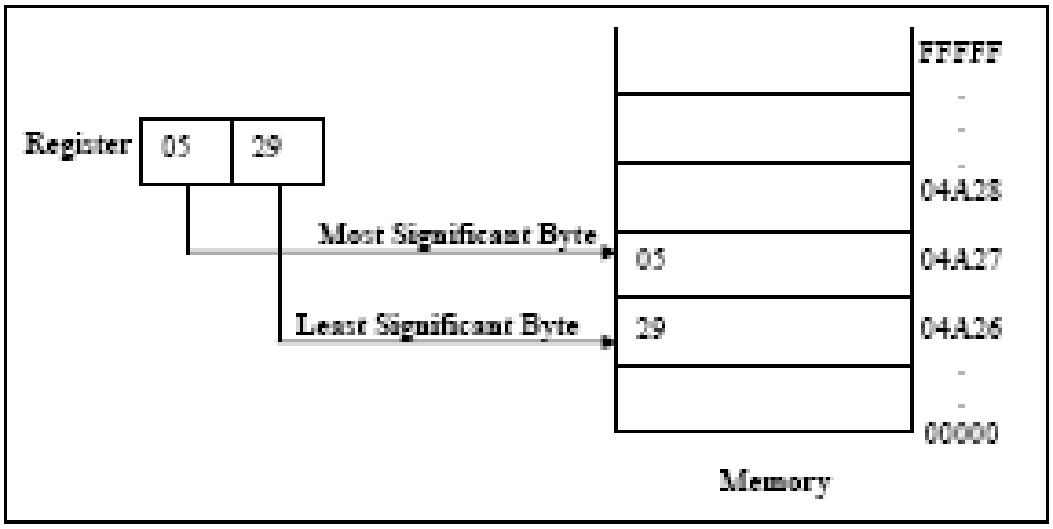
Microprocessors (0630371)
Fall 2010/2011 – Lecture Notes # 2

Outline of the Lecture

- **Addressing Data in Memory**
- **Execution Unit and Bus Interface Unit**
- **Registers of Microprocessor**

Addressing Data in Memory

Depending on the model, the processor can access one or more bytes of memory at a time. Consider the Hexa value (0529H) which requires two bytes. It consist of high order (most significant) byte 05 and a low order (least significant) byte 29. The processor store the data in memory in reverse byte sequence i.e. the low order byte in the low memory address and the high order byte in the high memory address. For example, the processor transfer the value 0529H from a register into memory addresses 04A26 H and 04A27H like this:



Memory addressing schemes:

1. An **Absolute Address**, such as 04A26H, is a 20 bit value that directly references a specific location.
2. A **Segment Offset Address** combines the starting address of a segment with an offset value.

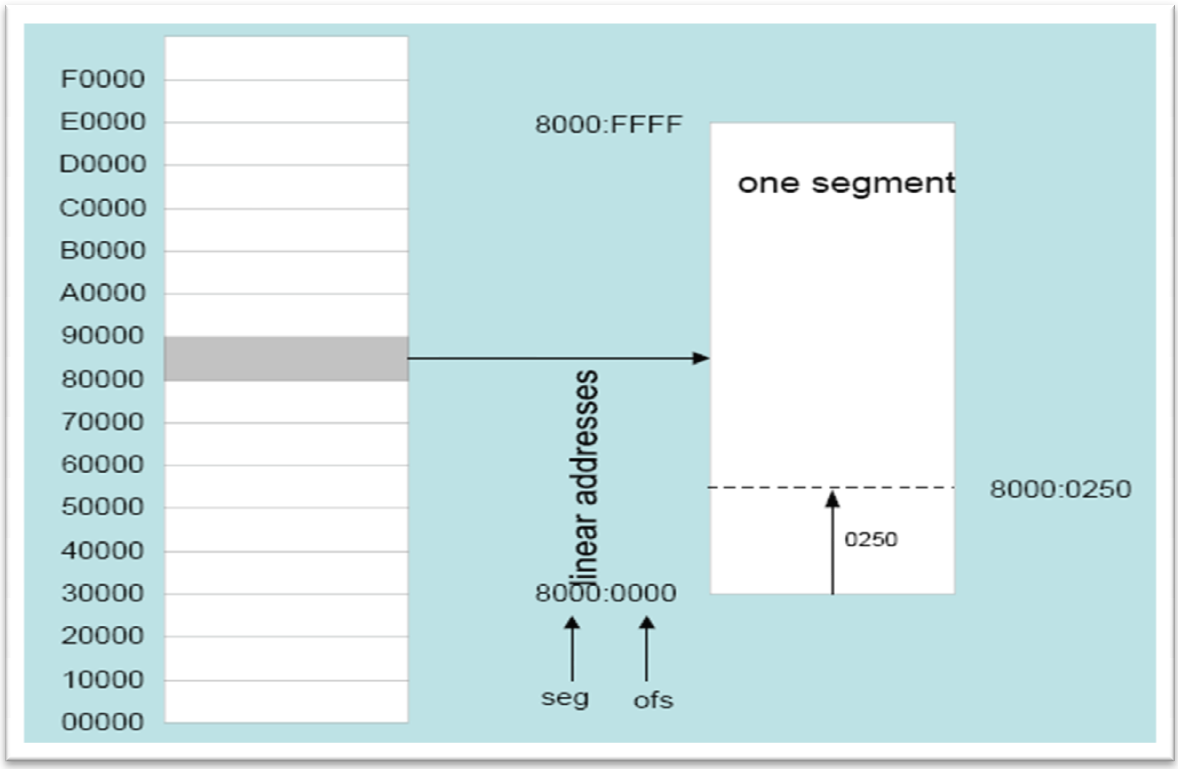
Segment and offset:

