80x86 Architecture & Addressing Modes

Saidalavi Kalady & S Sheerazuddin

National Institute of Technology, Calicut

References

1. The Intel Microprocessors : Architecture, Programming, and Interfacing, Barry B. Brey, 8th Ed., Prentice Hall, 2009.
2. Introduction to NASM, Jayaraj P B & Saidalavi Kalady, NIT Calicut, 2019

The First Microprocessor

- The world's first microprocessor, the INTEL 4004, was a 4-bit microprocessor-programmable controller on a chip.
- It addressed a mere 4096, 4-bit-wide memory locations. (A BIT is a binary digit with a value of one or zero. A 4-bit-wide memory location is called a NIBBLE.)
- The 4004 instruction set contained only 45 instructions.
- It was fabricated with the then-current state-of-the-art P-channel MOSFET technology that only allowed it to execute instructions at the slow rate of 50 KIPs (kilo-instructions per second).
- This was slow when compared to the 100,000 instructions executed per second by the 30-ton ENIAC computer in 1946.
- The main difference was that the 4004 weighed much less than an ounce.

The First Microprocessor

- The 4-bit microprocessor debuted in early video game systems and small microprocessor-based control systems.
- One such early video game, a shuffleboard game, was produced by Bailey.
- The main problems with this early microprocessor were its SPEED, WORD WIDTH, and MEMORY SIZE.

The Second Microprocessor

- The evolution of the 4-bit microprocessor ended when Intel released the 4040, an updated version of the earlier 4004.
- The 4040 operated at a higher speed, although it lacked improvements in word width and memory size.
- The 4-bit microprocessor still survives in low-end applications such as microwave ovens and small control systems and is still available from some microprocessor manufacturers.
- Most calculators are still based on 4-bit microprocessors that process 4-bit BCD (binary-coded decimal) codes.

The Age of Microprocessor

- Later in 1971, realizing that the microprocessor was a commercially viable product, Intel Corporation released the 8008 – an extended 8-bit version of the 4004 microprocessor.
- The 8008 addressed an expanded memory size (16K bytes) and contained additional instructions (a total of 48) that provided an opportunity for its application in more advanced systems.
- As engineers developed more demanding uses for the 8008 microprocessor, they discovered that its somewhat small memory size, slow speed, and instruction set limited its usefulness.
- Intel recognized these limitations and introduced the 8080 microprocessor in 1973 – the first of the modern 8-bit microprocessors.
- About six months after Intel released the 8080 microprocessor, **Motorola Corporation** introduced its $\mathrm{MC6800}$ microprocessor.
- The floodgates opened and the 8080 – and, to a lesser degree, the MC6800 – ushered in the age of the microprocessor.

What Was Special about the 8080?

- Not only could the 8080 address more memory and execute additional instructions, but it executed them 10 times faster than the 8008.
- An addition that took 20 μs (50,000 instructions per second) on an 8008-based system required only 2.0 μs (500,000 instructions per second) on an 8080-based system.
- The 8080 also addressed four times more memory (64K bytes) than the 8008 (16K bytes).
- These improvements are responsible for ushering in the era of the 8080 and the continuing saga of the microprocessor.
- Incidentally, the first personal computer, the MITS Altair 8800, was released in 1974.
- The BASIC language interpreter, written for the Altair 8800 computer, was developed in 1975 by Bill Gates and Paul Allen, the founders of Microsoft Corporation.

The 8085 Microprocessor

- In 1977, Intel Corporation introduced an updated version of the 8080 – the 8085.
- The 8085 was to be the last 8-bit, general-purpose microprocessor developed by Intel.
- Although only slightly more advanced than an 8080 microprocessor, the 8085 executed software at an even higher speed.
- An addition that took 2.0 μs (500,000 instructions per second on the 8080) required only 1.3 μs (769,230 instructions per second) on the 8085.
- The main advantages of the 8085 were its internal clock generator, internal system controller, and higher clock frequency.
- This higher level of component integration reduced the 8085's cost and increased its usefulness.

The Modern Microprocessor

- In 1978, Intel released the 8086 microprocessor; a year or so later, it released the 8088.
- Both devices are 16-bit microprocessors, which executed instructions in as little as 400 ns (2.5 MIPs, or 2.5 millions of instructions per second).
- This represented a major improvement over the execution speed of the 8085.
- In addition, the 8086 and 8088 addressed 1M byte of memory, which was 16 times more memory than the 8085.
- This higher execution speed and larger memory size allowed the 8086 and 8088 to replace smaller minicomputers in many applications.
- One other feature found in the 8086/8088 was a small 4- or 6-byte instruction $\textsc{cache}$ or queue that prefetched a few instructions before they were executed.
- The queue sped the operation of many sequences of instructions and proved to be the basis for the much larger instruction caches found in modern microprocessors.

The Modern Microprocessor

- The increased memory size and additional instructions in the 8086 and 8088 have led to many sophisticated applications for microprocessors.
- Improvements to the instruction set included multiply and divide instructions, which were missing on earlier microprocessors.
- In addition, the number of instructions increased from 45 on the 4004, to 246 on the 8085, to well over 20,000 variations on the 8086 and 8088 microprocessors.
- Note that these microprocessors are called CISC (complex instruction set computers) because of the number and complexity of instructions.
- The additional instructions eased the task of developing efficient and sophisticated applications, even though the number of instructions are at first overwhelming and time-consuming to learn.
- The 16-bit microprocessor also provided more internal register storage space than the 8-bit microprocessor.

The Modern Microprocessor

- The popularity of the Intel family was ensured in 1981, when IBM Corporation decided to use the 8088 microprocessor in its personal computer.
- Applications such as spreadsheets, word processors, spelling checkers, and computer-based thesauruses were memory-intensive and required more than the 64K bytes of memory found in 8-bit microprocessors to execute efficiently.
- The 16-bit 8086 and 8088 provided 1M byte of memory for these applications.
- Soon, even the 1M-byte memory system proved limiting for large databases and other applications.
- This led Intel to introduce the 80286 microprocessor, an updated 8086, in 1983.

The 80286 Microprocessor

- The 80286 microprocessor (also a 16-bit architecture microprocessor) was almost identical to the 8086 and 8088, except it addressed a 16M-byte memory system instead of a 1M-byte system.
- The instruction set of the 80286 was almost identical to the 8086 and 8088, except for a few additional instructions that managed the extra 15M bytes of memory.
- The clock speed of the 80286 was increased, so it executed some instructions in as little as 250 ns (4.0 MIPs) with the original release 8.0 MHz version.
- Some changes also occurred to the internal execution of the instructions, which led to an eight fold increase in speed for many instructions when compared to 8086/8088 instructions.

The 32-Bit Microprocessor

- Applications began to demand faster microprocessor speeds, more memory, and wider data paths.
- This led to the arrival of the 80386 in 1986 by Intel Corporation.
- The 80386 was Intel's first practical 32-bit microprocessor that contained a 32-bit data bus and a 32-bit memory address.
- Through these 32-bit buses, the 80386 addressed up to 4G bytes of memory.

The Pentium Microprocessor

- The Pentium, introduced in 1993, was similar to the 80386 and 80486 microprocessors.
- The two introductory versions of the Pentium operated with a clocking frequency of 60 MHz and 66 MHz, and a speed of 110 MIPs, with a higher-frequency 100 MHz one and one-half clocked version that operated at 150 MIPs.
- The double-clocked Pentium, operating at 120 MHz and 133 MHz, was also available, as were higher-speed versions.
- Another difference was that the cache size was increased to 16K bytes from the 8K cache found in the basic version of the 80486.

The Pentium Microprocessor

- The Pentium contained an 8K-byte instruction cache and an 8K-byte data cache, which allowed a program that transfers a large amount of memory data to still benefit from a cache.
- The memory system contained up to 4G bytes, with the data bus width increased from the 32 bits found in the 80386 and 80486 to a full 64 bits.
- The data bus transfer speed was either 60 MHz or 66 MHz, depending on the version of the Pentium.
- This wider data bus width accommodated double-precision floating-point numbers used for modem high-speed, vector-generated graphical displays.

Manufacturer	*Part Number*	*Data Bus Width*	*Memory Size*
Intel	8048	8	2K internal
	8051	8	8K internal
	8085A	8	64K
	8086	16	1M
	8088	8	1M
	8096	16	8K internal
	80186	16	1M
	80188	8	1M
	80251	8	16K internal
	80286	16	16M
	80386EX	16	64M
	80386DX	32	4G
	80386SL	16	32M
	80386SLC	16	32M + 8K cache
	80386SX	16	16M
	80486DX/DX2	32	4G + 8K cache
	80486SX	32	4G + 8K cache
	80486DX4	32	4G + 16 cache
	Pentium	64	4G + 16K cache
	Pentium OverDrive	32	4G + 16K cache
	Pentium Pro	64	64G + 16K L1 cache + 256K L2 cache
	Pentium II	64	64G + 32K L1 cache + 256K L2 cache
	Pentium III	64	64G + 32K L1 cache + 256K L2 cache
	Pentium 4	64	64G+32K L1 cache+ 512K L2 cache (or larger) (1T for 64-bit extensions)
	Pentium4 D (Dual Core)	64	1T + 32K L1 cache + 2 or 4 M L2 cache
	Core2	64	1T + 32K L1 cache + a shared 2 or 4 M L2 cache
	Itanium (Dual Core)	128	1T + 2.5 M L1 and L2 cache + 24 M L3 cache

Figure: The Many Microprocessors

Data Formats

- Successful programming requires a precise understanding of data formats.
- In this section, many common computer data formats are described as they are used with the Intel family of microprocessors.
- Commonly, data appear as ASCII, UNICODE, BCD, SIGNED and UNSIGNED INTEGERS, and FLOATING-POINT NUMBERS (real numbers).
- Other forms are available, but are not presented here because they are not commonly found.

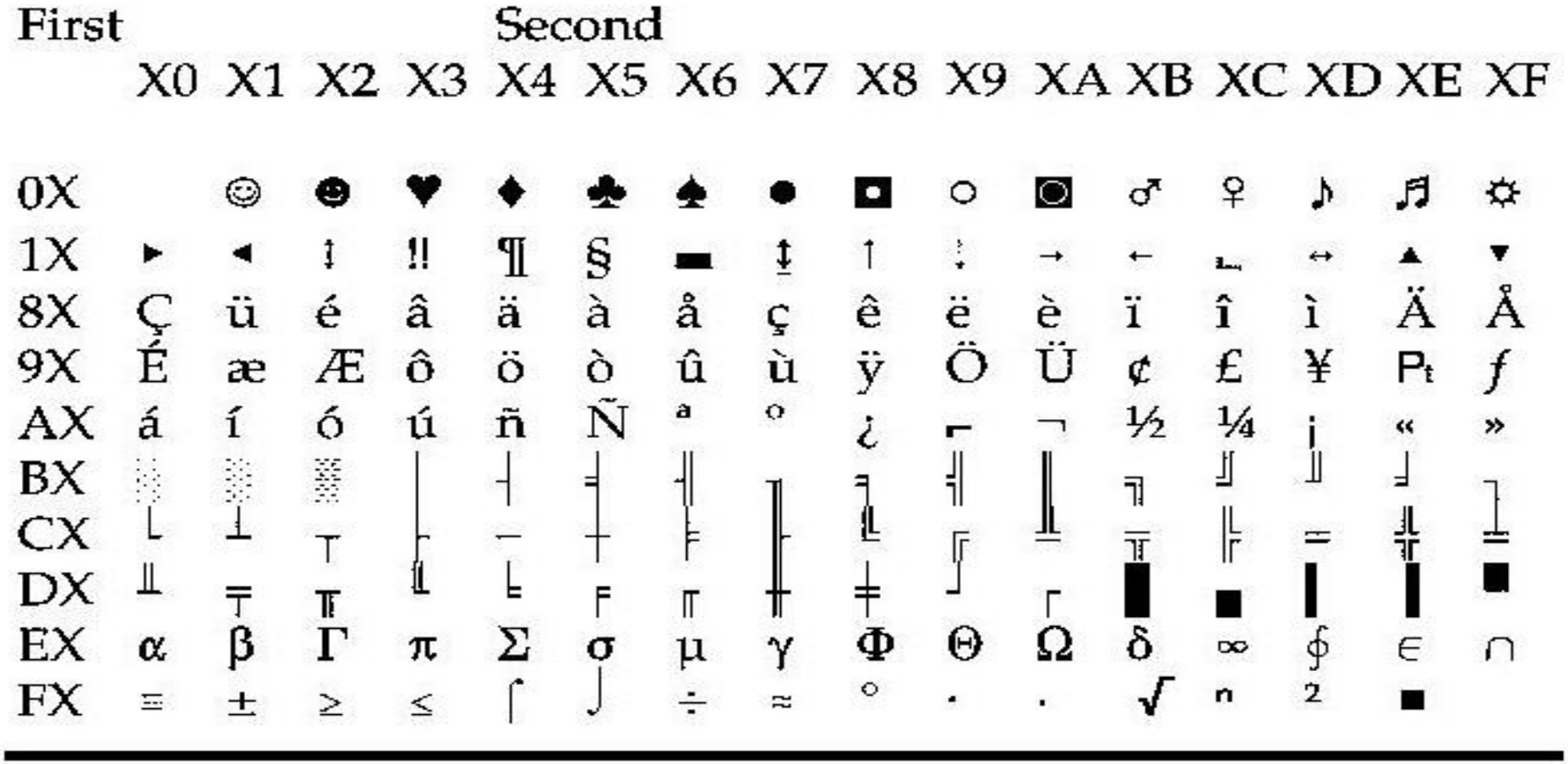
ASCII & Unicode

- ASCII (American Standard Code for Information Interchange) data represent alphanumeric characters in the memory of a computer system.
- The standard ASCII code is a 7-bit code, with the eighth and most significant bit used to hold parity in some antiquated systems.
- If ASCII data are used with a printer, the most significant bits are a 0 for alphanumeric printing and 1 for graphics printing.
- In the personal computer, an extended ASCII character set is selected by placing a 1 in the leftmost bit.
- The second Table shows the extended ASCII character set, using code 80H–FFH.
- The extended ASCII characters store some foreign letters and punctuation, Greek characters, mathematical characters, box-drawing characters, and other special characters.

ASCII Characters

	Second															
	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF
First																
0X	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1X	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EMS	SUB	ESC	FS	GS	RS	US
2X	SP	!	"	#	$	%	&	'	(	)	*	+	,	–	.	/
3X	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4X	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5X	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6X	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7X	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	⋮⋮⋮

Extended ASCII Characters



Binary Coded Decimal (BCD)

- Binary-coded decimal (BCD) information is stored in either packed or unpacked forms.
- Packed BCD data are stored as two digits per byte and unpacked BCD data are stored as one digit per byte.
- The range of a BCD digit extends from 0000_2 to 1001_2 , or 0–9 decimal. Unpacked BCD data are returned from a keypad or keyboard.
- Packed BCD data are used for some of the instructions included for BCD addition and subtraction in the instruction set of the microprocessor.
- The Table below shows some decimal numbers converted to both the packed and unpacked BCD forms.
- Applications that require BCD data are point-of-sales terminals and almost any device that performs a minimal amount of simple arithmetic.
- If a system requires complex arithmetic, BCD data are seldom used because there is no simple and efficient method of performing complex BCD arithmetic.

Packed and Unpacked BCD Data

Decimal	*Packed*		*Unpacked*		
12	0001 0010		0000 0001	0000 0010	
623	0000 0110	0010 0011	0000 0110	0000 0010	0000 0011
910	0000 1001	0001 0000	0000 1001	0000 0001	0000 0000

Byte-Sized Data

- Byte-sized data are stored as unsigned and signed integers.
- The Figure below illustrates both the unsigned and signed forms of the byte-sized integer.
- The difference in these forms is the weight of the leftmost bit position.
- Its value is 128 for the unsigned integer and minus 128 for the signed integer.
- In the signed integer format, the leftmost bit represents the sign bit of the number, as well as a weight of minus 128.
- For example, 80H represents a value of 128 as an unsigned number; as a signed number, it represents a value of minus 128.
- Unsigned integers range in value from 00H to FFH (0–255). Signed integers range in value from -128 to 0 to +127.

