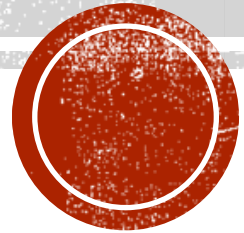
# FLAG REGISTERS



Course Title: Microprocessors

Course No:  CSE 3107

***Instructor:***

Mr . Sujan Sarker

***Presented By:***

Sabbir Hassan Abir (14.02.04.058)

Sarah Nuzhat Khan (14.02.04.064)

Nowshin Nawar Arony (14.02.04.067)

Sanjana Farial (14.02.04.100)

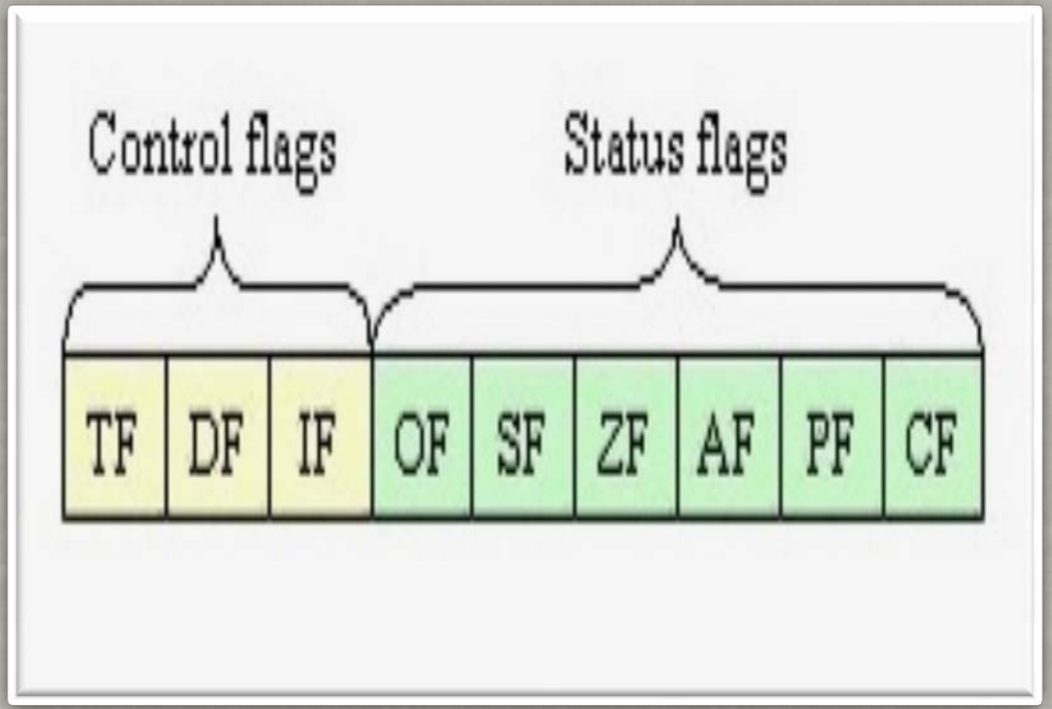
Mehjabin Nowshin (14.02.04.101)

