Computer Organization and Architecture

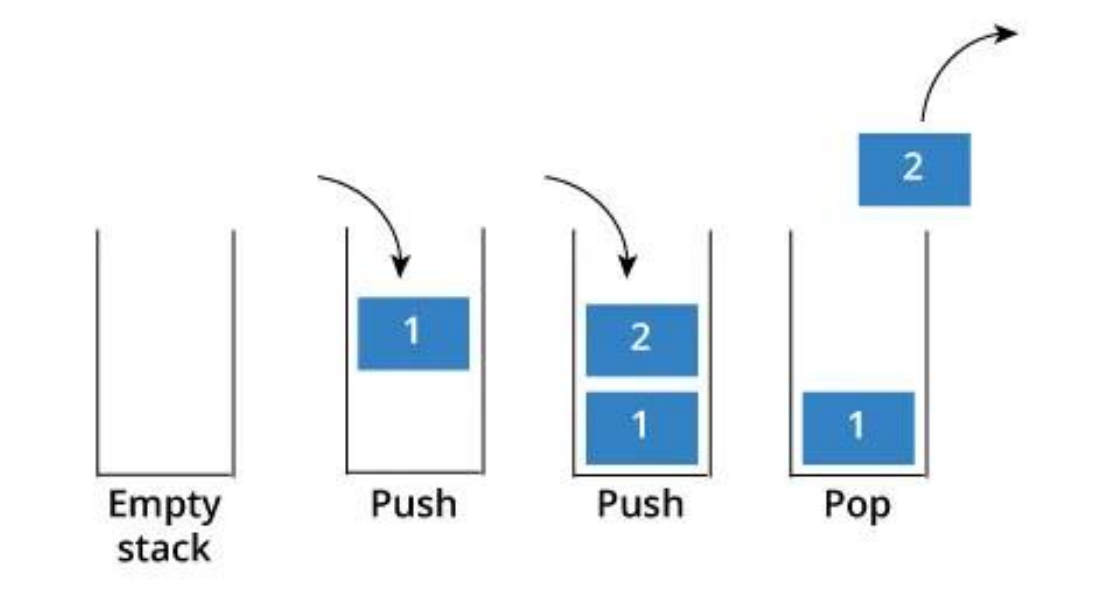
X86 Assembly

Content

Stack Segment
Directives
Defining Segments
Examples

Stack Segment

- The stack is a section of the RAM used by the CPU to store information temporarily.
  - The CPU needs this storage area since there are only a limited number of registers.
  - Stack is a linear data structure which follows a particular order in which the operations are performed, LIFO(Last In First Out).



Stack Segment

Why not design a CPU with more registers?

- The reason is that in the design of the CPU, every transistor is precious and not enough of them are available to build hundreds of registers.
- In addition, how many registers should a CPU have to satisfy every possible program and application?
    - All applications and programming techniques are not the same.

Stack Segment

- If the stack is a section of RAM, there must be registers inside the CPU to point to it.
- The two main registers used to access the stack are
  - SS (stack segment) register, stores information about the memory segment of the stack.
  - SP (stack pointer) register, points to the top of the stack.
  - These registers must be loaded before any instructions accessing the stack are used.
- Every register inside the 80x86 (except segment registers and SP) can be stored in the stack and brought back into the CPU from the stack memory.

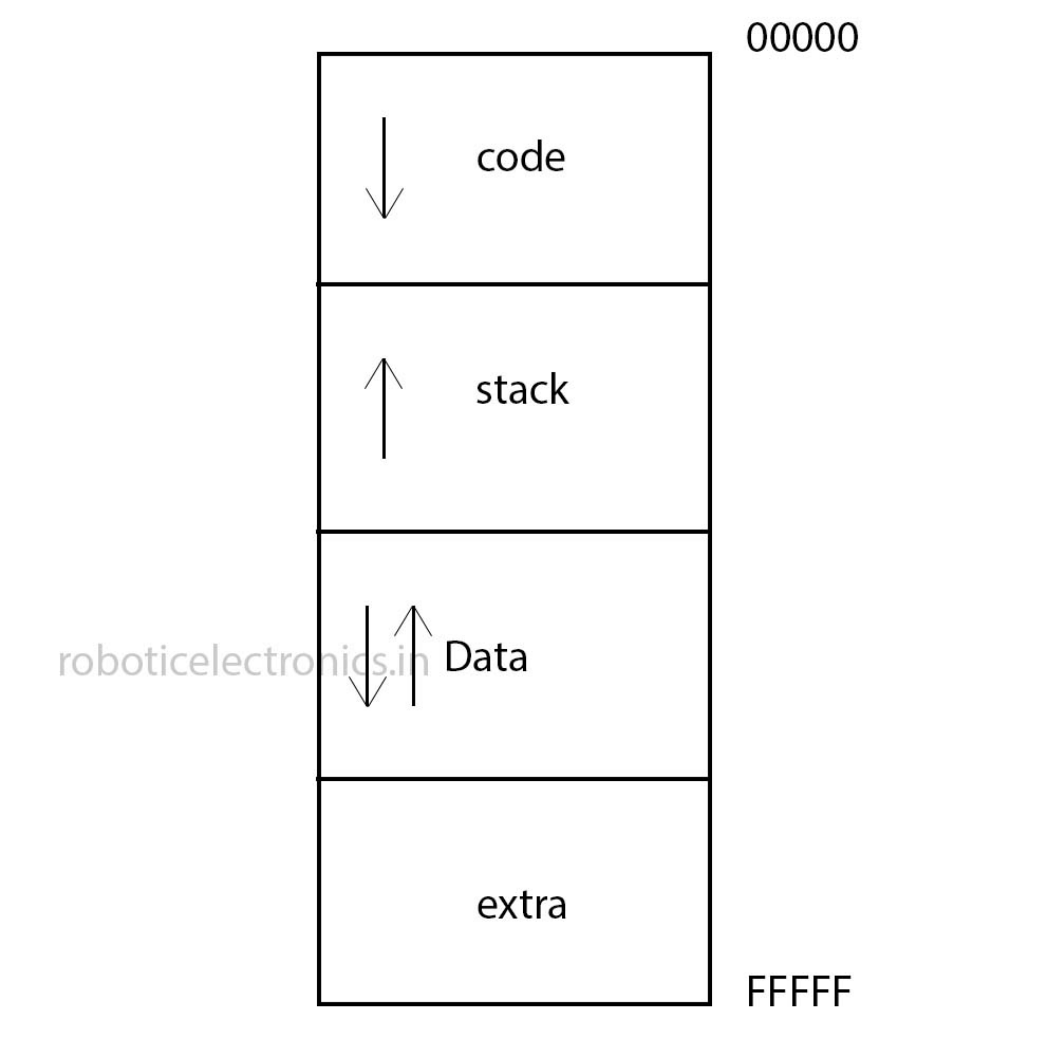
Stack Segment

- The storing of a CPU register in the stack is called a *push.*
- The loading the contents of the stack into the CPU register is called a *pop*.
- The stack pointer register (SP) points at the top of the stack.
  - Decremented when pushing data into the stack.
  - Incremented when popping data from the stack.
- When an instruction pushes or pops a general-purpose register, it must be the entire 16-bit register.
  - One must code "PUSH AX"; there are no instructions such as "PUSH AL" or "PUSH AH".

Stack Segment

- The stack pointer (SP) is the opposite of the instruction pointer (IP)
    - The IP points to the next instruction to be executed and is incremented as each instruction is executed.
    - The SP points to the top of the stack and is decremented after the push operation.
- The stack section and code section are located at opposite ends of the RAM memory to ensure that don’t write over each other.

