Assembly Flags

Flag register

This is a 16-bit register of which 9 bits are used by 8086 to indicate current state of the processor. The nine flags are categorized into two groups.

Flag register:1st group

Status flags: Six status flags indicate the status of currently executing instruction.

- Carry flag (CF)
- Parity flag (PF)
- Auxiliary flag (AF)
- Zero flag (ZF)
- Sign flag (SF)
- Overflow flag (OF)

Status Flags

Carry Flag (CF)
Purpose: Indicates if there was a carry out or borrow in an arithmetic operation.
Set when: In an addition, when the result exceeds the capacity of the register (overflow); in a subtraction, when a borrow occurs.
Used for: Detecting unsigned arithmetic overflow.

Status Flags

Parity Flag (PF)
Purpose: Indicates if the number of set bits (1s) in the result of the last operation is even or odd.
Set when: The result has an even number of 1-bits (parity is even).
Used for: Checking parity errors in data transmission or operations.

Status Flags

Auxiliary Flag (AF)
Purpose: Used in BCD (Binary Coded Decimal) arithmetic to indicate a carry or borrow between the lower and upper nibbles of a byte.
Set when: There is a carry from the lower nibble (4 bits) to the upper nibble or vice versa during addition or subtraction.
Used for: Decimal operations and adjustments.

Status Flags

Zero Flag (ZF)
Purpose: Indicates if the result of an operation is zero.
Set when: The result of the operation is 0.
Used for: Condition checks in loops or comparisons to determine equality.

Status Flags

Sign Flag (SF)
Purpose: Indicates the sign of the result of an operation (i.e., whether the result is positive or negative).
Set when: The result has a negative value (in two's complement, the most significant bit is 1).
Used for: Indicating the sign of the result.

Status Flags

Overflow Flag (OF)
Purpose: Indicates if an arithmetic operation has resulted in an overflow (the result is too large or too small to be represented in the register).
Set when: In signed arithmetic, when the result is out of the range that can be represented with the available number of bits (overflow or underflow).
Used for: Detecting overflow in signed arithmetic.

Flag register: 2nd Group

Control flags: There are three control flags that controls certain operations of the processor.

- Interrupt flag (IF)
- Direction flag (DF)
- Trap flag (TF)

Control Flags

Interrupt Flag (IF)

Purpose: Controls the enabling and disabling of interrupts. If this flag is set, interrupts are enabled; if it is cleared, interrupts are disabled.

Set when: This flag is set when interrupts are allowed, meaning the processor can respond to interrupt requests (IRQs) from external devices or other sources.

Used for: Enabling or disabling interrupt processing. The processor will only respond to interrupts when this flag is set.

Common Use: Often managed in operating systems or interrupt-driven programs to control the interrupt behavior.

Control Flags

Direction Flag (DF)

Purpose: Controls the direction of string operations, specifically the REP (repeat) prefix used in string manipulation instructions like MOVS, LODS, STOS, etc.

Set when: If the Direction Flag is set, string operations will move from higher memory to lower memory (decrementing the address registers like SI and DI). If the flag is cleared, string operations move from lower memory to higher memory (incrementing the address registers).

Used for: To define whether string operations should process data in forward (incrementing) or reverse (decrementing) order.

Control Flags

Direction Flag (DF)

Common Use: It is used in optimized code that deals with blocks of memory or arrays, and the flag is often adjusted before performing operations like copying or comparing strings.

Control Flags

Trap Flag (TF)

Purpose: Enables single-step debugging. When set, the processor generates a debug exception after each instruction is executed, which allows a debugger to take control between each instruction.

Set when: This flag is set when a debugger or the program itself wants the processor to stop after each instruction to inspect the execution step-by-step.

Used for: For debugging, as it forces the processor to generate an interrupt (commonly a debug interrupt) after every instruction, giving the debugger a chance to take action.

Control Flags

Trap Flag (TF)

Common Use: Single-step debugging in low-level software development, particularly useful for tracking down bugs and inspecting the behavior of programs one instruction at a time.

User Input

supports user input by setting a predefined value 01 or 01H in the AH register and then calling interrupt (INT). It will take a single character from the user and save the ASCII value of that character in the AL register. The emu8086 emulator displays all values in hexadecimal.

```
; input a character from user
MOV AH, 1
INT 21h    ; the input will be stored in AL register
```

Display Output

It also allows multi-character or string output. Similar to taking input, we have to provide a predefined value in the AH register and call interrupt. The predefined value for single character output is 02 or 02H and for string output 09 or 09H. The output value must be stored in the general-purpose data register before calling interrupt.

```
; Output a character
MOV AH, 2
MOV DL, 35
INT 21H

; Output a string
MOV AH, 9
LEA DX, output
INT 21H
```

LOOPS

Emu8086 emulator supports five types of loop syntax, LOOP, LOOPE, LOOPNE, LOOPNZ, LOOPZ, they are not flexible enough for many situations. We can create our self-defined loops using condition and jump statements.

