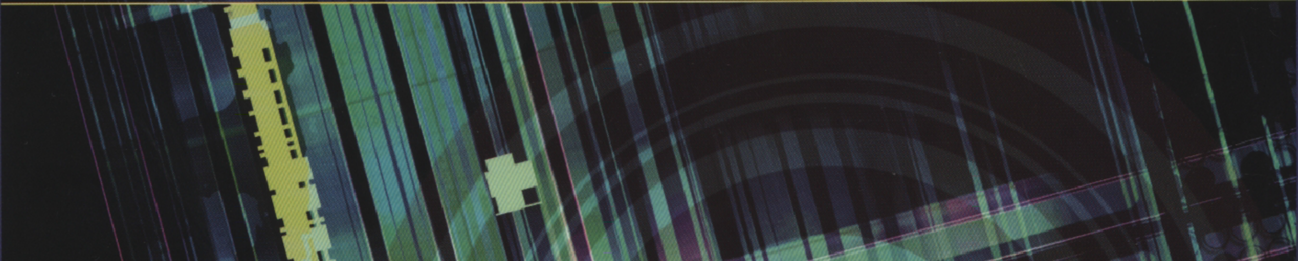
INTERNATIONAL EDITION

# THE 80x86 IBM PC AND COMPATIBLE COMPUTERS (VOLUMES I & II)

## ASSEMBLY LANGUAGE, DESIGN, AND INTERFACING

4th Edition



Muhammad Ali Mazidi
Janice Gillispie Mazidi

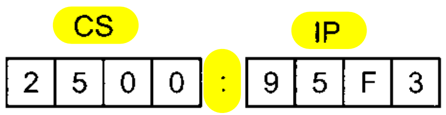
### Origin and definition of the segment

A segment is an area of memory that includes up to 64K bytes and begins on an address evenly divisible by 16 (such an address ends in 0H). The segment size of 64K bytes came about because the 8085 microprocessor could address a maximum of 64K bytes of physical memory since it had only 16 pins for the address lines ($2^{16} = 64K$). This limitation was carried into the design of the 8088/86 to ensure compatibility. Whereas in the 8085 there was only 64K bytes of memory for all code, data, and stack information, in the 8088/86 there can be up to 64K bytes of memory assigned to each category. Within an Assembly language program, these categories are called the code segment, data segment, and stack segment. For this reason, the 8088/86 can only handle a maximum of 64K bytes of code and 64K bytes of data and 64K bytes of stack at any given time, although it has a range of 1 megabyte of memory because of its 20 address pins ($2^{20}$ = 1 megabyte). How to move this window of 64K bytes to cover all 1 megabyte of memory is discussed below, after we discuss logical address and physical address.

### Logical address and physical address

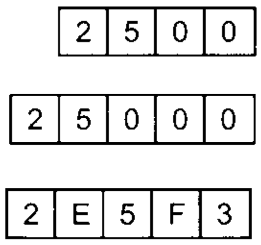
In Intel literature concerning the 8086, there are three types of addresses mentioned frequently: the physical address, the offset address, and the logical address. The *physical address* is the 20-bit address that is actually put on the address pins of the 8086 microprocessor and decoded by the memory interfacing circuitry. This address can have a range of 00000H to FFFFFH for the 8086 and real-mode 286, 386, and 486 CPUs. This is an actual physical location in RAM or ROM within the 1 megabyte memory range. The *offset address* is a location within a 64K-byte segment range. Therefore, an offset address can range from 0000H to FFFFH. The *logical address* consists of a segment value and an offset address. The differences among these addresses and the process of converting from one to another is best understood in the context of some examples, as shown next.

### Code segment



To execute a program, the 8086 fetches the instructions (opcodes and operands) from the code segment. The logical address of an instruction always consists of a CS (code segment) and an IP (instruction pointer), shown in CS:IP format. The physical address for the location of the instruction is generated by shifting the CS left one hex digit and then adding it to the IP. IP contains the offset address. The resulting 20-bit address is called the physical address since it is put on the external physical address bus pins to be decoded by the memory decoding circuitry. To clarify this important concept, assume values in CS and IP as shown in the diagram. The offset address is contained in IP; in this case it is 95F3H. The logical address is CS:IP, or 2500:95F3H. The physical address will be 25000 + 95F3 = 2E5F3H. The physical address of an instruction can be calculated as follows:

1. Start with CS.
2. Shift left CS.
3. Add IP.



The microprocessor will retrieve the instruction from memory locations starting at 2E5F3. Since IP can have a minimum value of 0000H and a maximum of FFFFH, the logical address range in this example is 2500:0000 to 2500:FFFF. This means that the lowest memory location of the code segment above will be 25000H (25000 + 0000) and the highest memory location will be 34FFFH (25000 + FFFF). What happens if the desired instructions are located beyond these two limits? The answer is that the value of CS must be changed to access those instructions. See Example 1-1.

