Pointers Registers

The pointers will always store some address or memory location. In 8086 Microprocessor, they usually store the offset through which the actual address is calculated:

1. **Instruction Pointer (IP):** is a 16-bit register. IP in 8086 acts as a Program Counter. It points to the address of the next instruction to be executed. Its content is automatically incremented when the execution of a program proceeds further. It cannot be manipulated by instructions i.e it cannot appear as an operand in any instruction. The contents of the IP and Code Segment Register are used to compute the memory address of the instruction code to be fetched. This is done during the Fetch Cycle.
2. **Stack Pointer (SP)** is a 16-bit register pointing to program Stack, also contains 16-Bit offset address.
3. **Base Pointer (BP)** is a 16-bit register pointing to data in stack segment. BP register is usually used for based indexed or register indirect addressing.

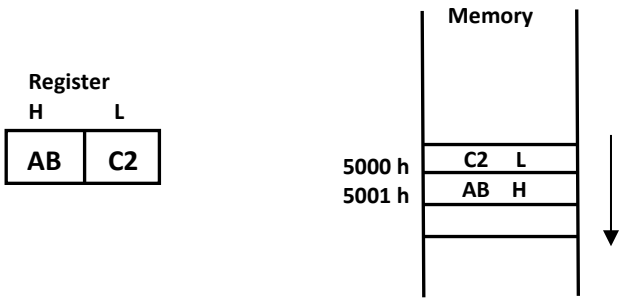
Index Registers

1. **Source Index (SI)** is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation Instructions
2. **Destination Index (DI)** is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data addresses in string manipulation instructions.

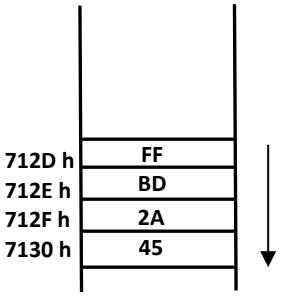
Note:
By default BX, SI and DI registers work with DS segment register
BP and SP work with SS segment register.
IP work with CS segment register.

Addressing data in memory

Consider the hex number ABC2 h , requires 2 bytes (one word) of memory. It consist of a high-order (most significant) byte , AB , and a low-order (least significant) byte , C2. The system stores the data in memory in reverse-byte sequence : The low-order byte in low memory address and the high-order byte in the high memory address. For example, the processor transfers ABC2 h from a register into memory addresses 5000 h and 5001 h :



Ex: Store the double word 452ABDFF h in the memory startind at address 712D h :



SEGMENTED MEMORY

A segment is a special area defined in a program that begins on a paragraph boundary , that is , at allocation evenly divisible by 16 dec (10 hex). This means that the beginning segment address is not arbitrary -it must begin at an address divisible by 16 or another way if saying that is the low-order hex digit must be 0. Although a segment may be located almost anywhere in memory and in real mode may be up to 64k bytes , it required only as much as the program requires for its execution.

Within the 1 MB of memory space the 8086/88 defines four 64K-byte memory blocks called the code segment, stack segment, data segment, and extra segment (four segment give a maximum of 256 K (64K * 4) bytes of active memory; can be utilized). Each of these blocks of memory is used differently by the processor as follows:

1. Code Segment (CS): contains the machine code instructions that are to be execute. Typically, the first executable instruction is at the start of this segment, and the operating system links to that location to begin program execution.

2. Stack Segment (SS) : contains any data and addresses that you need to save temporarily or for use by your own called subroutines.

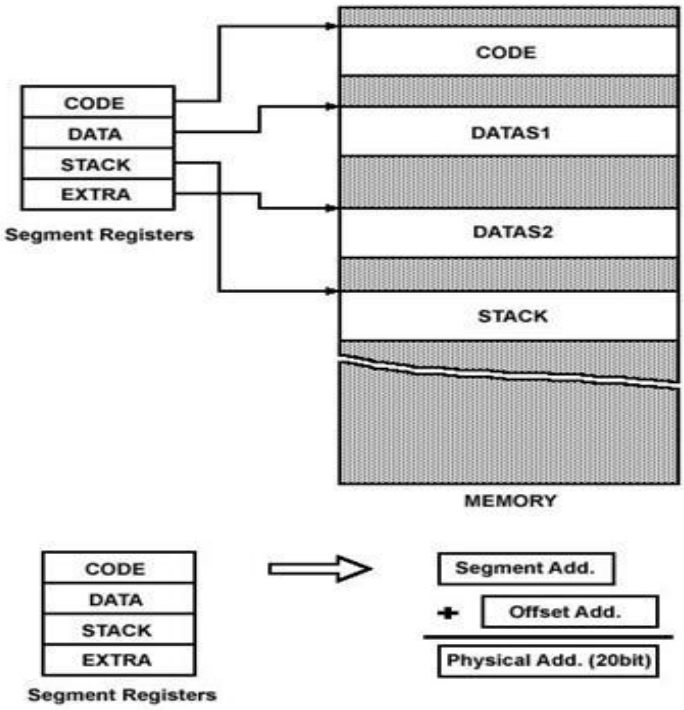
3. Data Segment (DS) : contains a program's data .

