Segment Registers

Within the 1MB of memory space the 8086/88 defines four 64K-byte memory blocks called the *code* segment, *stack* segment, *data* segment, and *extra* segment. Each of these blocks of memory is used differently by the processor.

The code segment holds the program instruction codes. The data segment stores data for the program. The extra segment is an extra data segment (often used for shared data). The stack segment is used to store interrupt and subroutine return addresses (explained more fully in Sec. 2.8).

You should realize that the concept of a segmented memory is a unique one. Older-generation microprocessors such as the 8-bit 8085 or Z-80 could access only one 64K-byte segment. This meant that the program instructions, data, and subroutine stack all had to share the same memory. This limited the amount of memory available for the program itself and led to disaster if the stack should happen to overwrite the data or program areas.

The four segment registers shown in Figs. 2.1 and 2.3 (CS, DS, ES, and SS) are used to "point" at location 0 (the base address) of each segment. This is a little "tricky" because the segment registers are only 16 bits wide, but the memory address is 20 bits wide. The BIU takes care of this problem by appending four 0's to the low-order bits of the segment register. In effect, this multiplies the segment register contents by 16. Fig. 2.9 shows an example.

The CS register contains B3FFH but is interpreted as pointing to address B3FF0H. The point to note is that the beginning segment address is not arbitrary—*it must begin at an address divisible by 16*. Another way of saying this is that the low-order hex digit must be 0.

Also note that the four segments need not be defined separately. In Fig. 2.9 the stack and extra segments are partially overlapped. Indeed, it is allowable for all four segments to completely overlap (CS = DS = ES = SS).

Example 2.2

Calculate the beginning and ending addresses for the data segment assuming that register DS = E000H.

Solution The base address is found by appending four 0's. Base address: E0000H. The ending address is found by adding FFFFH (64K). Ending address: E0000H + FFFFH = EFFFFH.

Memory locations not defined to be within one of the current segments cannot be accessed by the 8086/88 without first redefining one of the segment registers to include that location. Thus at any given instant a maximum of 256K (64K × 4) bytes of memory can be utilized. As we will see, the contents of the segment registers can only be specified via software. As you might imagine, instructions to load these registers should be among the first given in any 8086/88 program.

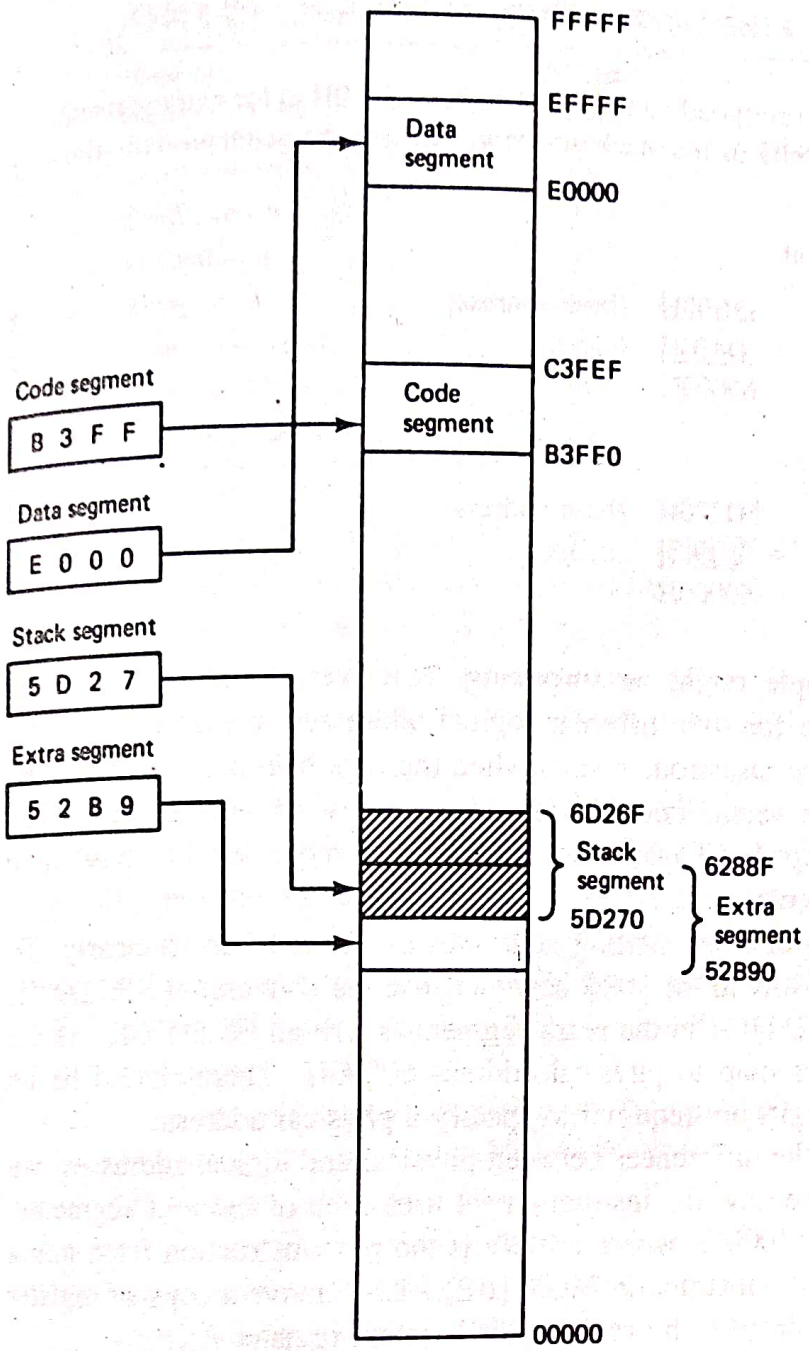

Figure 2.9 The 8086/88 divides its 1M bytes of memory address space into four segments, called the data, code, stack, and extra segments. The four segment registers DS, CS, SS, and ES point to location 0 of the current segment. In this example the stack and extra segments are partially overlapped. (From J. Uffenbeck, *Microcomputers and Microprocessors: The 8080, 8085, and Z-80*. Prentice-Hall, Englewood Cliffs, N.J., 1985.)