Byte Sized Data

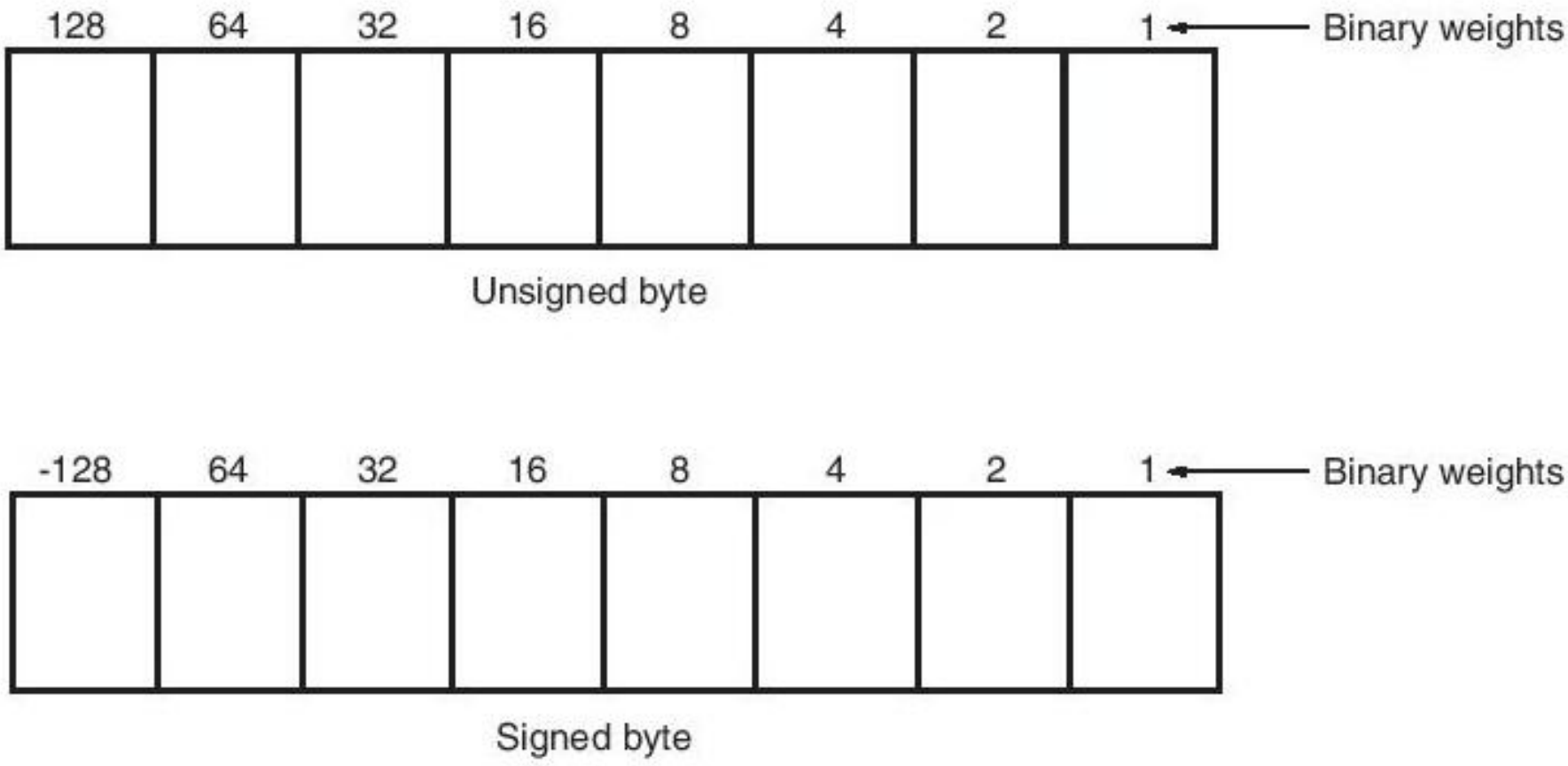


Figure: The unsigned and signed bytes illustrating the weights of each binary-bit position.

Word Sized Data

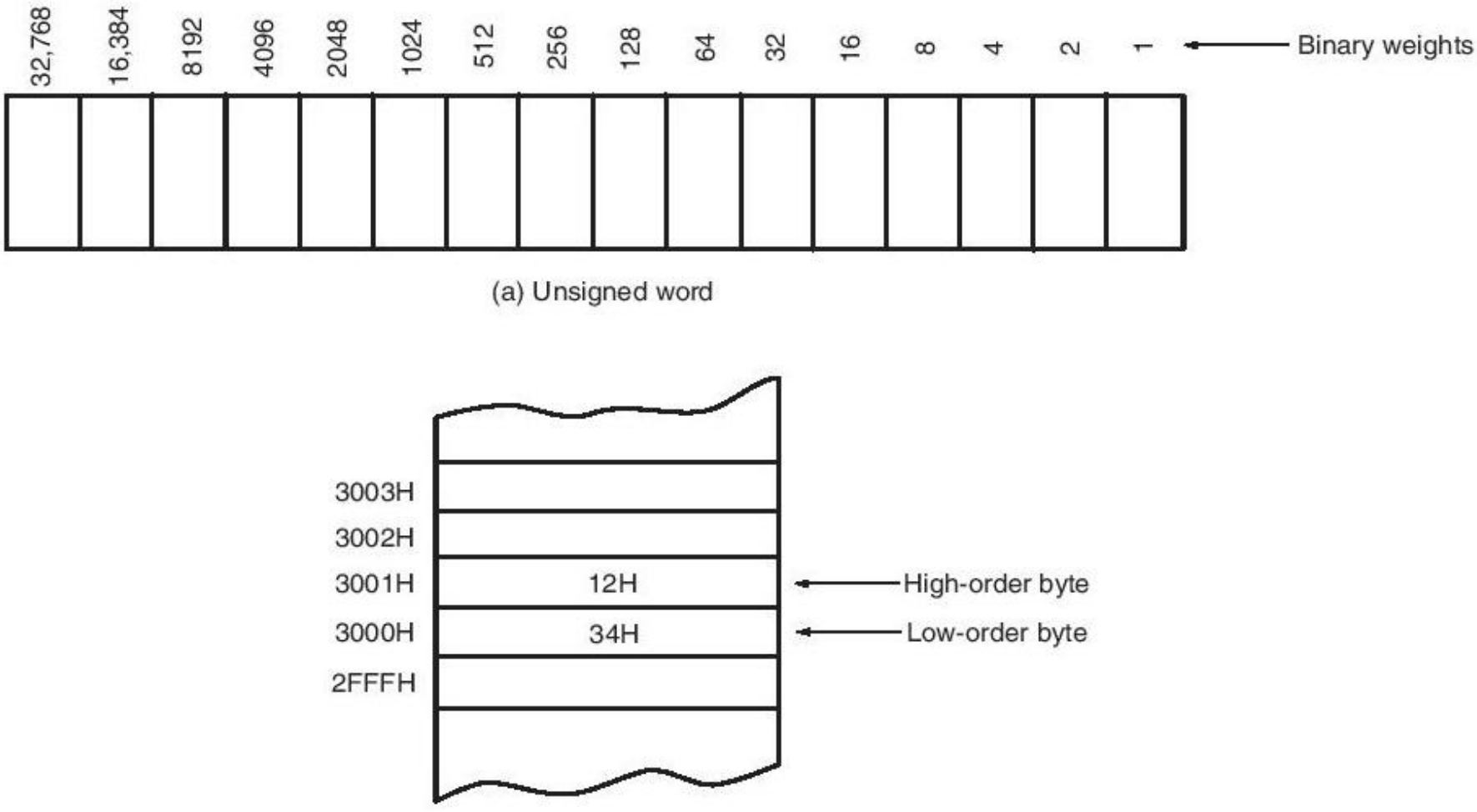


Figure: The storage format for a 16-bit word in (a) a register and (b) two bytes of memory.

Double word Sized Data

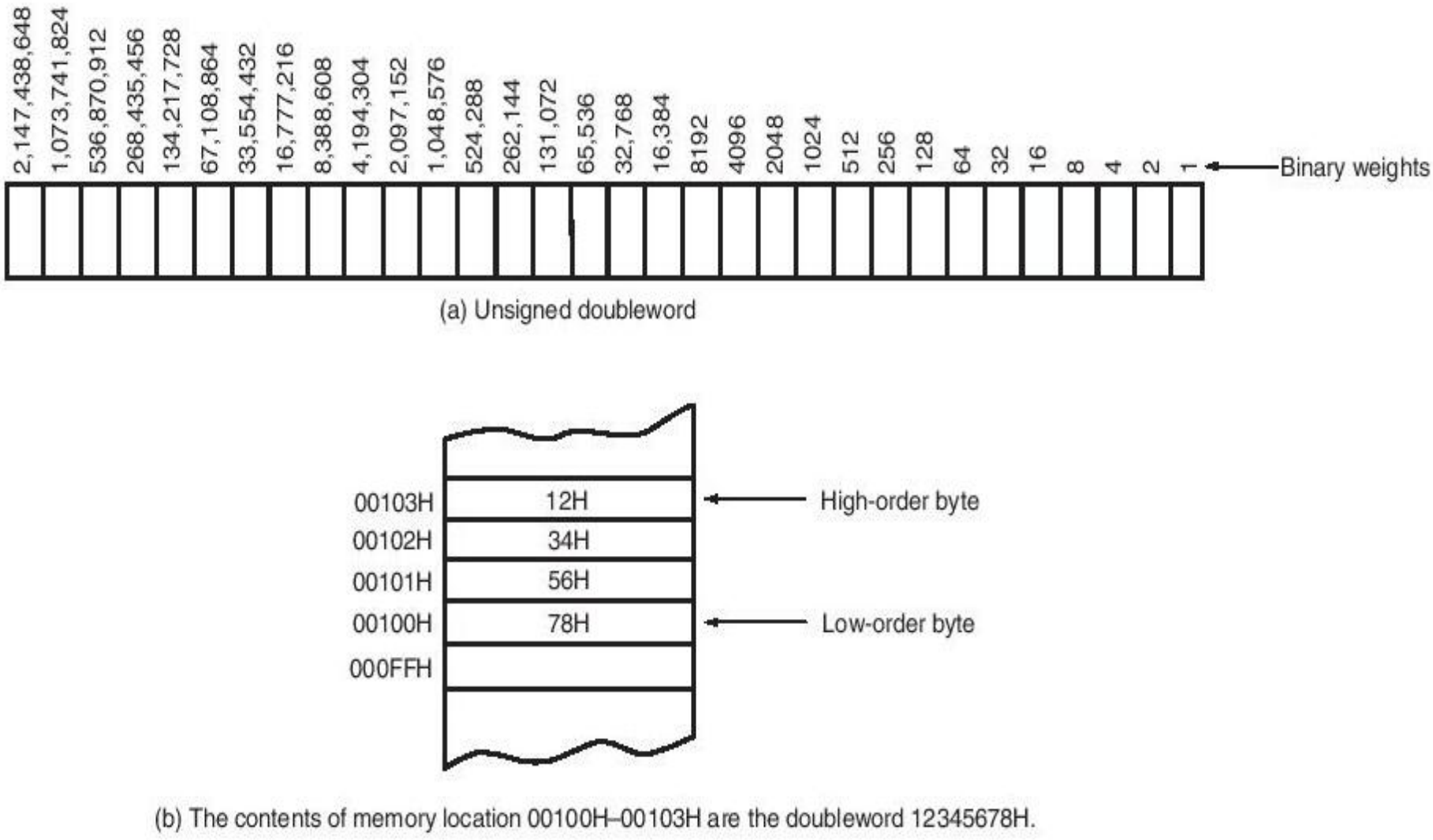


Figure: The storage format for a 32-bit word in (a) a register and (b) four bytes of memory.

Real Numbers

- Because many high-level languages use the Intel family of microprocessors, real numbers are often encountered.
- A real number, or a FLOATING-POINT NUMBER, as it is often called, contains two parts: a mantissa, SIGNIFICAND, or fraction; and an EXPONENT.
- The Figure below depicts both the 4- and 8-byte forms of real numbers as they are stored in any Intel system.
- Note that the 4-byte number is called SINGLE-PRECISION and the 8-byte form is called DOUBLE-PRECISION.
- The form presented here is the same form specified by the IEEE 10 standard, IEEE-754, version 10.0.

Real Numbers

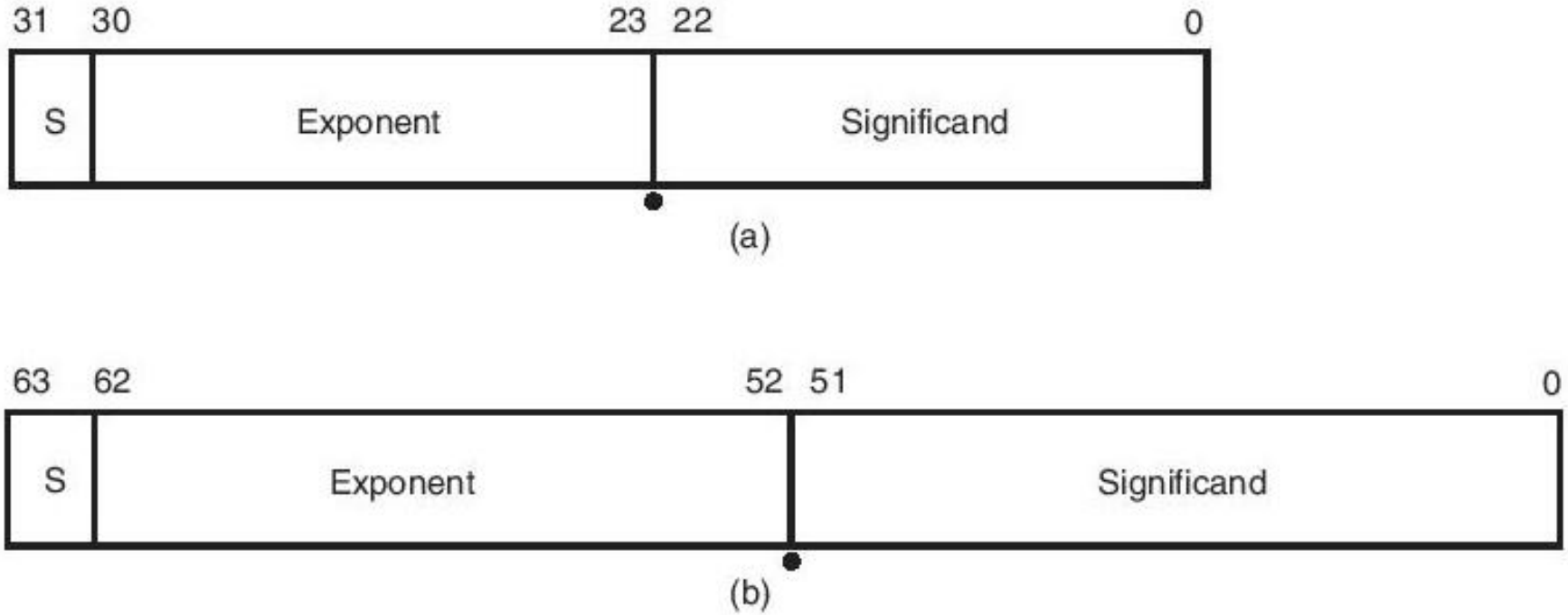


Figure: The floating-point numbers in (a) single-precision using a bias of 7FH and (b) double-precision using a bias of 3FFH.

Real Numbers

- Simple arithmetic indicates that it should take 33 bits to store all three pieces of data.
- Not true—the 24-bit mantissa contains an implied (hidden) one-bit that allows the mantissa to represent 24 bits while being stored in only 23 bits.
- The hidden bit is the first bit of the normalized real number.
- When normalizing a number, it is adjusted so that its value is at least 1, but less than 2.
- For example, if 12 is converted to binary (1100_2), it is normalized and the result is 1.1×2^3.
- The whole number 1 is not stored in the 23-bit mantissa portion of the number; the 1 is the hidden one-bit.
- The Table below shows the single-precision form of this number and others.

Real Numbers

- The exponent is stored as a biased exponent.
- With the single-precision form of the real number, the bias is 127 (7FH) and with the double-precision form, it is 1023 (3FFH).
- The bias and exponent are added before being stored in the exponent portion of the floating-point number.
- In the previous example, there is an exponent of 2^3 , represented as a biased exponent of $127 + 3$ or 130 (82H) in the single-precision form, or as 1026 (402H) in the double-precision form.