4. Extra Data (ES) : is used for certain string (character data) operations as the destination address.

Segment Registers

The BIU contains four 16-bit segment registers. They are: the extra segment (ES) register, the code segment (CS) registers, the data segment (DS) registers, and the stack segment (SS) registers. These segment registers are used to hold the upper 16 bits of the starting address for each of the segments. The part of a segment starting address stored in a segment register is often called the segment base.

1. **Code segment (CS) register** : Contains the starting address of a program's code segment. This segment address, plus an offset value in the instruction pointer (IP) register , indicates the address of an instruction to be fetched for execution. The CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.
2. **Stack segment (SS) register** : Permits the implementation of a stack in memory , which a program uses for temporary storage of addresses and data. The system stores the starting address of a program's stack segment in the SS register. This segment address , plus an offset value in the stack pointer (SP) register, indicates the current word in the stack being addressed.
3. **Data segment (DS) register** : Contains the starting address of a program's data segment. Instructions use this address to locate data : This address , plus an offset value in an instruction , causes a reference to a specific byte location in the data segment.
4. **Extra segment (ES) register** : Used by some string (character data) operations to handle memory addressing. The ES register is associated with the DI (index) register.



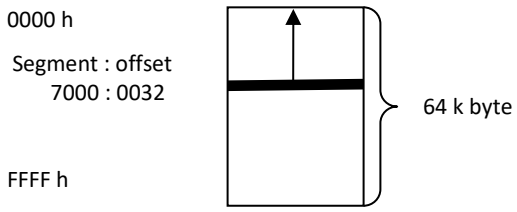
Advantages of segmented memory

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. some of the advantages of memory segmentation in the 8086 are as follows:

a) It allows the memory addressing capacity to be 1 MB even though the address associated with individual instruction is only 16-bit.
b) It allows instruction code, data, stack, and portion of program to be more than 64 KB long by using more than one code, data, stack segment, and extra segment.
c) It facilitates use of separate memory areas for program, data and stack.
d) It permits a program or its data to be put in different areas of memory, each time the program is executed i.e. program can be relocated which is very useful in multiprogramming.

Segment Offsets

Within a program , all memory locations are relative to a segment's starting address. The distance in byte from the segment address to another location within the segment is expressed as an offset (or displacement). A 2-byte (16 bits) offset can range from 0000 h through FFFF h (0 – 65,535) .To reference any memory location in a segment , the processor combines the segment address in a segment register with an offset value.



