Gramin Shikshan Prasarak Mandal's

GRAMIN POLYTECHNIC

(ISO 9001:2008 Certified Institute)

# Chapter 4-Assembly Language Programming

By

**Shinde G. B.**

M.TECH. (Electronics Engineering)

# Syllabus

4.1Model of 8086 assembly language program.

4.2 programming using assembler: Arithmetic operation on HEX and BCD numbers, sum of series, smallest and largest numbers from array, sorting numbers in ascending and descending orders, finding ODD,EVEN positive and negative numbers in the array, block transfer, string operations-length ,reverse ,compare , concatenation,copy,count numbers of 1 and 0 in 16 bit numbers.

# Unit Outcomes

- Use the given model of assembly language programs for the given problem.
- Develop the relevant program for the given problem.
- Apply the relevant control loops in the program for the given problem.
- Use string instructions for the given string /block to manipulate its elements.

# Model of Assembly Language Programming

- We will see the structure of assembly language program of 8086.
- The general assembly language programs of 8086 are given below.

# Model 1

- Using SEGMENT,ASSUME and ENDS directives.

MY_data SEGMENT

……

……

ENDS

MY_code SEGMENT

…….

…….

ENDS

# Model 2

- Using .DATA and .CODE directives.

```
.MODEL SMALL
.DATA
……
……
MY_code SEGMENT
……
……
ENDS
END
```

# Programming Using Assembler

- Addition of Two Numbers.
- Assume two 8 bit numbers are stored in AL and BL registers of 8086.
- AL=80H and BL=70H

# Program code

```
.model small
.code
Mov al,80H
Mov bl,70H
Add al,bl
Ends
End
```

# Programming Using Assembler

- Addition of Two Numbers.
- Assume two 8 bit numbers are stored in memory using variable name num1 and num2 respectively.

# Programming Using Assembler

- Subtraction of Two Numbers.
- Assume two 8 bit numbers are stored in AL and BL registers.
- AL=80H,BL=70H

# Model of assembly language programming model-1

```
Data segment
    --
    --
 Data ends
Code segment
Assume cs:code,ds:data
    --
    --
Code ends
end
```

# Model of assembly language programming model-1

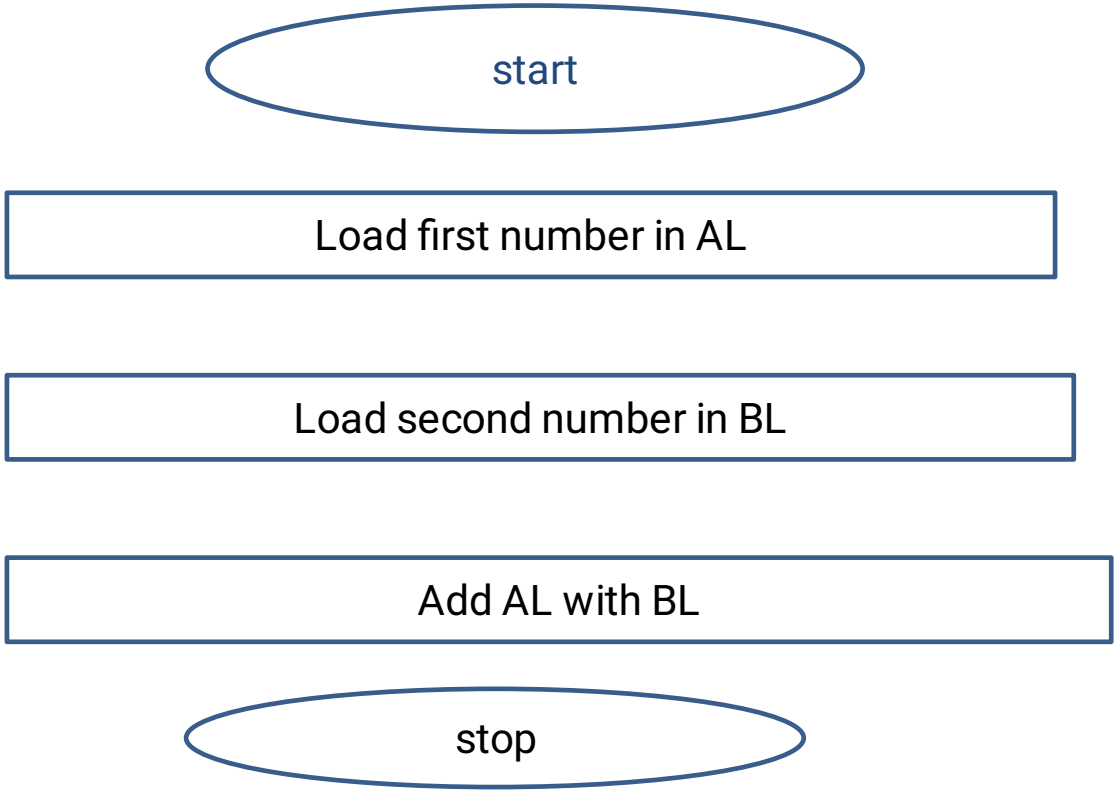
```
.model small
.data
  --
  --
.code
  --
  --
Ends
end
```

