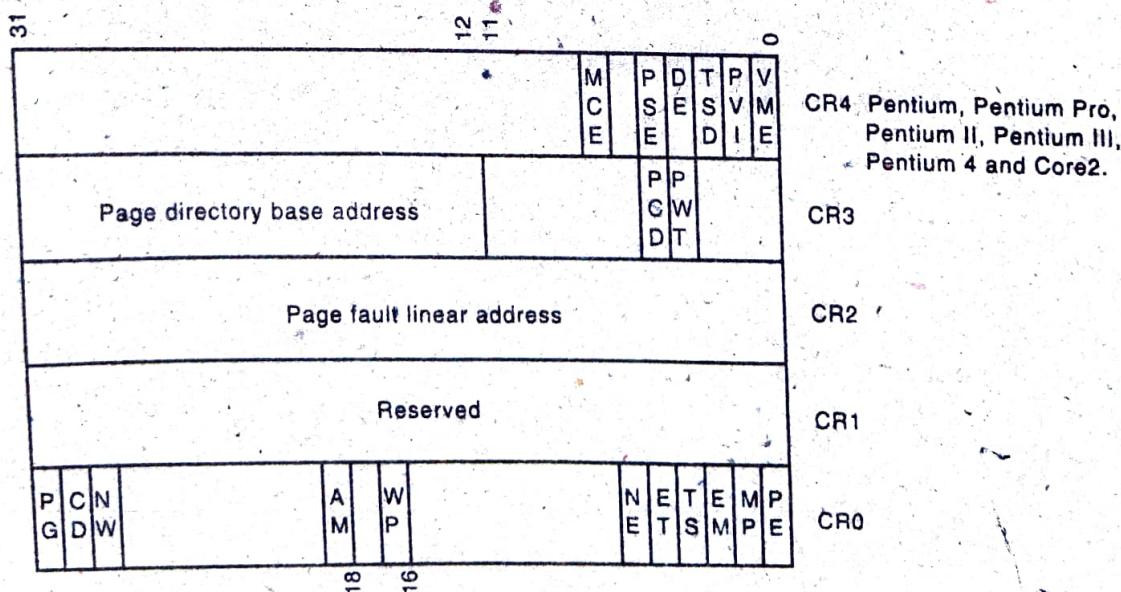


FIGURE 2-11 The control register structure of the microprocessor.