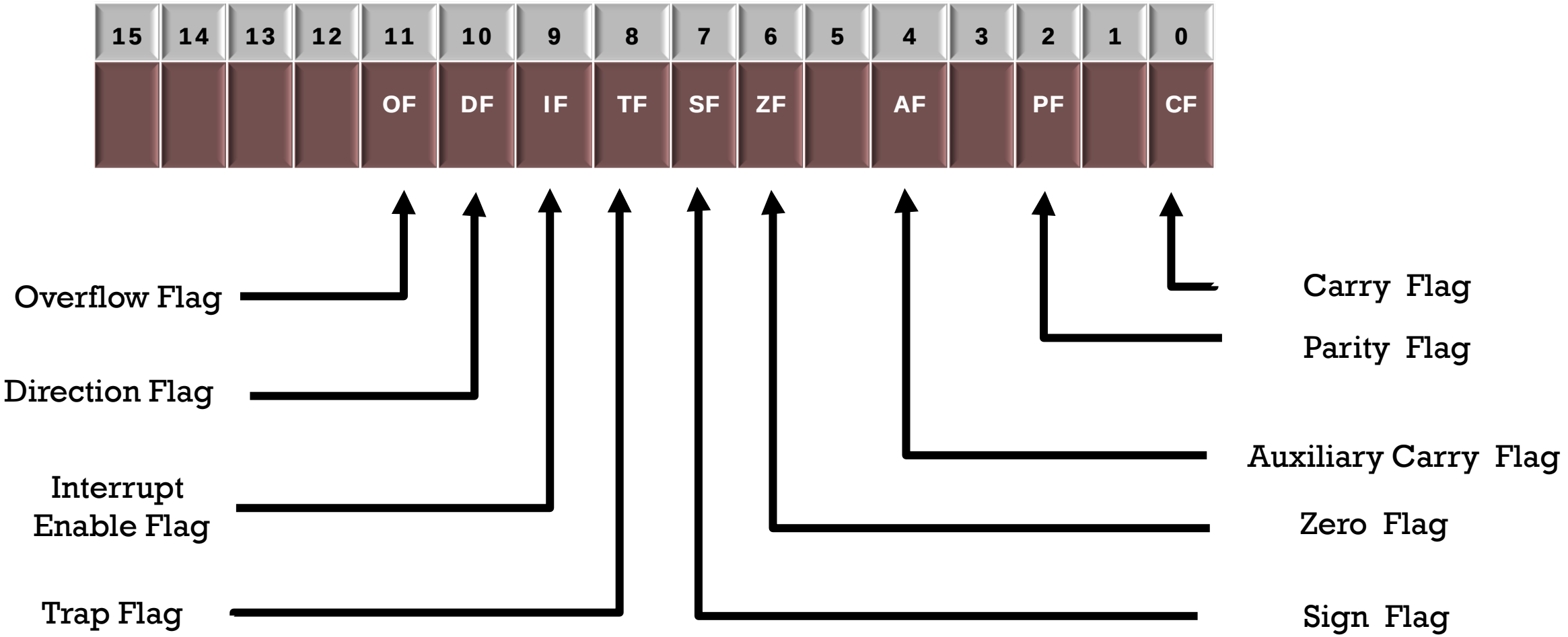
Md. Siam Ansary (14.02.04.104)

Hello ! I am a 8086 Microprocessor . I am the first 16 bit microprocessor introduced in 1978 by Intel .
I have 20 bit address bus.
However , now we will talk about my flag registers…

# INTRO TO FLAG REGISTERS

- One of the important features of a computer is the ability to make decisions.
- In 8086 processor, these decisions are made by flags.
- These flags are placed in Flag Registers and are classified as either status flags or control flags.




15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
OF DF IF TF SF ZF AF PF CF
Overflow Flag
Direction Flag
Interrupt Enable Flag
Trap Flag
Carry Flag
Parity Flag
Auxiliary Carry Flag
Zero Flag
Sign Flag

# ZERO FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 6 | ZF | The zero flag sets if the result of operation in ALU is zero and flag resets if the result is non-zero. The zero flag also sets if certain register content becomes zero following an increment or decrement operation of that register. |

# EXAMPLE OF ZERO FLAG

- ZF=1 for a zero result and ZF=0 or a non zero result.

```
  1111 1111 1111 1111
+ 0000 0000 0000 0001
-------------------
1 0000 0000 0000 0000
```

Result is zero

| ZF |
| --- |
| 1 |

# PARITY FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 2 | PF | It is set to 1 if the result of byte operation or lower byte of the word operation contains even number of ones, otherwise it is set to 0. |