**Example 1-1**

If CS = 24F6H and IP = 634AH, show:
(a) The logical address
(b) The offset address
and calculate:
(c) The physical address
(d) The lower range
(e) The upper range of the code segment

**Solution:**

(a) 24F6:634A (b) 634A
(c) 2B2AA (24F60 + 634A) (d) 24F60 (24F60 + 0000)
(e) 34F5F (24F60 + FFFF)

## Logical address vs. physical address in the code segment

In the code segment, CS and IP hold the logical address of the instructions to be executed. The following Assembly language instructions have been assembled (translated into machine code) and stored in memory. The three columns show the logical address of CS:IP, the machine code stored at that address and the corresponding Assembly language code. This information can easily be generated by the DEBUG program using the Unassemble command.

| Logical address<br>CS:IP | Machine language<br>opcode and operand | Assembly language<br>mnemonics and operand |
|---|---|---|
| 1132:0100 | B057 | MOV AL,57 |
| 1132:0102 | B686 | MOV DH,86 |
| 1132:0104 | B272 | MOV DL,72 |
| 1132:0106 | 89D1 | MOV CX,DX |
| 1132:0108 | 88C7 | MOV BH,AL |
| 1132:010A | B39F | MOV BL,9F |
| 1132:010C | B420 | MOV AH,20 |
| 1132:010E | 01D0 | ADD AX,DX |
| 1132:0110 | 01D9 | ADD CX,BX |
| 1132:0112 | 05351F | ADD AX,1F35 |

The program above shows that the byte at address 1132:0100 contains B0, which is the opcode for moving a value into register AL, and address 1132:0101 contains the operand (in this case 57) to be moved to AL. Therefore, the instruction "MOV AL,57" has a machine code of B057, where B0 is the opcode and 57 is the operand. Similarly, the machine code B686 is located in memory locations 1132:0102 and 1132:0103 and represents the opcode and the operand for the instruction "MOV DH,86". The physical address is an actual location within RAM (or even ROM). The following are the physical addresses and the contents of each location for the program above. Remember that it is the physical address that is put on the address bus by the 8086 CPU to be decoded by the memory circuitry:

| Logical address | Physical address | Machine code contents |
|---|---|---|
| 1132:0100 | 11420 | B0 |
| 1132:0101 | 11421 | 57 |
| 1132:0102 | 11422 | B6 |
| 1132:0103 | 11423 | 86 |
| 1132:0104 | 11424 | B2 |
| 1132:0105 | 11425 | 72 |
| 1132:0106 | 11426 | 89 |
| 1132:0107 | 11427 | D1 |
| 1132:0108 | 11428 | 88 |
| 1132:0109 | 11429 | C7 |
| 1132:010A | 1142A | B3 |
| 1132:010B | 1142B | 9F |
| 1132:010C | 1142C | B4 |
| 1132:010D | 1142D | 20 |
| 1132:010E | 1142E | 01 |
| 1132:010F | 1142F | D0 |
| 1132:0110 | 11430 | 01 |
| 1132:0111 | 11431 | D9 |
| 1132:0112 | 11432 | 05 |
| 1132:0113 | 11433 | 35 |
| 1132:0114 | 11434 | 1F |

### Data segment

Assume that a program is being written to add 5 bytes of data, such as 25H, 12H, 15H, 1FH, and 2BH, where each byte represents a person's daily overtime pay. One way to add them is as follows:

```
MOV   AL,00H        ;initialize AL
ADD   AL,25H        ;add 25H to AL
ADD   AL,12H        ;add 12H to AL
ADD   AL,15H        ;add 15H to AL
ADD   AL,1FH        ;add 1FH to AL
ADD   AL,2BH        ;add 2BH to AL
```

In the program above, the data and code are mixed together in the instructions. The problem with writing the program this way is that if the data changes, the code must be searched for every place the data is included, and the data retyped. For this reason, the idea arose to set aside an area of memory strictly for data. In 80x86 microprocessors, the area of memory set aside for data is called the data segment. Just as the code segment is associated with CS and IP as its segment register and offset, the data segment uses register DS and an offset value.

The following demonstrates how data can be stored in the data segment and the program rewritten so that it can be used for any set of data. Assume that the offset for the data segment begins at 200H. The data is placed in memory locations:

```
DS:0200 = 25
DS:0201 = 12
DS:0202 = 15
DS:0203 = 1F
DS:0204 = 2B
```

and the program can be rewritten as follows:

```
MOV   AL,0          ;clear AL
ADD   AL,[0200]     ;add the contents of DS:200 to AL
ADD   AL,[0201]     ;add the contents of DS:201 to AL
ADD   AL,[0202]     ;add the contents of DS:202 to AL
ADD   AL,[0203]     ;add the contents of DS:203 to AL
ADD   AL,[0204]     ;add the contents of DS:204 to AL
```

Notice that the offset address is enclosed in brackets. The brackets indicate that the operand represents the address of the data and not the data itself. If the brackets were not included, as in "MOV AL,0200", the CPU would attempt to move 200 into AL instead of the contents of offset address 200. Keep in mind that there is one important difference in the format of code for MASM and DEBUG in that DEBUG assumes that all numbers are in hex (no "H" suffix is required), whereas MASM assumes that they are in decimal and the "H" must be included for hex data.