Single Precision Real Numbers : Examples

Decimal	*Binary*	*Normalized*	*Sign*	*Biased Exponent*	*Mantissa*
+12	1100	1.1×2^3	0	10000010	10000000 00000000 00000000
−12	1100	1.1×2^3	1	10000010	10000000 00000000 00000000
+100	1100100	1.1001×2^6	0	10000101	10010000 00000000 00000000
−1.75	1.11	1.11×2^0	1	01111111	11000000 00000000 00000000
+0.25	0.01	1.0×2^{-2}	0	01111101	00000000 00000000 00000000
+0.0	0	0	0	00000000	00000000 00000000 00000000

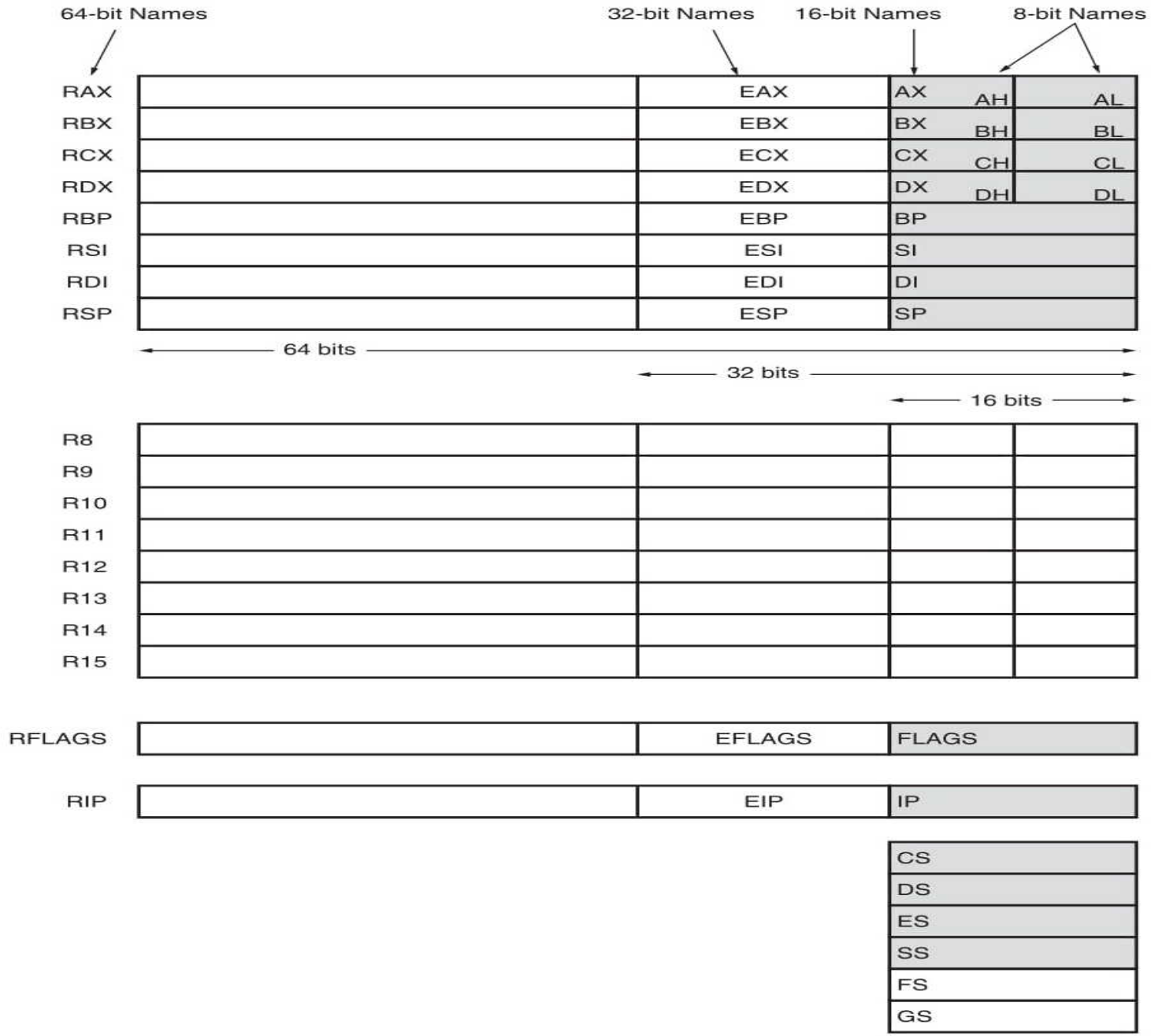
Real Numbers

- There are two exceptions to the rules for floating-point numbers.
- The number 0.0 is stored as all zeros.
- The number infinity is stored as all ones in the exponent and all zeros in the mantissa.
- The sign-bit indicates either a positive or a negative infinity.

80x86 Architecture

- Before a program is written or any instruction investigated, the internal configuration of the microprocessor must be known.
- Here we detail 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 programming model of the 8086 and above is considered to be PROGRAM VISIBLE because its registers are used during application programming and are specified by the instructions.
- Other registers are considered to be PROGRAM INVISIBLE because they are not addressable directly during applications programming, but may be used indirectly during system programming.

The Programming Model



General-purpose (Multipurpose) Registers

- General purpose registers are used to store temporary data within the microprocessor.
- There are eight general purpose registers: EAX, EBX, ECX, EDX, EBP, ESI, EDI, ESP.
- We can refer to the lower 8 and 16 bits of these registers. This is to maintain the backward compatibility of instruction sets.
- These registers are also known as scratchpad area as they are used by the processor to store intermediate values in a calculation and also for storing address locations.

General-purpose (Multipurpose) Registers

The General Purpose Registers are used for :

1. EAX: Accumulator Register - Contains the value of some operands in some operations (E.g.: multiplication).
2. EBX: Base Register - Pointer to some data in Data Segment.
3. ECX: Counter Register - Acts as loop counter, used in string operations etc.
4. EDX: Used as pointer to I/O ports.
5. ESI: Source Index - Acts as source pointer in string operations. It can also act as a pointer in Data Segment (DS).
6. EDI: Destination Index - Acts as destination pointer in string operations. It can also act as a pointer in Extra Segment (ES).
7. ESP: Stack Pointer - Always points to the top of system stack.
8. EBP: Base Pointer - It points to the starting of system stack (ie.bottom/base of stack).

Special-purpose Registers

FLAGS are special purpose registers inside the CPU that contains the status of CPU / the status of last operation executed by the CPU. Some of the bits in FLAGS need special mention:

- Carry Flag: When a processor does a calculation, if there is a carry then the Carry Flag will be set to 1.
- Zero Flag: If the result of the last operation was zero, Zero Flag will be set to 1, else it will be zero.
- Sign Flag : If the result of the last signed operation is negative then the Sign Flag is set to 1, else it will be zero.
- Parity Flag: If there are odd number of ones in the result of the last operation, parity flag will be set to 1.
- Interrupt Flag: If interrupt flag is set to 1, then only it will listen to external interrupts.

FLAGS Register

The other bits in FLAGS register are:

- Auxiliary Carry Flag: The auxiliary carry holds the carry (half-carry) after addition or the borrow after subtraction between bit positions 3 and 4 of the result.
- Trap Flag: If the Trap flag is enabled, the microprocessor interrupts the flow of the program on conditions as indicated by the debug registers and control registers.
- Direction Flag: 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.
- Overflow Flag: Overflows occur when signed numbers are added or subtracted. An overflow indicates that the result has exceeded the capacity of the machine.

FLAGS Register

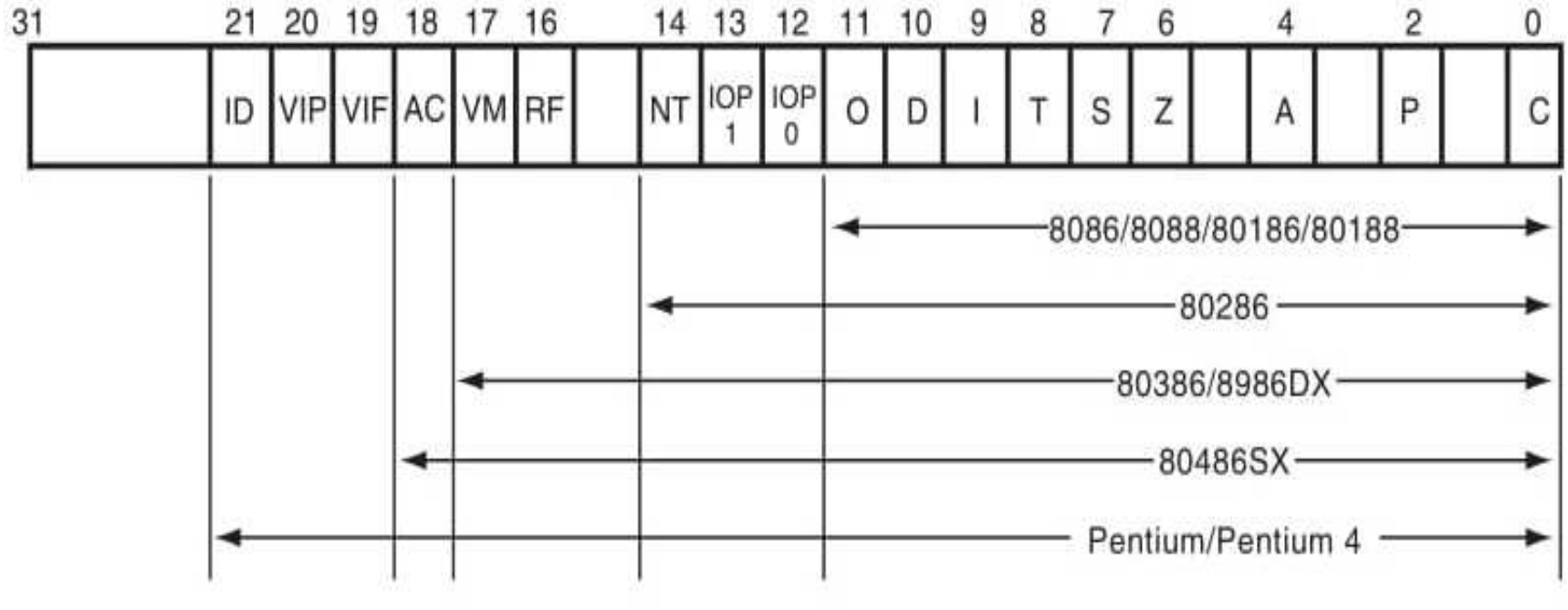


Figure: The EFLAGS and FLAGS register counts

Special-purpose Registers

- Another special-purpose register is the EIP.
- EIP is the INSTRUCTION POINTER, it points to the next instruction to be executed.
- In memory there are basically two classes of things stored: DATA and PROGRAM.
- When we start a program, it will be copied into the main memory and EIP is the pointer which points to the starting of this program in memory and execute each instruction sequentially.
- Branch statements like JMP, RET, CALL, JNZ alter the value of EIP.

Segment Registers

- In 80x86 processors, for accessing the memory there are two types of registers used; Segment Register and Offset.
- Segment register contains the base address of a particular data section and Offset will contain how many bytes should be displaced from the segment register to access the particular data.
- CS contains the base address of Code Segment and EIP is the offset. It keeps on updating while executing each instruction.
- SS or Stack Segment contains the address of top most part of system stack. ESP and EBP are the offset for SS.

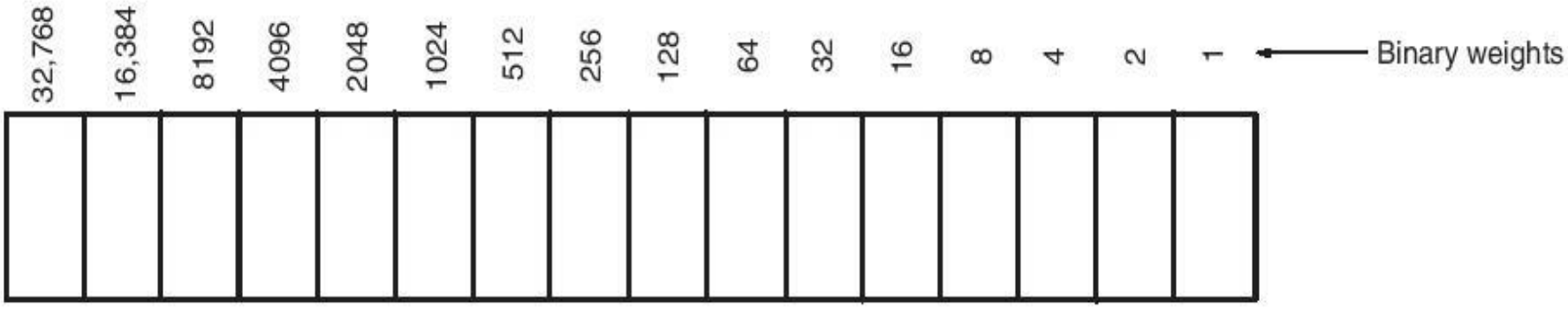
Segment Registers

- The segment registers DS, ES, FS and GS acts as BASE REGISTERS for a lot of data operations like array addressing, string operations etc.
- ESI, EDI and EBX register can act as OFFSETS for these segment registers.
- Unlike other registers, Segment registers are still 16 bit wide in 32-bit processors.
- In modern 32 bit processor the segment address is just an entry into a DESCRIPTOR TABLE in memory and using the offset it gets the exact memory locations through some manipulations.

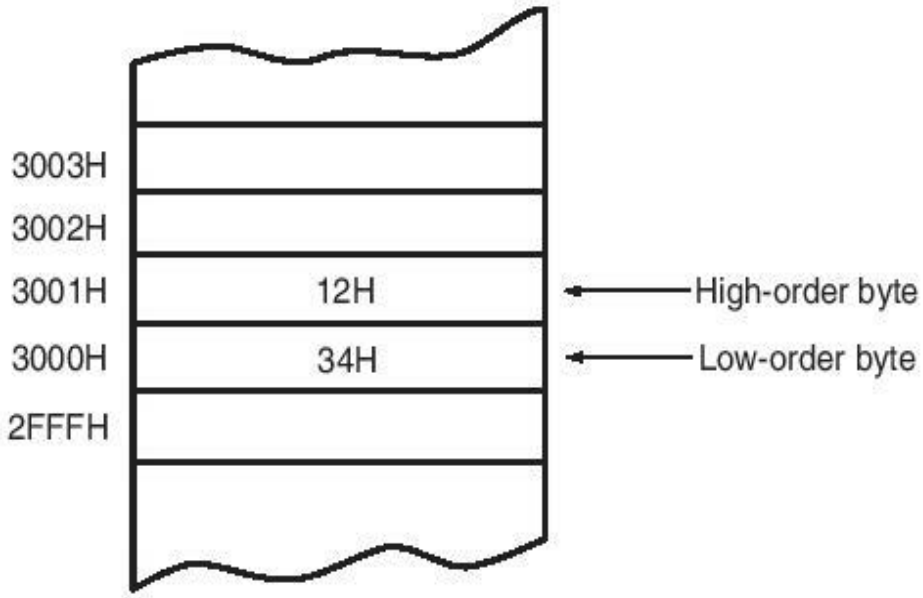
Little-Endian vs. Big-Endian

- A word (16-bits) is formed with two bytes of data.
- The least significant byte is always stored in the lowest-numbered memory location, and the most significant byte is stored in the highest.
- This method of storing a number is called the little endian format.
- An alternate method, not used with the Intel family of microprocessors, is called the big endian format.
- In the big endian format, numbers are stored with the lowest location containing the most significant data.
- The big endian format is used with the Motorola family of microprocessors.

Little-Endian Format



(a) Unsigned word



(b) The contents of memory location 3000H and 3001H are the word 1234H.

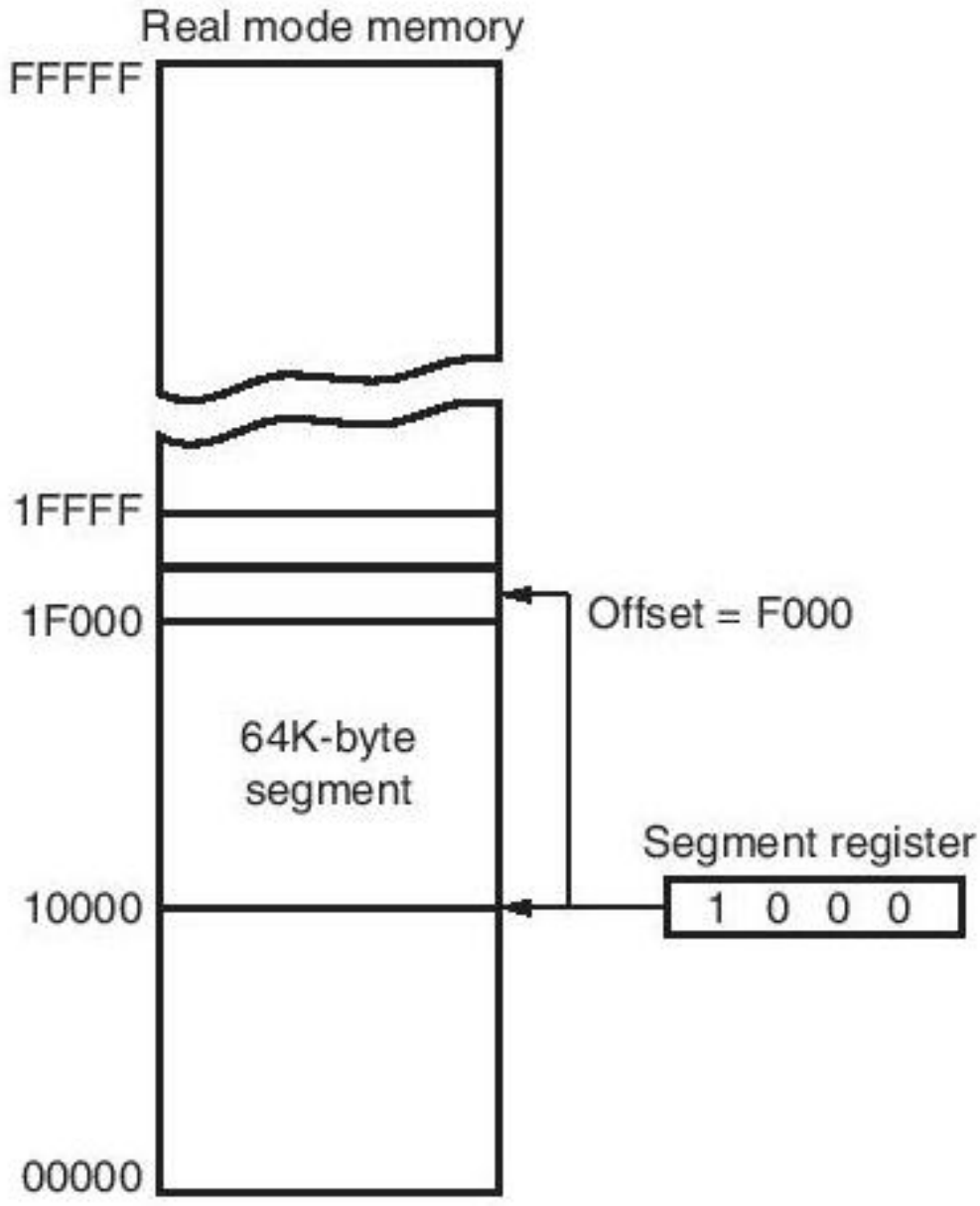
Execution Modes

- The 80286 and above operate in either the REAL or PROTECTED MODE.
- Only the 8086 and 8088 operate exclusively in the real mode.
- Real mode operation allows the microprocessor to address only the first 1M byte of memory space.
- Note that the first 1M byte of memory is called the REAL MEMORY, CONVENTIONAL MEMORY, or DOS MEMORY SYSTEM.
- 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.
- In all cases, each of these microprocessors begins operation in the real mode by default whenever power is applied or the microprocessor is reset.