Stack Segment



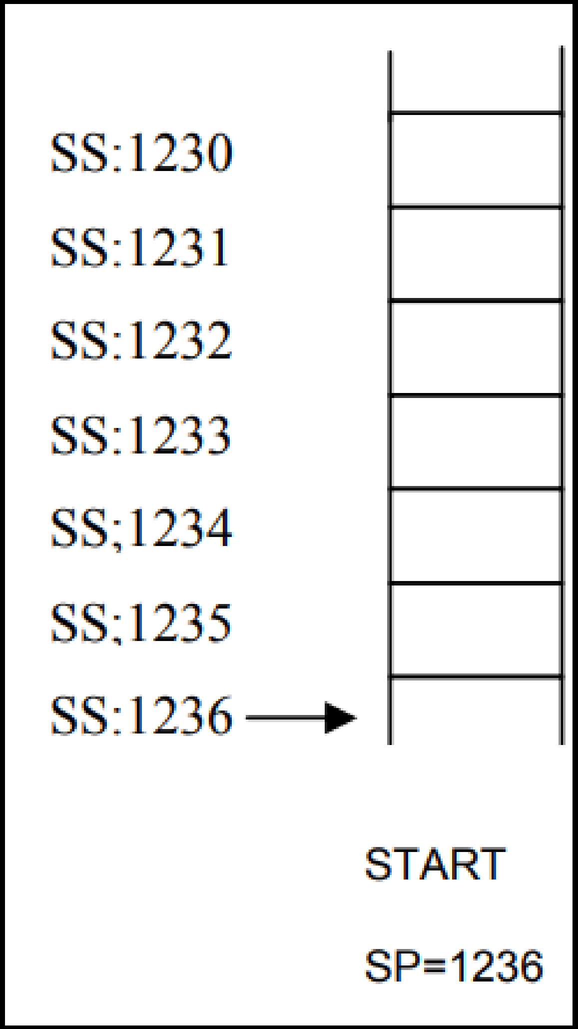
Stack Segment

- Example:

Assuming that SP = 1236, AX = 24B6, DI = 85C2, and DX = 5F93, show the contents of the stack as each of the following instructions is executed:

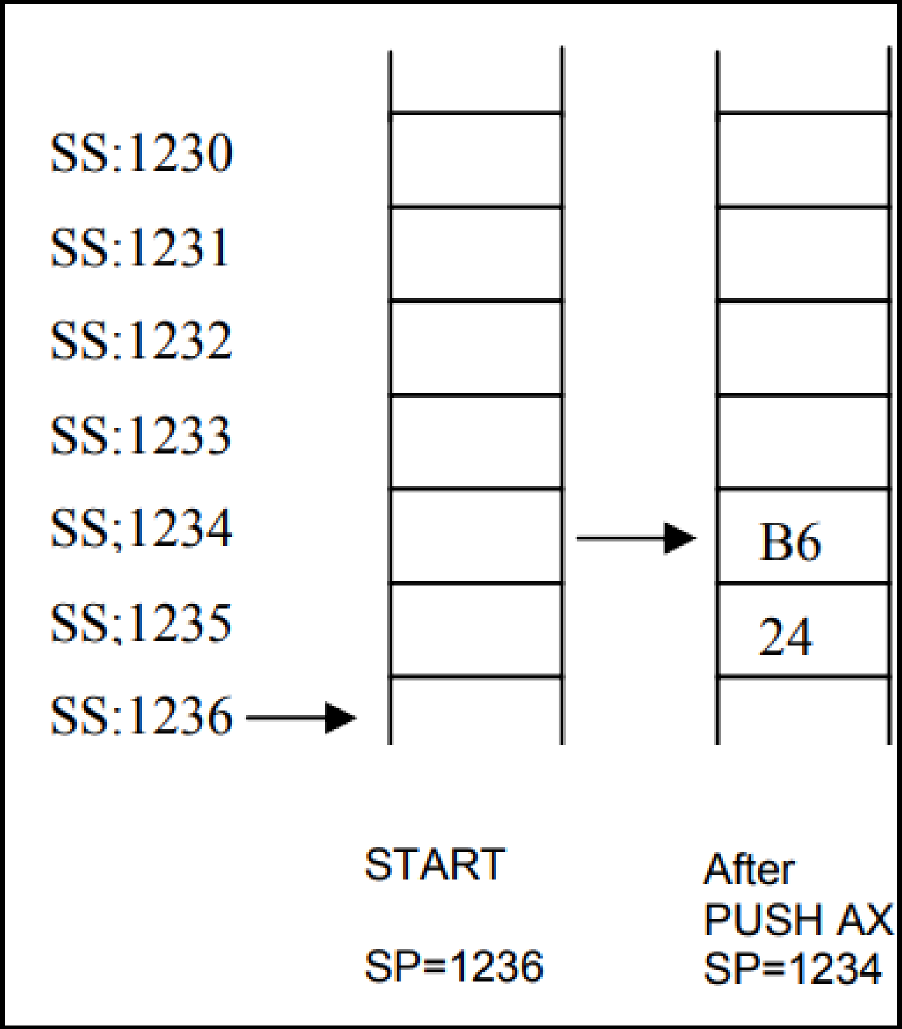
$$PUSH\ DX$$
$$PUSH\ DI$$
$$PUSH\ AX$$

Stack Segment



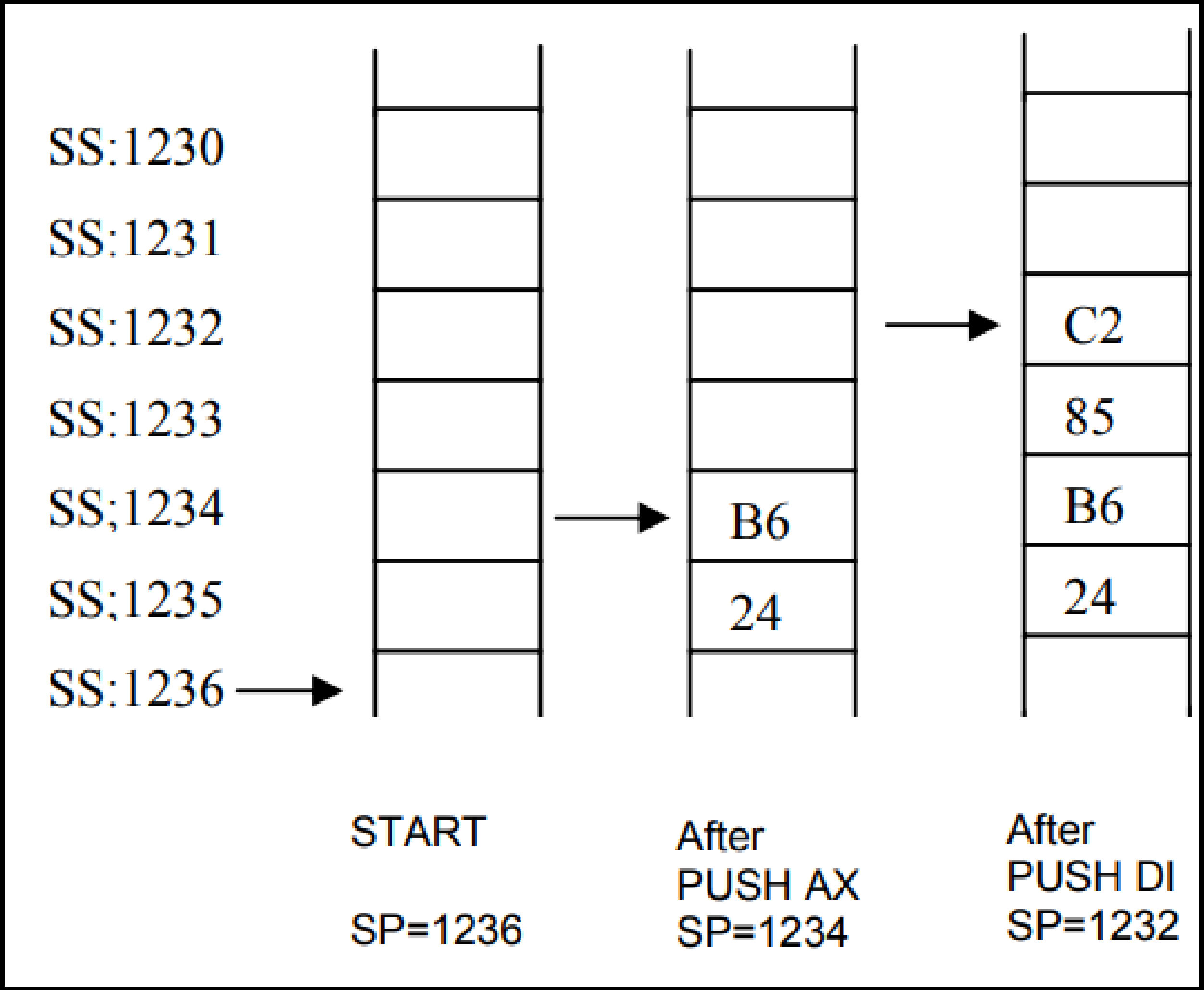
Stack Segment

AX = 24B6



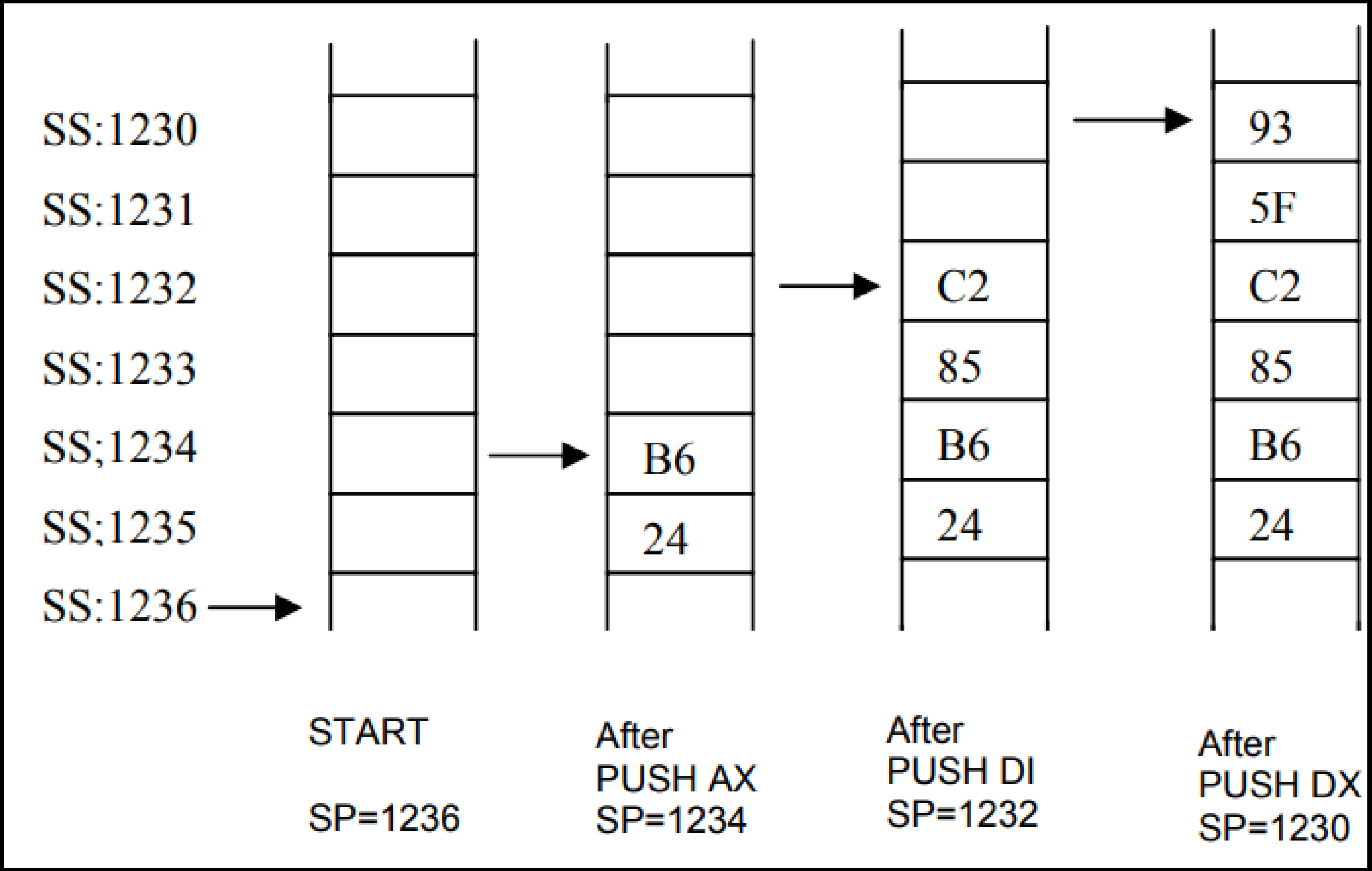
Stack Segment

DI = 85C2



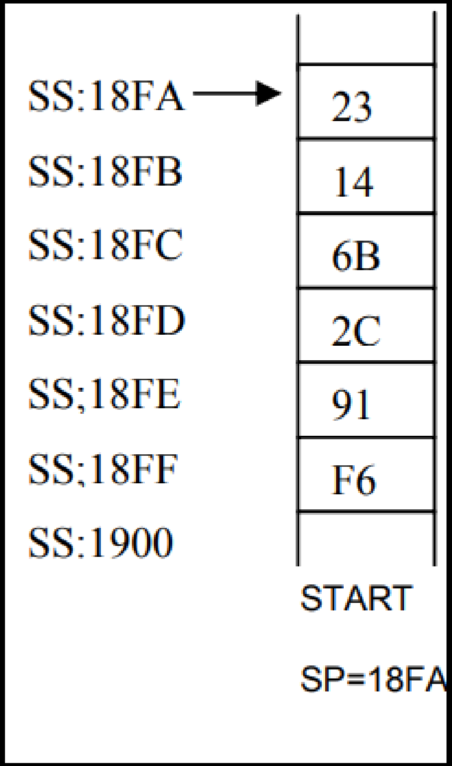
Stack Segment

DX = 5F93



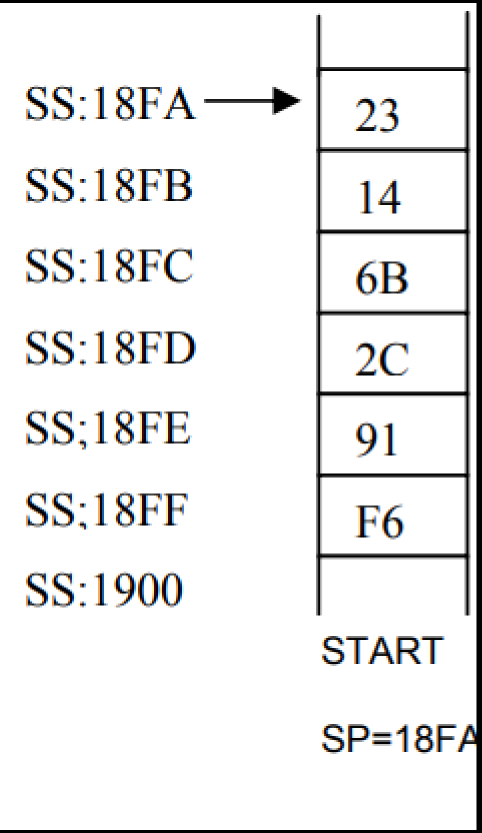
Stack Segment

- Example:
  Assume that the stack is shown below, and SP=18FA, show the contents of the stack and registers as each of the following instructions is executed.



