address. For example, the word in Fig. 2–4(a) has its least significant byte at address 00724_{16}. Therefore, it is stored at an even-address boundary.

A word of data stored at an even-address boundary, such as 00000_{16}, 00002_{16}, 00004_{16}, and so on, is said to be an *aligned word*—that is, all aligned words are located at an address that is a multiple of 2. On the other hand, a word of data stored at an odd-address boundary, such as 00001_{16}, 00003_{16}, or 00005_{16} and so on, is called a *misaligned word*. Figure 2–5 shows some aligned and misaligned words of data. Here words 0, 2, 4, and 6 are examples of aligned-data words, while words 1 and 5 are misaligned words. Notice that misaligned word 1 consists of byte 1 from aligned word 0 and byte 2 from aligned word 2.

When expressing addresses and data in hexadecimal form, it is common to use the letter H to specify the base. For instance, the number $00AB_{16}$ can also be written as 00ABH.

EXAMPLE 2.1

What is the data word shown in Fig. 2–4(b)? Express the result in hexadecimal form. Is it stored at an even- or odd-addressed word boundary? Is it an aligned or misaligned word of data?

Solution

The most significant byte of the word is stored at address $0072C_{16}$ and equals

$$11111101_2 = FD_{16} = FDH$$

Its least significant byte is stored at address $0072B_{16}$ and is

$$10101010_2 = AA_{16} = AAH$$

Together the two bytes give the word

$$1111110110101010_2 = FDAA_{16} = FDAAH$$

Expressing the address of the least significant byte in binary form gives

$$0072BH = 0072B_{16} - 00000000011100101011_2$$

Because the rightmost bit (LSB) is logic 1, the word is stored at an odd-address boundary in memory; therefore, it is a misaligned word of data.

The *double word* is another data form that can be processed by the 8088 microcomputer. A double word corresponds to four consecutive bytes of data stored in memory; an example of double-word data is a *pointer*. A pointer is a two-word address element that is used to access data or code in memory. The

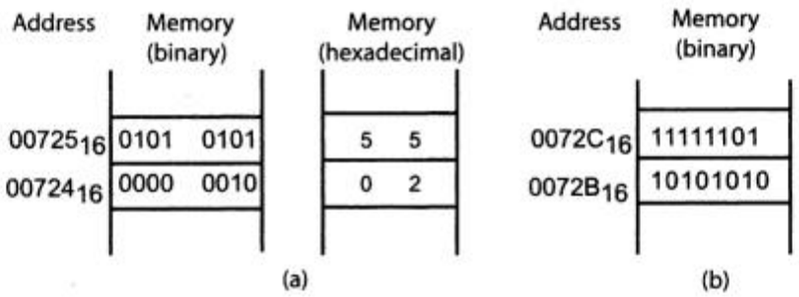


Figure 2–4 (a) Storing a word of data in memory. (b) An example.

Copyrighted material

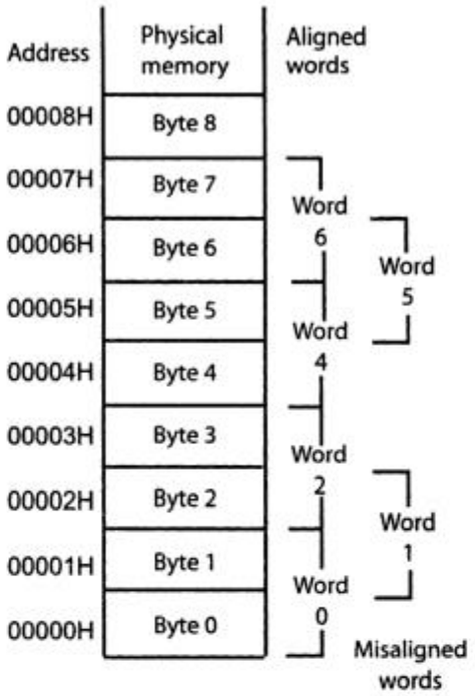


Figure 2–5 Examples of aligned and misaligned data words.

word of this pointer that is stored at the higher address is called the *segment base address*, and the word at the lower address is called the *offset*.

Just like for words, a double word of data can be aligned or misaligned. An aligned double word is located at an address that is a multiple of 4 (e.g., 00000_{16}, 00004_{16}, and 00008_{16}). A number of aligned and misaligned double words of data are shown in Fig. 2–6. Of these six examples, only double words 0 and 4 are aligned double words.