Segments are special area defined in a program for containing the code, the data, and the stack. Segment Offset within a program, all memory locations within a segment are relative to the segment starting address. The distance in bytes from the segment address to another location within the segment is expressed as an offset (or displacement).

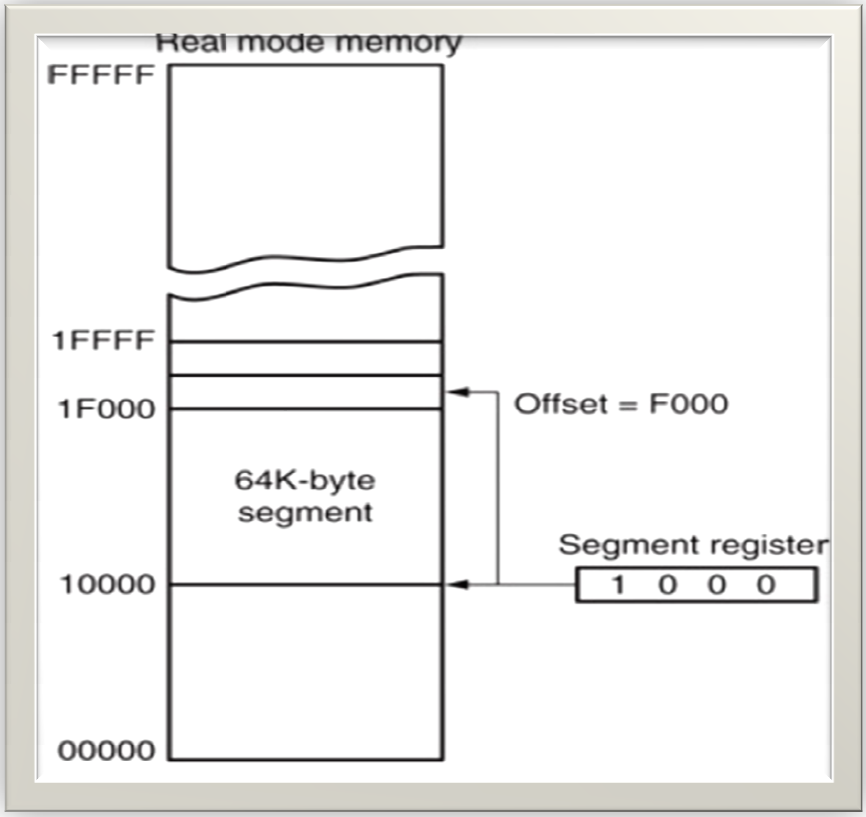
A segment is an area of memory that includes up to 64K bytes as shown in the following figures.

The offset address is always added to the segment starting address to locate the data.

All real mode memory addresses must consist of a segment address plus an offset address. –Segment address defines the beginning address of any 64K-byte memory segment offset address selects any location within the64K byte memory segment.



Linear and Segmented Memory



The real mode memory-addressing scheme, using a segment address plus an offset.

Assembly Language Program consists of three segments:

- ***Code segment:*** contains the program code (instructions)
- ***Data segment:*** used to store data (information) to be processed by the program
- ***Stack segment***: used to store information temporarily.