Write a program to add two 8 bit numbers present in AL and BL

algorithm:

1.load first number in AL.
2.load second number in BL.
3.add AL with BL.
4.stop.

# FLOWCHART:


start
Load first number in AL
Load second number in BL
Add AL with BL
stop

program:

```
.model small
.code
mov al,50h
mov bl,30h
add al,bl
ends
end
```

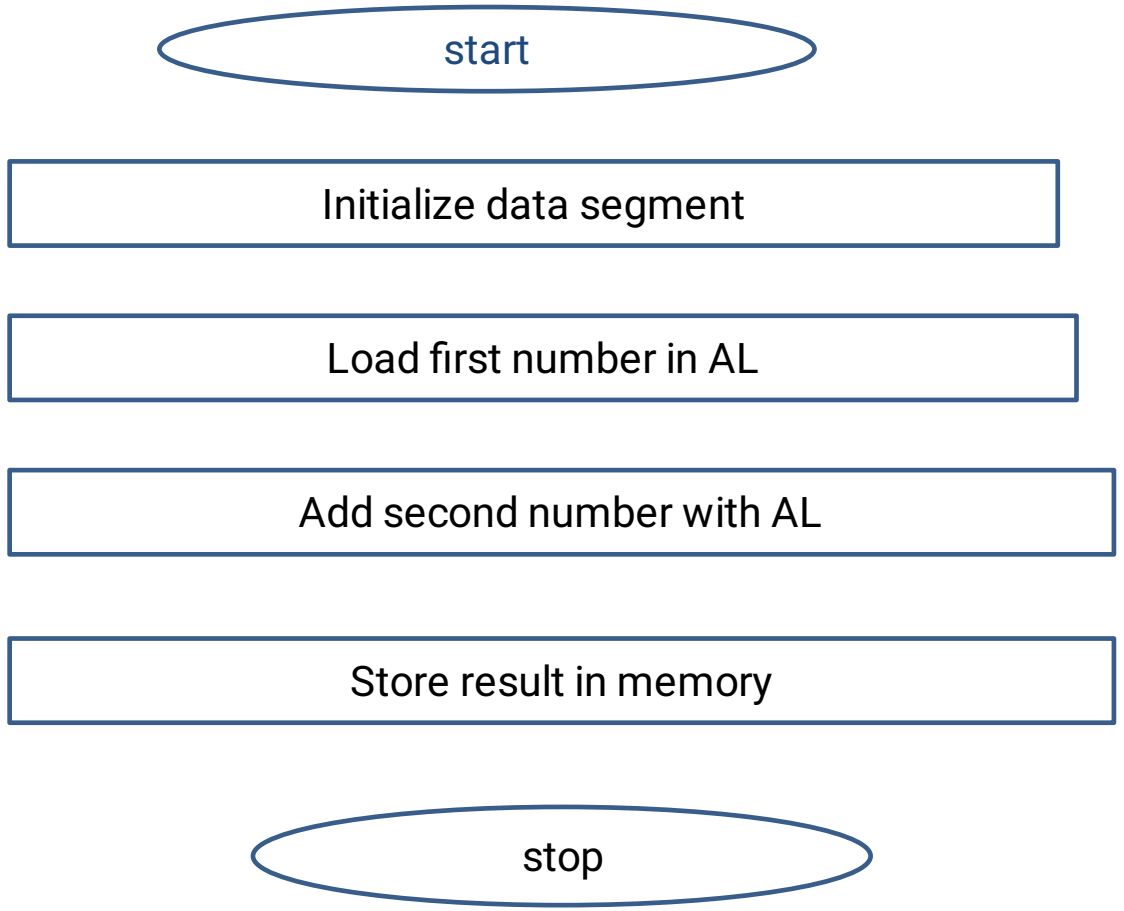
.