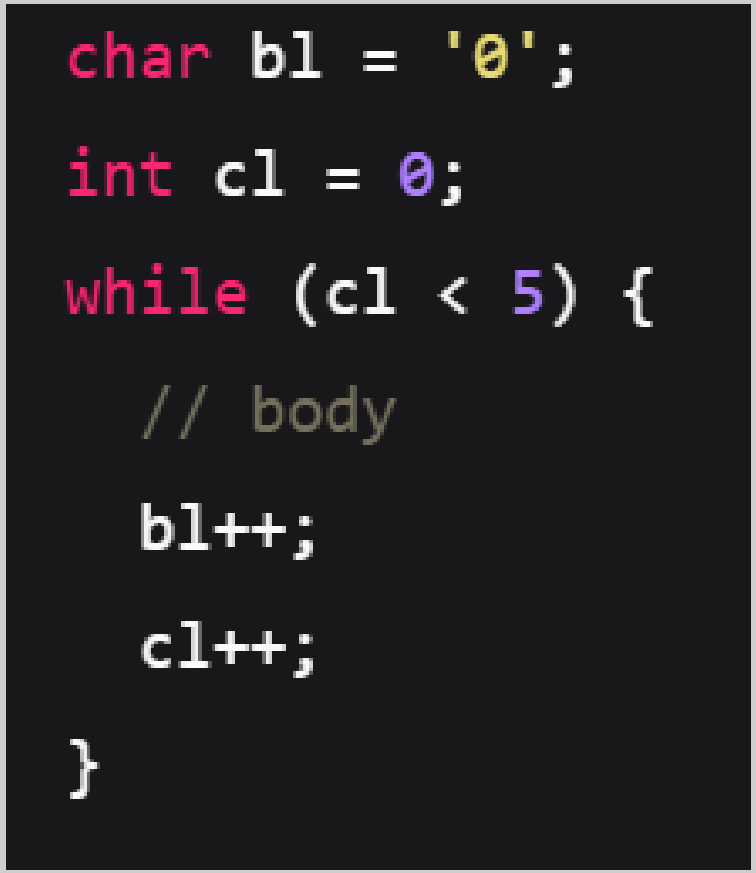
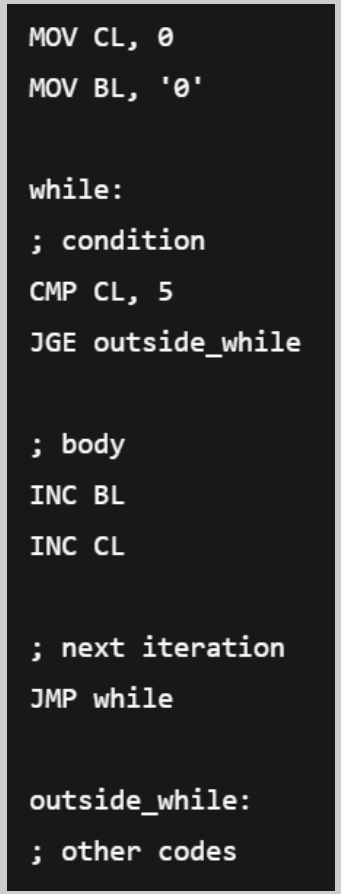
For loop

C Language	Equivalent Assembly

```c
char bl = '0';
for (int cl = 0; cl < 5; cl++) {
  // body
  bl++;
}
```

```assembly
MOV BL, '0'

init_for:
; initialize loop variables
MOV CL, 0

for:
; condition
CMP CL, 5
JGE outside_for

; body
INC BL

; increment/decrement and next iteration
INC CL
JMP for

outside_for:
; other codes
```