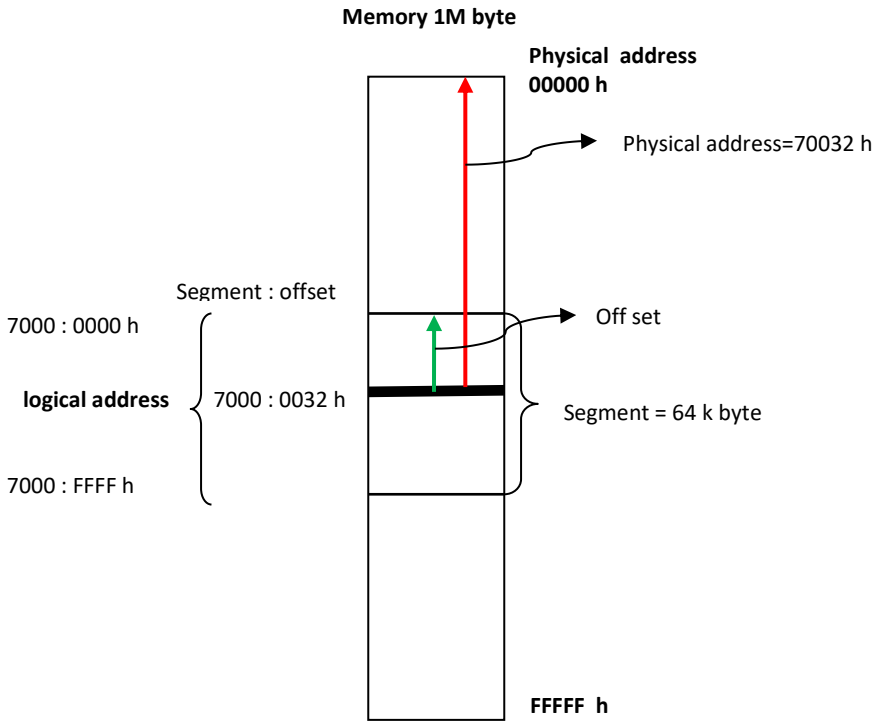
Logical And Physical Address

Addresses within a segment can range from address 0000 h to address FFFF h. This corresponds to the 64K-byte length of the segment. An address within a segment is called an offset or logical address. A logical address gives the offset (or displacement) from the address base of the segment to the desired location within it, as opposed to its "real" address, which maps directly anywhere into the 1 MB memory space. This "real" address is called the physical 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.

Each segment register is 16 bits wide while the address bus is 20 bits wide. The BIU takes care of this by appending four 0 s to the low order bits of the segment register. This means that the value in a Segment register is multiplied by 16 dec (10 h) and then the value in an offset register is added to it. So, the Physical Address (or Absolute address) for any combination of Segment and offset pairs is found by using this formula (Converting the logical address to the physical address):

Physical Address = (Segment address * 10 h) + Offset value (Absolute Memory Location)



After working through some examples, this will become much clearer to understand: The physical or Absolute address for the Segment:Offset pair, F000:FFFD can be computed by inserting a zero at the end of the Segment value (which is the same as multiplying by 10 h) and then adding the Offset value:

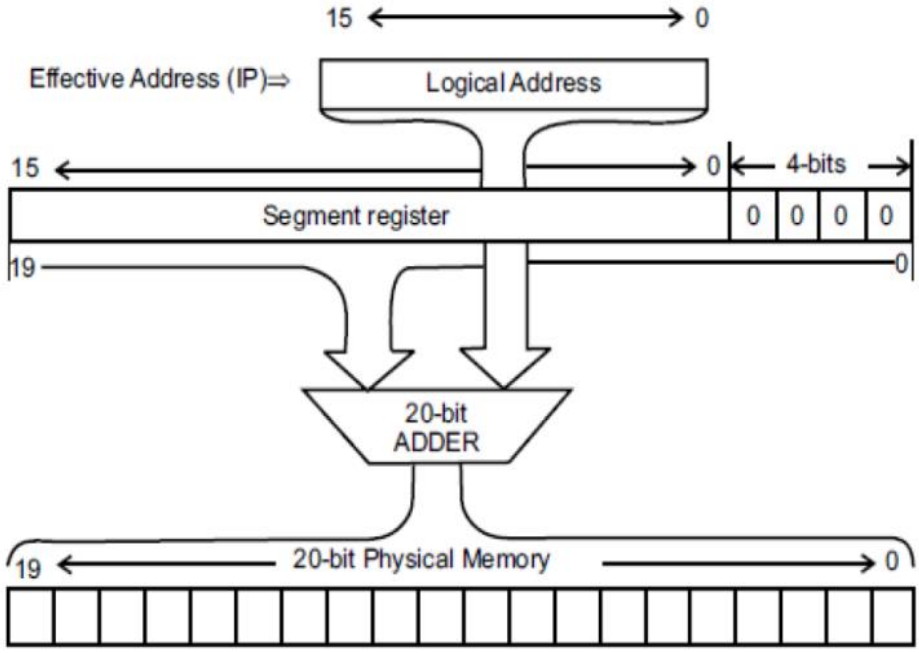
Segment address * 10 h	F0000
+ Offset value	+ FFFD
---	---
Physical Address	FFFFD

Example1: the logical address is 923F:E2FF, to find the physical address:

923F0
+ E2FF

A06EF

Example 2: If DS =2000H and the offset value is in register SI = 1234H, find the physical address?
PA = (2000)*10 + 1234H = 20000H + 1234H = 21234H



Types of Segmentation

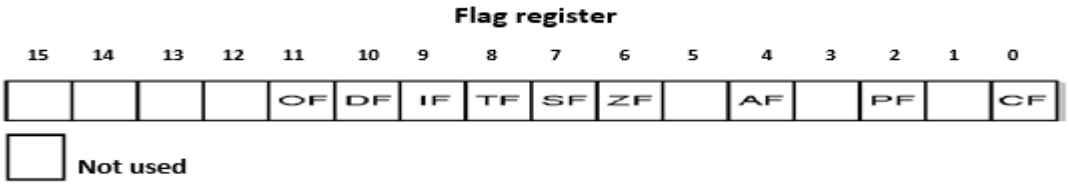
1. **Overlapping Segment**: A segment starts at a particular address and its maximum size can go up to 64kilobytes. But if another segment starts along with this 64kilobytes location of the first segment, then the two are said to be Overlapping Segment.

   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.

2. **Non-Overlapped Segment:** A segment starts at a particular address and its maximum size can go up to 64kilobytes. But if another segment starts before this 64kilobytes location of the first segment, then the two segments are said to be Non-Overlapped Segment.

Flags Registers:

The 8086 microprocessor has a 16 bits for Flag register. In this register nine of the 16 bits are active for flags to indicate the current status of the computer and the results of processing. These 9 bits are divided into two parts: 6 bits are status flags and 3 bits are control flags. The remaining 7 bits flags marked 'U' are undefined flags (unused).



Status Flags: In 8086 there are 6 different status flags which are set or reset after 8-bit or 16-bit operations. They are modified automatically by CPU after mathematical operations. This allows to determine the type of the result. These flags and their functions are listed below.

1. **CF (Carry Flag)**

Contains carries from a high-order (leftmost) bit following an arithmetic operation when there is unsigned overflow ; also , contains the content of the last bit of a shift or rotate operation.

2. **PF (Parity Flag)**

This flag is used to indicate the parity of result. If lower order (rightmost) 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity Flag is reset.

3. **AF (Auxiliary Flag)**

If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. b0 – b3) to upper nibble (i.e. b4 – b7), the AF flag is set i.e. carry given by b3 bit to b4 is AF flag. This is not a general-purpose flag; it is used internally by the processor to perform Binary to BCD conversion.

4. **ZF (Zero Flag)**

It is set; if the result of arithmetic or logical operation is zero else it is reset.

5. **SF (Sign Flag)**

In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set else it is reset if the result of operation is positive .

6. **OF (Overflow Flag)**

It will be set if the result of a signed operation is too large to fit in the number of bits available to represent it.

Control flags: In 8086 there are 3 different flags which are used to enable or disable some basic operations of the microprocessor. These flags and their functions are listed below:

1. **TF (Trap Flag):** controls the operation of the microprocessor. (TF=0 Normal operation , TF=1 Single Step operation).

2. **IF (Interrupt Flag):** controls the interrupt operation in 8086μP. (IF=0 Disable interrupt, IF=1 Enable interrupt).
3. **DF (Direction Flag) :** controls the direction of operation of string instructions. (DF=0 Ascending order, DF=1 Descending order)

Example: How the flag register changed after add 7F with 1 ?

```
   7F
+   1
_____
```

CF=0 80

CF=0 ; there is no carry out of bit 7
PF=0 ; the lower order (rightmost) 8-bits (80) has odd number of logic ones
AF=1 ; there is a carry out of bit 3 into bit 4
ZF=0 ; the result is not zero
SF=1 ; the bit 7 is 1 means the result is negative.
OF=1; the sign bit has changed (the two number are positive but the result is negative)

Another example : How the flag register changed after add these words A2EE h with 3605h ?

```
   A2EE
+  7605
_______
```

CF=1 18F3

CF=1; there is a carry out of bit 15
PF=1 ; the lower order (rightmost) 8-bits (F3) has even number of logic ones
AF=1 ; there is a carry out of bit 3 into bit 4
ZF=0 ; the result is not zero.
SF=0 ; the bit 15 is 0 means the result is positive.
OF=0; there is not overflow