# EXAMPLE OF PARITY FLAG

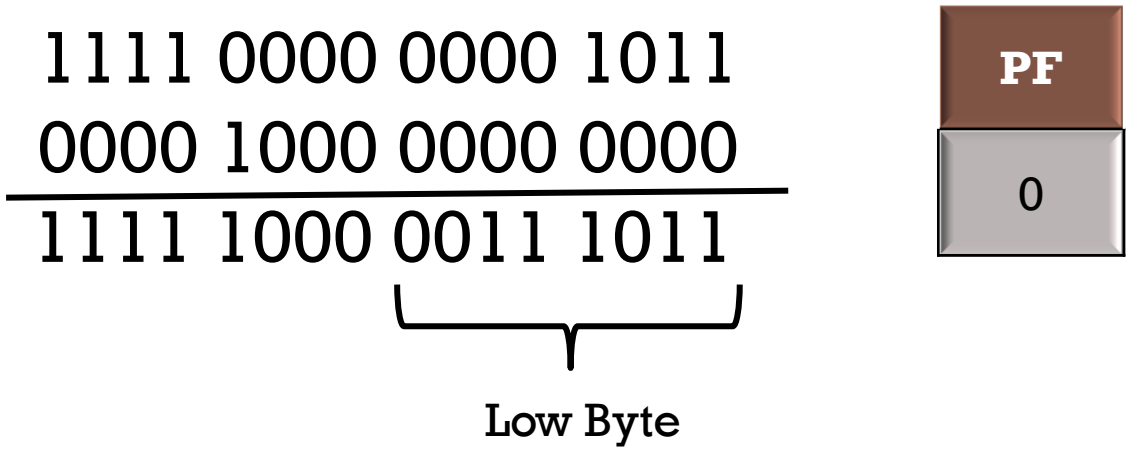
- If the low byte of a result has an even number of one bits (even parity) then PF=1.

```
  0001 0000 0011 1001
+ 0000 1000 0000 0000
-------------------
  0001 1000 0011 1001
```

Low Byte

# EXAMPLE OF PARITY FLAG

- If the low byte of a result has an odd number of one bits (odd parity) then PF=0.

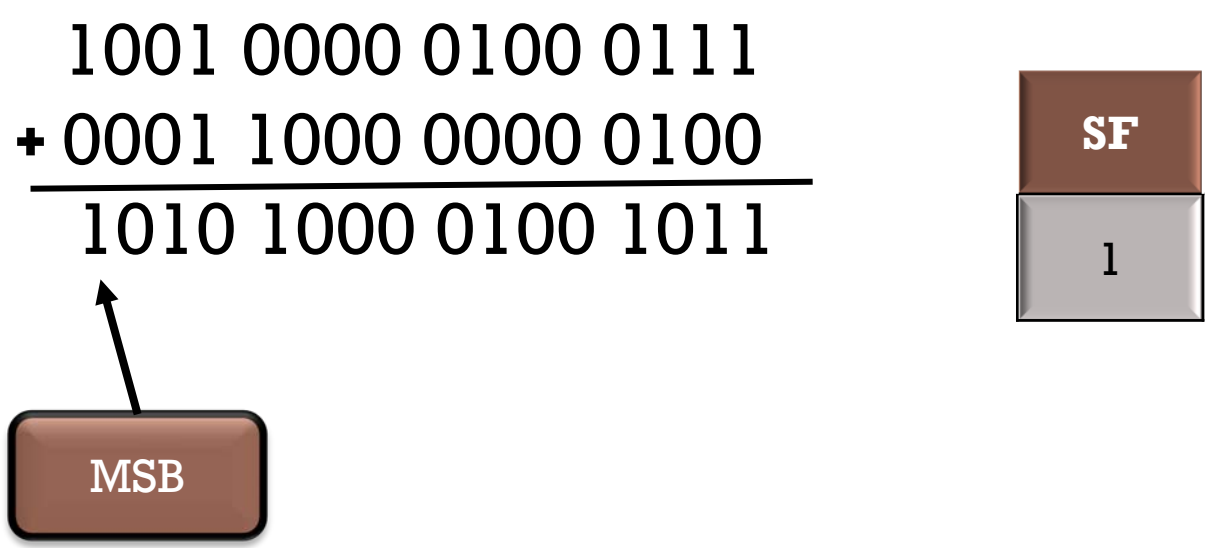


# SIGN FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 7 | SF | After the execution of arithmetic or logical operations , if the MSB of the result is 1 , the sign bit is set . Sign bit indicates the result is negative , otherwise it is positive. |