# Write a program to add two 8 bit numbers which are stored in memory location

## algorithm:

1.initialize data segment
2.load first number from memory in AL
3.add second number with first number
4.store result in memory location
5.stop

# FLOWCHART:


start
Initialize data segment
Load first number in AL
Add second number with AL
Store result in memory
stop

```
.model small
.data
a db 40h
b db 30h
c db ?
.code
mov ax,@data
mov ds,ax
mov al,a
add al,b
mov c,al
ends
end
```

# Addition of Two 8 bit numbers

```
data segment
a db 09h
b db 02h
c dw ?
data ends

code segment
assume cs:code,ds:data
start:
mov ax,data
mov ds,ax
mov al,a
mov bl,b
add al,bl
mov c,ax
mov ah,4ch
int 21h
code ends
```

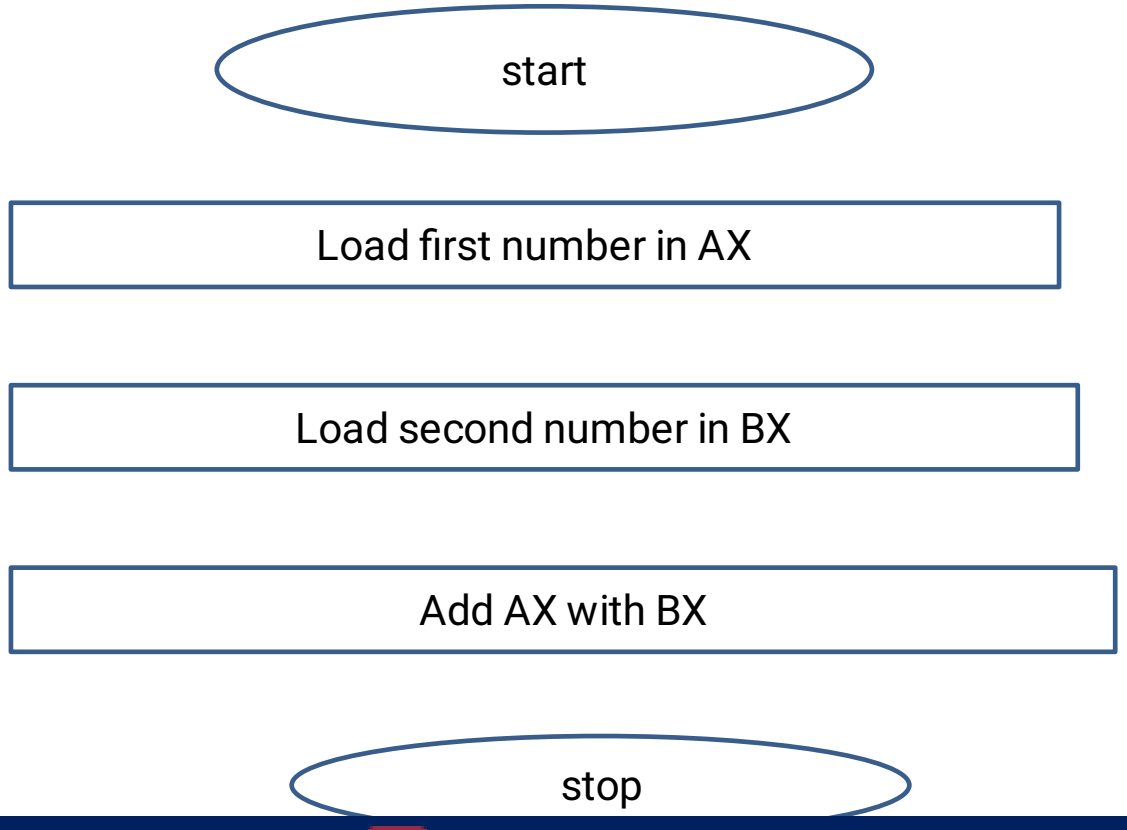
# Program for addition of two 16 bit numbers.

- 1.assume two 16 bit numbers are present in two registers.
- 2.assume two 16 bit numbers are present in memory.

# 1.assume two 16 bit numbers are present in two registers.

- algorithm:
  1.load first number in AX.
  2.load second number in BX.
  3.add AX with BX.
  4.stop.