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.

Real-mode Addressing



Segments and Offsets

- The Figure above 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 IFFFFH—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.
- The segment register contains 1000H, yet it addresses a starting segment at location 10000H.

Segments and Offsets

- 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.

Default Segment and Offset Registers

- The microprocessor has a set of rules that apply to segments whenever memory is addressed.
- These rules 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.

Default Segment and Offset Registers

- 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.

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 8bit/16 bit number	Data Address
ES	DI for string instructions	String Destination Address

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, EDI, ESI, an 8bit/16 bit number	Data Address
ES	EDI for string instructions	String Destination Address
FS	No default	General Address
GS	No default	General Address

Segment and Offset Addressing Scheme Allows Relocation

- 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.

QUESTIONS

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.
- 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.
- 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.

Protected-mode Memory Addressing

- 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.

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.

Selectors and Descriptors

- 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.
- A global descriptor may be called SYSTEM DESCRIPTOR and call a local descriptor 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 $64\mathrm{T}$ bytes.

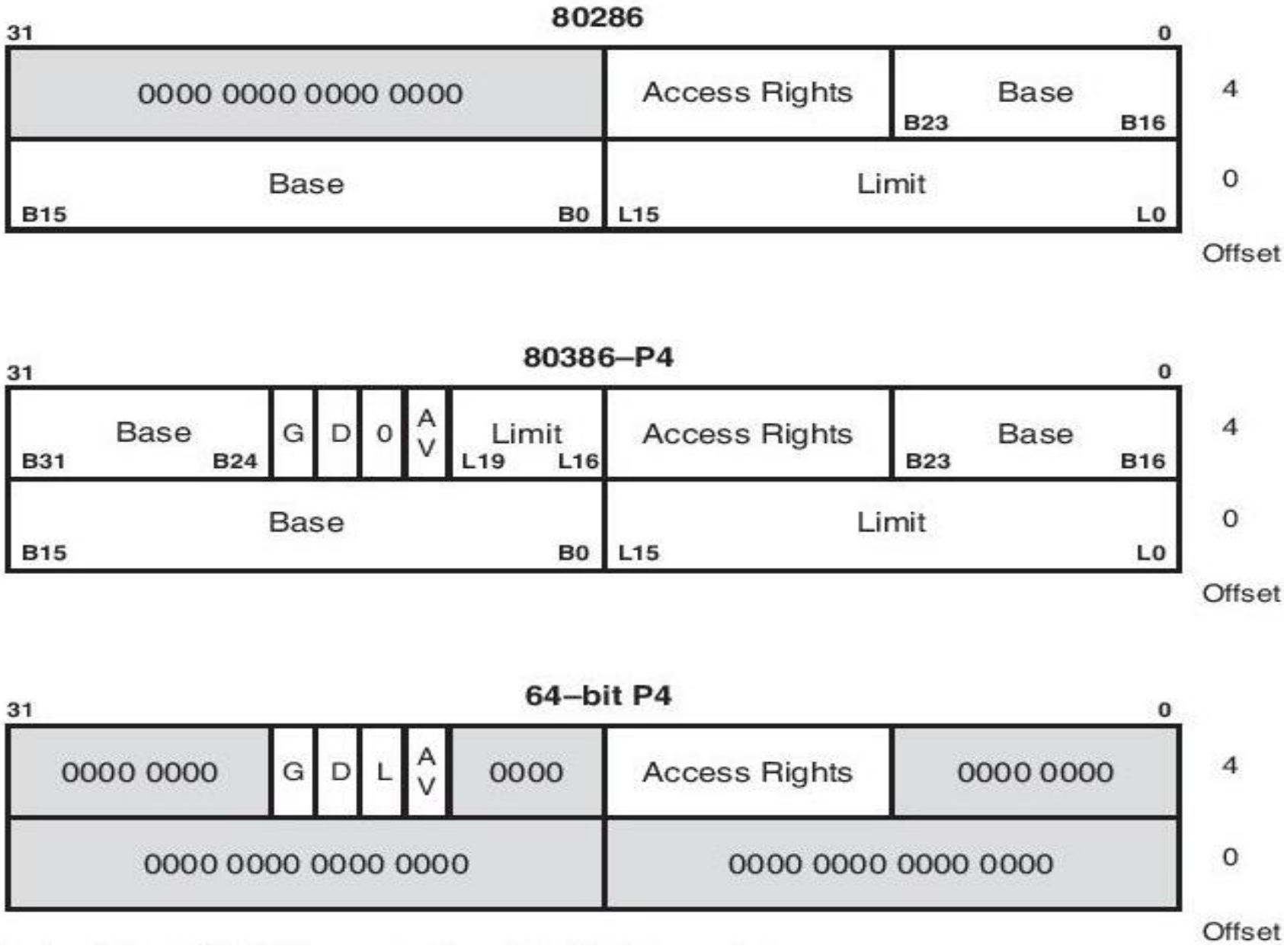


Figure: The 80286 through Core2 64-bit descriptors

Selectors and Descriptors

- The Figure above shows the format of a descriptor for the 80286 through the Core2.
- Each descriptor is 8 bytes in length, so the global and local descriptor tables are each a maximum of $64\mathrm{K}$ 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.
- The paragraph boundary limitation is removed in these microprocessors when operated in the protected mode, so segments may begin at any address.

Selectors and Descriptors

- 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.
- 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.

Selectors and Descriptors

- 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 $\mathrm{GRANULARITY\ BIT}$.
- 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.

Selectors and Descriptors

- In the 64-bit descriptor, the L 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.
- 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.

Selectors & Descriptors

- 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.
- The default for register size and offset address is overridden in both the 16- and 32-bit instruction modes.

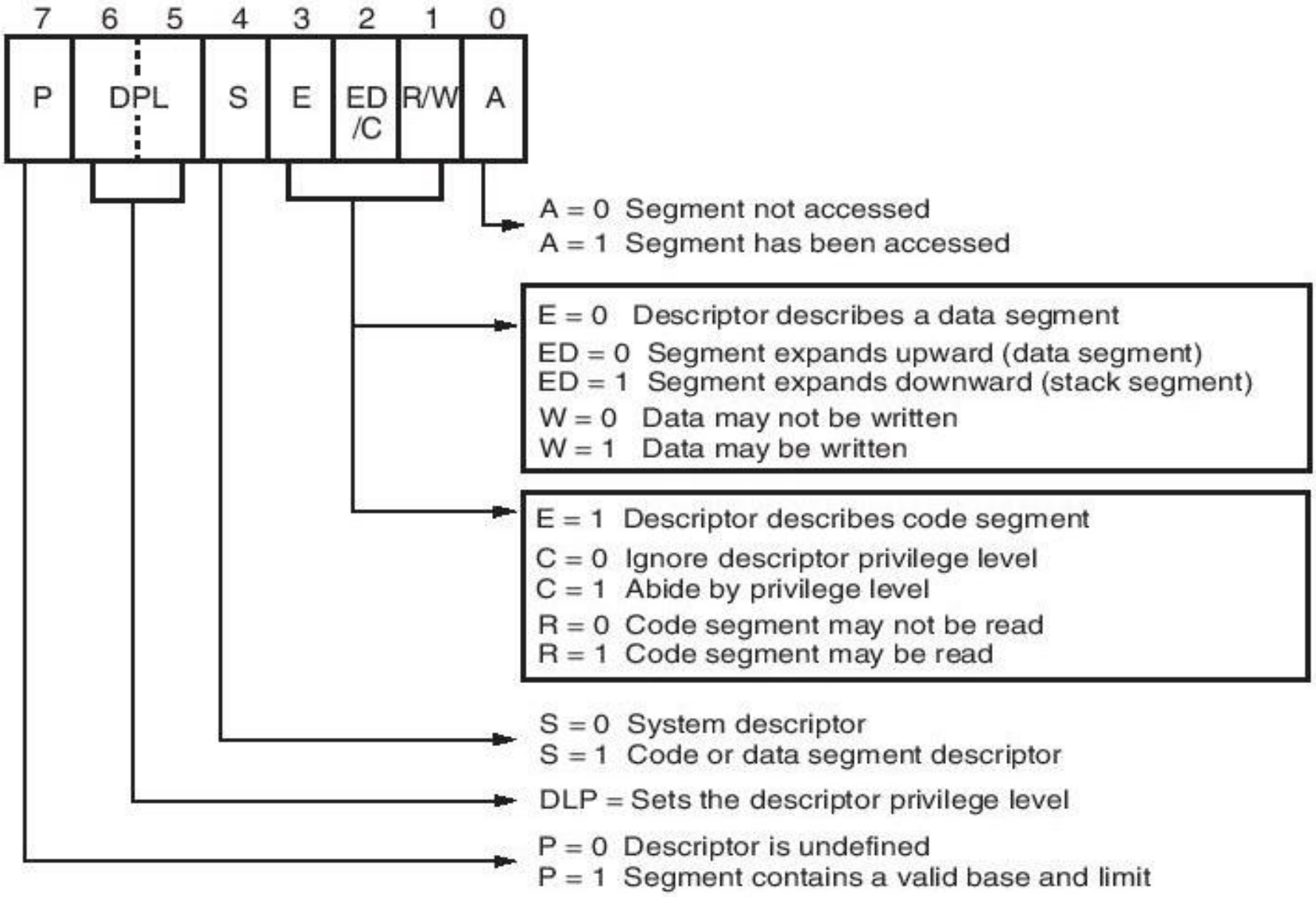


Figure: The access rights byte for the 80286 through Core2 descriptor

Selectors & Descriptors

- The ACCESS RIGHTS BYTE 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.
- We 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.
- 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.

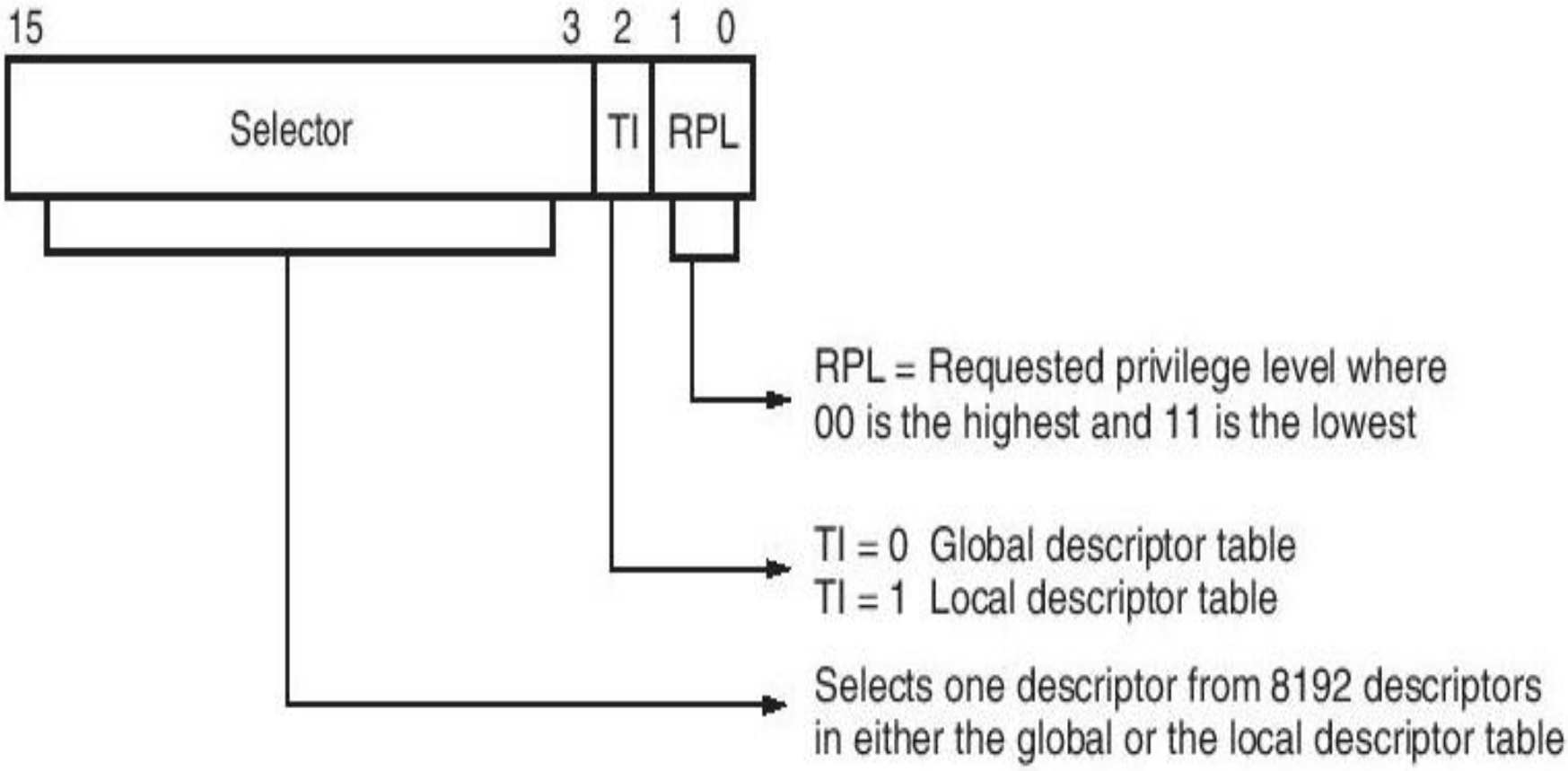


Figure: The contents of a segment register during protected mode operation.

Selectors & Descriptors

- Descriptors are chosen from the descriptor table by the segment register.
- The Figure above 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.

Selectors & Descriptors

- 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 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.
- If privilege levels are violated, the system normally indicates an application or privilege level violation.

