- **DS (Data Segment):** the data segment register is the default base location for variables. The CPU calculates their location using the segment value in DS.
- **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.

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. The base pointer register is often used by a subroutine to locate variables that were passed on the stack by a calling program.
- ✓ **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, in which the source string is pointed to by the source index register.
- ✓ **DI (Destination Index):** the **DI** register acts as the destination for string movement instruction

Status and Control Register

1. **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.

2. **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. The Figure9 describe the **8086/8088** flags register:

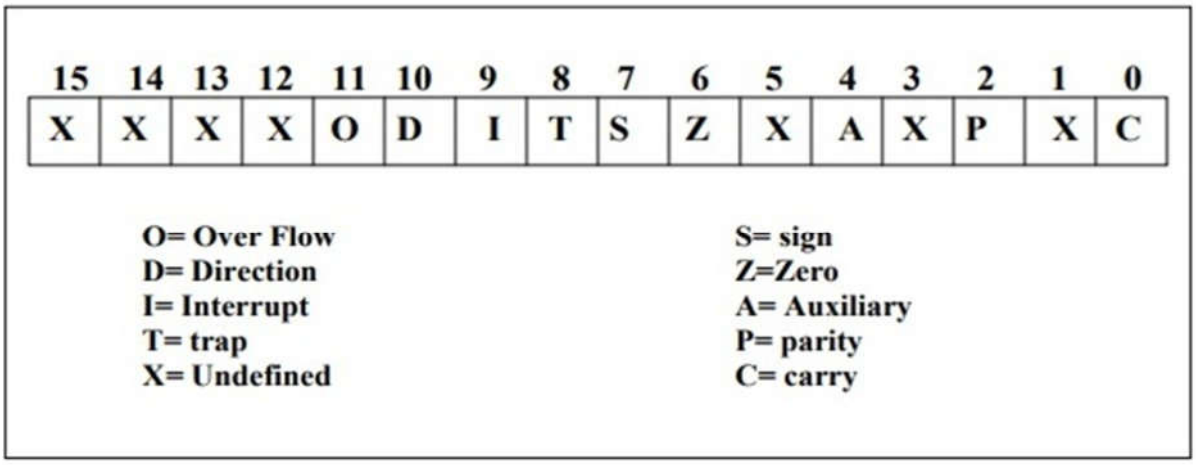


Figure 9: Flag Register

There Two Basic Types of Flags

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

1. **Control Flags:** individual bits can be set in the flag register by the programmer to control the CPU operation , these are

- **The Direction Flag (DF):** affects block data transfer instructions, such as **MOVS**, **CMPS**, **SCAS**. The flag values are **0 = up** and **1 = down**.

- **The Interrupt flag (IF):** dictates whether or not a system interrupt can occur. Such as keyboard, disk drive, and the system clock timer. A program will sometimes briefly disable the interrupt when performing a critical operation that cannot be interrupted. The flag values are **1 = enable**, **0 = disable**.

- **The Trap flag (TF):** Determine whether or not the CPU is halted after each instruction. When this is set, a debugging program can let a programmer to enter single stepping (trace) through a program one instruction at a time. The flag values are **1 = on**, **0 = off**. The flag can be set by **INT 3** instruction.

2. Status Flags: The status flags reflect the outcomes of arithmetic and logical operations performed by the CPU, these are:

- **The Carry Flag (CF):** is set when the result of an unsigned arithmetic operation is too large to fit into the destination for example, 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**.

- **The Overflow (OF):** is set when the result of a signed arithmetic operation is too wide (too many bits) to fit into destination. **1 = overflow**, **0 = no overflow.**

- **Sign Flag (SF):** is set when the result of an arithmetic of logical operation generates a negative result, **1= negative**, **0 = positive**.

- **Zero Flag (ZF):** is set when the result of an arithmetic of logical operation generates a result of zero, the flag is used primarily by jump or loop instructions to allow branching to a new location in a program based on the comparison of two values. The flag **value = 1 = zero**, **& 0 = not zero**.

- **Auxiliary Flag:** is set when an operation causes a carry from bit 3 to bit 4 (or borrow from bit 4 to bit 3) of an operand. The flag **value = 1 = carry**, **0 = no carry**.
- **Parity Flag:** reflect the number of 1 bits in the result of an operation. If there is an **even number** of bit, the parity is even. If there is an **odd number** of bits, parity is **odd**. This flag is used by the OS to verify memory integrity and by communication software to verify the correct transmission of data.

Instruction Execution and Addressing

An assembly language programmer writhe a program in **symbolic code** and uses the **assembler** to translate it into machine code as .EXE program. For program execution, the system looks only the machine code into memory.

Every instruction consists of at least one **operation**, such as MOV, ADD. Depending on the operation, an instruction may also have one or more **operands** that reference the data the operation is to process.

The Basic Steps the Processor

The basic steps the processor takes in executing on instruction are:

1. **Fetch the next instruction** to be executed from memory and place it in the instruction queue.
2. **Decode the instruction** calculates addressed that reference memory, deliver data to the Arithmetic Logic Unit, and increment the instruction pointer **(IP)** register.

