Registers of 8086: Status & Control Flags

Overview

Intel 8086 Flag Register Overview

The Intel 8086 microprocessor contains a dedicated **16-bit Flag Register**, also called the **Program Status Word (PSW)**. This register is crucial for monitoring operation outcomes and controlling CPU behavior.

- **Flag Usage:** Out of 16 bits, **9 are active flags**, while **7 are reserved/unused**.
- **Purpose:** Each flag is a single-bit indicator representing the **status of an operation** or providing **control instructions** to the CPU.
- **Categories:** Based on functionality, flags are divided into:

  Condition Flags

  Reflect results of arithmetic or logical operations (e.g., Zero, Sign, Carry, Overflow). Help in decision making and conditional branching.

  Control Flags

  Control the way the CPU executes instructions (e.g., enabling/disabling interrupts, string direction). Act like switches that modify execution flow.

- **Functions of Flags:**
  - Detect errors or arithmetic overflows.
  - Enable conditional jumps and loops.
  - Control low-level processor operations such as debugging, interrupts, and data movement.

Condition Flags (Status Indicators)

Condition Flags

The **Condition Flags** in the 8086 processor reflect the outcome of arithmetic or logical operations. They allow the CPU (and programmer) to make decisions based on results.

Carry Flag (CF)

- **Definition:** Indicates *unsigned overflow* in arithmetic operations.
- **Set (CF=1):** When there is a carry out of the most significant bit (MSB) in addition, or a borrow is needed in subtraction.
- **Usage:** Essential for `JC/JNC` (Jump on Carry/No Carry).
- **Example:** `FFh + 01h` → CF=1.

Parity Flag (PF)

- **Definition:** Indicates whether the lower byte of the result contains an *even number of 1s.*
- **Set (PF=1):** When the count of 1s in the least significant byte is even.
- **Usage:** Often used in error checking during communication.
- **Example:** Result = `00110110b` (4 ones) → PF=1.

Auxiliary Carry Flag (AF)

- **Definition:** Indicates a carry/borrow between bit 3 and bit 4 (nibble boundary).
- **Set (AF=1):** If an operation produces a carry out of bit 3 or requires borrow into bit 4.
- **Usage:** Important for *BCD (Binary-Coded Decimal)* arithmetic.

Zero Flag (ZF)

- **Definition:** Shows whether the operation result is zero.
- **Set (ZF=1):** If the result of an operation is 0.
- **Usage:** Common in branching instructions like `JZ/JNZ`.
- **Example:** `SUB AX,AX` → ZF=1.

Sign Flag (SF)

- **Definition:** Copies the most significant bit of the result (sign bit).
- **Set (SF=1):** If result is negative (MSB = 1 in signed numbers).
- **Usage:** Affects signed conditional jumps (`JS/JNS`).
- **Example:** Result = `10000001b` (–127 signed) → SF=1.

Overflow Flag (OF)

- **Definition:** Detects overflow in *signed* arithmetic.
- **Set (OF=1):** When the result is too large or small to fit in signed representation.
- **Usage:** Important in signed comparisons (`JO/JNO`).
- **Example:** `7Fh + 01h = 80h` → OF=1.

Control Flags (Execution Control)

The **Control Flags** are special bits in the 8086 Flag Register that act as switches to control the behavior of the CPU. Unlike condition flags, they do not reflect operation results but rather **influence how instructions are executed**.

Control Flags

- **Trap Flag (TF)**
  - **Purpose:** Enables **single-step execution mode**.
  - **Effect:** After every instruction, the CPU generates a **trap interrupt (type 1)**, allowing step-by-step program tracing.
  - **How to Set/Reset:** Push the flag register onto the stack, modify TF, then pop it back.
  - **Practical Use:** Debugging and error detection in programs (works like a breakpoint mechanism).

- **Interrupt Flag (IF)**
  - **Purpose:** Controls whether the CPU accepts **maskable hardware interrupts (INTR)**.
  - **Effect:**
    - ∗ IF = 1 ⇒ Interrupts are enabled.
    - ∗ IF = 0 ⇒ Interrupts are disabled (ignored).
  - **Instructions:** `STI` (Set IF → enable interrupts), `CLI` (Clear IF → disable interrupts).
  - **Default State:** After system reset, IF = 0 (disabled).
  - **Practical Use:** Protects critical sections of code from being interrupted.