# FLOWCHART:


start
Load first number in AX
Load second number in BX
Add AX with BX
stop

## 2.assume two 16 bit numbers are present in memory.

- 1.initialize data segment

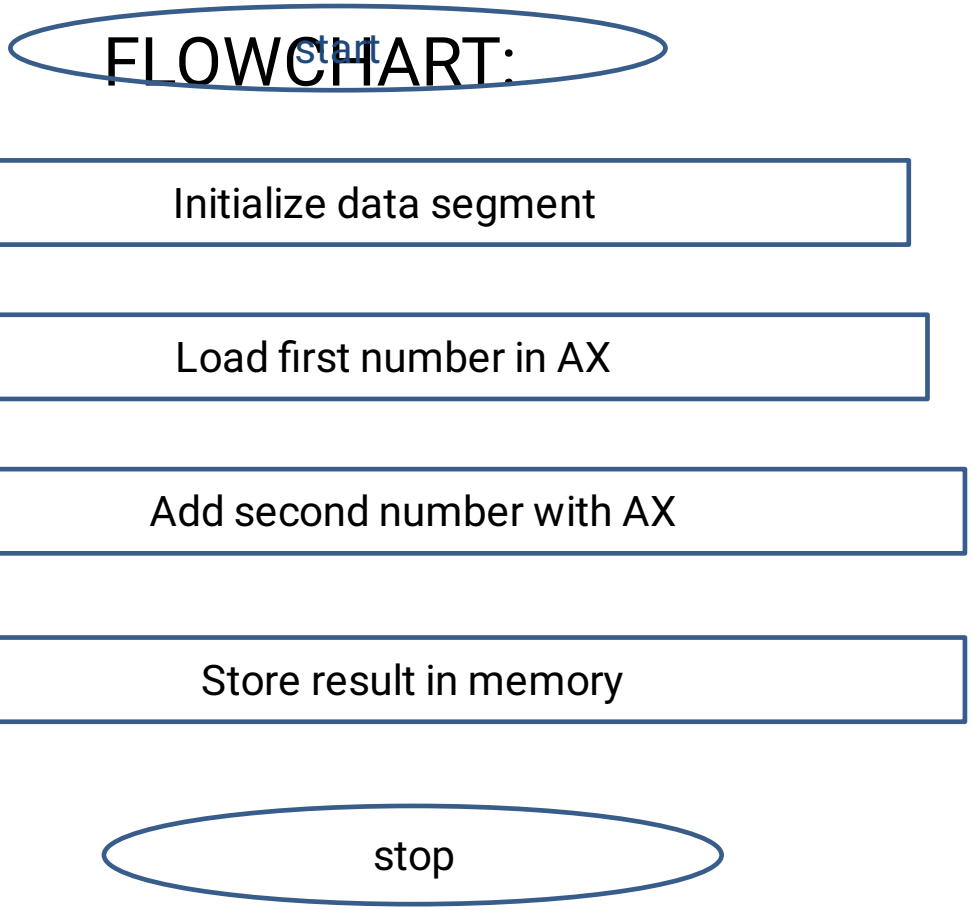
  2.load first number from memory in register