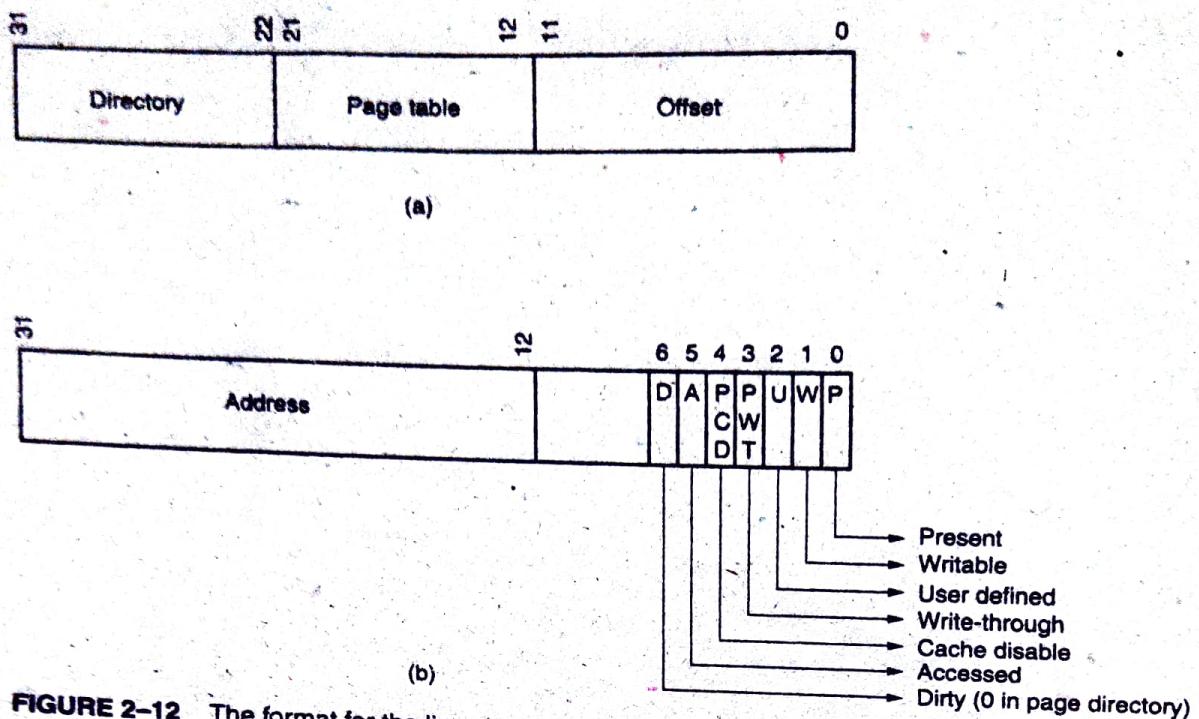


FIGURE 2-12 The format for the linear address (a) and a page directory or page table entry (b).

The linear address, as it is generated by the software, is broken into three sections that are used to access the **page directory entry**, **page table entry**, and **memory page offset address**. Figure 2-12 shows the linear address and its makeup for paging. Notice how the leftmost 10 bits address an entry in the page directory. For linear address 00000000H-003FFFFFH, the first page directory is accessed. Each page directory entry represents or repages a 4M section of the memory system. The contents of the page directory select a page table that is indexed by the next 10 bits of the linear address (bit positions 12-21). This means that address 00000000H-00000FFFH selects page directory entry of 0 and page table entry of 0. Notice this is a 4K-byte address range. The offset part of the linear address (bit positions 0-11) next selects a byte in the 4K-byte memory page. In Figure 2-12, if the page table entry 0 contains address 00100000H, then the physical address is 00100000H-00100FFFH for linear address 00000000H-00000FFFH. This means that when the program accesses a location between 00000000H and 00000FFFH, the microprocessor physically addresses location 00100000H-00100FFFH.

Because the act of repaging a 4K-byte section of memory requires access to the page directory and a page table, which are both located in memory, Intel has incorporated a special type of cache called the **TLB (translation look-aside buffer)**. In the 80486 microprocessor, the cache holds the 32 most recent page translation addresses. This means that the last 32 page table translations are stored in the TLB, so if the same area of memory is accessed, the address is already present in the TLB, and access to the page directory and page tables is not required. This speeds program execution. If a translation is not in the TLB, the page directory and page table must be accessed, which requires additional execution time. The Pentium-Pentium 4 microprocessors contain separate TLBs for each of their instruction and data caches.

The Page Directory and Page Table

Figure 2-13 shows the page directory, a few page tables, and some memory pages. There is only one page directory in the system. The page directory contains 1024 doubleword addresses that locate up to 1024 page tables. The page directory and each page table are 4K bytes in length. If