- **Direction Flag (DF)**
  - **Purpose:** Controls the direction of string operations involving `SI` (Source Index) and `DI` (Destination Index).
  - **Effect:**
    - ∗ DF = 0 ⇒ Increment mode (strings processed from lower to higher memory).
    - ∗ DF = 1 ⇒ Decrement mode (strings processed from higher to lower memory).
  - **Instructions:** `CLD` (Clear DF → Increment mode), `STD` (Set DF → Decrement mode).
  - **Practical Use:** Efficient manipulation of arrays and strings in forward/backward directions.

Flag Register Structure

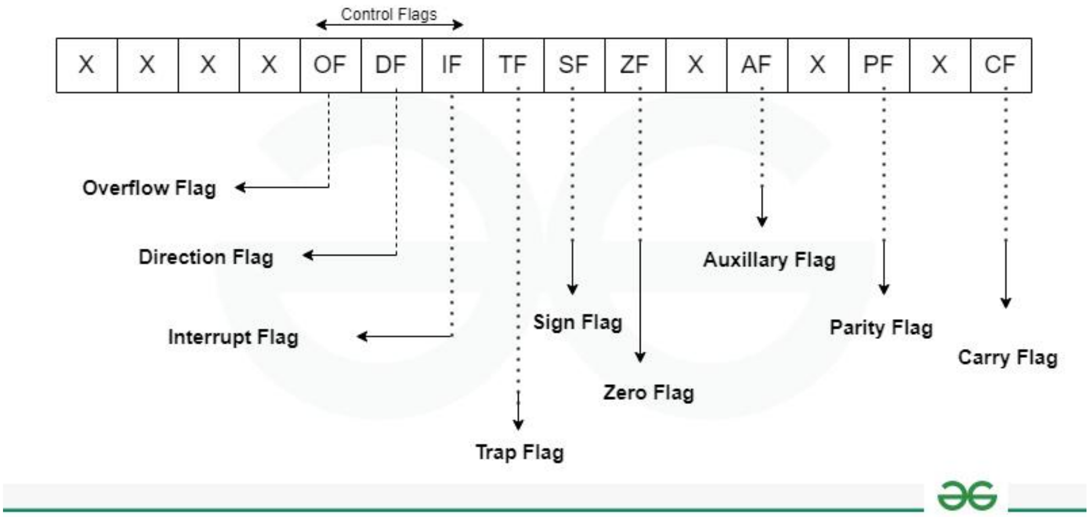

Figure: 16-bit Flag Register of Intel 8086

Summary Table

Flag	Name	Meaning when Set (1)
CF	Carry	Unsigned overflow: Carry/borrow from MSB.
PF	Parity	Even parity in lower 8 bits.
AF	Aux Carry	Carry/borrow between bit 3 and 4 (BCD arithmetic).
ZF	Zero	Result = 0. Used in conditional jumps.
SF	Sign	MSB = 1 ⇒ Negative result (signed).
OF	Overflow	Signed overflow occurred.
TF	Trap	Enables single-step execution mode.
IF	Interrupt	Enables/disables maskable interrupts.
DF	Direction	String operations decrement (1) or increment (0).

Practical Examples

Example 1: Using Zero Flag (ZF) for Conditional Jump

```
MOV AX, 5
SUB AX, 5       ; AX = 0, ZF = 1
JZ ZERO_LABEL   ; Jump if Zero Flag = 1
ZERO_LABEL:
; Executes if result was zero
```

Example 2: Overflow Flag in Signed Arithmetic

```
MOV AL, 7Fh    ; 127 in decimal
ADD AL, 1      ; AL = 80h (-128 signed)
; OF = 1 because result overflowed signed 8-bit range
```

Example 3: Direction Flag in String Operation

```
CLD            ; Clear DF -> SI, DI increment
REP MOVSB      ; Copy string forward
STD            ; Set DF -> SI, DI decrement
REP MOVSB      ; Copy string backward
```

Example 4: Interrupt Flag Control

```
CLI            ; Disable interrupts (IF=0)
; critical code section
STI            ; Enable interrupts (IF=1)
```

Applications of Flags

- **Program Flow Control:** Jumps, loops, and conditional branching depend on ZF, SF, CF, OF.
- **Debugging:** TF enables single-step debugging for tracing execution.
- **Interrupt Handling:** IF allows or blocks hardware interrupts.
- **String Processing:** DF controls direction in repeated string operations.
- **Error Checking:** PF is used in parity-based error detection schemes.