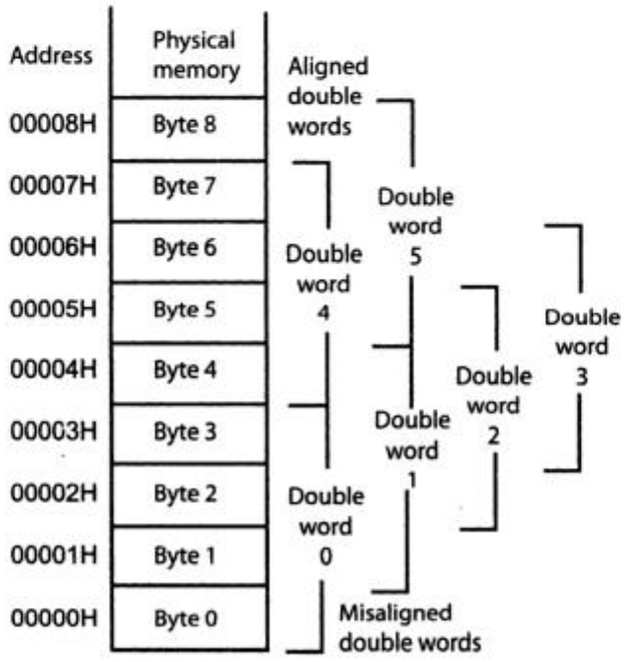


Figure 2–6 Examples of aligned and misaligned double words of data.

Copyrighted material

An example showing the storage of a pointer in memory is given in Fig. 2–7(a). Here the higher-addressed word, which represents the segment base address, is stored starting at even-address boundary 00006_{16}. The most significant byte of this word is at address 00007_{16} and equals $00111011_2 = 3B_{16}$. Its least significant byte is at address 00006_{16} and equals $01001100_2 = 4C_{16}$. Combining these two values, we get the segment base address, which equals $0011101101001100_2 = 3B4C_{16}$.

The offset part of the pointer is the lower-addressed word. Its least significant byte is stored at address 00004_{16}; this location contains $01100101_2 = 65_{16}$. The most significant byte is at address 00005_{16}, which contains $00000000_2 = 00_{16}$. The resulting offset is $0000000001100101_2 = 0065_{16}$. The complete double word is $3B4C0065_{16}$. Since this double word starts at address 00004_{16}, it is an example of an aligned double word of data.

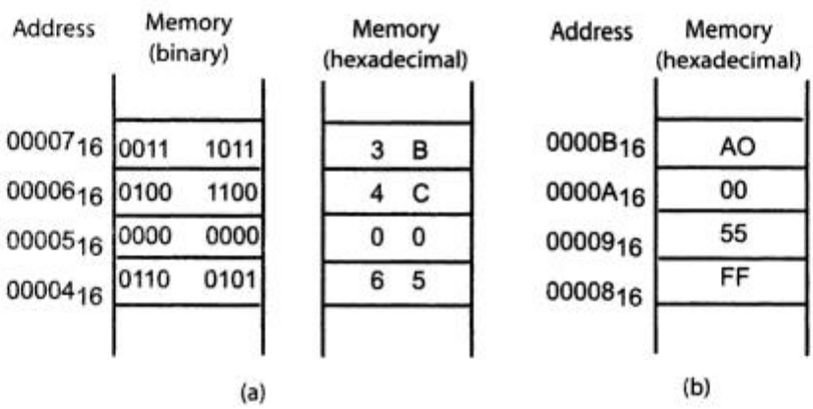


Figure 2–7 (a) Storing a 32-bit pointer in memory. (b) An example.

EXAMPLE 2.2

How should the pointer with segment base address equal to $A000_{16}$ and offset address $55FF_{16}$ be stored at an even-address boundary starting at 00008_{16}? Is the double word aligned or misaligned?

Solution

Storage of the two-word pointer requires four consecutive byte locations in memory, starting at address 00008_{16}. The least-significant byte of the offset is stored at address 00008_{16} and is shown as FF_{16} in Fig. 2–7(b). The most significant byte of the offset, 55_{16}, is stored at address 00009_{16}. These two bytes are followed by the least significant byte of the segment base address, 00_{16}, at address $0000A_{16}$, and its most significant byte, $A0_{16}$, at address $0000B_{16}$. Since the double word is stored in memory starting at address 00008_{16}, it is aligned.