Specifying addresses

To reference any memory location in a segment, the processor combines the segment address in a segment register with the offset value of that location, that is, its distance in byte from the start of the segment.

To represent a segment address and its relative offset we use the notation:

Segment: offset

Thus 020A:1BCD denotes offset 1BCDH from segment 020AH.

The actual address it refers to is obtained in the following way:

1. Add zero to the right hand side of the segment address.
2. Add to this the offset.

Hence the actual address referred to by 020A:1BCD is 03C6D.

Address Bus in the 8086 is 20 bits wide (20 lines) i.e. the processor can access memory of size (1MB).

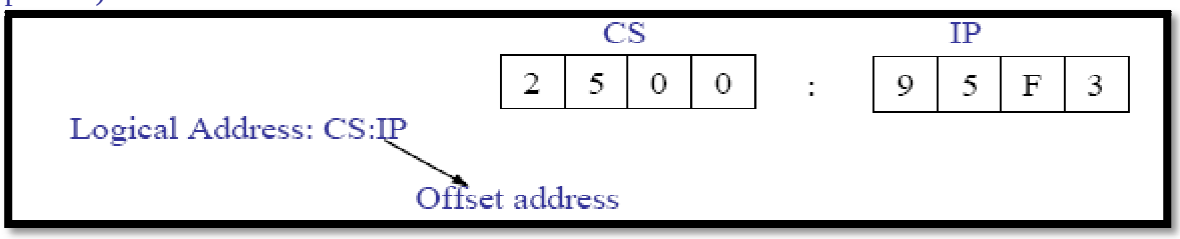
Instruction Pointer = 16 bit register which means the processor can only address (65535) bytes of memory. But we need to write instructions in any of the 1MB of memory. This can be solved by using **memory segmentation**, where each segment registers is 16-bit

Logical and Physical Address

- ***Physical Address*** is the 20-bit address that actually put on the address bus. (in 8086)• Has a range of 00000H – FFFFFH.
- ***Offset Address*** is a location within 64K byte segment range. Has a range of 0000H - FFFFH
- ***Logical Address*** consists of segment address and offset address.

Addressing in Code segment

The logical address of an instruction consists of **CS** (Code Segment) and **IP** (instruction pointer)



Example: CS:IP => 2500:95F3H

1. **Start with CS** 2500
2. **Shift left CS** 25000
3. **Add IP** 2E5F3 (25000+95F3)

Ex: If CS=24F6H and IP=634AH, determine:

a) The logical address
b) The offset address
c) The physical address
d) The lower range of the code segment
e) The upper range of the code segment

Ans:

a) The logical address is; 24F6:634A
b) The offset address is; 634A
c) The Physical address is; 24F60+634A= 2B2AA
d) The lower range of the code segment: 24F6:0000 => 24F60+0000 =24F60
e) The upper range of the code segment: 24F6:FFFF => 24F60+FFFF=34F5F

Addressing in Data segment

- The area of memory allocated strictly for data is called ***data segment.***
- The data segment uses DS and BX, SI and DI are used to hold the offset address.

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

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

Ans:

a) The Physical address is; 7FA20+438E= 83DAE
b) The lower range: 7FA20(7FA20+0000)
c) The upper range: 8FA1F(7FA20+FFFF)
d) The logical address is; 7FA2:438E

Addressing in Stack segment

Calculating the physical address for the stack, the same principle is applied as was used for the code and data segments. Physical address depends on the value of stack segment (SS) register and the stack pointer (SP).

Ex: If SS=3500H and SP:FFFEH

a) Calculate the physical address: 35000+FFFE = 44FFE
b) Calculate the lower range of the stack: 35000+0000 = 35000
c) Calculate the upper range of the stack segment: 35000+FFFF = 44FFF
d) Show the logical address of the stack: 3500:FFFE