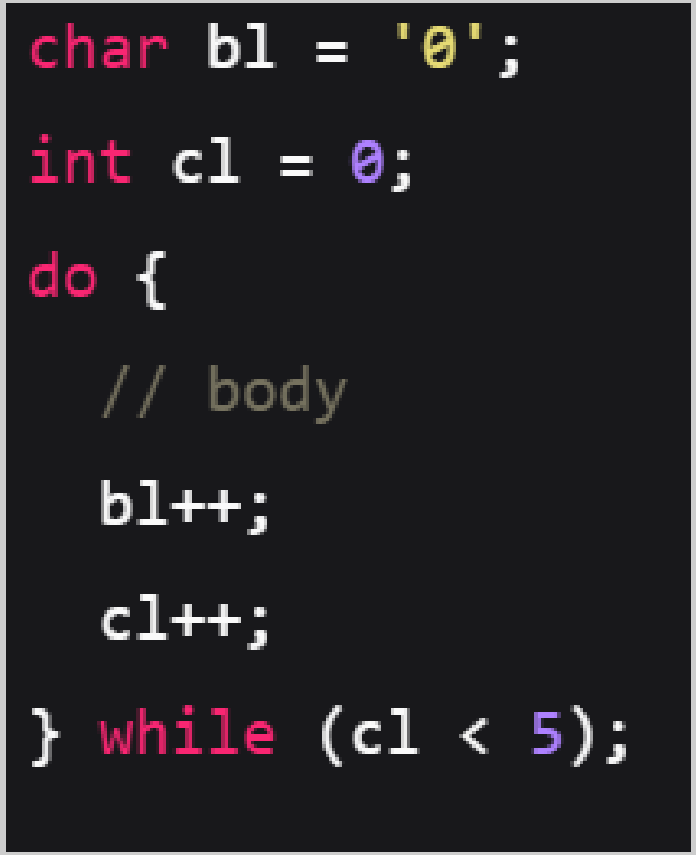
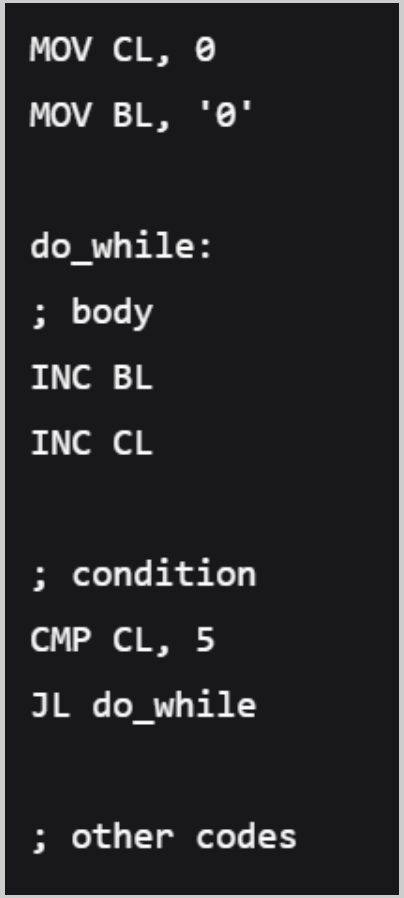
While loop

C Language	Equivalent Assembly
	

```c
char bl = '0';
int cl = 0;
while (cl < 5) {
  // body
  bl++;
  cl++;
}
```

```
MOV CL, 0
MOV BL, '0'

while:
; condition
CMP CL, 5
JGE outside_while

; body
INC BL
INC CL

; next iteration
JMP while

outside_while:
; other codes
```

Do-While loop

C Language	Equivalent Assembly
	

```c
char bl = '0';
int cl = 0;
do {
  // body
  bl++;
  cl++;
} while (cl < 5);
```

```
MOV CL, 0
MOV BL, '0'

do_while:
; body
INC BL
INC CL

; condition
CMP CL, 5
JL do_while

; other codes
```