This program will run with any set of data. Changing the data has no effect on the code. Although this program is an improvement over the preceding one, it can be improved even further. If the data had to be stored at a different offset address, say 450H, the program would have to be rewritten. One way to solve this problem would be to use a register to hold the offset address, and before each ADD, to increment the register to access the next byte. Next a decision must be made as to which register to use. The 8086/88 allows only the use of registers BX, SI, and DI as offset registers for the data segment. In other words, while CS uses only the IP register as an offset, DS uses only BX, DI, and SI to hold the offset address of the data. The term *pointer* is often used for a register holding an offset address. In the following example, BX is used as a pointer:

```
MOV   AL,0         ;initialize AL
MOV   BX,0200H     ;BX points to the offset addr of first byte
ADD   AL,[BX]      ;add the first byte to AL
INC   BX           ;increment BX to point to the next byte
ADD   AL,[BX]      ;add the next byte to AL
INC   BX           ;increment the pointer
ADD   AL,[BX]      ;add the next byte to AL
INC   BX           ;increment the pointer
ADD   AL,[BX]      ;add the last byte to AL
```

The "INC" instruction adds 1 to (increments) its operand. "INC BX" achieves the same result as "ADD BX,1". For the program above, if the offset address where data is located is changed, only one instruction will need to be modified and the rest of the program will be unaffected. Examining the program above shows that there is a pattern of two instructions being repeated. This leads to the idea of using a loop to repeat certain instructions. Implementing a loop requires familiarity with the flag register, discussed later in this chapter.

### Logical address and physical address in the data segment

The physical address for data is calculated using the same rules as for the code segment. That is, the physical address of data is calculated by shifting DS left one hex digit and adding the offset value, as shown in Examples 1-2, 1-3, and 1-4.

**Example 1-2**

Assume that DS is 5000 and the offset is 1950. Calculate the physical address of the byte.

**Solution:**

```
     DS        :   offset
| 5 | 0 | 0 | 0 | : | 1 | 9 | 5 | 0 |
```

The physical address will be 50000 + 1950 = 51950.

```
1. Start with DS.          | 5 | 0 | 0 | 0 |

2. Shift DS left.      | 5 | 0 | 0 | 0 | 0 |

3. Add the offset.     | 5 | 1 | 9 | 5 | 0 |
```

**Example 1-3**

If DS = 7FA2H and the offset is 438EH,

(a) Calculate the physical address. (b) Calculate the lower range.
(c) Calculate the upper range of the data segment. (d) Show the logical address.

**Solution:**

(a) 83DAE (7FA20 + 438E) (b) 7FA20 (7FA20 + 0000)
(c) 8FA1F (7FA20 + FFFF) (d) 7FA2:438E

**Example 1-4**

Assume that the DS register is 578C. To access a given byte of data at physical memory location 67F66, does the data segment cover the range where the data is located? If not, what changes need to be made?

**Solution:**

No, since the range is 578C0 to 678BF, location 67F66 is not included in this range. To access that byte, DS must be changed so that its range will include that byte.

**Little endian convention**

Previous examples used 8-bit or 1-byte data. In this case the bytes are stored one after another in memory. What happens when 16-bit data is used? For example:

```
MOV   AX,35F3H       ;load 35F3H into AX
MOV   [1500],AX      ;copy the contents of AX to offset 1500H
```

In cases like this, the low byte goes to the low memory location and the high byte goes to the high memory address. In the example above, memory location DS:1500 contains F3H and memory location DS:1501 contains 35H.

```
DS:1500 = F3         DS:1501 = 35
```

This convention is called little endian versus big endian. The origin of the terms *big endian* and *little endian* is from a *Gulliver's Travels* story about how an egg should be opened: from the little end or the big end. In the big endian method, the high byte goes to the low address, whereas in the little endian method, the high byte goes to the high address and the low byte to the low address. See Example 1-5. All Intel microprocessors and many minicomputers, notably the Digital VAX, use the little endian convention. Motorola microprocessors (used in the Macintosh),

**Example 1-5**

Assume memory locations with the following contents: DS:6826 = 48 and DS:6827 = 22. Show the contents of register BX in the instruction "MOV BX,[6826]".

**Solution:**

According to the little endian convention used in all 80x86 microprocessors, register BL should contain the value from the low offset address 6826 and register BH the value from offset address 6827, giving BL = 48H and BH = 22H.

DS:6826 = 48
DS:6827 = 22

| BH | BL |
|---|---|
| 22 | 48 |