Logical and Physical Addresses

Addresses within a segment can range from address 0 to address FFFFH. This corresponds to the 64K-byte length of the segment. An address within a segment is called an *offset* or *logical address*. For example, logical address 0005H in the code segment shown in Fig. 2.9 actually corresponds to the real address B3FF0H + 5 = B3FF5H. This "real" address is called the *physical address*.

What is the difference between the physical and the logical address? The physical address is 20 bits long and corresponds to the actual binary code output by the BIU on the address bus lines. The logical address is an offset from location 0 of a given segment.

Example 2.3

Calculate the physical address corresponding to logical address D470H in the extra segment. Repeat for logical address 2D90H in the stack segment. Assume the segment definitions shown in Fig. 2.9.

Solution For the extra segment

```
  52B90H   (base address)
+ D470H    (offset)
  60000H
```

and for the stack segment

```
  5D270H   (base address)
+ 2D90H    (offset)
  60000H
```

The result of this example might be surprising. However, when two segments overlap it is certainly possible for two different logical addresses to map to the same physical address. This can have disastrous results when the data begins to overwrite the subroutine stack area, or vice versa. For this reason you must be very careful when segments are allowed to overlap. In Chaps. 3 and 4 we provide more detail on how these segments are defined in a program.

You should also be careful when writing addresses on paper to do so clearly. To specify the logical address 2D90H in the stack segment, use the convention SS:2D90H. Similarly, the logical address D470H in the extra segment is written ES:D470H. As we have just seen, both addresses map to physical address 60000H. There should be no ambiguity here, as five hex digits are required to specify a physical address.

Now that we have seen the differences between physical and logical addresses, we must dig a little deeper and see how the instruction set uses each of the four segments. For example, if register IP = 1000H, where exactly is the next instruction fetch going to come from? Or where will the instruction MOV [BP],AL—"move a copy of register AL to the memory location pointed to by register BP"—store register AL?

The answer is contained in Table 2.1. Every instruction that references memory has a *default* segment register, as shown. Instruction fetches occur only from the code segment, with IP supplying the offset or logical address. Similarly, register BP used as a pointer defaults to the stack segment.

Table 2.1 is programmed into the BIU. If IP = 1000H and CS = B3FFH, the BIU will form the physical address B3FF0H + 1000H = B4FF0H and fetch the byte stored at this physical address.

Example 2.4

What physical memory location is accessed by the instruction MOV [BP],AL if BP = 2C30H? Assume the segment definitions shown in Fig. 2.9.

Solution Table 2.1 indicates that the stack segment will be used. The physical address is

```
  5D270H
+ 2C30H
  5FEA0H
```

TABLE 2.1 SEGMENT REGISTER ASSIGNMENTS

Type of memory reference	Default segment	Alternate segment	Offset (logical address)
Instruction fetch	CS	None	IP
Stack operation	SS	None	SP
General data	DS	CS, ES, SS	Effective address
String source	DS	CS, ES, SS	SI
String destination	ES	None	DI
BX used as pointer	DS	CS, ES, SS	Effective address
BP used as pointer	SS	CS, ES, DS	Effective address

Table 2.1 indicates that some memory references can have their segment definitions changed. For example, BP can also be used as a pointer into the code, data, or extra segments. On the other hand, instruction codes can only be stored in the code segment with IP used as the offset. Similarly, string destinations always use the extra segment. In Sec. 2.4 we show how the *segment override* is used to access these alternate segments.

Advantages of Segmented Memory

Segmented memory can seem confusing at first. What you must remember is that the program op-codes will be fetched from the code segment, while program data variables will be stored in the data and extra segments. Stack operations use registers BP or SP and the stack segment. As we begin writing programs the consequences of these definitions will become clearer.

An immediate advantage of having separate data and code segments is that one program can work on several different sets of data. This is done by reloading register DS to point to the new data.

Perhaps the greatest advantage of segmented memory is that programs that reference logical addresses only can be loaded and run anywhere in memory. This is because the logical addresses always range from 0000 to FFFFH, independent of the code segment base.

Consider a *multitasking* environment in which the 8086/88 is doing several different jobs at once. An inactive program can be temporarily saved on a magnetic disk and a new program brought in to take its place—without concern for the physical location of this new program. Such programs are said to be *relocatable*, meaning that they will run at any location in memory. The requirements for writing relocatable programs are that no references be made to physical addresses, and no changes to the segment registers are allowed.

Defining Memory Locations

As a programmer you will seldom need to know the physical address of a memory location. Usually, only the logical addresses are important. Indeed, as we have just seen, the physical address depends on the contents of the segment registers even though the