  3.add second number with first number

  4.store result in memory location

  5.stop

# FLOWCHART:

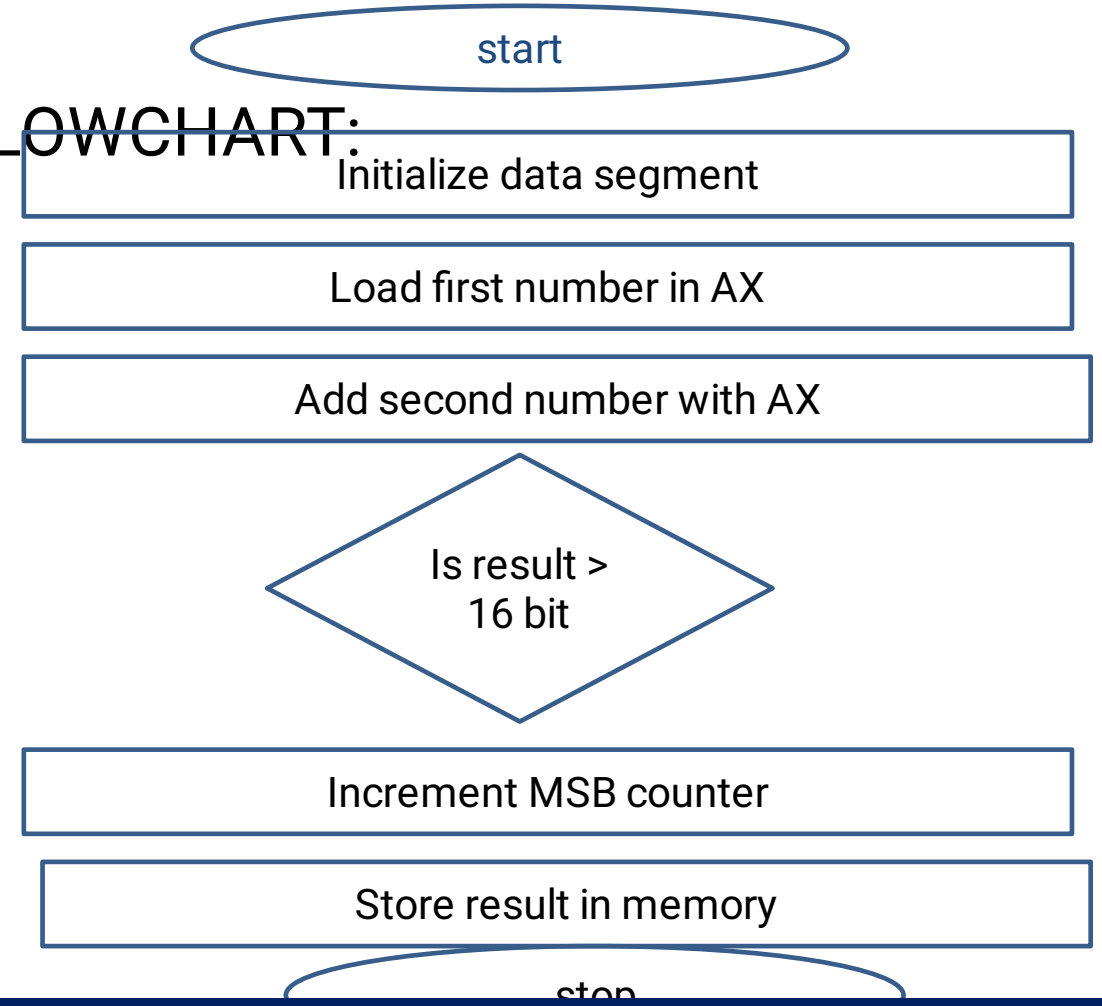

Start
Initialize data segment
Load first number in AX
Add second number with AX
Store result in memory
stop

```
.model small
.data
a dw 1234h
b dw 4537h
c dw ?
.code
mov ax,@data
mov ds,ax
mov ax,a
add ax,b
mov c,ax
ends
end
```

# Addition of two 16 bit numbers where result may be more than 16 bit

- 1.initialize data segment
  2.load first number from memory in register
  3.add second number with first number
  4.check result if it is>16 bit . If yes go to step 5 else go to step 6
  5.increment MSB counter
  6.store result in memory location
  7.stop

# FLOWCHART:


start
Initialize data segment
Load first number in AX
Add second number with AX
Is result >
16 bit
Increment MSB counter
Store result in memory
stop