# EXAMPLE OF SIGN FLAG

- SF=1 , if the MSB of a result is 1, it means if the result is negative in case of signed interpretation. And SF=0 if MSB is 0.



# CARRY FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 0 | CF | In case of addition , this flag is set if there is a carry out of the MSB. The carry flag also serves as a borrow flag for subtraction. In case of subtraction, it is set when borrow is needed. It is also effected by shift and rotate operations. |

# EXAMPLE OF CARRY FLAG

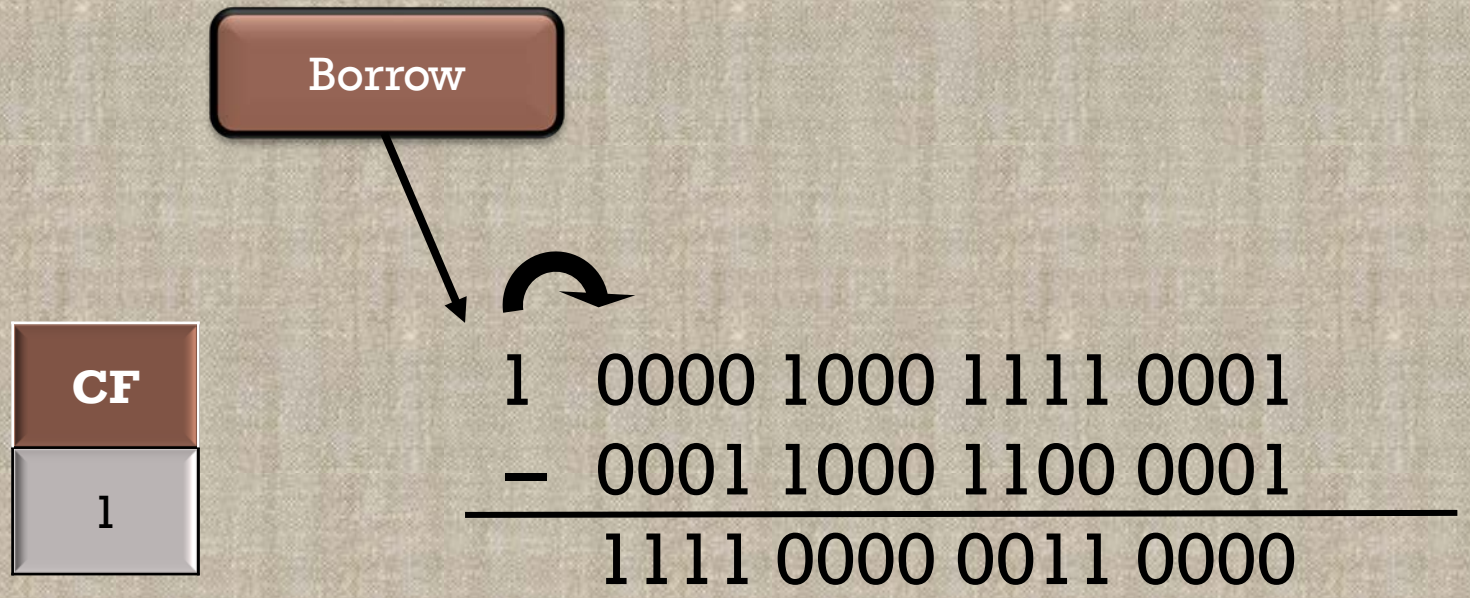
- The carry flag is set if the addition of two numbers causes a carry out of the most significant (leftmost) bits added.

```
   1111 0000 0011 0001
+  0111 1000 0000 0000
-------------------------
1 0000  1000 0011 0001
```