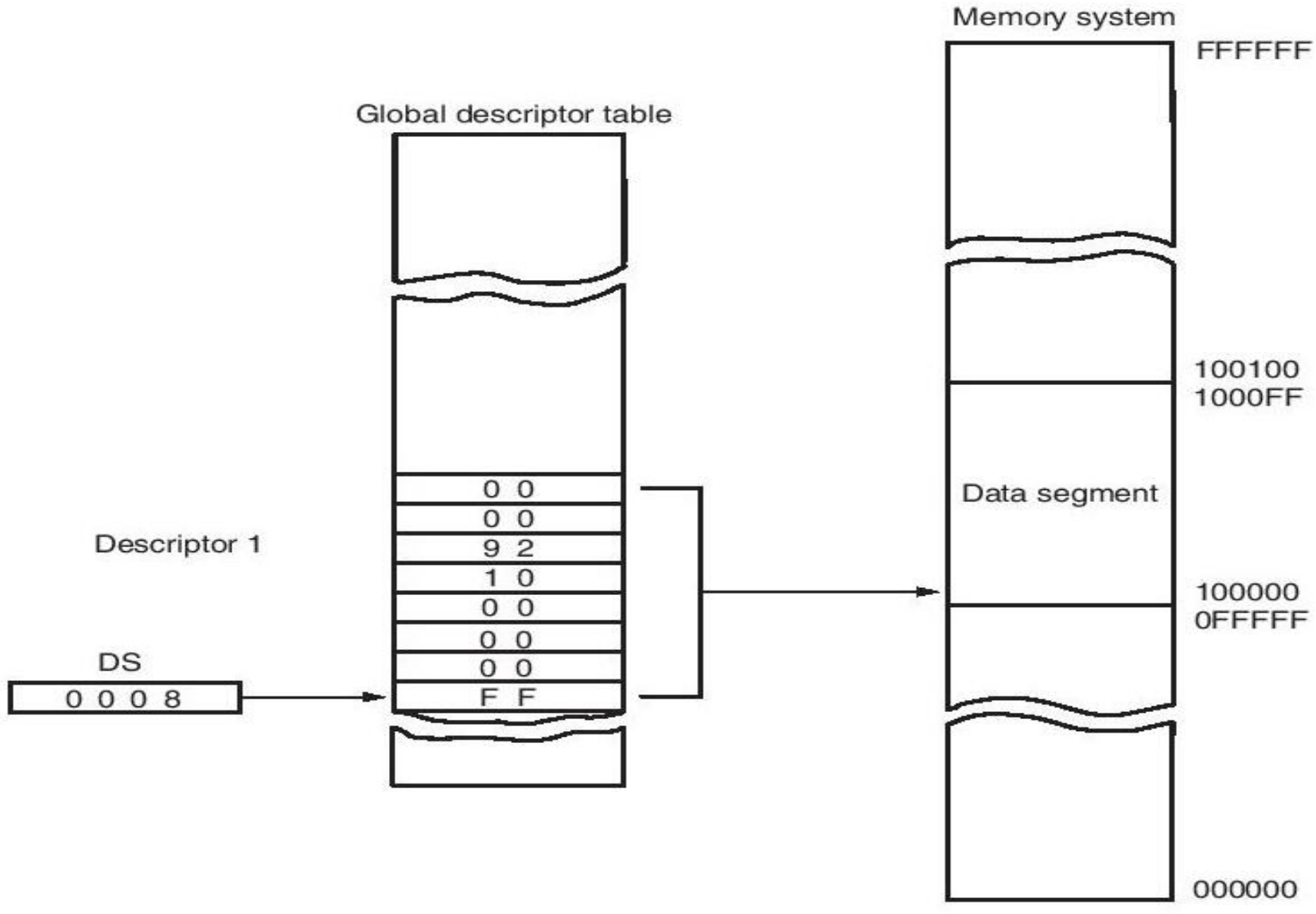


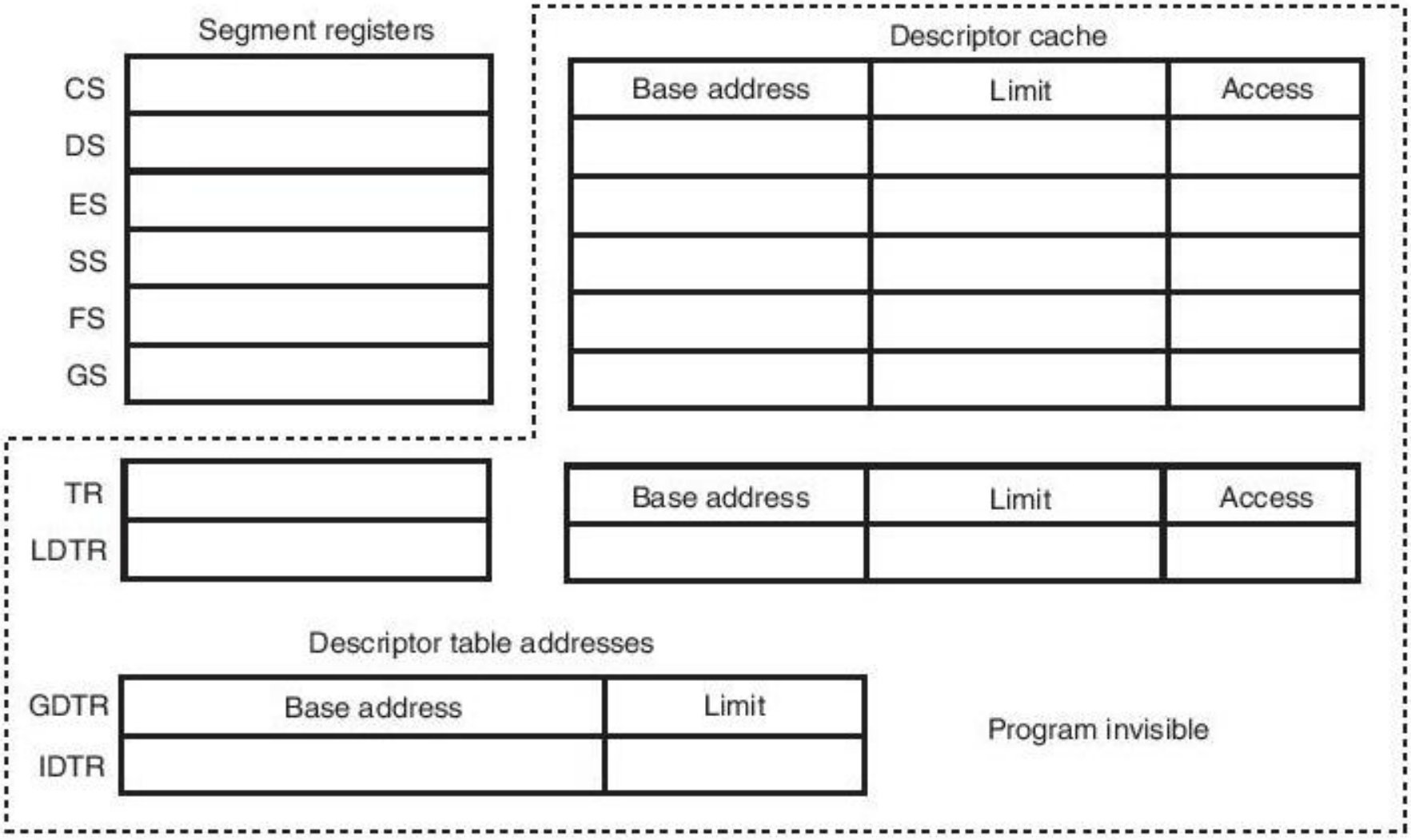
Figure: Using the DS register to select a description from the global descriptor table. In this example, the DS register accesses memory locations 00100000H–001000FFH as a data segment.

Selectors & Descriptors

- The Figure above 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 00100000H with a segment limit of 000FFH.
- This means that a value of 0008H loaded into DS causes the microprocessor to use memory locations 00100000H–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.

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.
- The Figure below illustrates the program-invisible registers as they appear in the 80286 through the Core2.
- These registers control the microprocessor when operated in protected mode.



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: The program-invisible register within the 80286–Core2 microprocessors.

Program-Invisible Registers

- 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).

Program-Invisible Registers

- 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 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.
- 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.

Program-Invisible Registers

- 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.
- 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 μs.
- Task switching allows the microprocessor to switch between tasks in a fairly short amount of time.

Hello World Program in 80x86

```
section     .text
global      _start                  ;must be declared for linker (ld)

_start:                             ;tell linker entry point

    mov     edx,len                 ;message length
    mov     ecx,msg                 ;message to write
    mov     ebx,1                   ;file descriptor (stdout)
    mov     eax,4                   ;system call number (sys_write)
    int     0x80                    ;call kernel

    mov     eax,1                   ;system call number (sys_exit)
    int     0x80                    ;call kernel

section     .data

msg     db  'Hello, world!',0xa     ;our dear string
len     equ $ - msg                 ;length of our dear string
```

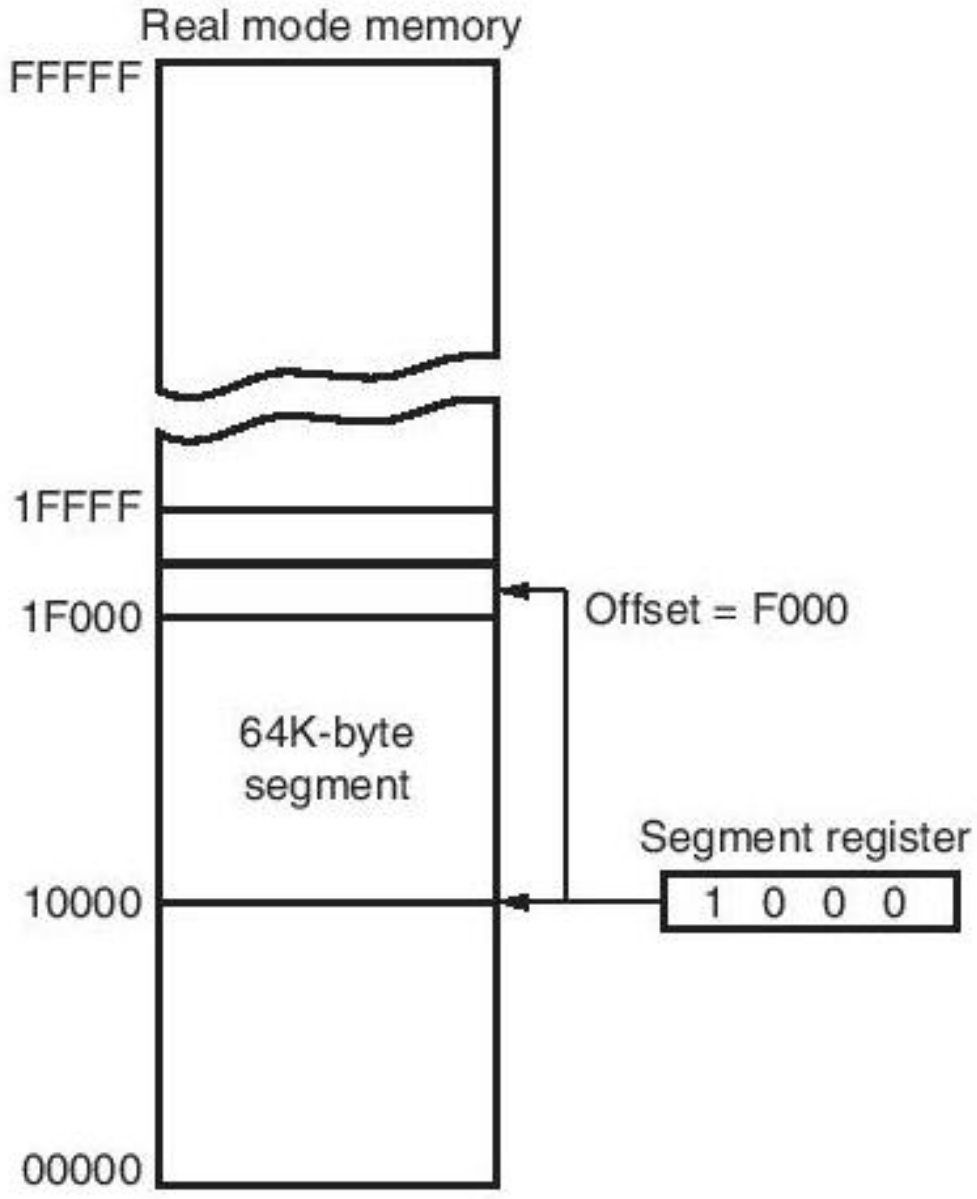
Execution Modes

- The 80286 and above operate in either the REAL or PROTECTED MODE.
- Only the 8086 and 8088 operate exclusively in the real mode.
- Real mode operation allows the microprocessor to address only the first 1M byte of memory space.
- The first 1M byte of memory is called the REAL MEMORY, CONVENTIONAL MEMORY, or DOS MEMORY SYSTEM.
- 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.
- In all cases, each of these microprocessors begins operation in the real mode by default whenever power is applied or the microprocessor is reset.

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.

Real-mode Addressing



Segments and Offsets

- The Figure above 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 IFFFFH – 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.
- The segment register contains 1000H, yet it *addresses a starting segment at location 10000H*.

Segments and Offsets

- 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.

Default Segment and Offset Registers

- The microprocessor has a set of rules that apply to segments whenever memory is addressed.
- These rules 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.

Default Segment and Offset Registers

- 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.

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-bit/ 16-bit number	Data Address
ES	DI for string instructions	String Destination Address

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, EDI, ESI, an 8-bit/ 16-bit number	Data Address
ES	EDI for string instructions	String Destination Address
FS	No default	General Address
GS	No default	General Address

Protected-mode

- Protected mode, also called protected virtual address mode, is an *operational mode* of x86-compatible central processing units (CPUs).
- It allows system software to use features such as virtual memory, paging and safe multi-tasking designed to increase an operating system's control over application software.
- When a processor that supports x86 protected mode is powered on, it begins executing instructions in real mode.
- Protected mode may only be entered after the system software sets up one descriptor table and enables the Protection Enable (PE) bit in the control register 0 (CR0).
- Protected mode was first added to the x86 architecture in 1982, with the release of Intel's 80286 (286) processor, and later extended with the release of the 80386 (386) in 1985.

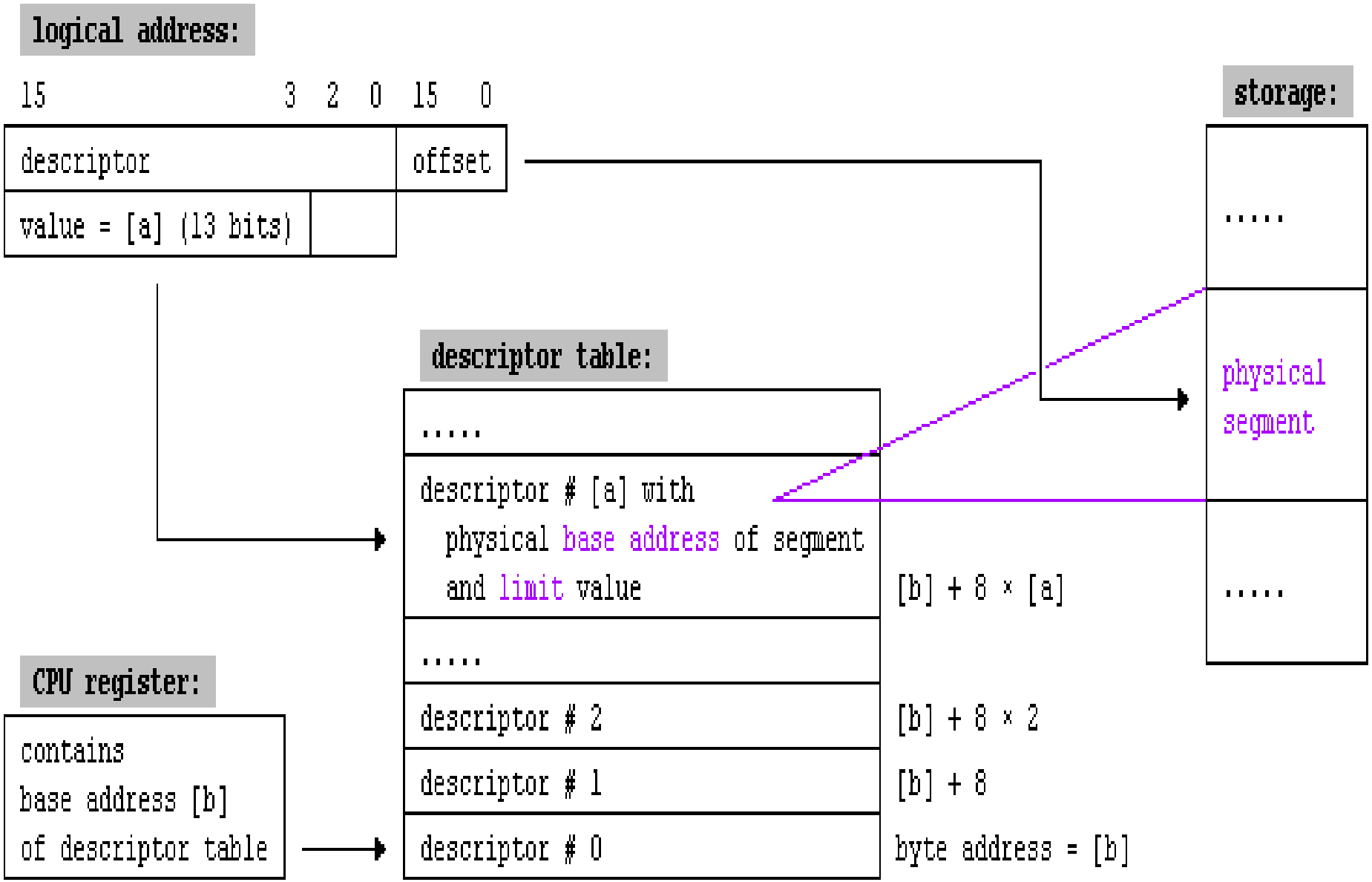
Protected-mode

- 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.
- 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.

Protected-mode

- 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.

Protected-mode - Segment Addressing in 286



Addressing Modes

- Efficient software development for the microprocessor requires a complete familiarity with the addressing modes employed by each instruction.
- The different ways in which a source operand is denoted in an instruction is known as ADDRESSING MODE.
- Here we use the MOV (move data) instruction to describe the data-addressing modes.
- The MOV instruction transfers bytes or words of data between two registers or between registers and memory in the 8086 through the 80286.
- Bytes, words, or doublewords are transferred in the 80386 and above by a MOV.
- In describing the program memory-addressing modes, the CALL and JUMP instructions show how to modify the flow of the program.