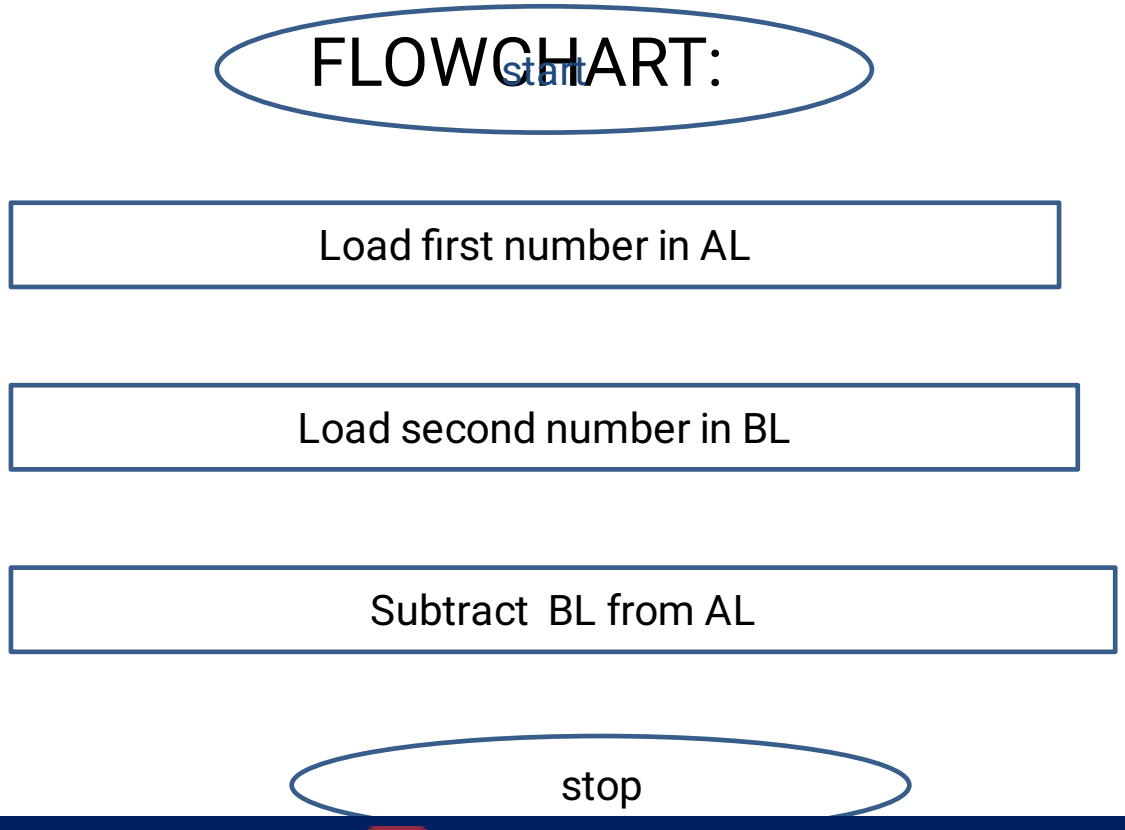
```
.model small
.data
a dw 0ffffh
b dw 0ffffh
result dw ?
c dw ?
.code
mov ax,@data
mov ds,ax
mov ax,a
add ax,b
jnc exit
inc c
exit:
mov result,ax
ends
end
```

Write a program to subtract two 8 bit numbers present in AL and BL

algorithm:

1.load first number in AL.
2.load second number in BL.
3.subtract second number from first number.
4.stop