Carry Out

| CF |
| --- |
| 1 |

# EXAMPLE OF CARRY FLAG



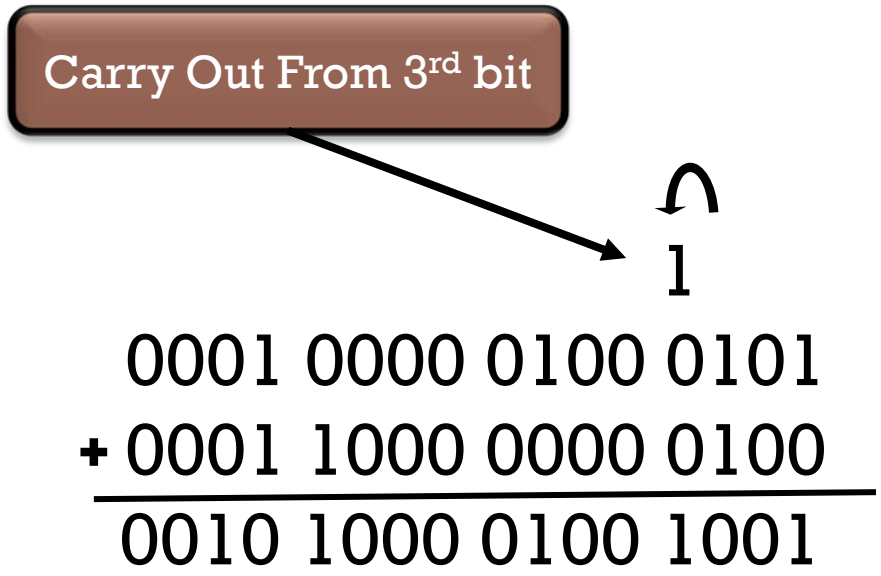

- The carry (borrow) flag is also set if the subtraction of two numbers requires a borrow into the most significant (leftmost) bits subtracted.

# AUXILIARY CARRY FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 4 | AF | This flag is set if there is an overflow out of bit. Carry from lower nibble to higher nibble(D3 bit to D4 bit). This flag is used for BCD operation that is not available to programmers. |

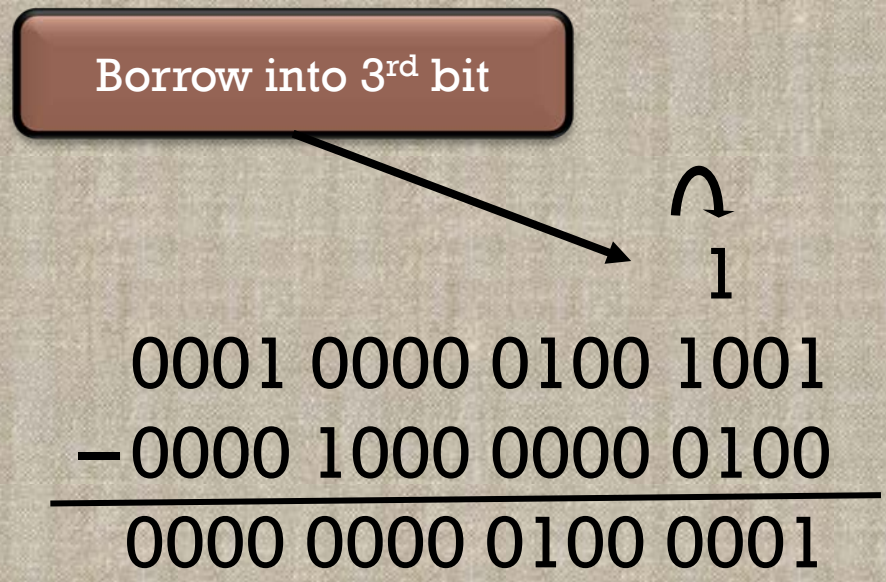
# EXAMPLE OF AUXILIARY CARRY FLAG

- If there is carry out from bit 3 on addition, or borrow into bit 3 on subtraction then AF=1.