Addressing Modes

- The data-addressing modes include REGISTER, IMMEDIATE, DIRECT, REGISTER INDIRECT, BASE-PLUS INDEX, REGISTER-RELATIVE, and BASE RELATIVE-PLUS-INDEX in the 8086 through the 80286 microprocessor.
- The 80386 and above also include a SCALED-INDEX mode of addressing memory data.
- The program memory-addressing modes include program RELATIVE, DIRECT, and INDIRECT.
- We also explain the operation of the stack memory so that the PUSH and POP instructions and other stack operations will be understood.

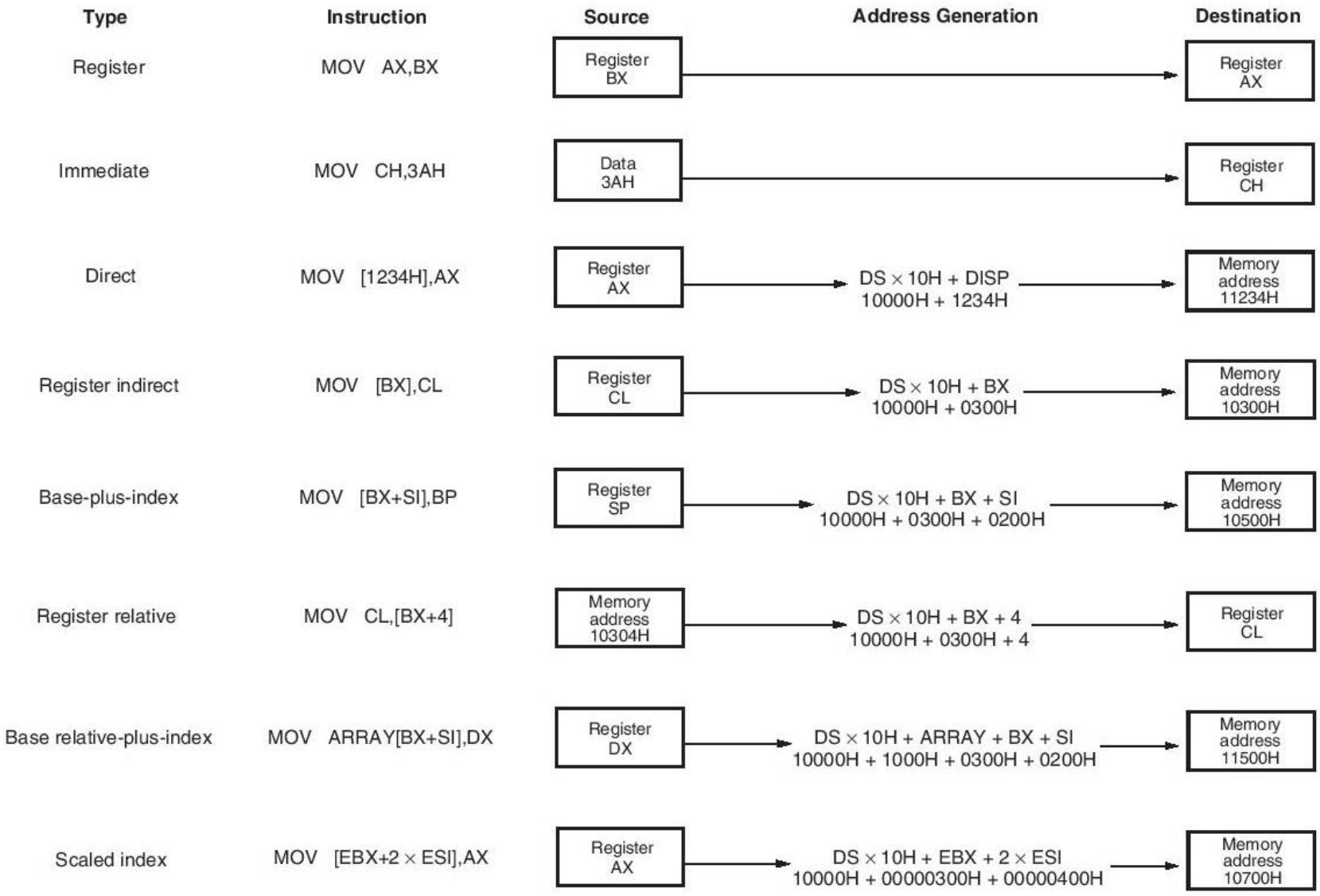
Data-Addressing Modes

- A MOV instruction may be defined as follows:

$$\mathrm{MOV\ AX, BX}$$

- The source of the operation is to the right and the destination is to the left, next to the opcode MOV.
- Notice that a comma always separates the destination from the source in an instruction.
- Also, note that memory-to-memory transfers are not allowed by any instruction except for the MOVS instruction.
- The $\mathrm{MOV\ AX, BX}$ instruction transfers the word contents of the source register (BX) into the destination register (AX).
- The source never changes, but the destination always changes.
- The Figure in the next slide shows all possible variations of the data-addressing modes using the MOV. instruction.

Data Addressing Modes



Notes: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, and DS = 1000H

Register Addressing

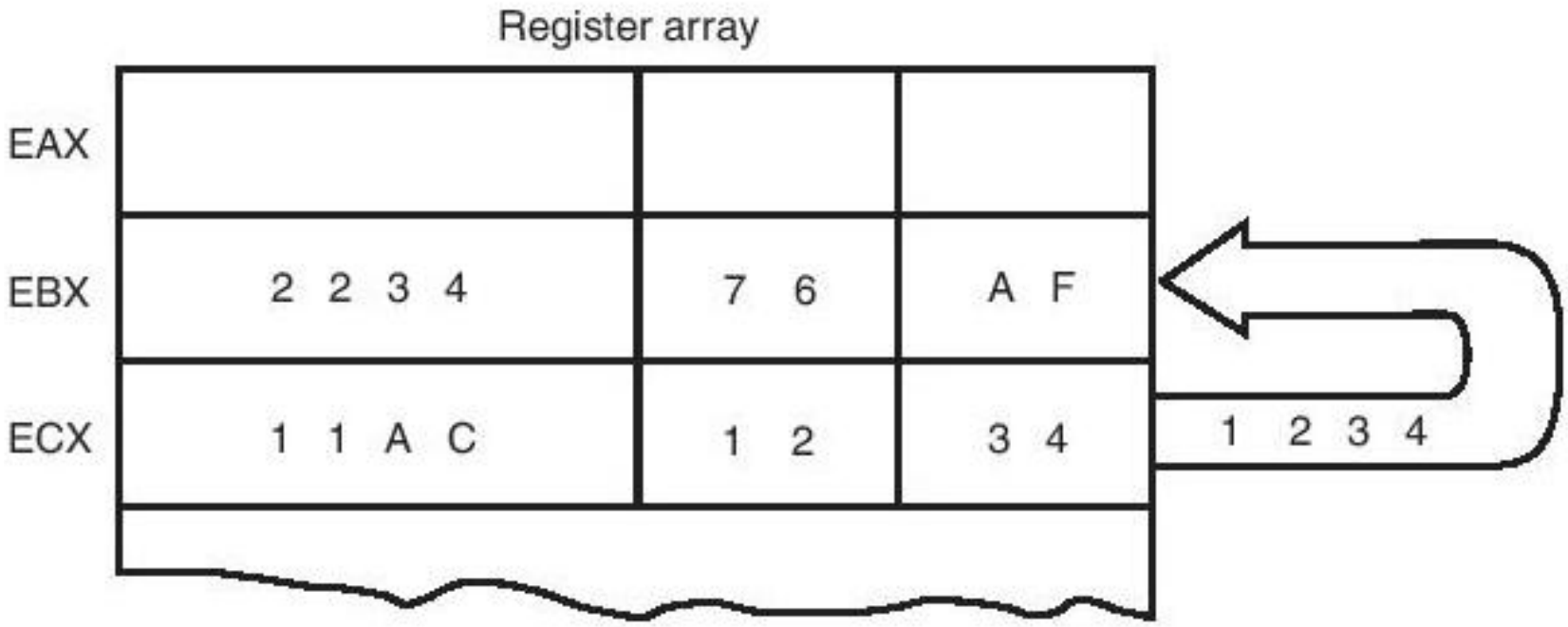


Figure: The effect of executing the $\mathrm{MOV\ BX, CX}$ instruction at the point just before the BX register changes. Note that only the rightmost 16 bits of register EBX change.

Assembly Language	*Size*	*Operation*
MOV AL,BL	8 bits	Copies BL into AL
MOV CH,CL	8 bits	Copies CL into CH
MOV R8B,CL	8 bits	Copies CL to the byte portion of R8 (64-bit mode)
MOV R8B,CH	8 bits	Not allowed
MOV AX,CX	16 bits	Copies CX into AX
MOV SP,BP	16 bits	Copies BP into SP
MOV DS,AX	16 bits	Copies AX into DS
MOV BP,R10W	16 bits	Copies R10 into BP (64-bit mode)
MOV SI,DI	16 bits	Copies DI into SI
MOV BX,ES	16 bits	Copies ES into BX
MOV ECX,EBX	32 bits	Copies EBX into ECX
MOV ESP,EDX	32 bits	Copies EDX into ESP
MOV EDX,R9D	32 bits	Copies R9 into EDX (64-bit mode)
MOV RAX,RDX	64 bits	Copies RDX into RAX
MOV DS,CX	16 bits	Copies CX into DS
MOV ES,DS	—	Not allowed (segment-to-segment)
MOV BL,DX	—	Not allowed (mixed sizes)
MOV CS,AX	—	Not allowed (the code segment register may not be the destination register)

Figure: Examples of register-addressed instructions.

Immediate Addressing

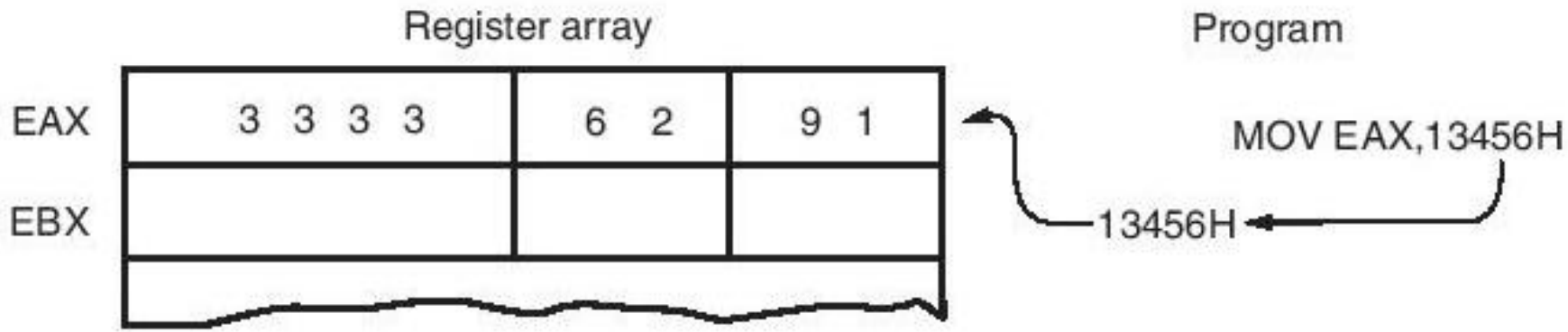


Figure: The operation of the $\mathrm{MOV\ EX,\ 13456H}$ instruction. This instruction copies the immediate data (13456H) into EAX. As with the $\mathrm{MOV\ BX,\ CX}$ instruction the source data overwrites the destination data.

Assembly Language	*Size*	*Operation*
MOV BL,44	8 bits	Copies 44 decimal (2CH) into BL
MOV AX,44H	16 bits	Copies 0044H into AX
MOV SI,0	16 bits	Copies 0000H into SI
MOV CH,100	8 bits	Copies 100 decimal (64H) into CH
MOV AL,'A'	8 bits	Copies ASCII A into AL
MOV AH,1	8 bits	Not allowed in 64-bit mode, but allowed in 32- or 16-bit modes
MOV AX,'AB'	16 bits	Copies ASCII BA* into AX
MOV CL,11001110B	8 bits	Copies 11001110 binary into CL
MOV EBX,12340000H	32 bits	Copies 12340000H into EBX
MOV ESI,12	32 bits	Copies 12 decimal into ESI
MOV EAX,100B	32 bits	Copies 100 binary into EAX
MOV RCX,100H	64 bits	Copies 100H into RCX

*Note: This is not an error. The ASCII characters are stored as BA, so exercise care when using word-sized pairs of ASCII characters.

Figure: Examples of immediate addressing using the MOV instruction..

Direct Data-Addressing

- Most instructions can use the direct data-addressing mode.
- There are two basic forms of DIRECT DATA-ADDRESSING:
  - DIRECT ADDRESSING, which applies to a MOV between a memory location and AL, AX, or EAX,
  - DISPLACEMENT ADDRESSING, which applies to almost any instruction in the instruction set.
- In either case, the address is formed by adding the displacement to the default data segment address or an alternate segment address.
- In 64-bit operation, the direct-addressing instructions are also used with a 64-bit linear address, which allows access to any memory location.

Direct Addressing

Direct addressing with a MOV instruction transfers data between a memory location, located within the data segment, and the AL (8-bit), AX (I6-bit), or EAX (32-bit) register. A MOV instruction using this type of addressing is usually a 3-byte long instruction.

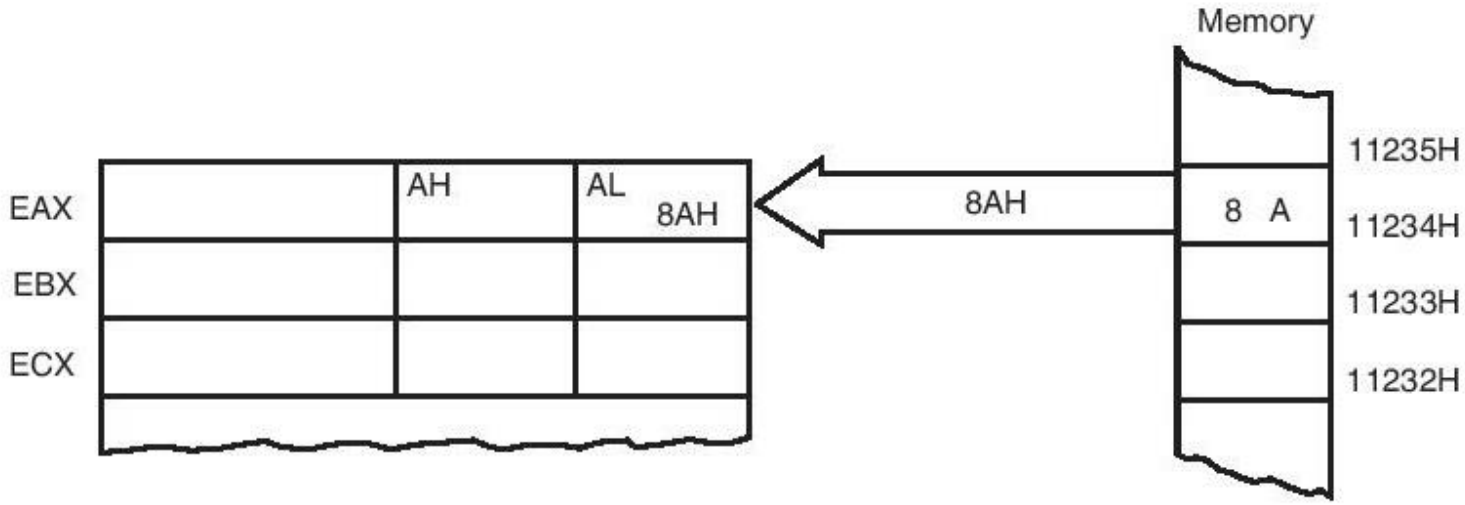


Figure: The operation of the $\mathrm{MOV\ AL,[1234H]}$ instruction when DS = 1000H .

Assembly Language	*Size*	*Operation*
MOV AL,NUMBER	8 bits	Copies the byte contents of data segment memory location NUMBER into AL
MOV AX,COW	16 bits	Copies the word contents of data segment memory location COW into AX
MOV EAX,WATER*	32 bits	Copies the doubleword contents of data segment location WATER into EAX
MOV NEWS,AL	8 bits	Copies AL into byte memory location NEWS
MOV THERE,AX	16 bits	Copies AX into word memory location THERE
MOV HOME,EAX*	32 bits	Copies EAX into doubleword memory location HOME
MOV ES:[2000H],AL	8 bits	Copies AL into extra segment memory at offset address 2000H
MOV AL,MOUSE	8 bits	Copies the contents of location MOUSE into AL; in 64-bit mode MOUSE can be any address
MOV RAX,WHISKEY	64 bits	Copies 8 bytes from memory location WHISKEY into RAX

*Note: The 80386–Pentium 4 at times use more than 3 bytes of memory for 32-bit instructions.

Figure: Direct addressed instructions using EAX, AX, and AL and RAX in 64-bit mode.