3. **Execute the instruction**, performs the request operation, store the result in a register or memory, and set flags such as zero or carry where required.

For an .EXE program the **CS** register provide the address of the beginning of a program code segment, and **DS** provide the address of the beginning of the data segment.

The **CS** contains instructions that are to be executed, where as the **DS** contain data that the instruction reference. The **IP** register indicates the offset address of the current instruction in the **CS** that is to be executed. An instruction operand indicates on offset address in the **DS** to be referenced.

Consider and example in which the program loader has determined that it is to be load on .EXE program into memory beginning at location **05BE0H**. The loader accordingly initialize CS with segment address **05BE[0]H** and **IP** with zero.

CS: IP together determine the address of the first instruction to execute **05BE0H + 0000H = 05BE0H**. In this way the first instruction in CS being execution, if the first instruction is two byte long, the processor increment **IP** by **2**, so that , the next instruction to be executed is **05BE0H + 0002H = 05BE2H**.

Assume the program continues executing, and **IP** contain the offset **0023H. CS: IP** now determine the address of the next instruction to execute, as follows:

CS address: **05BE0H**

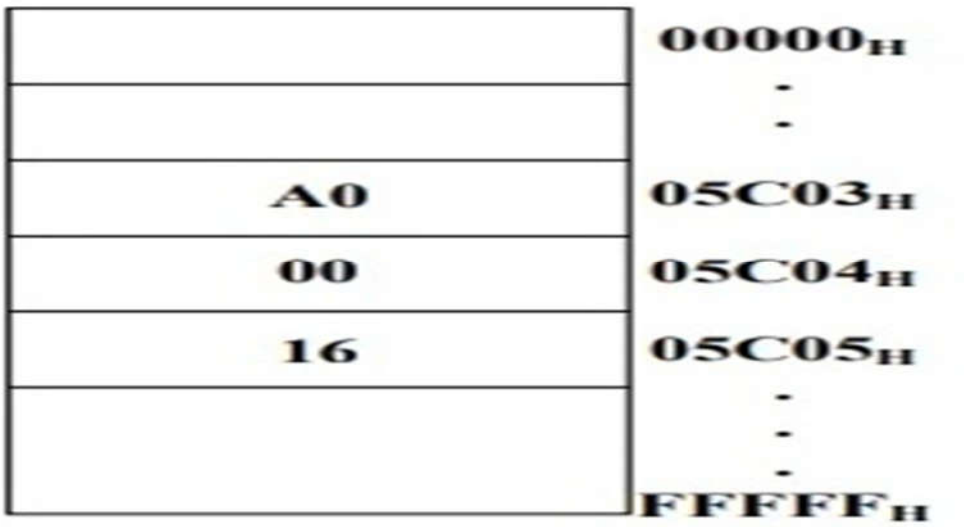
IP offset: **0023H** +

Instruction address: **05C03H**

EX: let's say that **MOV** instruction beginning at **0FC03H** copies the content of a byte in memory into the **AL** register. The byte is at offset **0016H** in the **DS**. Her are the machine code and the symbolic code for this operation.

Address	Symbolic Code	MIC code
0FC03	MOV AL, [0016]	A0 1600

Address **0FC03H** contain the first byte **(A0H)** of the MIC code instruction the processor is to access



The second and third byte contains the offset value in reversed byte sequence. In symbolic code, the operand **[0016]** in square brackets (an index operator) indicates an offset value to distinguish it from the actual storage address 16. Lest say that the program has initialized the **DS** register with **DS** address **05D1[0]H**. To access the data item, the processor determines its location from the segment address in **DS** + the offset **(0016H)** in the instruction. Operand become **DS** contain **0FD1[0]H**, the actual location of the reference data item is

DS:	**05D10H**	
Offset:	**0016H**	+
Address of data item:	**05D26H**	

Assume the address **05D26H** contain **4AH**, the processor now extract the **4AH** at address **05D26H** and copy it into **AL** register.

An instruction may also access more than one byte at a time

EX: Suppose an instruction is to store the content of the **AX** register **(0248H)** in two adjacent byte in the **DS** beginning at offset **0016H**.

The symbolic code **MOV [0016], AX**

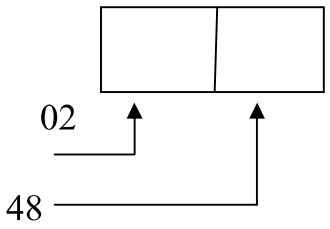
The processor stores the two byte in memory in revered byte sequence as

Content of AX: 02 48

Offset in DS: 0017 0016

Another instruction, **MOV AX, [0016]**, subsequently could retrieve these byte by copy them from memory back into **AX**.

The operation reverses (and corrects) the byte in **AX** as:



Number of Operands

Operands specify the value an instruction is to operate on, and where the result is to be stored. Instruction sets are classified by the number of operands used. An instruction may have no, one, two, or three operands.