# EXAMPLE OF AUXILIARY CARRY FLAG

- Similarly if there is borrow into bit 3 on subtraction then AF=1.




| AF |
| --- |
| 1 |

# DIFFICULT ONE...

- OVERFLOW FLAG IS THE MOST DIFFICULT FLAG AMONG ALL SATUS FLAGS.
- OVERFLOW DENOTES THAT THE RESULT WHICH WE GET, IS OUT OF RANG MAY BE FOR UNSIGN, SIGN OR BOTH REPRESENTATION. (IT WILL DISCUSS IN DETAIL LATER)
- **ALWAYS KEEP IN MIND THAT, OF=1 IF SIGN OVERFLOW OCCURRED OTHERWISE IT IS 0.**

# OVERFLOW FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 11 | OF | The flag is set if the result is out of range. For addition, the flag is set when there is a carry into the MSB and no carry out of the MSB or vice-versa. For subtraction, it is set when MSB needs a borrow and there is no borrow from the MSB , or vice-versa. |

# OVERFLOW

- For signed numbers a 16 bit word is represented in the range -32768 to 32767.
- For unsigned numbers, the range is 0 to 65535.
- If the result of an operation falls outside these ranges, overflow occurs and the result which is obtained will be incorrect.
- The processor sets OF=1 for signed overflow and CF=1 for unsigned overflow.

# TYPES OF OVERFLOW

- THEY ARE FOUR IN NUMBER
  - NO OVERFLOW
  - SIGNED OVERFLOW BUT NOT UNSIGNED
  - UNSIGNED OVERFLOW BUT NOT SIGNED OVERFLOW
  - BOTH SIGN AND UNSIGN OVERFLOW

# NO OVERFLOW

0101 0101

+0001 0101

0110 1010

HERE, CF AND OF BOTH ARE ZERO BECAUSE NO OVERFLOW OCCURS

# UNSIGN BUT NOT SIGN OVERFLOW

| | | | | |
|---|---|---|---|---|
| | 1111 | 1111 | 1111 | 1111 |
| + | 0000 | 0000 | 0000 | 0001 |
| 1 | 0000 | 0000 | 0000 | 0000 |

**If we are giving an unsigned interpretation, the correct answer Is lOOOOh =65536, but this is out of range for a word operation. A I is carried out of the msb and the answer stored in AX, OOOOh, is wrong, so usigned overflow occurred. But the stored anwer is correct as a signed number, for FFFFh = -1. OOOlh = 1, and FFFFh + OOOlh = -1 + 1 = 0, so signed overflow did not occur.**

# SIGNED BUT NOT UNSIGNED OVERFLOW

```
  0111  1111   1111   1111
+ 0111  1111   1111   1111
  1000  0000   0000   0000 = FFFEH
```

**The signed and unsigned decimal interpretation of 7FFFh IS 32767. Thus for both signed and unsigned addition, 7FFFh + 7FFFh = 32767 + 32767 = 65534. This is out of range for signed numbers; the signed interpretation of the stored answer FFFEh is -2. so signed overflow occurred. However, the unsigned interpretation of FFFEh is 65534, which is the right answer, so there is no unsigned overflow.**

# SIGNED AND UNSIGNED OVERFLOW

| | | | |
|---|---|---|---|
| **1000** | **0000** | **1111** | **0000 -> A** |
| **+ 1000** | **1111** | **0000** | **0000 -> B** |
| **1 0000** | **1111** | **1111** | **0000 ->C** |

AT THE TIME OF SIGNED REPRESENTATION:

A=-32,528(**1000 0000 1111 0000**) AND B=-28928(**10001111 0000 0000** ) AND
THE RESULT HAVE TO BE, C = -61456
BUT WE GET C=32528 (0000 1111 1111 0000) (AS WE CAN TAKE ONLY 16 BIT)

WE ALSO KNOW 16 BITS SIGNED REPRESENTATION RANG IS (-32768 TO 32767)
THE RESULT IS OUTSIDE THIS RANGE SO A OVERFLOW OCCOUR.