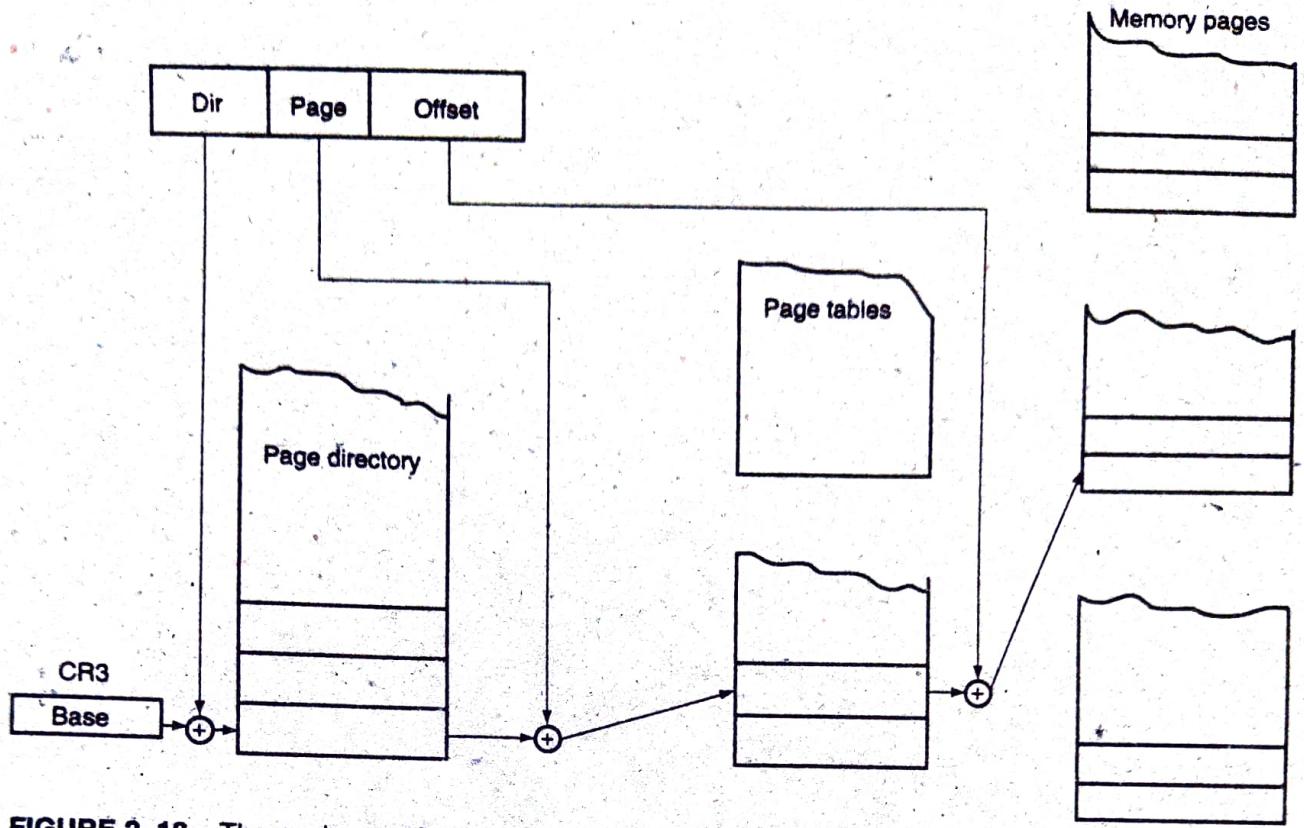
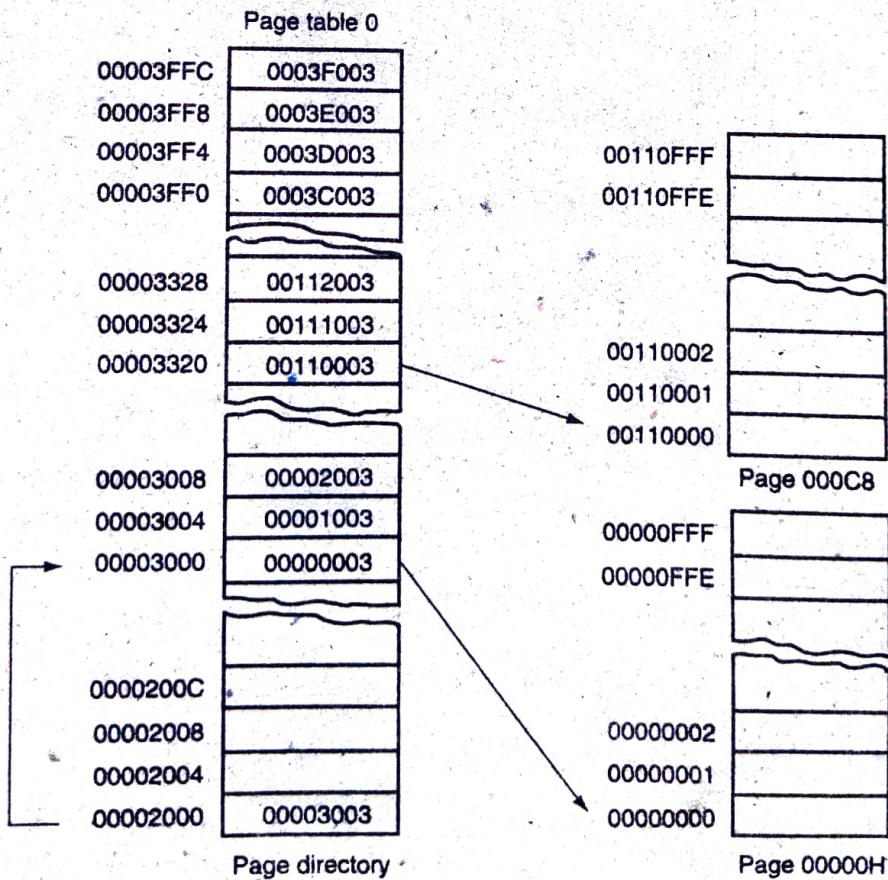