▲ 2.4 DATA TYPES

The preceding section identified the fundamental data formats of the 8088 as the byte (8 bits), word (16 bits), and double word (32 bits). It also showed how each of these elements is stored in memory. The next step is to examine the types of data that can be coded into these formats for processing.

The 8088 microprocessor directly processes data expressed in a number of different data types. Let us begin with the *integer data type*. The 8088 can process data as either *unsigned* or *signed integer* numbers; each type of integer can be either byte-wide or word-wide. Figure 2–8(a) represents an unsigned byte integer; this data type can be used to represent decimal numbers in the range 0 through 255. The unsigned word integer is shown in Fig. 2–8(b); it can be used to represent decimal numbers in the range 0 through 65,535.

Copyrighted material

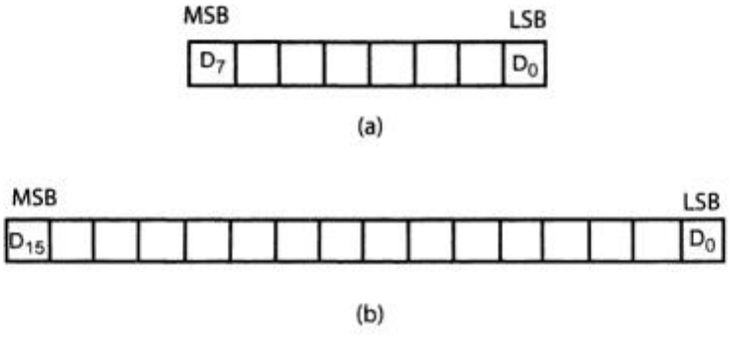


Figure 2–8 (a) Unsigned byte integer. (b) Unsigned word integer.

EXAMPLE 2.3

What value does the unsigned word integer 1000_{16} represent?

Solution

First, the hexadecimal integer is converted to binary form:

$$1000_{16} = 0001000000000000_2$$

Next, we find the value for the binary number:

$$0001000000000000_2 = 2^{12} = 4096$$

The signed byte integer and signed word integer in Figs. 2–9(a) and (b) are similar to the unsigned integer data types just introduced; however, here the most significant bit is a sign bit. A zero in this bit position identifies a positive number. For this reason, the signed integer byte can represent decimal numbers in the range +127 to –128, and the signed integer word permits numbers in the range +32,767 to –32,768, respectively. For example, the number +3 expressed as a signed integer byte is 00000011_2 (03_{16}). On the other hand, the 8088 always expresses negative numbers in 2's-complement notation. Therefore, –3 is coded as 11111101_2 (FD_{16}).

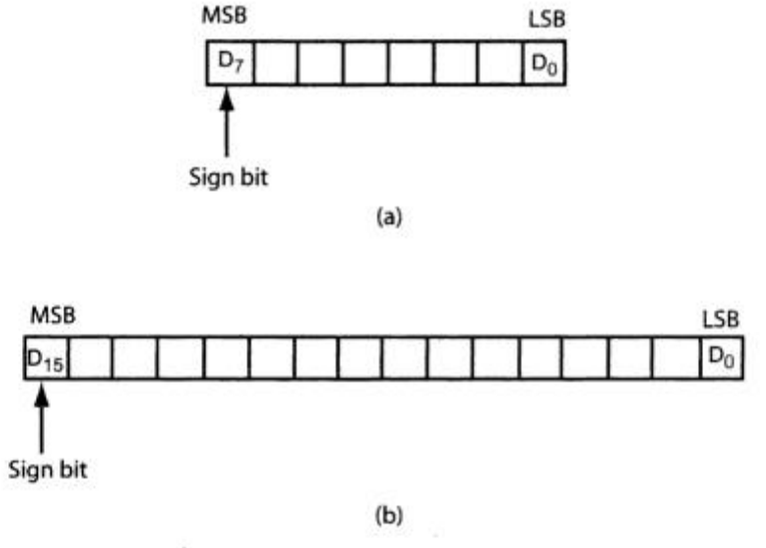


Figure 2–9 (a) Signed byte integer. (b) Signed word integer.

Copyrighted material

EXAMPLE 2.4

A signed word integer equals $FEFF_{16}$. What decimal number does it represent?