# FLOWCHART:


start
Load first number in AL
Load second number in BL
Subtract BL from AL
stop

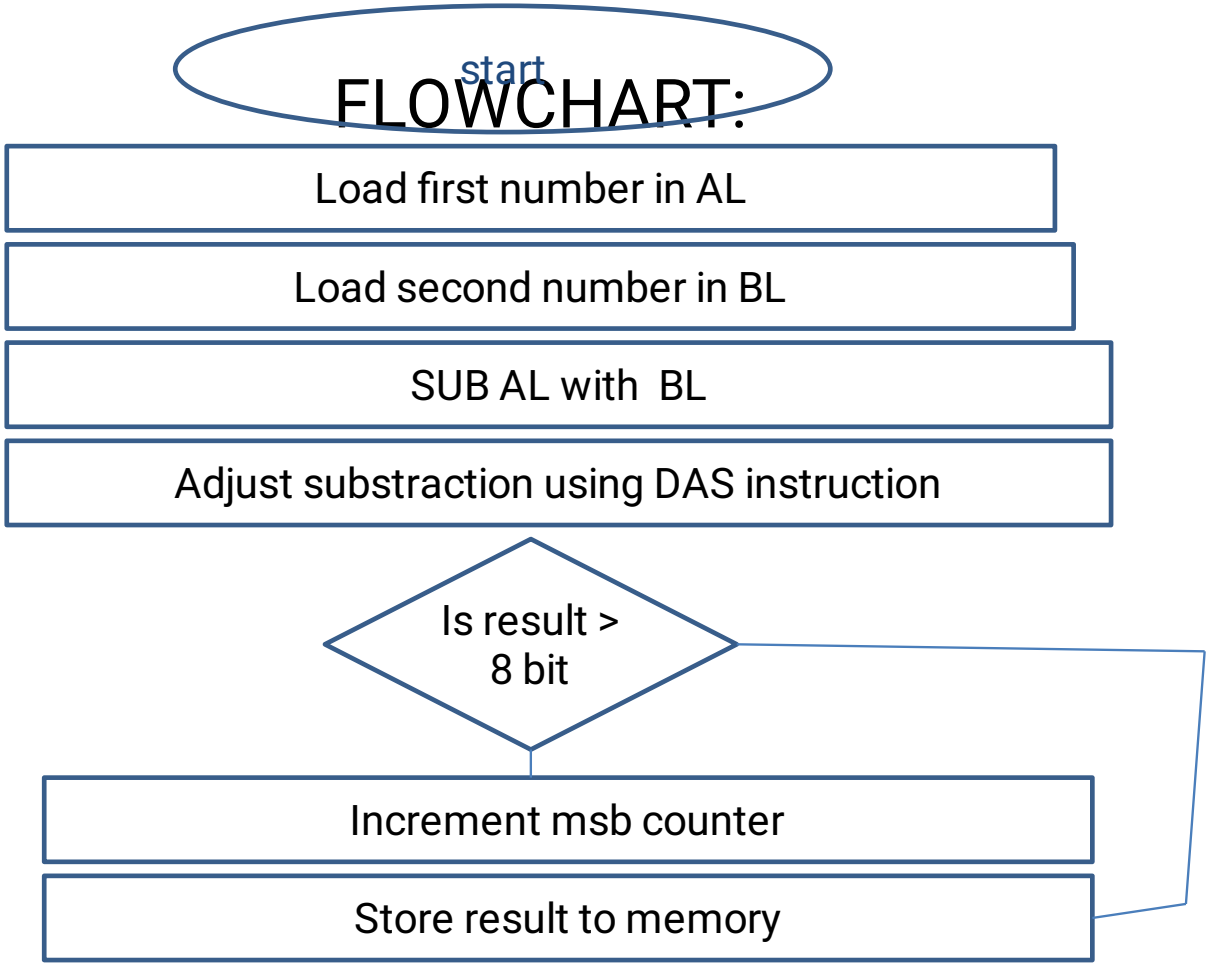
# Addition of BCD numbers

- Write an ALP to add two 8 bit BCD numbers

Algorithm:

1. **Initialize data segment.**
2. **Load first number in AL**
3. **load second number in BL**
4. **Add first number with second number.**
5. **Adjust result to BCD**
6. **If result>8 bit go to next step**
7. **Increment MSB counter by 1**
8. **Store result**
9. **stop**

# FLOWCHART:


start
Load first number in AL
Load second number in BL
SUB AL with BL
Adjust substraction using DAS instruction
Is result > 8 bit
Increment msb counter
Store result to memory

```
.model small
.data
        a db 75h
        b db 21h
        result db ?
       carry db 0
 .code
        Mov ax,@data
        Mov ds,ax
        Mov al,a
        Mov bl,b
        Sub al,bl
        Das
        Jnc next
         Inc carry
Next: mov result,al
         Ends
         end
```

```
.model small
.data
Num1 dw 9999h
Num2 dw 9999h
Result dw ?
Carry db ?
.code
Mov ax,@data
Mov ds,ax
Mov al,byte ptr num1
Add al,byte ptr num2
Daa
Mov byte ptr result,al
Mov al,byte ptr num1+1
Adc al,byte ptr num2+2
Daa
Mov byte ptr result+1,al
Jnc exit
Inc carry
Exit:ends
end
```

# Write an ALP to multiply two 8 bit numbers

- The multiplication of BCD numbers can not performed directly
- There is no such instruction available to adjust result to BCD after multiplication.
- Therefore to multiply successive addition method is used where we can use ADD/ADC and DAA instruction.
- Eg.

  suppose we have to multiply 7 by 4 then we add 7 four times or 4 seven times.

```
.data
Num1 db 55h
Num2 db  05h
Result db 0
Carry db 0
.code
Mov ax,@data   ;initialize data segment
Mov ds,ax
Mov cl,num2      ; multiplicand as a addition counter
Mov al,0          ;initialize AL with 0
Up:
      Add al,num1     :add al with num1 , num2 times
      Daa               ; decimal adjust after BCD addition
      Jnc Next          ; check result is>8 bit if yes
       inc carry         ;incerment carry bit
Next:
      Dec cl            ;decerment addition counter
      Jnz up             ; if addition counter not 0 then go to up
Mov result,al            ;else store result
Ends
```

# Block transfer

- In the block transfer a block of data should be present on the N numbers stored in a memory.
- Now these block of N numbers are  moved from source location to destination locations.
- For doing this we must have to initialize byte counter in CX register. (number of blocks or N)
- Two memory pointers are required to point source block and destination block that are SI And DI
- For doing block transfer we must have to use string instructions such as MOVS/MOVSB/MOVSW.
- Two arrays must be declared where one must contain actual blocks and one is empty