FIGURE 2–13 The paging mechanism in the 80386 through Core2 microprocessors.

FIGURE 2–14 The page directory, page table 0, and two memory pages. Note how the address of page 000C8000–000C9000 has been moved to 00110000–00110FFF.



the entire 4G byte of memory is paged, the system must allocate 4K bytes of memory for the page directory, and 4K times 1024 or 4M bytes for the 1024 page tables. This represents a considerable investment in memory resources.

The DOS system and EMM386.EXE use page tables to redefine the area of memory between locations C8000H-EFFFFH as upper memory blocks. This is done by repaging extended memory to backfill this part of the conventional memory system to allow DOS access to additional memory. Suppose that the EMM386.EXE program allows access to 16M bytes of extended and conventional memory through paging and locations C8000H-EFFFFH must be repaged to locations 110000H-138000H, with all other areas of memory paged to their normal locations. Such a scheme is depicted in Figure 2-14.

Here, the page directory contains four entries. Recall that each entry in the page directory corresponds to 4M bytes of physical memory. The system also contains four page tables with 1024 entries each. Recall that each entry in the page table repages 4K bytes of physical memory. This scheme requires a total of 16K of memory for the four page tables and 16 bytes of memory for the page directory.

As with DOS, the Windows program also repages the memory system. At present, Windows version 3.11 supports paging for only 16M bytes of memory because of the amount of memory required to store the page tables. Newer versions of Windows repage the entire memory system. On the Pentium-Core2 microprocessors, pages can be 4K, 2M, or 4M bytes in length. In the 2M and 4M variations, there is only a page directory and a memory page, but no page table.

2-5

FLAT MODE MEMORY

The memory system in a Pentium-based computer (Pentium 4 or Core2) that uses the 64-bit extensions uses a flat mode memory system. A *flat mode memory* system is one in which there is no segmentation. The address of the first byte in the memory is at 00 0000 0000H and the last location is at FF FFFF FFFFH (address is 40-bits). The flat model does not use a segment register to address a location in the memory. The CS segment register is used to select a descriptor from the descriptor table that defines the access rights of only a code segment. The segment register still selects the privilege level of the software. The flat model does not select the memory address of a segment using the base and limit in the descriptor (see Figure 2-6). In 64-bit mode the actual address is not modified by the descriptor as in 32-bit protected mode. The offset address is the actual physical address in 64-bit mode. Refer to Figure 2-15 for the flat mode memory model.

This form of addressing is much easier to understand, but offers little protection to the system, through the hardware, as did the protected mode system discussed in Section 2.3. The real mode system is *not* available if the processor operates in the 64-bit mode. Protection and paging are allowed in the 64-bit mode. The CS register is still used in the protected mode operation in the 64-bit mode.

In the 64-bit mode if set to IA32 compatibility (when the L bit = 0 is in the descriptor), an address is 64-bits, but since only 40 bits of the address are brought out to the address pins, any address above 40 bits is truncated. Instructions that use a displacement address can only use a 32-bit displacement, which allows a range of $\pm 2G$ from the current instruction. This addressing mode is called RIP relative addressing, and is explained in Chapter 3. The move immediate instruction allows a full 64-bit address and access to any flat mode memory location. Other instructions do not allow access to a location above 4G because the offset address is still 32-bits.

If the Pentium is operated in the full 64-bit mode (where the L = 1 in the descriptor), the address may be 64-bits or 32-bits. This is shown in examples in the next chapter with addressing modes and in more detail in Chapter 4. Most programs today are operated in the IA32 compatible mode so current versions of Windows software operates properly, but this will change in a