Solution

Expressing the hexadecimal number in binary form gives

$$FEFF_{16} = 1111111011111111_2$$

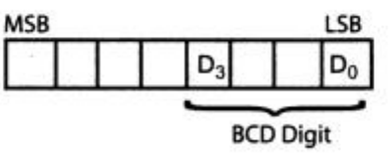
Since the most significant bit is 1, the number is negative and is in 2's complement form. Converting to its binary equivalent by subtracting 1 from the least significant bit and then complementing all bits gives

$$FEFF_{16} = -0000000100000001_2$$
$$= -257$$

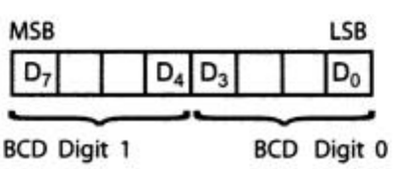
The 8088 can also process data that is coded as *binary-coded decimal* (BCD) *numbers*. Figure 2–10(a) lists the BCD values for decimal numbers 0 through 9. BCD data can be stored in either unpacked or packed form. For instance, the unpacked BCD byte in Fig. 2–10(b) shows that a single BCD digit is stored in the four least significant bits, and the upper four bits are set to 0. Figure 2–10(c) shows a byte with packed BCD digits. Here two BCD numbers are stored in a byte. The upper four bits represent the most significant digit of a two-digit BCD number.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

(a)



(b)



(c)

Figure 2–10 (a) BCD numbers. (b) An Unpacked BCD digit. (c) Packed BCD digits.

Copyrighted material

EXAMPLE 2.5

The packed BCD data stored at byte address 01000_{16} equal 10010001_2. What is the two-digit decimal number?

Solution

Writing the value 10010001_2 as separate BCD digits gives

$$10010001_2 = 1001_{BCD}0001_{BCD} = 91_{10}$$

Information expressed in ASCII (*American Standard Code for Information Interchange*) can also be directly processed by the 8088 microprocessor. The chart in Fig. 2–11 (a) shows how numbers, letters, and control characters are coded in ASCII. For instance, the number 5 is coded as

$$H_1H_0 = 011010l_2 = 35H$$

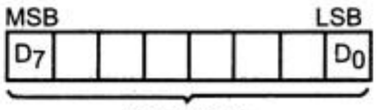
where H denotes that the ASCII-coded number is in hexadecimal form. As shown in Fig. 2–11(b), ASCII data are stored as one character per byte.

	b_7 b_6 b_5	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1	H_1 / H_0	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P		p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	A	LF	SUB	*	:	J	Z	j	z
1 0 1 1	B	V	ESC	+	;	K	[	k	}
1 1 0 0	C	FF	FS	,	<	L	\	l	|
1 1 0 1	D	CR	GS	-	=	M	]	m	{
1 1 1 0	E	SO	RS	.	>	N	^	n	~
1 1 1 1	F	SI	US	/	?	O	-	o	DEL

(a)

Figure 2–11 (a) ASCII table.

Copyrighted material



(b)

Figure 2–11 (b) ASCII digit.

EXAMPLE 2.6

Byte addresses 01100_{16} through 01104_{16} contain the ASCII data 01000001, 01010011, 01000011, 01001001, and 01001001, respectively. What do the data stand for?

Solution

Using the chart in Fig. 2–11(a), the data are converted to ASCII as follows:

$$(01100H) = 01000001_{ASCII} = A$$

$$(01101H) = 01010011_{ASCII} = S$$

$$(01102H) = 01000011_{ASCII} = C$$

$$(01103H) = 01001001_{ASCII} = I$$

$$(01104H) = 01001001_{ASCII} = I$$

▲ 2.5 SEGMENT REGISTERS AND MEMORY SEGMENTATION

Even though the 8088 has a 1 Mbyte address space, not all this memory is active at one time. Actually, the 1 Mbytes of memory are partitioned into 64 Kbyte (65,536) *segments*. A segment represents an independently addressable unit of memory consisting of 64 K consecutive byte-wide storage locations. Each segment is assigned a *base address* that identifies its starting point—that is, its lowest address byte-storage location.

Only four of these 64 Kbyte segments are active at a time: the *code segment*, *stack segment*, *data segment*, and *extra segment*. The segments of memory that are active, as shown in Fig. 2–12, are identified by the values of addresses held in the 8088's four internal segment registers: *CS* (code segment), *SS* (stack segment), *DS* (data segment), and *ES* (extra segment). Each of these registers contains a 16-bit base address that points to the lowest addressed byte of the segment in memory. Four segments give a maximum of 256 Kbytes of active memory. Of this, 64 Kbytes are for *program storage* (code), 64 Kbytes are for a *stack*, and 128 Kbytes are for *data storage*.