Displacement Addressing

- Displacement addressing is almost identical to direct addressing, except that the instruction is 4 bytes wide instead of 3.
- In the 80386 through the Pentium 4, this instruction can be up to 7 bytes wide if both a 32-bit register and a 32-bit displacement are specified.

Assembly Language	*Size*	*Operation*
MOV CH,DOG	8 bits	Copies the byte contents of data segment memory location DOG into CH
MOV CH,DS:[1000H]*	8 bits	Copies the byte contents of data segment memory offset address 1000H into CH
MOV ES,DATA6	16 bits	Copies the word contents of data segment memory location DATA6 into ES
MOV DATA7,BP	16 bits	Copies BP into data segment memory location DATA7
MOV NUMBER,SP	16 bits	Copies SP into data segment memory location NUMBER
MOV DATA1,EAX	32 bits	Copies EAX into data segment memory location DATA1
MOV EDI,SUM1	32 bits	Copies the doubleword contents of data segment memory location SUM1 into EDI

Register Indirect Addressing

- Register indirect addressing allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI, and SI.
- For example, if register BX contains 1000H and the $\mathrm{MOV\ AX,[BX]}$ instruction executes, the word contents of data segment offset address 1000H are copied into register AX.
- If the microprocessor is operated in the real mode and DS = 0100H , this instruction addresses a word stored at memory bytes 2000H and 2001H, and transfers it into register AX.
- Note that the contents of 2000H are moved into AL and the contents of 2001H are moved into AH.
- The [ ] symbols denote INDIRECT ADDRESSING in assembly language.
- In addition to using the BP, BX, DI, and SI registers to indirectly address memory, the 80386 and above allow register indirect addressing with any extended register except ESP.

Register Indirect Addressing

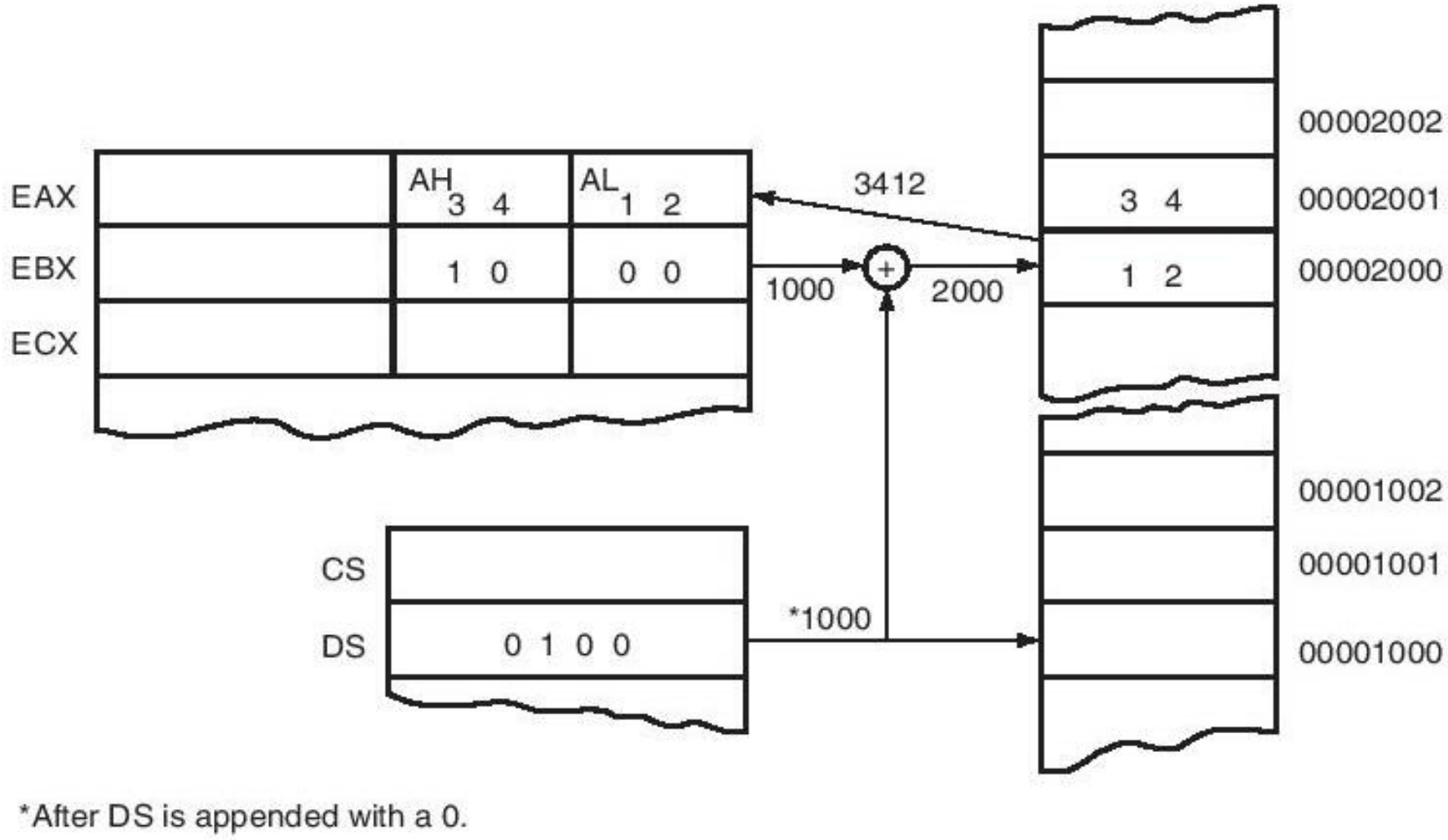


Figure: The operation of the $\mathrm{MOV\ AX,[BX]}$ instruction when BX = 1000H and DS = 0100H.

Base-Plus-Index Addressing

- Base-plus-index addressing is similar to indirect addressing because it indirectly addresses memory data.
- In the 8086 through the 80286, this type of addressing uses one base register (BP or BX) and one index register (DI or SI) to indirectly address memory.
- The base register often holds the beginning location of a memory array, whereas the index register holds the relative position of an element in the array.
- Remember that whenever BP addresses memory data, both the stack segment register and BP generate the effective address.
- In the 80386 and above, this type of addressing allows the combination of any two 32-bit extended registers except ESP.
- For example, the $\mathrm{MOV\ DL,[\ EAX{+}EBX\ ]}$ instruction is an example using EAX (as the base) plus EBX (as the index).
- If the EBP register is used, the data are located in the stack segment instead of in the data segment.

Base-Plus-Index Addressing

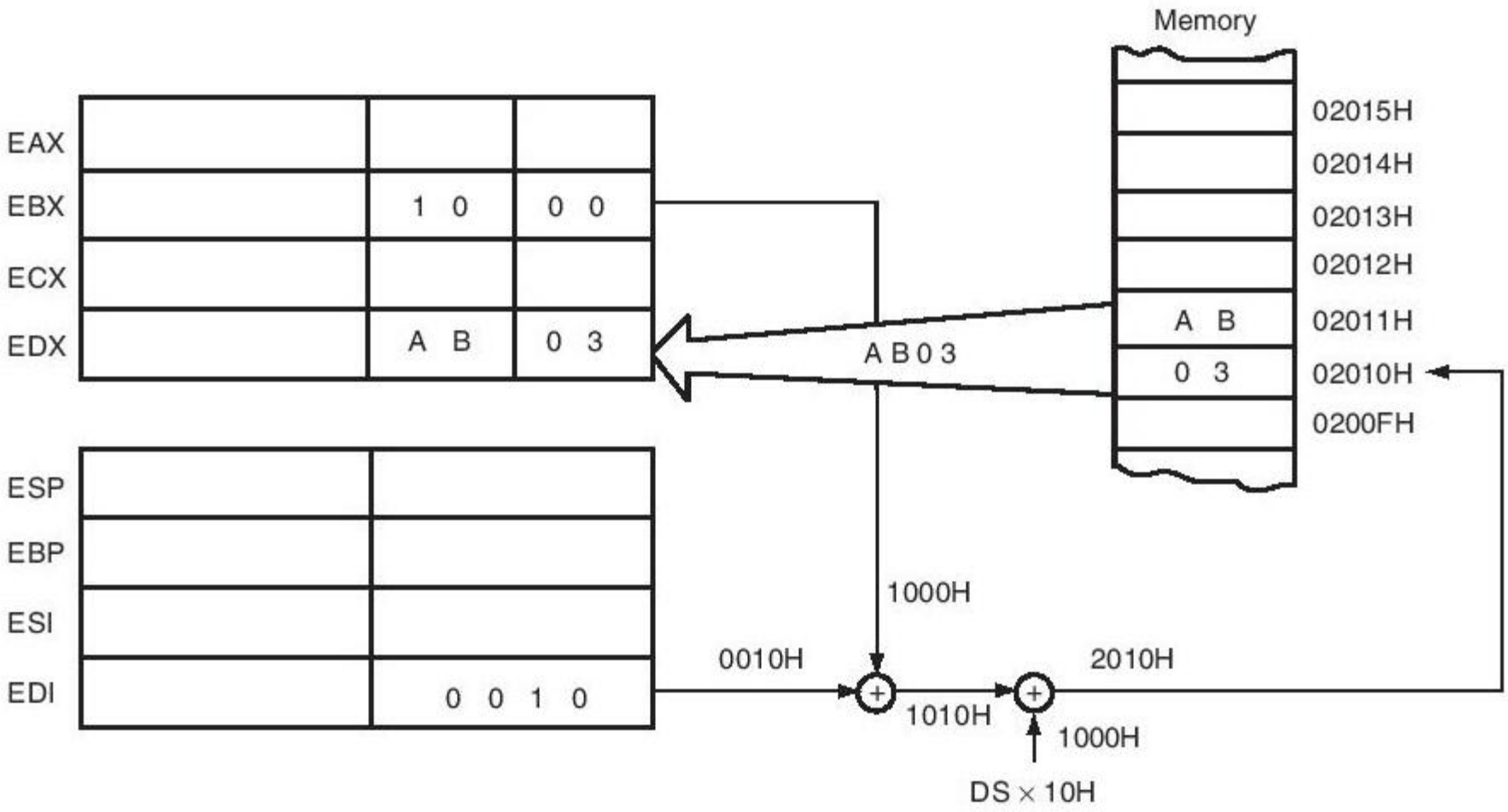


Figure: An example showing how the base-plus-index addressing mode functions for the $\mathrm{MOV\ DX,[BX+DI]}$ instruction. Notice that memory address 02010H is accessed because DS = 0100H, BX = 1000H, and DI = 0010H .

Register Relative Addressing

- Register relative addressing is similar to base-plus-index addressing and displacement addressing.
- In register relative addressing, the data in a segment of memory are addressed by adding the displacement to the contents of a base or an index register (BP, BX, DI, or SI).
- The Figure below shows the operation of the $\mathrm{MOV\ AX,[BX+1000H]}$ instruction.
- In this example, BX = 0100H and DS = 0200H , so the address generated is the sum of DS $\times$ 10H, BX, and the displacement of 1000H, which addresses location 03100H.
- Remember that BX, DI, or SI addresses the data segment and BP addresses the stack segment.
- In the 80386 and above, the displacement can be a 32-bit number and the register can be any 32-bit register except the ESP register.
- Remember that the size of a real mode segment is 64K bytes long.

Register Relative Addressing

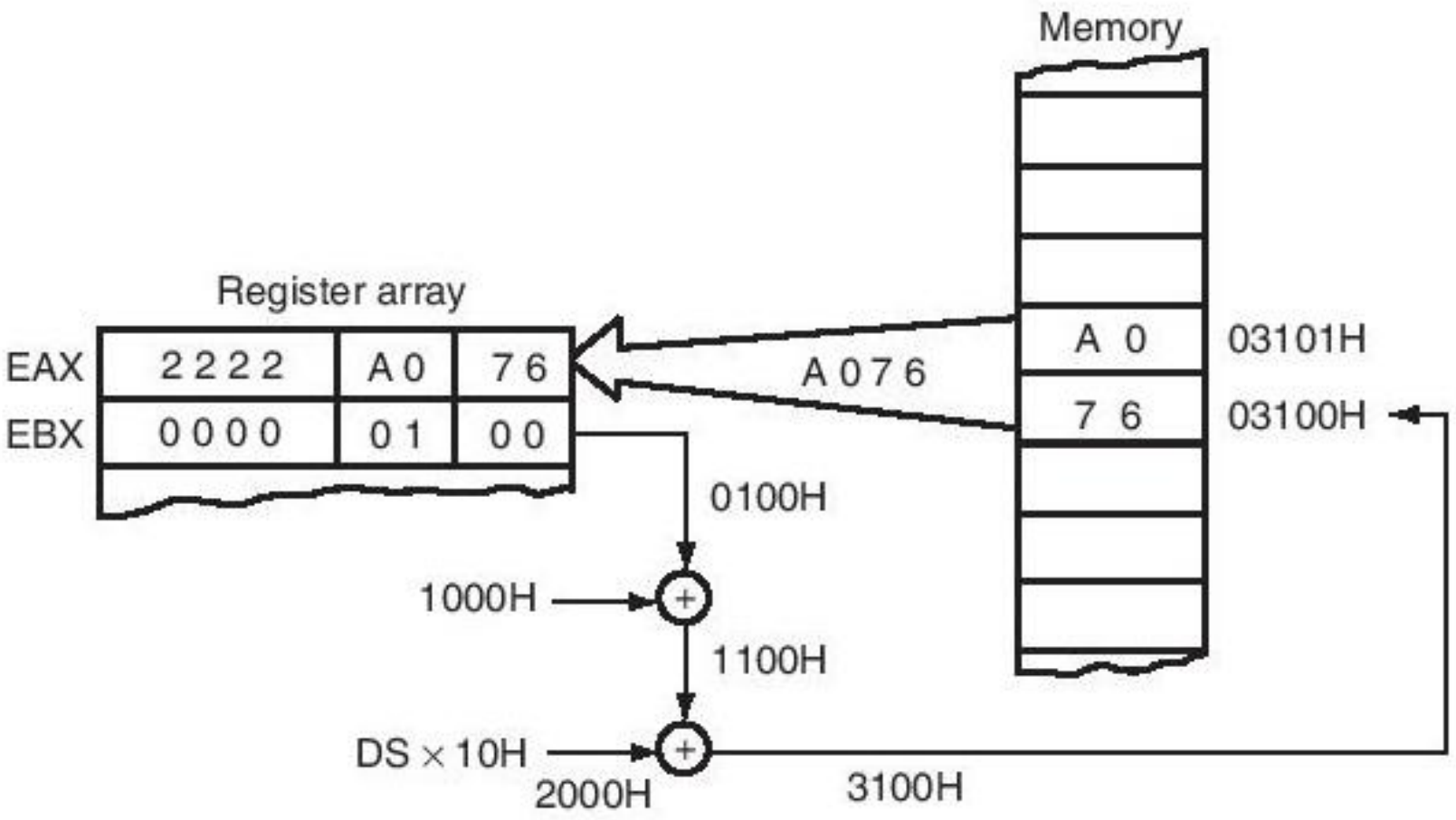


Figure: The operation of the $\mathrm{MOV\ AX, [BX{+}1000H]}$ instructon, when BX = 0100H and DS = 0200H.

Base Relative-Plus-Index Addressing

The base relative-plus-index addressing mode is similar to base-plus-index addressing, but it adds a displacement, besides using a base register and an index register, to form the memory address.