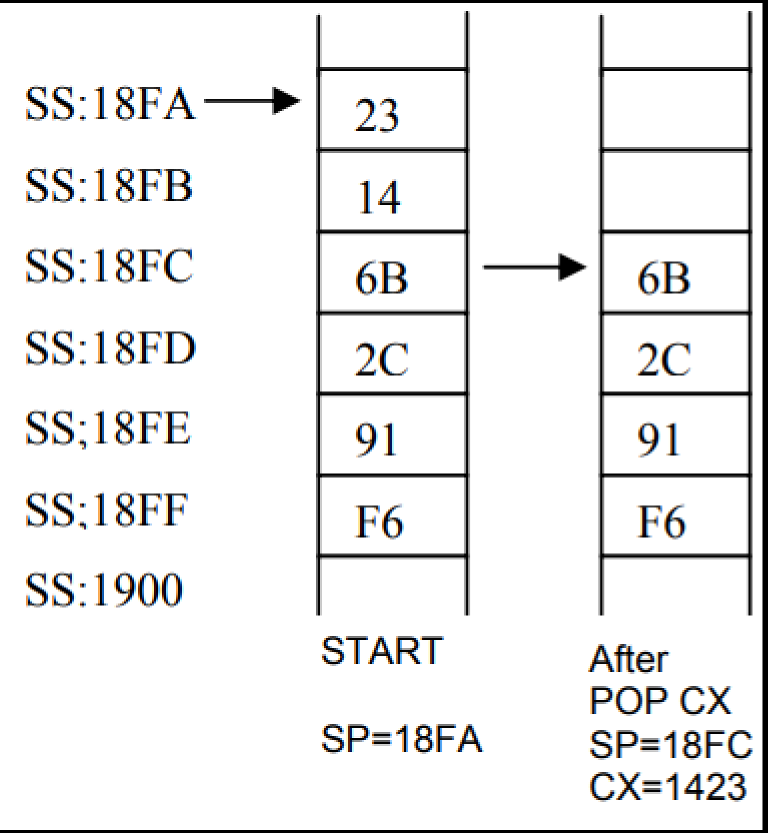
$POP\ CX$
$POP\ DX$
$POP\ BX$

Stack Segment



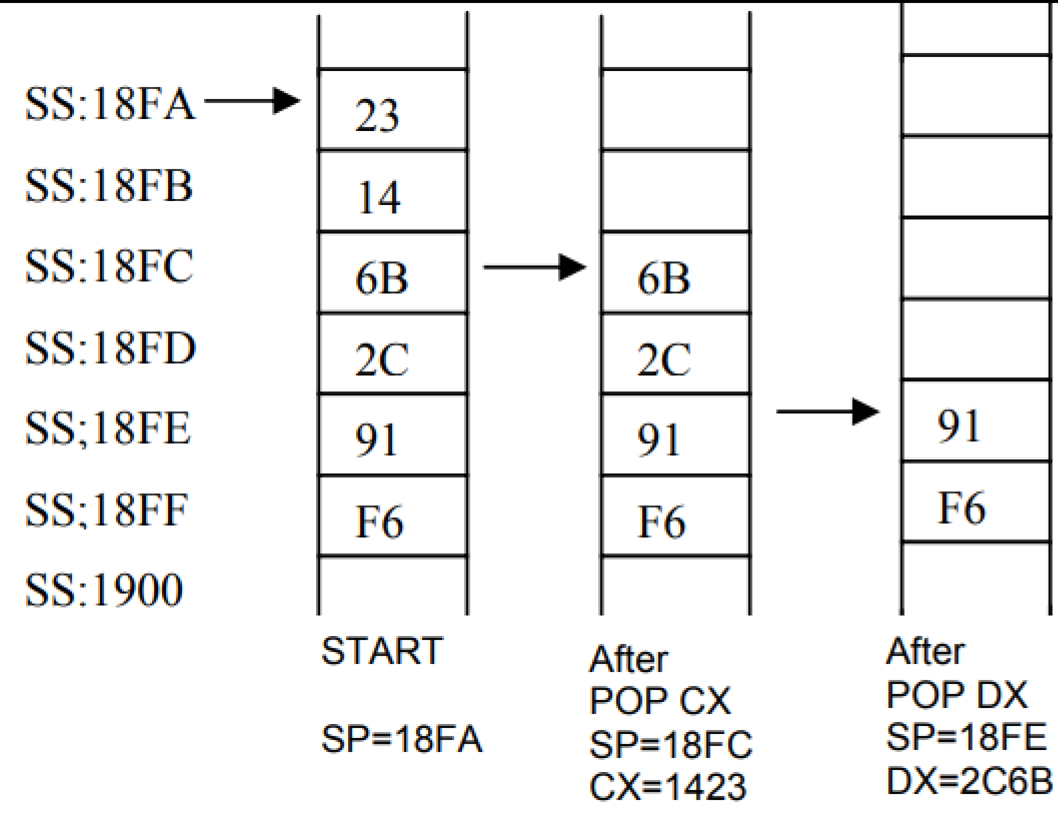
Stack Segment

POP CX



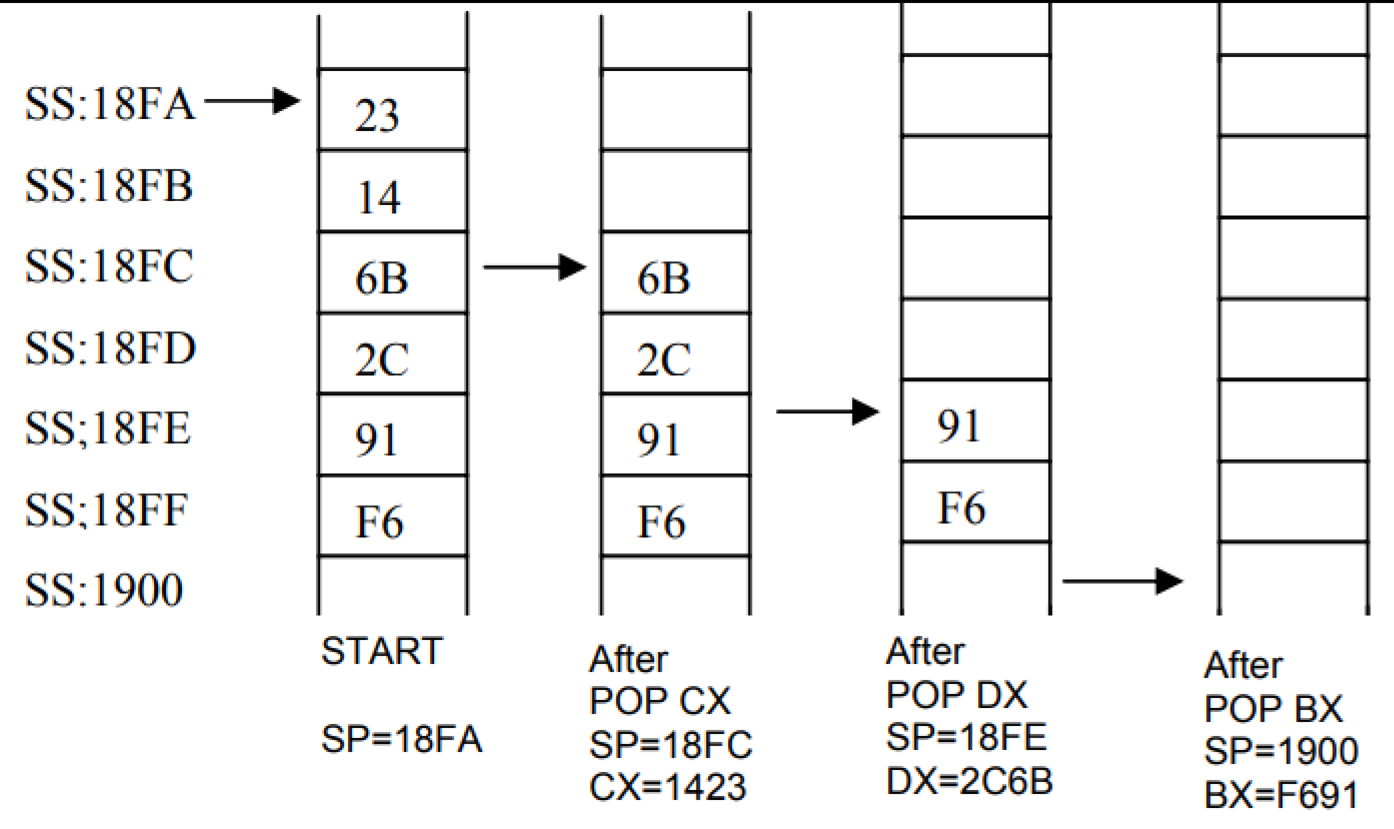
Stack Segment

POP DX



Stack Segment

POP BX



Content

Agenda
Stack Segment
Directives
Defining Segments
Examples

Directives

- Directives give directions to the assembler about how it should translate the Assembly language instructions into machine code.
- An Assembly language instruction consists of four fields:

$$[label:] \quad mnemonic \quad [operands] \quad [;comment]$$

Brackets indicate that the field is optional. Do not type in the brackets.

Directives

- An Assembly language instruction consists of four fields:

$$[label{:}] \quad mnemonic \quad [operands] \quad [;comment]$$

Brackets indicate that the field is optional. Do not type in the brackets.

- The label field allows the program to refer to a line of code by name.
    - The label field cannot exceed 31 characters.

Directives

- An Assembly language instruction consists of four fields:

$$[label:] \quad mnemonic \quad [operands] \quad [;comment]$$

Brackets indicate that the field is optional. Do not type in the brackets.

- The Assembly language mnemonic (instruction) and operand(s) fields together perform the real work of the program and accomplish the tasks for which the program was written.

Directives

- An Assembly language instruction consists of four fields:

$$[label:]\quad mnemonic\quad [operands]\quad [;comment]$$

Brackets indicate that the field is optional. Do not type in the brackets.

- The assembler ignores comments, recommended to make it easier for someone to read and understand the program.

Directives

The .MODEL Directive

- Specifies the size of the memory for each program segments the program needs.
  - Based on this directive, the assembler assigns the required amount of memory to data and code.

Memory Model	Size of Code	Size of Data
TINY	Code + Data < 64KB	Code + data < 64KB
SMALL	Less than 64KB	Less than 64KB
MEDIUM	Can be more than 64KB	Less than 64 KB
COMPACT	Less than 64KB	Can be more than 64KB
LARGE*	Can be more than 64K	Can be more than 64KB
HUGE**	Can be more than 64K	Can be more than 64KB

Array size can not exceed 64 KB.

Array size can exceed 64 KB.

Content

Agenda
Stack Segment
Directives
Defining Segments
Examples

Defining Segments

A program consists of at least three segments:

- stack segment: $.STACK$; marks the beginning of the stack segment
- data segment: .DATA ; marks the beginning of the data segment
- code segment .CODE ; marks the beginning of the code segment

Defining Segments

- To define a stack segment:

$$.STACK\ 64$$

This directive reserves 64 bytes of memory for the stack.

Defining Segments

- To define a data segment:

$$label \quad DataType \quad value$$

- The data type can be
  - DB = Define Byte
  - DW = Define Word (2 bytes)
- Example:

$$\begin{array}{lll} DATA1 & DB & 52H \\ DATA2 & DB & 29H \\ SUM & DB & ? \end{array}$$

  - The “?” means to set the storage space but without a value.

Defining Segments

- To define a code segment:

$$.CODE$$

$$LABEL \qquad PROC \qquad FAR/NEAR$$

$$instructions\ \ldots$$

$$LABEL \qquad ENDP$$

$$END \qquad LABEL$$

Defining Segments

- To define a code segment:

$$
\begin{array}{ccc}
 & .CODE & \\
LABEL & PROC & FAR/NEAR \\
 & instructions\ \ldots & \\
LABEL & ENDP & \\
 & END & LABEL
\end{array}
$$

- The LABEL is the name of the procedure.
- The PROC (procedure) directive defines a group of instructions to accomplish a specific task.
    - Like a method or a function.

Defining Segments

- To define a code segment:

$$
\begin{array}{lll}
 & .CODE & \\
LABEL & PROC & FAR/NEAR \\
 & instructions\ \ldots & \\
LABEL & ENDP & \\
 & END & LABEL
\end{array}
$$

- FAR and NEAR are control transfer instructions used to transfer program control to different locations.
  - NEAR used to transfer control to a memory location within the current code segment.
  - FAR used to transfer control to a memory location outside the current code segment.

Defining Segments

- To define a code segment:

$$
\begin{array}{lll}
 & .CODE & \\
LABEL & PROC & FAR/NEAR \\
 & instructions\ \ldots & \\
LABEL & ENDP & \\
 & END & LABEL
\end{array}
$$

- FAR = intersegment (between segments), used at the program entry point.
    - both Instruction Pointer(IP) and the Code Segment(CS) register will be changed.
- NEAR = intrasegment (within segment).
    - Only Instruction Pointer(IP register) contents will be changed.

Defining Segments

- To define a code segment:

$$
\begin{array}{ccc}
 & .CODE & \\
LABEL & PROC & FAR/NEAR \\
 & instructions\ \ldots & \\
LABEL & ENDP & \\
 & END & LABEL
\end{array}
$$

- The ENDP directive marks the end of the current procedure.

Defining Segments

- To define a code segment:

$$
\begin{array}{ccc}
 & .CODE & \\
LABEL & PROC & FAR/NEAR \\
 & instructions\ \ldots & \\
LABEL & ENDP & \\
 & END \quad LABEL &
\end{array}
$$

- The END directive marks the exist point of the whole program.
    - Marks the last line to be assembled.

Content

Agenda
Stack Segment
Directives
Defining Segments
Examples

Examples

- Example: write a program to add two numbers stored in variable DATA1 and DATA2 and store the result in a variable SUM.

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN    PROC    FAR                 ;this is the program entry point
   mov ax, @DATA                    ;load the data segment address
   mov ds, ax                       ;assign value to DS
   mov al, DATA1                    ;get the first operand
   mov bl, DATA2                    ;get the second operand
   add al, bl                       ;add the operands
   mov SUM, al                      ;store the result in location SUM
   MOV AH,4CH                       ;set up to return to DOS
   INT 21H
MAIN ENDP
    END    MAIN
```

Examples

```
  .MODEL SMALL
  .STACK 64

  .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

  .CODE
MAIN    PROC    FAR                 ;this is the program entry point
  mov ax, @DATA                     ;load the data segment address
  mov ds, ax                        ;assign value to DS
  mov al, DATA1                     ;get the first operand
  mov bl, DATA2                     ;get the second operand
  add al, bl                        ;add the operands
  mov SUM, al                       ;store the result in location SUM
  MOV AH,4CH                        ;set up to return to DOS
  INT 21H
MAIN ENDP
   END    MAIN
```

Define memory model SMALL

Examples