The values held in these registers are referred to as the *current-segment register values*; for example, the value in CS points to the first word-wide storage location in the current code segment. Code is always fetched from memory as words, not as bytes.

Copyrighted material

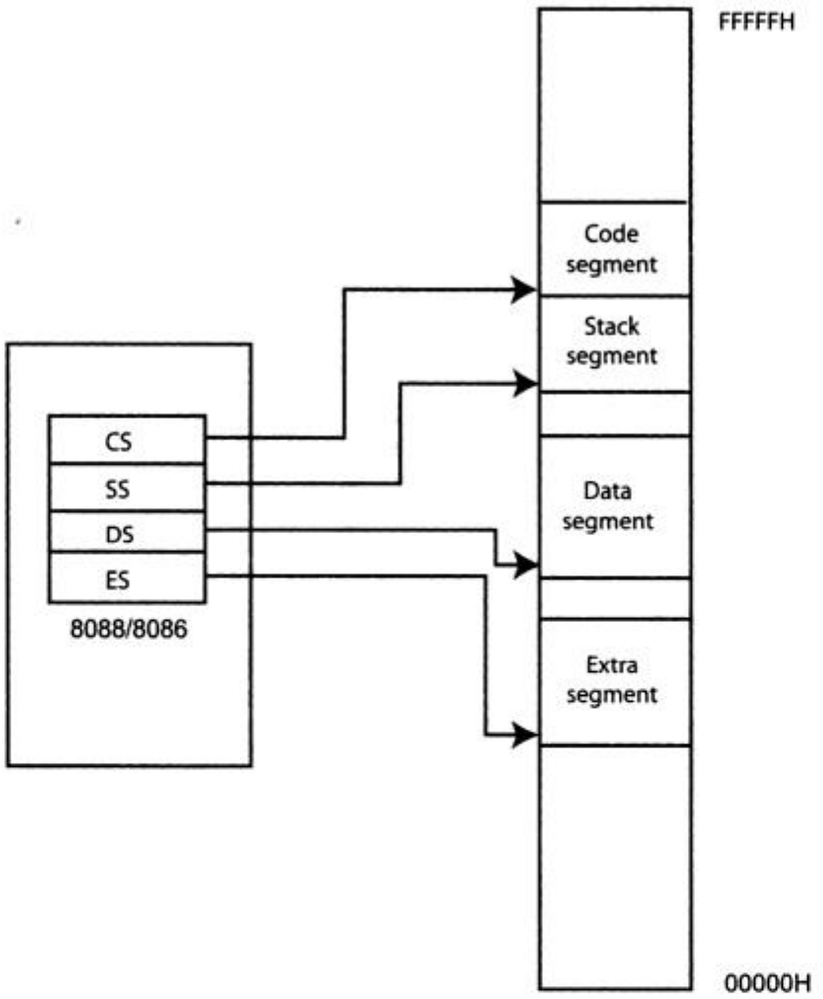


Figure 2–12 Active segments of memory.

Figure 2–13 illustrates the *segmentation of memory*. In this diagram, the 64 Kbyte segments are identified with letters such as A, B, and C. The data segment (DS) register contains the value B. Therefore, the second 64 Kbyte segment of memory from the top, labeled B, acts as the current data storage segment This is one of the segments in which data that are to be processed by the microcomputer are stored. For this reason, this part of the microcomputer's memory address space must contain read/write storage locations that can be accessed by instructions as storage locations for source and destination operands. CS selects segment E as the code segment. It is this segment of memory from which instructions of the program are currently being fetched for execution. The stack segment (SS) register contains H, thereby selecting the 64 Kbyte segment labeled as H for use as a stack. Finally, the extra segment (ES) register is loaded with value J such that segment J of memory functions as a second 64 Kbyte data storage segment.

The segment registers are said to be user accessible. This means that the programmer can change their contents through software. Therefore, for a program to gain access to another part of memory, one simply has to change the value of the appropriate register or registers. For instance, a new data space, with up to 128 Kbytes, is brought in simply by changing the values in DS and ES.