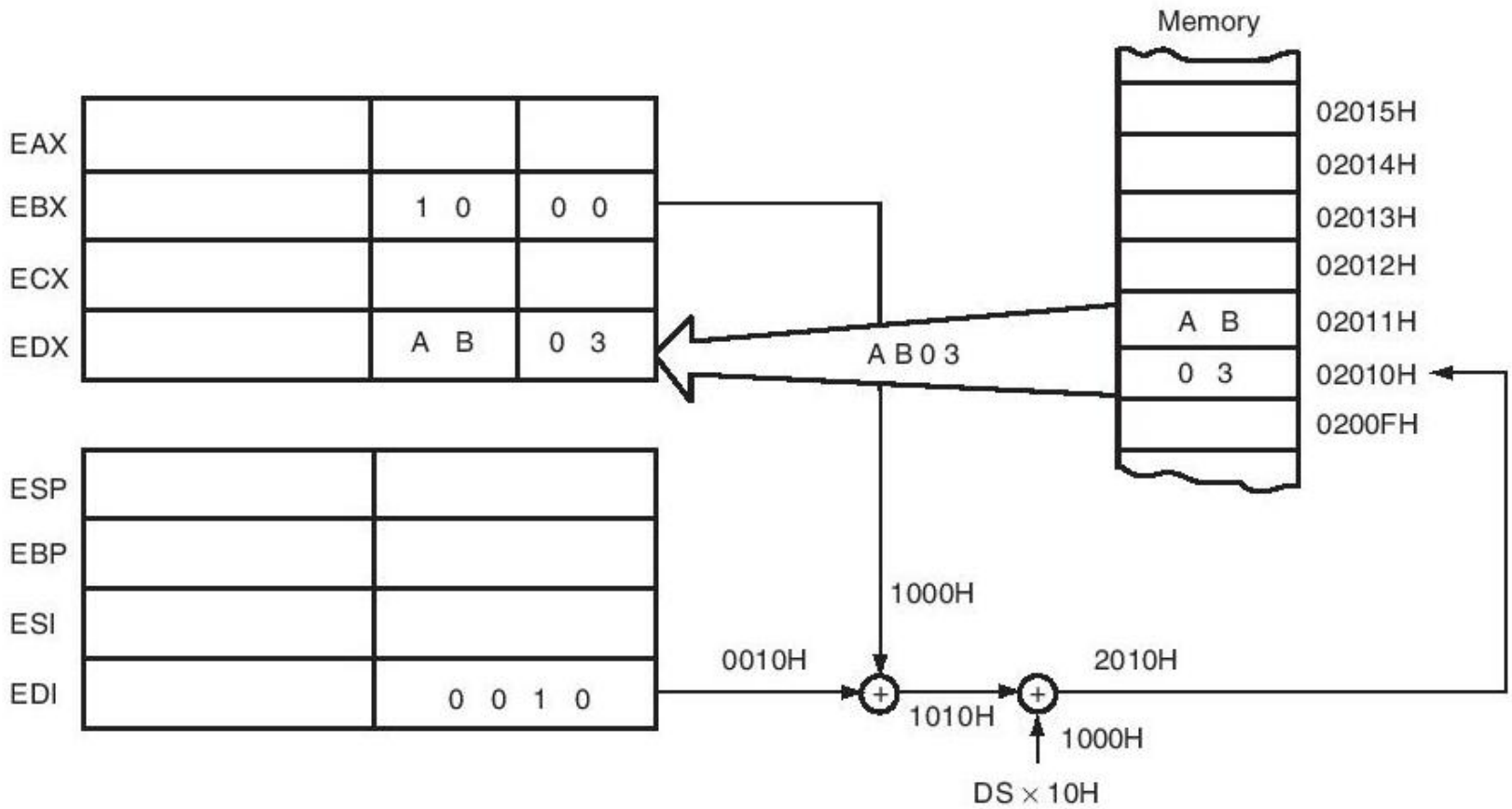


Figure: An example of base relative-plus-index addressing using a $\mathrm{MOV\ DX,[BX+DI+100H]}$ instruction. Note: DS = 100H.

Scaled-Index Addressing

- Scaled-index addressing is the last type of data-addressing mode discussed.
- This data-addressing mode is unique to the 80386 through the Core2 microprocessors.
- Scaled-index addressing uses two 32-bit registers (a base register and an index register) to access the memory.
- The second register (index) is multiplied by a scaling factor.
- The scaling factor can be $1\times$, $2\times$, $4\times$, or $8\times$.
- A scaling factor of $1\times$ is implied and need not be included in the assembly language instruction ($\mathrm{MOV\ AL,[EBX+ECX]}$).
- A scaling factor of $2\times$ is used to address word-sized memory arrays, a scaling factor of $4\times$ is used with doubleword-sized memory arrays, and a scaling factor of $8\times$ is used with quadword-sized memory arrays.

Scaled-Index Addressing

- An example instruction is $\mathrm{MOV\ AX,[EDI+2*ECX]}$. This instruction uses a scaling factor of 2× , which multiplies the contents of ECX by 2 before adding it to the EDI register to form the memory address.
- If ECX contains a 00000000H, word-sized memory element 0 is addressed; if ECX contains a 00000001H, word-sized memory element 1 is accessed, and so forth.
- This scales the index (ECX) by a factor of 2 for a word-sized memory array.
- As you can imagine, there are an extremely large number of the scaled-index addressed register combinations.
- Scaling is also applied to instructions that use a single indirect register to access memory.
- The $\mathrm{MOV\ EAX,[4*EDI]}$ is a scaled-index instruction that uses one register to indirectly address memory.
- In the 64-bit mode, an instruction such as $\mathrm{MOV\ RAX,[8*RDI]}$ might appear in a program.

Assembly Language	*Size*	*Operation*
MOV EAX,[EBX+4*ECX]	32 bits	Copies the doubleword contents of the data segment memory location addressed by the sum of 4 times ECX plus EBX into EAX
MOV [EAX+2*EDI+100H],CX	16 bits	Copies CX into the data segment memory location addressed by the sum of EAX, 100H, and 2 times EDI
MOV AL,[EBP+2*EDI+2]	8 bits	Copies the byte contents of the stack segment memory location addressed by the sum of EBP, 2, and 2 times EDI into AL
MOV EAX,ARRAY[4*ECX]	32 bits	Copies the doubleword contents of the data segment memory location addressed by the sum of ARRAY and 4 times ECX into EAX

Figure: Examples of scaled-index addressing

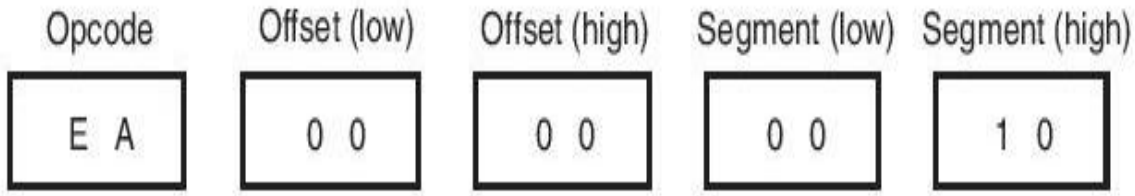
Program Memory Addressing Modes

- Program memory-addressing modes are used with the JMP (jump) and CALL instructions.
- They consist of three distinct forms: DIRECT, RELATIVE, and INDIRECT.
- Here we introduce these three addressing forms, using the JMP instruction to illustrate their operation.

Direct Program Memory Addressing

- Direct program memory addressing is what many early microprocessors used for all jumps and calls.
- Direct program memory addressing is also used in high-level languages, such as the BASIC language GOTO and GOSUB instructions.
- The instructions for direct program memory addressing store the address with the opcode.
- For example, if a program jumps to memory location 10000H for the next instruction, the address (10000H) is stored following the opcode in the memory.

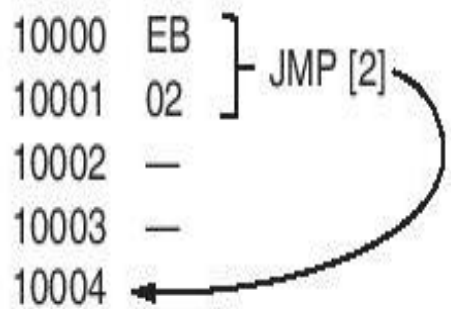
Direct Program Memory Addressing



- Figure above shows the direct inter-segment JMP instruction and the 4 bytes required to store the address 10000H.
- This JMP instruction loads CS register with 1000H and IP (or EIP) register with 0000H to jump to memory location 10000H for the next instruction.
- An inter-segment jump is a jump to any memory location within the entire memory system.
- The direct jump is often called a FAR JUMP because it can jump to any memory location for the next instruction.

Relative Program Memory Addressing

- The term relative means "relative to the instruction pointer (IP)."
- For example, if a JMP instruction skips the next 2 bytes of memory, the address in relation to the instruction pointer is a 2 that adds to the instruction pointer.
- This develops the address of the next program instruction.
- An example of the relative JMP instruction is shown in the Figure.



Relative Program Memory Addressing

- Notice that the JMP instruction is a 1-byte instruction, with a 1-byte or a 2-byte displacement that adds to the instruction pointer.
- A 1-byte displacement is used in short jumps, and a 2-byte displacement is used with near jumps and calls.
- Both types are considered to be intra-segment jumps.
- An intra-segment jump is a jump anywhere within the current code segment.
- In the 80386 and above, the displacement can also be a 32-bit value, allowing them to use relative addressing to any location within their 4G-byte code segments.

Indirect Program Memory Addressing

Assembly Language	*Operation*
JMP AX	Jumps to the current code segment location addressed by the contents of AX
JMP CX	Jumps to the current code segment location addressed by the contents of CX
JMP NEAR PTR[BX]	Jumps to the current code segment location addressed by the contents of the data segment location addressed by BX
JMP NEAR PTR[DI+2]	Jumps to the current code segment location addressed by the contents of the data segment memory location addressed by DI plus 2
JMP TABLE[BX]	Jumps to the current code segment location addressed by the contents of the data segment memory location address by TABLE plus BX
JMP ECX	Jumps to the current code segment location addressed by the contents of ECX
JMP RDI	Jumps to the linear address contained in the RDI register (64-bit mode)

- The microprocessor allows several forms of program indirect memory addressing for the JMP and CALL instructions.
- The table above lists some acceptable program indirect jump instructions, which can use any 16-bit register (AX, BX, CX, DX, SP, BP, DI, or SI); any relative register ([BP], [BX], [DI], or [SI]); and any relative register with a displacement.

Indirect Program Memory Addressing

- In the 80386 and above, an extended register can also be used to hold the address or indirect address of a relative JMP or CALL.
- For example, the $\mathrm{JMP\ EAX}$ jumps to the location address by register EAX.
- If a 16-bit register holds the address of a JMP instruction, the jump is near.
- For example, if the BX register contains 1000H and a $\mathrm{JMP\ BX}$ instruction executes, the microprocessor jumps to offset address 1000H in the current code segment.

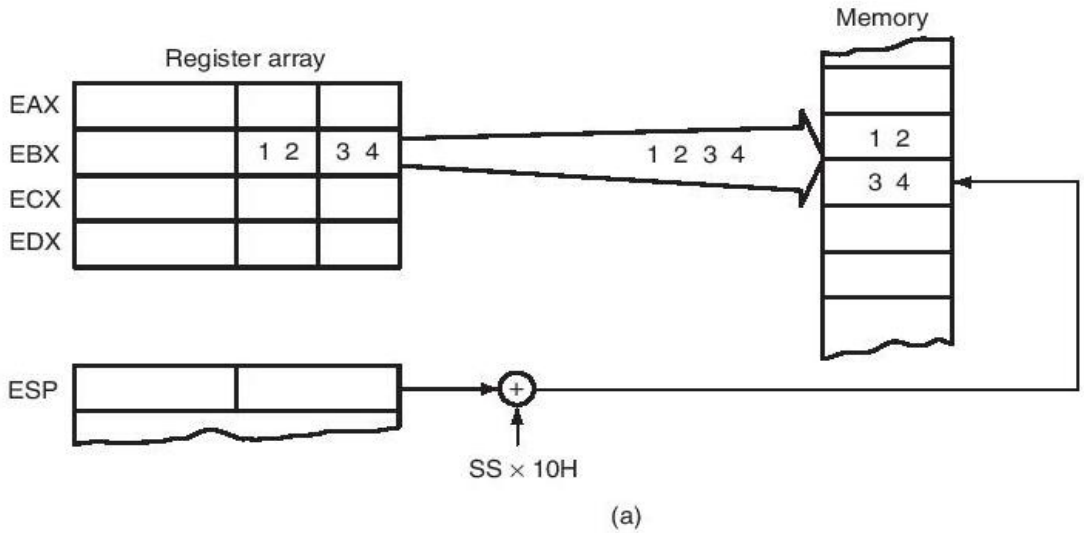
Indirect Program Memory Addressing

- If a relative register holds the address, the jump is also considered to be an indirect jump.
- For example, JMP [BX] refers to the memory location within the data segment at the offset address contained in BX.
- At this offset address is a l6-bit number that is used as the offset address in the intra-segment jump.
- This type of jump is sometimes called an INDIRECT-INDIRECT or DOUBLE-INDIRECT JUMP.

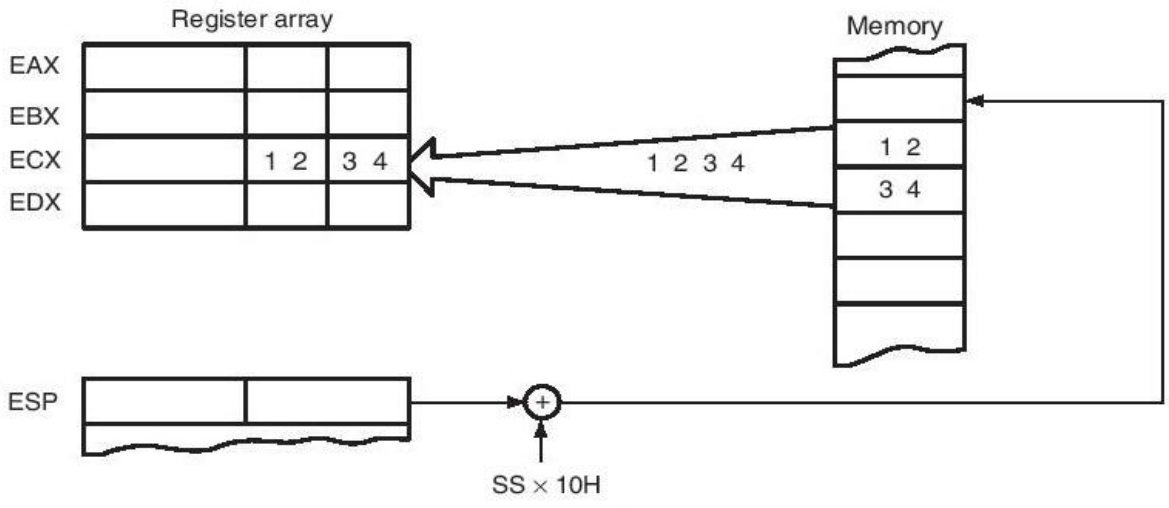
Stack Memory-Addressing

- The stack plays an important role in all microprocessors. It holds data temporarily and stores the return addresses used by procedures.
- The stack memory is an LIFO (last-in, first-out) memory, which describes the way that data are stored and removed from the stack.
- Data are placed onto the stack with a PUSH instruction and removed with a POP instruction.
- The CALL instruction also uses the stack to hold the return address for procedures and a RET (return) instruction to remove the return address from the stack.

Stack Memory-Addressing



(a)



(b)

Stack Memory-Addressing

- The stack memory is maintained by two registers: the stack pointer (SP or ESP) and the stack segment register (SS).
- Whenever a word of data is pushed onto the stack, the high-order 8 bits are placed in the location addressed by SP - 1.
- The low-order 8 bits are placed in the location addressed by SP - 2.
- The SP is then decremented by 2 so that the next word of data is stored in the next available stack memory location.
- The SP/ESP register always points to an area of memory located within the stack segment.
- The SP/ESP register adds to SS * 10H to form the stack memory address in the real mode.
- In protected mode operation, the SS register holds a selector that accesses a descriptor for the base address of the stack segment.

Stack Memory-Addressing

- Whenever data are popped from the stack, the low-order 8 bits are removed from the location addressed by SP.
- The high-order 8 bits are removed from the location addressed by SP $+ 1$. The SP register is then incremented by 2.
- The Table in the next slide lists some of the PUSH and POP instructions available to the microprocessor.
- Note that PUSH and POP store or retrieve words of data – never bytes – in the 8086 through the 80286 microprocessors.
- The 80386 and above allow words or double-words to be transferred to and from the stack.

Assembly Language	*Operation*
POPF	Removes a word from the stack and places it into the flag register
POPFD	Removes a doubleword from the stack and places it into the EFLAG register
PUSHF	Copies the flag register to the stack
PUSHFD	Copies the EFLAG register to the stack
PUSH AX	Copies the AX register to the stack
POP BX	Removes a word from the stack and places it into the BX register
PUSH DS	Copies the DS register to the stack
PUSH 1234H	Copies a word-sized 1234H to the stack
POP CS	This instruction is illegal
PUSH WORD PTR[BX]	Copies the word contents of the data segment memory location addressed by BX onto the stack
PUSHA	Copies AX, CX, DX, BX, SP, BP, DI, and SI to the stack
POPA	Removes the word contents for the following registers from the stack: SI, DI, BP, SP, BX, DX, CX, and AX
PUSHAD	Copies EAX, ECX, EDX, EBX, ESP, EBP, EDI, and ESI to the stack
POPAD	Removes the doubleword contents for the following registers from the stack: ESI, EDI, EBP, ESP, EBX, EDX, ECX, and EAX
POP EAX	Removes a doubleword from the stack and places it into the EAX register
POP RAX	Removes a quadword from the stack and places it into the RAC register (64-bit mode)
PUSH EDI	Copies EDI to the stack
PUSH RSI	Copies RSI into the stack (64-bit mode)
PUSH QWORD PTR[RDX]	Copies the quadword contents of the memory location addressed by RDX onto the stack

Stack Memory-Addressing

- A short program that pushes the contents of AX, BX, and CX onto the stack.
- The first POP retrieves the value that was pushed onto the stack from CX and places it into AX.
- The second POP places the original value of BX into CX.
- The last POP places the value of AX into BX.

```
MOV AX, 1000H
MOV BX, 2000H
MOV CX, 3000H
PUSH AX
PUSH BX
PUSH CX
POP AX
POP BX
POP CX
```

THANK YOU