```
.MODEL SMALL
.STACK 64

.DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

.CODE
MAIN    PROC    FAR                 ;this is the program entry point
   mov ax, @DATA                    ;load the data segment address
   mov ds, ax                       ;assign value to DS
   mov al, DATA1                    ;get the first operand
   mov bl, DATA2                    ;get the second operand
   add al, bl                       ;add the operands
   mov SUM, al                      ;store the result in location SUM
   MOV AH,4CH                       ;set up to return to DOS
   INT 21H
MAIN ENDP
   END    MAIN
```

Define stack of size 64 bytes

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN    PROC    FAR              ;this is the program entry point
   mov ax, @DATA                 ;load the data segment address
   mov ds, ax                    ;assign value to DS
   mov al, DATA1                 ;get the first operand
   mov bl, DATA2                 ;get the second operand
   add al, bl                    ;add the operands
   mov SUM, al                   ;store the result in location SUM
   MOV AH,4CH                    ;set up to return to DOS
   INT 21H
MAIN ENDP
    END    MAIN
```

Define the data segment with:
DATA1 is a byte variable = 52
DATA2 is a byte variable = 29
SUM is a byte variable not intialized

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN    PROC    FAR                 ;this is the program entry point
   mov ax, @DATA                    ;load the data segment address
   mov ds, ax                       ;assign value to DS
   mov al, DATA1                    ;get the first operand
   mov bl, DATA2                    ;get the second operand
   add al, bl                       ;add the operands
   mov SUM, al                      ;store the result in location SUM
   MOV AH,4CH                       ;set up to return to DOS
   INT 21H
MAIN ENDP
    END    MAIN
```

The program entry point is the MAIN procedure.

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN    PROC    FAR                   ;this is the program entry point
   mov ax, @DATA                      ;load the data segment address
   mov ds, ax                         ;assign value to DS
   mov al, DATA1                      ;get the first operand
   mov bl, DATA2                      ;get the second operand
   add al, bl                         ;add the operands
   mov SUM, al                        ;store the result in location SUM
   MOV AH,4CH                         ;set up to return to DOS
   INT 21H
MAIN ENDP
    END    MAIN
```

Load the data segment address into the AX, then load it into the DS (Data Segment) register.

We cannot load the address directly into the DS register.

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN    PROC    FAR                  ;this is the program entry point
   mov ax, @DATA                     ;load the data segment address
   mov ds, ax                        ;assign value to DS
   mov al, DATA1                     ;get the first operand
   mov bl, DATA2                     ;get the second operand
   add al, bl                        ;add the operands
   mov SUM, al                       ;store the result in location SUM
   MOV AH,4CH                        ;set up to return to DOS
   INT 21H
MAIN ENDP
    END    MAIN
```

Load first operand to AL.
Load second operand to BL.
Add the contents of the BL to the AL, and store the result in the AL.

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN    PROC    FAR                    ;this is the program entry point
   mov ax, @DATA                       ;load the data segment address
   mov ds, ax                          ;assign value to DS
   mov al, DATA1                       ;get the first operand
   mov bl, DATA2                       ;get the second operand
   add al, bl                          ;add the operands
   mov SUM, al                         ;store the result in location SUM
   MOV AH,4CH                          ;set up to return to DOS
   INT 21H
MAIN ENDP
    END    MAIN
```

Store the result in the AL to SUM variable.

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN    PROC    FAR                 ;this is the program entry point
   mov ax, @DATA                    ;load the data segment address
   mov ds, ax                       ;assign value to DS
   mov al, DATA1                    ;get the first operand
   mov bl, DATA2                    ;get the second operand
   add al, bl                       ;add the operands
   mov SUM, al                      ;store the result in location SUM
   MOV AH,4CH                       ;set up to return to DOS
   INT 21H
MAIN ENDP
    END    MAIN
```

This to return control to the operating system, which in this case is the DOS.

The value in the AH, which is 4C, is the return code.
The INT 21H is the interrupt instruction for the DOS.

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN    PROC    FAR                 ;this is the program entry point
   mov ax, @DATA                    ;load the data segment address
   mov ds, ax                       ;assign value to DS
   mov al, DATA1                    ;get the first operand
   mov bl, DATA2                    ;get the second operand
   add al, bl                       ;add the operands
   mov SUM, al                      ;store the result in location SUM
   MOV AH,4CH                       ;set up to return to DOS
   INT 21H
MAIN ENDP
    END    MAIN
```

The end of the MAIN procedure

Examples

```
   .MODEL SMALL
   .STACK 64

   .DATA
DATA1   DB  52H
DATA2   DB  29H
SUM     DB  ?

   .CODE
MAIN   PROC   FAR              ;this is the program entry point
   mov ax, @DATA               ;load the data segment address
   mov ds, ax                  ;assign value to DS
   mov al, DATA1               ;get the first operand
   mov bl, DATA2               ;get the second operand
   add al, bl                  ;add the operands
   mov SUM, al                 ;store the result in location SUM
   MOV AH,4CH                  ;set up to return to DOS
   INT 21H
MAIN ENDP
   END   MAIN
```

The exist point of the program.

Examples

- We can use this shell as a code template.

```
    .MODEL SMALL

    .STACK 64

    .DATA
;; MY DATA HERE ;;

    .CODE
MAIN    PROC    FAR     ;this is the program entry point
    mov ax, @DATA       ;load the data segment address
    mov ds, ax          ;assign value to DS

    ;; MY CODE HERE ;;

    MOV AH,4CH          ;set up to return to DOS
    INT 21H
MAIN ENDP
     END    MAIN
```

Examples

- Example: write a program to compute the sum of 5 Byte numbers and store the result in a variable SUM. The numbers are: 25H, 12H, 15H, 1 FH,2BH

Examples

Flowchart

Initialize loop counter = 5

```
MOV CX,05
```

Examples

Flowchart

Initialize loop counter = 5 → Initialize base counter to point to the first number