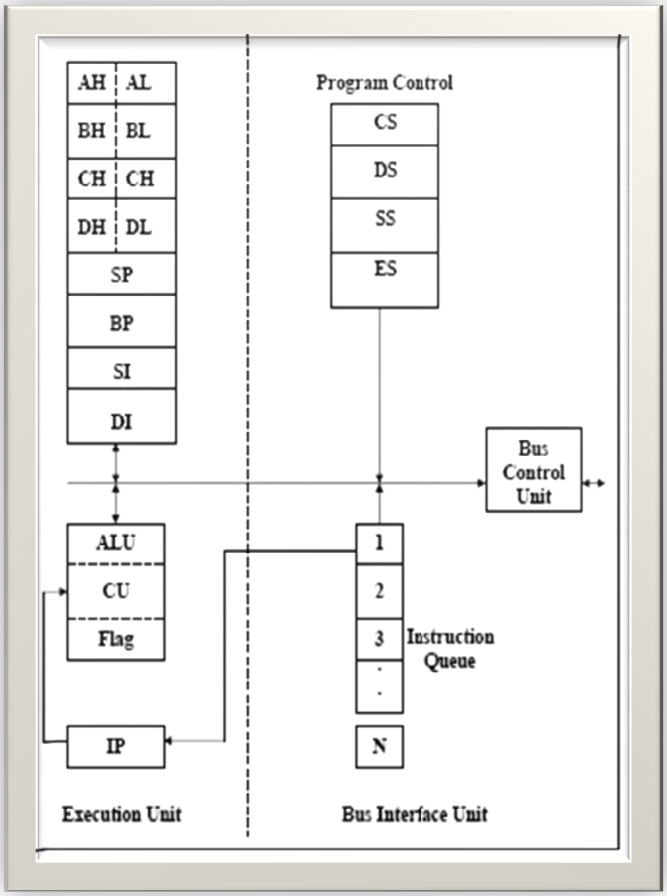
Execution Unit and Bus Interface Unit

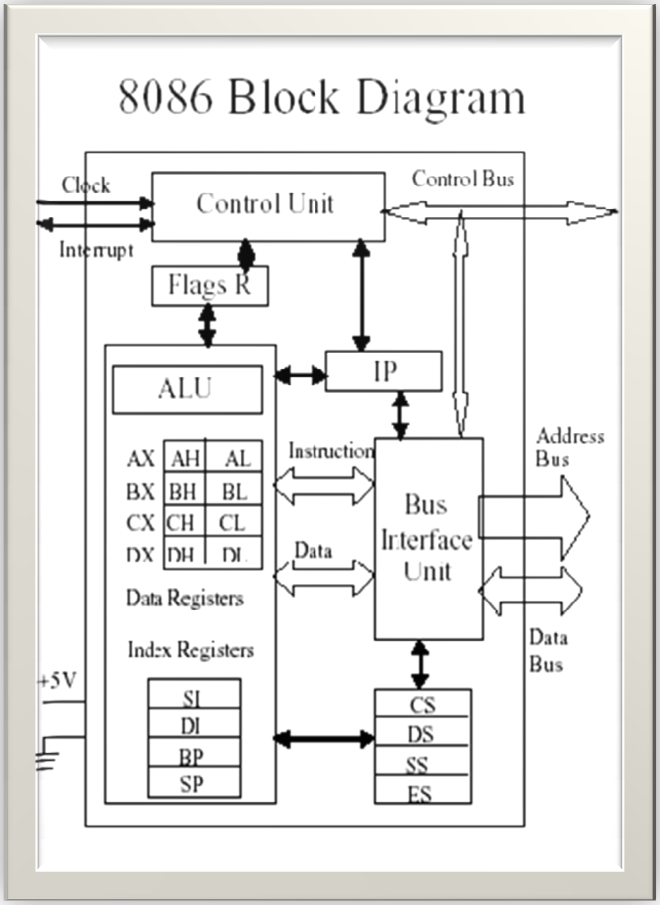
The processor is partitioned into two logical units as shown in figure:

1. Execution Unit (EU) to execute instruction and perform arithmetic and logical operations. The EU contains ALU, CU and number of registers.
2. Bus Interface Unit (BIU) to deliver the instruction and data to EU. The most important function of BIU is to manage the bus control unit, segment registers and instruction queue.

Another function of the BIU is to provide access to instructions, because the instructions for a program that is executing are kept in memory, the BIU must access instruction from memory and place them in an instruction queue, which varies in size depending on the processor. This feature enables the BIU to look ahead and prefetch instructions, so that there is always a queue of instructions ready to execute.

The EU and BIU work in parallel, The top instruction is the currently executable one, and while the EU is occupied executing an instruction, the BIU fetch another instruction from memory. This fetching overlaps with execution and speeds up processing.





Execution unit and Bus interface unit.

Registers of Microprocessor

In the CPU, registers are used to store information temporarily. Registers are 8, 16, or 32-bit high speed storage locations directly inside the CPU, designed to be accessed at much higher speed than conventional memory.