There is one restriction on the value assigned to a segment as a base address: it must reside on a 16-byte address boundary. This is because increasing the 16-bit value in a segment register by

Copyrighted material

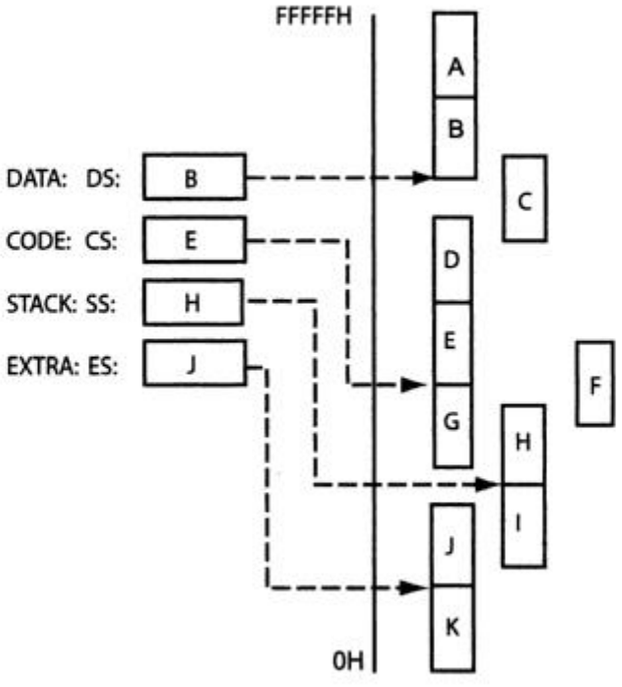


Figure 2–13 Contiguous, adjacent, disjointed, and overlapping segments. (Reprinted by permission of Intel Corp., Copyright/Intel Corp. 1979)

1 actually increases the corresponding memory address by 16; examples of valid base addresses are 00000_{16}, 00010_{16}, and 00020_{16}. Other than this restriction, segments can be set up to be contiguous, adjacent, disjointed, or even overlapping; for example, in Fig. 2–13, segments A and B are contiguous, whereas segments B and C are overlapping.

▲ 2.6 DEDICATED, RESERVED, AND GENERAL-USE MEMORY

Any part of the 8088 microcomputer's 1 Mbyte address space can be implemented for the user's access; however, some address locations have *dedicated functions* and should not be used as general memory for storage of data or instructions of a program. Let us now look at these reserved, dedicated use, and general-use parts of memory.

Figure 2–14 shows the *reserved, dedicated-use*, and *general-use* parts of the 8088/8086's *address space*. Notice that storage locations from address 00000_{16} to 00013_{16} are dedicated, and those from address 00014_{16} to $0007F_{16}$ are reserved. These 128 bytes of memory are used for storage of pointers to interrupt service routines. The dedicated part is used to store the pointers for the 8088's internal interrupts and exceptions. On the other hand, the reserved locations are saved to store pointers that are used by the user-defined interrupts. As indicated earlier, a pointer is a two-word address element and requires 4 bytes of memory. The word of this pointer at the higher address is called the segment base address and the word at the lower address is the offset. Therefore, this section of memory contains up to 32 pointers.

The part of the address space labeled *open in* Fig. 2–14 is *general-use memory* and is where data or instructions of the program are stored. Notice that the general-use area of memory is the range from addresses 80_{16} through $FFFEF_{16}$.

Copyrighted material

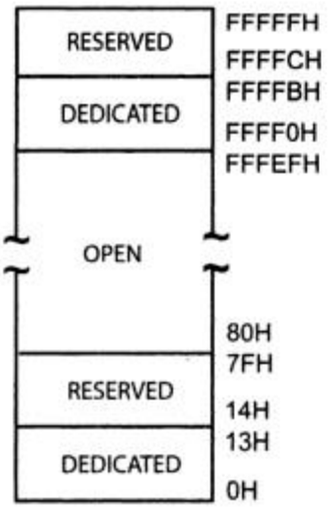


Figure 2–14 Dedicated-use,reserved, and general-use memory. (Reprinted by permission of Intel Corp., Copyright/Intel Corp. 1979)

At the high end of the memory address space is another reserved pointer area, located from address $FFFFC_{16}$ through $FFFFF_{16}$. These four memory locations are reserved for use with future products and should not be used. Intel Corporation, the original manufacturer of the 8088, has identified the 12 storage locations from address $FFFF0_{16}$ through $FFFFB_{16}$ as dedicated for functions such as storage of the hardware reset jump instruction. For instance, address $FFFF0_{16}$ is where the 8088/8086 begins execution after receiving a reset.