MOV BX,OFFSET DATA IN

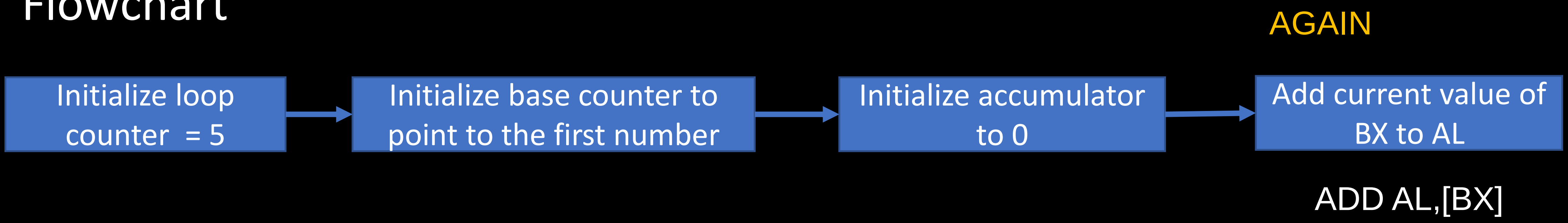
Examples

Flowchart

MOV AL,0

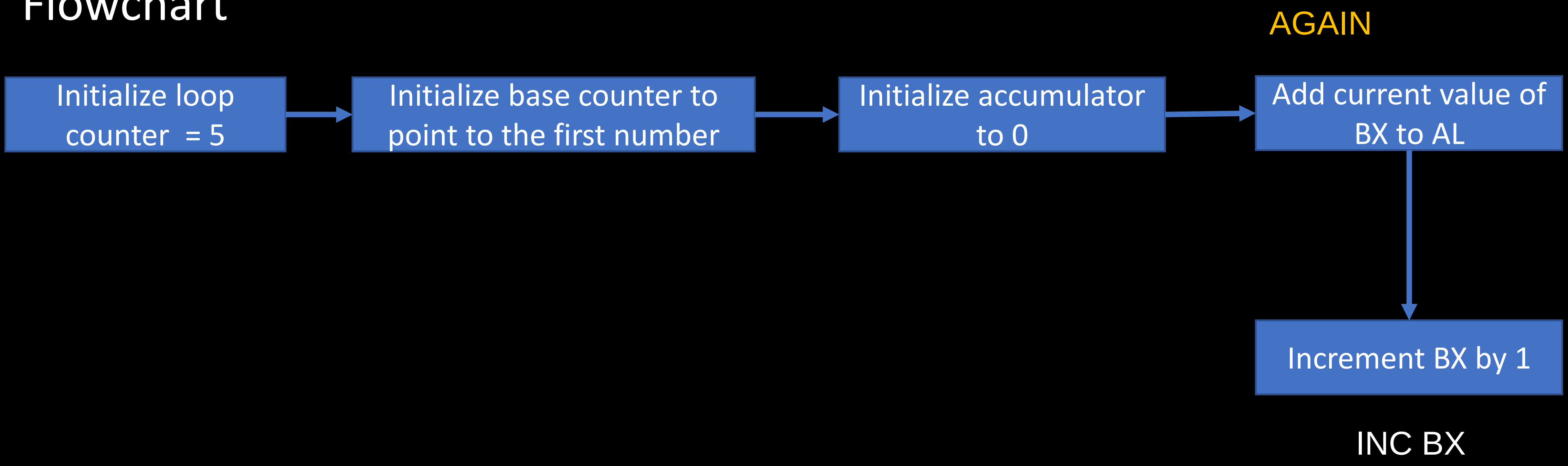
Examples

Flowchart



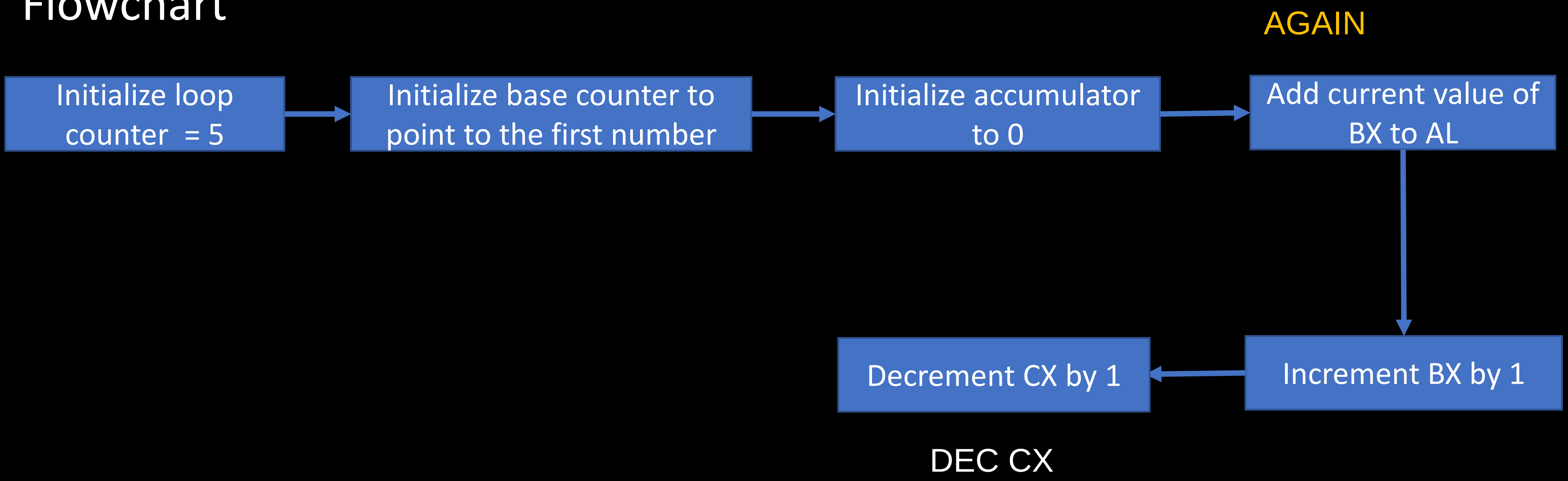
Examples

Flowchart



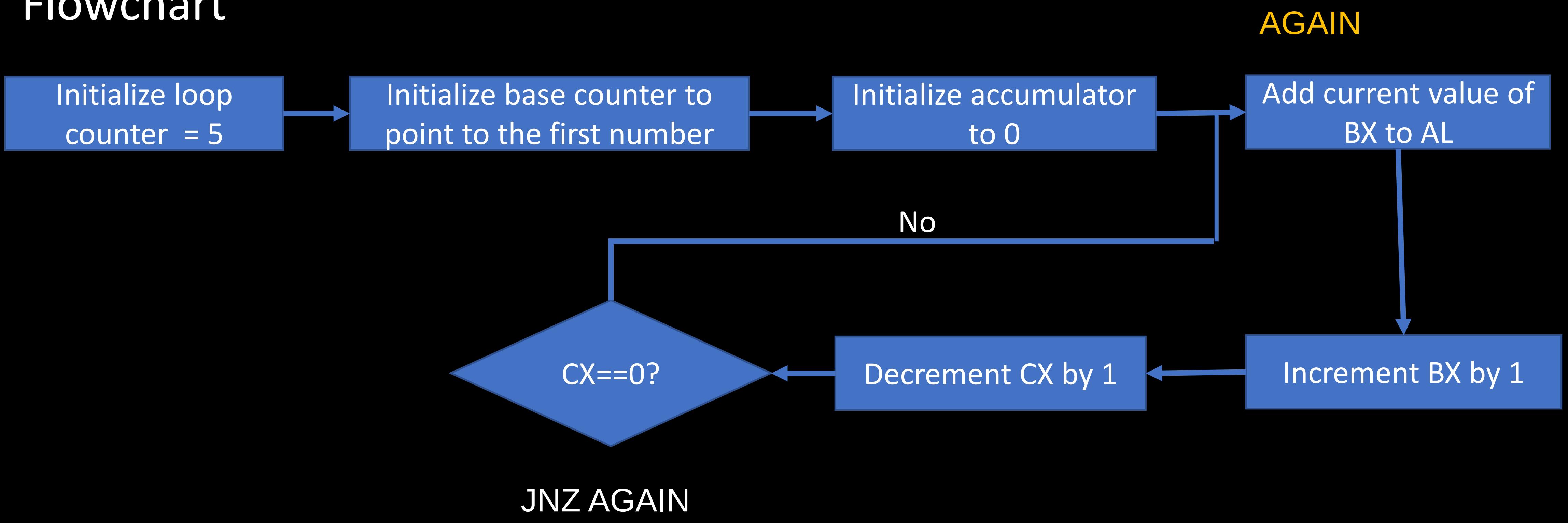
Examples

Flowchart



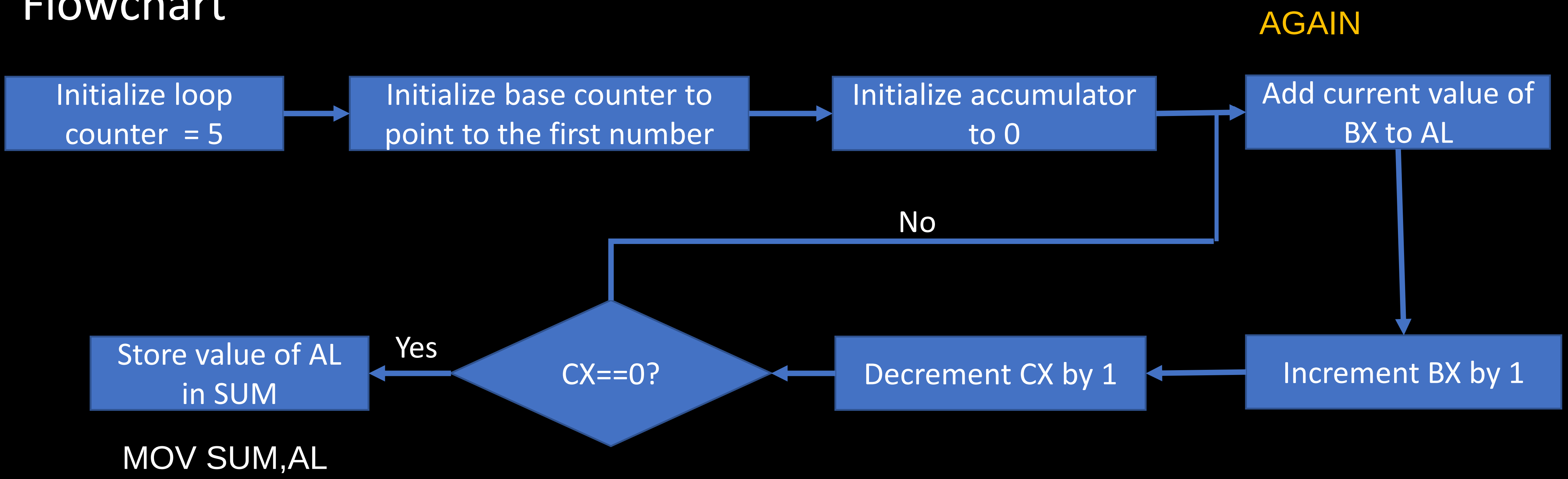
Examples

Flowchart



Examples

Flowchart



Examples

Code

```
   .MODEL SMALL
   .STACK 64
   .DATA
DATA_IN DB  25H, 12H, 15H, 1FH, 2BH
SUM       DB  ?

   .CODE
MAIN    PROC    FAR                          ;this is the program entry point
            mov ax, @DATA                    ;load the data segment address
            mov ds, ax                       ;assign value to DS
            mov cx, 05                       ;set up loop counter CX=5
            mov bx, OFFSET DATA_IN           ;set up data pointer BX
            mov al, 0                        ;initialize AL
AGAIN:      add al, [bx]                     ;add next data item to AL
            inc bx                           ;make BX point to next data item
            dec cx                           ;decrement loop counter
            jnz AGAIN                        ;jump if loop counter not zero
            mov SUM, al                      ;load result into sum
            MOV AH,4CH                       ;set up to return to DOS
            INT 21H
MAIN    ENDP
            END    MAIN
```

Examples

- Write a program that perform the following operation: $\frac{15+12}{18-9} * (3+1)$
- The multiply instruction

Algorithm:

when operand is a **byte**:
AX = AL * operand.

when operand is a **word**:
(DX AX) = AX * operand.

Example:

```
MOV AL, 200   ; AL = 0C8h
MOV BL, 4
MUL BL        ; AX = 0320h (800)
RET
```

The Divide instruction

Algorithm:

when operand is a **byte**:
AL = AX / operand
AH = remainder (modulus)

when operand is a **word**:
AX = (DX AX) / operand
DX = remainder (modulus)

Example:

```
MOV AX, 203   ; AX = 00CBh
MOV BL, 4
DIV BL        ; AL = 50 (32h), AH = 3
RET
```

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX
  3. Add 12 to the AX
  4. Store the result in a variable
  5. Move the 18 into the AX
  6. Subtract 9 from AX
  7. Store the result in the BX
  8. Load the variable into the AX
  9. Divide the AX by BX
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX
  4. Store the result in a variable
  5. Move the 18 into the AX
  6. Subtract 9 from AX
  7. Store the result in the BX
  8. Load the variable into the AX
  9. Divide the AX by BX
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable
  5. Move the 18 into the AX
  6. Subtract 9 from AX
  7. Store the result in the BX
  8. Load the variable into the AX
  9. Divide the AX by BX
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX
  6. Subtract 9 from AX
  7. Store the result in the BX
  8. Load the variable into the AX
  9. Divide the AX by BX
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX
  7. Store the result in the BX
  8. Load the variable into the AX
  9. Divide the AX by BX
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX $sub\ ax, 9$
  7. Store the result in the BX
  8. Load the variable into the AX
  9. Divide the AX by BX