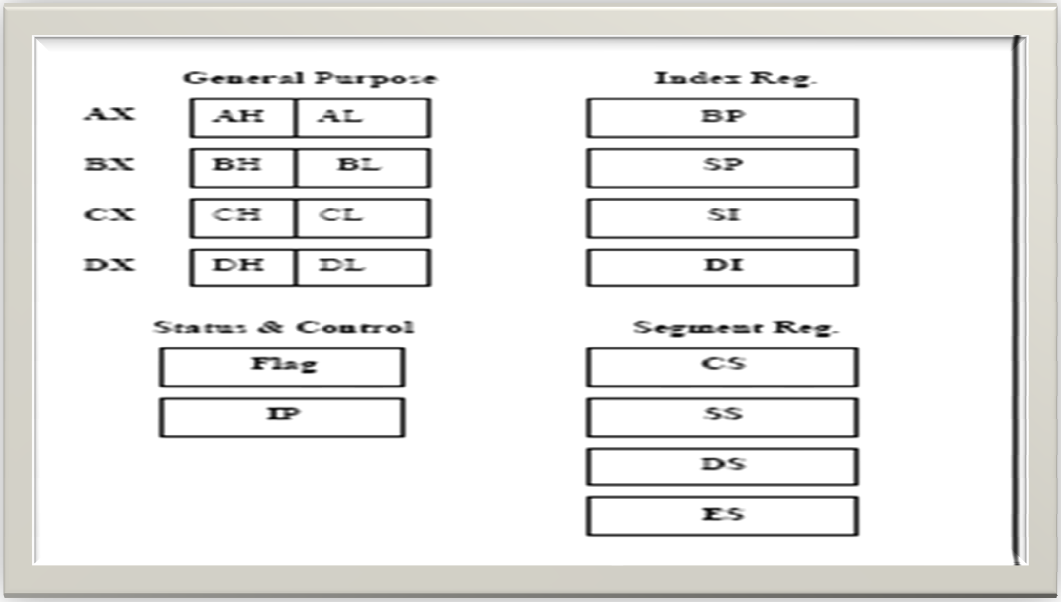
The bits of the registers are numbered in descending order:

For 8-bit register:

D7	D6	D5	D4	D3	D2	D1	D0

For 16-bit register:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0



Intel 16-bit registers

Types of registers (see the figure above)

1. **General purpose registers (Data Registers)**: are used for arithmetic and data movement. Each register can be addressed as either 16-bit or 8 bit value. Example, AX register is a 16-bit register, its upper 8-bit is called AH, and its lower 8-bit is called AL.

In 8088/8086 general-purpose registers can be accessed as either 16-bit or 8-bit registers. All other registers can be accessed as full 16-bit registers.

Different registers are used for different functions.

- **AX** is used for the ***accumulator*** because it is favored by the CPU for arithmetic operations.
- **BX** is used for ***base addressing register*** hold the address of a procedure or variable. Three other registers with this ability are SI, DI and BP. The BX register can also perform arithmetic and data movement.
- **CX** is used for ***counter loop operations.*** These instructions automatically repeat and decrement CX.
- **DX** is used to point out ***data in I/O operations*** DX register has a special role in multiply and divide operation.

2. **Segment Registers:** the CPU contain four segment registers, used as base location for program instruction, and for the stack.

- **CS** (Code Segment): The code segment register holds the base address of all executable instructions (code) in a program.

- **DS** (Data Segment): the data segment register is the default base location for variables.
- **SS** (Stack Segment): the stack segment register contain the base location of the stack.
- **ES** (Extra Segment): The extra segment register is an additional base location for memory variables.

3. **Index registers:** index registers contain the offset of data and instructions. The term offset refers to the distance of a variable, label, or instruction from its base segment. The index registers are:

- **BP** (Base Pointer): the BP register contain an assumed offset from the stack segment register, as does the stack pointer.
- **SP** (Stack Pointer): the stack pointer register contain the offset of the top of the stack. The stack pointer and the stack segment register combine to form the complete address of the top of the stack.
- **SI** (Source Index): This register takes its name from the string movement instruction
- **DI** (Destination Index): the DI register acts as the destination for string movement instruction.