▲ 2.7 INSTRUCTION POINTER

The register that we will consider next in the 8088's software model shown in Fig. 2–2 is the *instruction pointer* (IP). IP is also 16 bits in length and identifies the location of the next word of instruction code to be fetched from the current code segment of memory. The IP is similar to a program counter; however, it contains the offset of the next word of instruction code instead of its actual address. This is because IP and CS are both 16 bits in length, but a 20-bit address is needed to access memory. Internal to the 8088, the offset in IP is combined with the current value in CS to generate the address of the instruction code. Therefore, the value of the address for the next code access is often denoted as CS:IP.

During normal operation, the 8088 fetches instructions from the code segment of memory, stores them in its instruction queue, and executes them one after the other. Every time a word of code is fetched from memory, the 8088 updates the value in IP such that it points to the first byte of the next sequential word of code—that is, IP is incremented by 2. Actually, the 8088 prefetches up to four bytes of instruction code into its internal code queue and holds them there waiting for execution.

After an instruction is read from the output of the instruction queue, it is decoded; if necessary, operands are read from either the data segment of memory or internal registers. Next, the operation specified in the instruction is performed on the operands and the result is written back either in an internal register or a storage location in memory. The 8088 is now ready to execute the next instruction in the code queue.

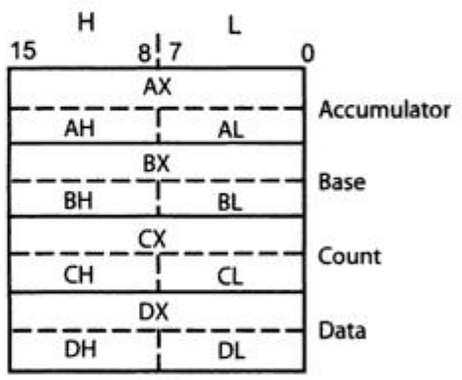
Copyrighted material

Executing an instruction that loads a new value into the CS register changes the active code segment; thus, any 64 Kbyte segment of memory can be used to store the instruction code.

▲ 2.8 DATA REGISTERS

As Fig. 2–2 shows, the 8088 has four *general-purpose data registers*. During program execution, they hold temporary values of frequently used intermediate results. Software can read, load, or modify their contents. Any of the general-purpose data registers can be used as the source or destination of an operand during an arithmetic operation such as ADD or a logic operation such as AND. For instance, the values of two pieces of data, A and B, could be moved from memory into separate data registers and operations such as addition, subtraction, and multiplication performed on them. The advantage of storing these data in internal registers instead of memory during processing is that they can be accessed much faster.

The four registers, known as the *data registers*, are shown in detail in Fig. 2–15(a). Notice that they are referred to as the *accumulator register* (A), the *base register* (B), the *count register* (C), and the *data register* (D). These names imply special functions they are meant to perform for the 8088 microprocessor. Figure 2–15(b) summarizes these operations. Notice that string and loop operations



(a)

Register	Operations
AX	Word multiply, word divide, word I/O
AL	Byte multiply, byte divide, byte I/O, translate, decimal arithmetic
AH	Byte multiply, byte divide
BX	Translate
CX	String operations, loops
CL	Variable shift and rotate
DX	Word multiply, word divide, indirect I/O

(b)

Figure 2–15 (a) General-purpose data registers. (Reprinted by permission of Intel Corp., Copyright/Intel Corp. 1979) (b) Dedicated register functions. (Reprinted by permission of Intel Corp., Copyright/Intel Corp. 1979)

use the C register. For example, the value in the C register is the number of bytes to be processed in a string operation. This is the reason it is given the name *count register*. Another example of the dedicated use of data registers is that all input/output operations must use accumulator register AL or AX for data.

Each of these registers can be accessed either as a whole (16 bits) for word data operations or as two 8-bit registers for byte-wide daa operations. An X after the register letter identifies the reference of a register as a word; for instance the 16-bit accumulator is referenced as AX. Similarly, the other three word registers are referred to as BX, CX, and DX.

On the other hand, when referencing one of these registers on a byte-wide basis, following the register name with the letter H or L, respectively, identifies the high byte and low byte. For the A register, the most significant byte is referred to as AH and the least significant byte as AL; the other byte-wide register pairs are BH and BL, CH and CL, and DH and DL. When software places a new value in one byte of a register, for instance AL, the value in the other byte (AH) does not change. This ability to

Copyrighted material