- To declare empty array we can use DUP directive.
- The statement 2 dup(0) allocates two memory locations and initialize them with 0.
- The source must be data segment and the destination must be in extra segment.
- For MOVSW instruction the default memory pointer for source and destination blocks are DS:SI and ES:DI

# Write an ALP to perform transfer of 10 bytes using string instructions

Algorithm:

- Initialize data and extra segment.
- Initialize word counter.
- Initialize memory pointers for source
- Initialize memory pointer for destination array.
- Use sting related instruction
- stop

```
.model small
.data
      Sdata dw 1234h,4321h,2589h,6497h,4563h
      Ddata dw 5dup(0)
.code
        mov ax,@data
        mov ds,ax
        mov es,ax
        mov cx,5
        mov si,offset Sdata
        mov di,offset Ddata
 up: movsw
        Loop up   ;check counter if not 0 go to up
        ends
        end
```

# Write an ALP to perform transfer of 10 bytes without using string instruction

Algorithm:

1. Initialize data and extra segment.
2. Initialize word counter.
3. Initialize memory pointers for source
4. Initialize memory pointer for destination array.
5. Read number from source array
6. Copy it to the destination array
7. Increment memory pointers for source and destination array for next number.
8. Decrement word counter by one
9. If word counter is not equal to zero go to step 5
10. stop

```
.model small
.data
      Sdata dw 1234h,4321h,2589h,6497h,4563h
      Ddata dw 5dup(0)
.code
        mov ax,@data
        mov ds,ax
        mov es,ax
        mov cx,5
        mov si,offset Sdata
        mov di,offset Ddata
  up:
        Mov ax,[si]
        Mov [di],ax
        Add si,2
        Add di,2
        Loop up
        ends
        end
```

# Smallest number from an array

- Array is a set of N numbers having similar data types
- To find smallest number from an array the numbers in the array must be compared with each other.
- Array may consist of 8 bit numbers i.e. byte or 16 bit numbers i.e. word
- To retrieve these numbers we need memory pointer.
- To know how many numbers present in array we have to take one counter, must be taken in the program to read it.

```
.model small
.data
 Array db 10h,20h,05h,50h,60h
 Small db 0
.code
mov ax,@data
mov ds,ax
mov cx,5
mov si,offset array
mov al,[si]
dec cx
Up:
inc si
cmp al,[si]
jc next
mov al,[si]
Next:
Loop up
Mov small,al
```

# largest number from an array

```
.model small
    .data
     Array db 10h,20h,05h,50h,60h
     Small db 0
    .code
    mov ax,@data
    mov ds,ax
    mov cx,5
    mov si,offset array
    mov al,[si]
    dec cx
Up:
    inc si
    cmp al,[si]
    jnc next
    mov al,[si]
Next:
    Loop up
    Mov small,al
```

# Write an ALP to add five 8 bit numbers in series result may be greater than 16 bit

```
.model small
 .data
  Array db 0ffh, 0ffh,, 0ffh, 0ffh, 0ffh,
  Sum db 0
  Carry db 0
.code
   mov ax,@data
   mov ds,ax
   mov cx,5
   mov si,offset array
Up:
  mov al,[si]
  Add sum,al
   Jnc next
    Inc carry
Next:
  Inc si
```

# ODD number or EVEN number

# Write an ALP to find out a number to be odd or even

- In 8 bit or 16 bit numbers the odd or even numbers is decided by the D0 bit of a number
- When we add two odd or two even numbers then result is always even but when we add odd number with even number then the result is always odd.
- When D0 bit of any number is 1 that number is odd and if it is 0 that number is even.
- To check D0 bit of any number rotate the bit of that number in towards right or left by 1 by using ROR or ROL instructions.
- Then D0 bit goes into the carry flag hence by checking carry flag number can be tested for odd or even.

```
.model small
.data
    num db 89h
    odd db 0
    even db 0
.code
    mov ax,@data
    mov ds,ax
    mov al,num
    ROR al,1
    Jnc dn
     ROL al,1
    mov odd,al
    Jmp exit
Dn:
    rol al,1
    Mov even,al
Exit:
    Ends
    end
```

- Algorithm:

1. Initialize data segment
2. Load number in register
3. Check number is odd or even
4. If number is odd then store result to odd
5. Store result to even
6. Stop.

*Thank You!!!*