4. **Status and Control register:**

- **IP** (Instruction Pointer): The instruction pointer register always contain the offset of the next instruction to be executed within the current code segment. The instruction pointer and the code segment register combine to form the complete address of the next instruction.
- **Flag** Registers and bit fields

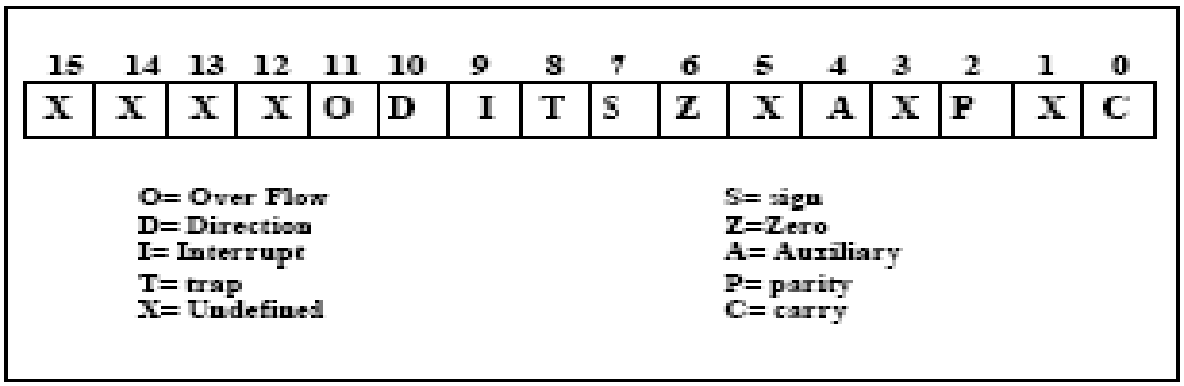
The flag register is a 16-bit register sometimes referred as the ***status*** register. Not all the bits are used.

The Flag Register: is a special register with individual bit positions assigned to show the status of the CPU or the result of arithmetic operations.

There two basic types of flags: (control flags and status flags)

Control flags: 6 of the flags are called the conditional flags, meaning that they indicate some condition that resulted after an instruction was executed. These 6 are: CF, PF, AF, ZF, SF, and OF.

Individual bits can be set in the flag register by the programmer to control the CPU operation, these are



The 16 bits of the flag registers:

- **CF, the Carry Flag:** This flag is set whenever there is a carry out, either from d7 after an 8-bit operation or from d15 after a 16-bit data operation. if the sum of 71 and 99 where stored in the 8-bit register AL, the result cause the carry flag to be 1. The flag values = 1 = carry, 0 = no carry.
- **PF, the Parity Flag:** After certain operations, the parity of the result's low-order byte is checked. If the byte has an even number of 1s, the parity flag is set to 1; otherwise, it is cleared. This flag is used by the OS to verify memory integrity and by communication software to verify the correct transmission of data.
- **AF, the Auxiliary Carry Flag:** If there is a carry from d3 to d4 (or borrow from bit 4 to bit 3) of an operation this bit is set to 1, otherwise cleared (set to 0).
- **ZF, the Zero Flag:** The ZF is set to 1 if the result of the arithmetic or logical operation is zero; otherwise, it is cleared (set to 0).
- **SF, the Sign Flag:** MSB is used as the sign bit of the binary representation of the signed numbers. After arithmetic or logical operations the MSB is copied into SF to indicate the sign of the result. The flag values 1=negative, 0 = positive
- **OF, the Overflow Flag:** This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. The flag values 1 = overflow, 0 = no overflow.

Status flags

- **TF, the Trap Flag:** When this flag is set it allows the program to single step, meaning to execute one instruction at a time. Used for debugging purposes. The flag values are 1 = on, 0 = off.
- **IF, Interrupt Enable Flag:** This bit is set or cleared to enable or disable only the external interrupt requests as keyboard, disk drive, and the system clock timer. The flag values are 1= enable, 0 = disable.
- **DF, the Direction Flag:** This bit is used to control the direction of the string operations and used for block data transfer instructions, such as MOVS, CMPS, SCAS, it selects increment or decrement mode for the DI and/or SI registers.

Registers of 8086/8088

Category	Bits	Register Names
General	16	AX, BX, CX, DX
	8	AH, AL, BH, BL, CH, CL, DH, DL
Pointer	16	SP (stack pointer), BP(base pointer)
Index	16	SI (source index), DI(destination index)
Segment	16	CS (code segment), DS(data segment) SS (stack segment), ES(extra segment)
Instruction	16	IP (instruction pointer)
Flag	16	FR (flag register)