AT THE TIME OF UNSIGNED REPRESENTATION:

A= 33008 (**1000 0000 1111 0000**) AND B= 36608 (**10001111 0000 0000** )
AND THE RESULT HAVE TO BE, C=69616 (1 0000 1111 1111 0000)
BUT WE GET C= 4080 (0000 1111 1111 0000)
BECAUSE WE CAN TAKE ONLY 16 BITS THAT'S WHY A UNSIGNED OVER FLOW OCCOURS.

# DETERMINING THE VALUES OF CF AND OF

- To set(1) or reset(0) CF flag:
  - If there is any carry out from MSB or BORROW in to MSB then carry flag CF=1 other wise CF=0.
- To set(1) or reset(0) OF flag:
  - Check the SIGNS of MSB of two operands for any kind of operation.

| MSB of operand ONE | MSB of operand TWO | IS SIGN OVERFLOW POSSIBLE? |
|---|---|---|
| 0 | 0 | YES, POSSIBLE IF THE MSB OF RESULT IS 1 |
| 0 | 1 | NEVER POSSIBLE |
| 1 | 0 | NEVER POSSIBLE |
| 1 | 1 | YES, POSSIBLE IF THE MSB OF RESULT IS 0 |

# CONCLUSION . . .

- WE USE OF FOR SIGNED OVERFLOW
- WE USE CF FOR UNSIGNED OVERFLOW

| OF | CF | OVERFLOW |
|---|---|---|
| 0 | 0 | NO OVERFLOW |
| 0 | 1 | UNSIGNED OVERFLOW |
| 1 | 0 | SIGNED OVERFLOW |
| 1 | 1 | SIGNED + UNSIGNED OVERFLOW |

# TRAP FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 8 | TF | Used for on-chip debugging.<br>If this flag is set, the processor enters the single step execution mode.<br>In single step mode processor executes one instruction at a time. |