Using Loop Syntax

We can use predefined loop syntax using the CX register as a counter. Following is an example of loop syntax, which does the same thing as previous loops.

```
MOV BL, '0'

; initialize counter
MOV CX, 5

loop1:
INC BL
LOOP loop1
```

```
.model small
.stack 100H

.code

main PROC

    ; FOR loop example
    MOV BL, '0'

    init_for:
    ; initialize loop variables
    MOV CL, 0

    for:
    ; condition
    CMP CL, 5
    JGE outside_for

    ; body
    INC BL

    ; increment/decrement and next iteration
    INC CL
    JMP for

    outside_for:
    ; other codes


    ; WHILE loop example
    MOV CL, 0
    MOV BL, '0'

    while:
    ; condition
    CMP CL, 5
    JGE outside_while

    ; body
    INC BL
    INC CL

    ; next iteration
    JMP while

    outside_while:
    ; other codes


    ; DO-WHILE loop example
    MOV CL, 0
    MOV BL, '0'

    do_while:
    ; body
    INC BL
    INC CL

    ; condition
    CMP CL, 5
    JL do_while

    ; other codes


    ; LOOP syntax example
    MOV BL, '0'

    ; initialize counter
    MOV CX, 5

    loop1:
    INC BL
    LOOP loop1

    exit:
    MOV AH, 4CH
    INT 21H
    main ENDP
END main
```

```
.model small
.stack 100H

.data
x DB ?
block db '#'
input_promt DW "Please enter the number of lines (1-9): $"
output_message DW "The reverse triangle:", 10, 13, "$"
invalid_message DW "Invalid input! Cannot create a triangle.$"

.code

main PROC
    MOV AX, @data
    MOV DS, AX

    ; Taking user input
    MOV AH, 9
    LEA DX, input_promt
    INT 21H
    MOV AH, 1
    INT 21h
    SUB AL, '0'
    MOV x, AL

    ; Printing new-line
    MOV AH, 2
    MOV DL, 10
    INT 21H
    MOV DL, 13
    INT 21H

    ; Checking for invalid input
    CMP x, 1
    JL invalid
    CMP x, 9
    JG invalid
    JMP create_triangle

    invalid:
    MOV AH, 9
    LEA DX, invalid_message
    INT 21H
    JMP exit

    create_triangle:
    ; Printing output message
    MOV AH, 9
    LEA DX, output_message
    INT 21H

    ; Initialize outer loop counter
    MOV BL, 0

    outer_loop: ; using while loop
    CMP BL, x
    JE exit

    ; Printing new-line
    MOV AH, 2
    MOV DL, 10
    INT 21H
    MOV DL, 13
    INT 21H

    ; Initialize inner loop counter
    MOV CH, 0
    MOV CL, x
    SUB CL, BL

    inner_loop:

    ; Print a single character
    MOV AH, 2
    MOV DL, block
    INT 21H

    LOOP inner_loop

    ; Increment outer loop counter
    INC BL
    JMP outer_loop

    exit:
    MOV AH, 4CH
    INT 21H
    main ENDP
END main
```

Label

is a symbolic name for the address of the instruction that is given immediately after the label declaration. It can be placed at the beginning of a statement and serve as an instruction operand. The exit: used before is a label. Labels are of two types

Symbolic Label

A symbolic label consists of an identifier or symbol followed by a colon (:). They must be defined only once as they have global scope and appear in the object file's symbol table.

```
; Symbolic label
label:
MOV AX, 5
```

Numeric Label

A numeric label consists of a single digit in the range zero (0) through nine (9) followed by a colon (:). They are used only for local reference and excluded in the object file's symbol table. Hence, they have a limited scope and can be re-defined repeatedly.

```
; Numeric label
1:
MOV AX, 5
```

Interrupts

are special instructions in assembly language that allow a program to trigger a predefined action.

They serve to interact with the system's hardware or to request specific services from the operating system.

Interrupts

Category	Interrupts
Video Services	INT 10h/00h, INT 10h/01h, INT 10h/02h, INT 10h/03h, INT 10h/05h, INT 10h/06h, INT 10h/07h, INT 10h/08h, INT 10h/09h, INT 10h/0Ah, INT 10h/0Ch, INT 10h/0Dh, INT 10h/0Eh, INT 10h/13h, INT 10h/1003h
System Information	INT 11h, INT 12h
Disk Services	INT 13h/00h, INT 13h/02h, INT 13h/03h
Hardware & System Services	INT 15h/86h
Keyboard Services	INT 16h/00h, INT 16h/01h
General Interrupts	INT 19h, INT 1Ah/00h, INT 20h
DOS Services	INT 21h, INT 21h/01h, INT 21h/02h, INT 21h/05h, INT 21h/06h, INT 21h/07h, INT 21h/09h, INT 21h/0Ah, INT 21h/0Bh, INT 21h/0Ch, INT 21h/0Eh, INT 21h/19h, INT 21h/25h, INT 21h/2Ah, INT 21h/2Ch, INT 21h/35h, INT 21h/39h, INT 21h/3Ah, INT 21h/3Bh, INT 21h/3Ch, INT 21h/3Dh, INT 21h/3Eh, INT 21h/3Fh, INT 21h/40h, INT 21h/41h, INT 21h/42h, INT 21h/47h, INT 21h/4Ch, INT 21h/56h
Mouse Services	INT 33h/0000h, INT 33h/0001h, INT 33h/0002h, INT 33h/0003h