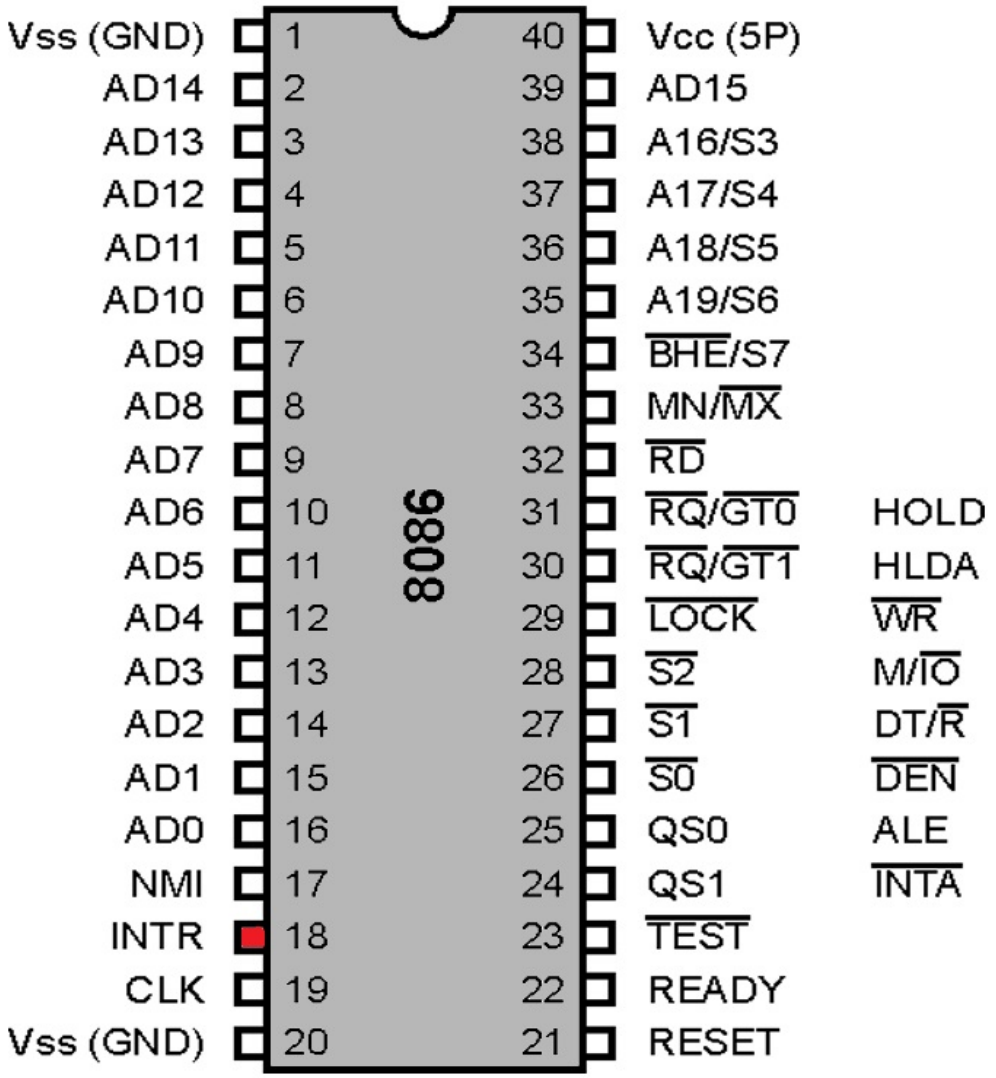
# TRAP FLAG

The 8086 has no instruction to directly set or reset the trap flag. These operations are done by pushing the flag register on the stack, changing the trap flag bit to what the programmer wants it to be, and then popping the flag register back off the stack. The instructions to set the trap flag are:

```
PUSHF                    ; Push flags on stack
MOV BP,SP                ; Copy SP to BP for use as index
OR WORD PTR[BP+0],0100H  ; Set TF flag
POPF                     ; Restore flag Register
```

# INTERRUPT FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 9 | IF | When the flag is set to 1 ,<br>CPU reacts to interrupts from external devices. |




# INTERRUPT

- This flag (IF) is set when the mas able inter or INTR is received by the processor
- IF=1 if the INTR pin (18) is enabled

# DIRECTION FLAG

| Bit No | Symbol | Description |
|---|---|---|
| 10 | DF | This flag is used for string manipulation instructions, i.e. the direction flag selects the increment or decrement mode for DI or SI registers in string instructions. |

# DIRECTION FLAG

- Using the clear-direction-flag instruction *CLD* the direction flag is set to zero.
- *Auto-incrementing* mode
- String is processed beginning from lowest to highest address.


- Using the set-direction-flag instruction *STD* the direction flag is set to one.
- *Auto-decrementing* mode
- String is processed from highest to lowest address.

# EFFECT ON ALL FLAGS

- Example: SUB AX,BX where AX= 8000h and BX = 0001h

- In signed sense, we are subtracting a positive number from a negative one, which is like adding two negatives. Because the result is positive (wrong sign) so overflow occurs.

```
  1000 0000 0000 0000
– 0000 0000 0000 0001
-------------------
  0111 1111 1111 1111
```

| SF | PF | ZF | CF | OF |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 |

"You don't have conversations with microprocessors. You tell them what to do, then helplessly watch the disaster when they take you literally!"
-David Brin

# Any Questions?