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX $sub\ ax, 9$
  7. Store the result in the BX $mov\ bx, ax$
  8. Load the variable into the AX
  9. Divide the AX by BX
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX $sub\ ax, 9$
  7. Store the result in the BX $mov\ bx, ax$
  8. Load the variable into the AX $mov\ ax, res$
  9. Divide the AX by BX
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX $sub\ ax, 9$
  7. Store the result in the BX $mov\ bx, ax$
  8. Load the variable into the AX $mov\ ax, res$
  9. Divide the AX by BX $div\ bx$
  10. Store the result in a variable
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX $sub\ ax, 9$
  7. Store the result in the BX $mov\ bx, ax$
  8. Load the variable into the AX $mov\ ax, res$
  9. Divide the AX by BX $div\ bx$
  10. Store the result in a variable $mov\ res, ax$
  11. Move 3 into the AX
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX $sub\ ax, 9$
  7. Store the result in the BX $mov\ bx, ax$
  8. Load the variable into the AX $mov\ ax, res$
  9. Divide the AX by BX $div\ bx$
  10. Store the result in a variable $mov\ res, ax$
  11. Move 3 into the AX $mov\ ax, 3$
  12. Increment AX
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX $sub\ ax, 9$
  7. Store the result in the BX $mov\ bx, ax$
  8. Load the variable into the AX $mov\ ax, res$
  9. Divide the AX by BX $div\ bx$
  10. Store the result in a variable $mov\ res, ax$
  11. Move 3 into the AX $mov\ ax, 3$
  12. Increment AX $inc\ ax$
  13. Move the result into the BX
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section — $res\ DW\ \ ?$
  2. Move the 15 into the AX — $mov\ ax, 15$
  3. Add 12 to the AX — $add\ ax, 12$
  4. Store the result in a variable — $mov\ res, ax$
  5. Move the 18 into the AX — $mov\ ax, 18$
  6. Subtract 9 from AX — $sub\ ax, 9$
  7. Store the result in the BX — $mov\ bx, ax$
  8. Load the variable into the AX — $mov\ ax, res$
  9. Divide the AX by BX — $div\ bx$
  10. Store the result in a variable — $mov\ res, ax$
  11. Move 3 into the AX — $mov\ ax, 3$
  12. Increment AX — $inc\ ax$
  13. Move the result into the BX — $mov\ bx, res$
  14. Multiply the AX by the BX

Examples

- Steps:
  1. Define a variable of size WORD in the data section $res\ DW\ \ ?$
  2. Move the 15 into the AX $mov\ ax, 15$
  3. Add 12 to the AX $add\ ax, 12$
  4. Store the result in a variable $mov\ res, ax$
  5. Move the 18 into the AX $mov\ ax, 18$
  6. Subtract 9 from AX $sub\ ax, 9$
  7. Store the result in the BX $mov\ bx, ax$
  8. Load the variable into the AX $mov\ ax, res$
  9. Divide the AX by BX $div\ bx$
  10. Store the result in a variable $mov\ res, ax$
  11. Move 3 into the AX $mov\ ax, 3$
  12. Increment AX $inc\ ax$
  13. Move the result into the BX $mov\ bx, res$
  14. Multiply the AX by the BX $mul\ bx$

```
   .MODEL SMALL
   .STACK 64
   .DATA
res DW  ?
   .CODE
MAIN   PROC   FAR
   mov ax, @DATA
   mov ds, ax

   mov ax, 15
   add ax, 12
   mov res, ax
   mov ax, 18
   sub ax, 9
   mov bx, ax
   mov ax, res
   div bx
   mov res, ax
   mov ax, 3
   inc ax
   mov bx, res
   mul bx

   MOV AH,4CH
   INT 21H
MAIN ENDP
    END   MAIN
```

TASK

- What are DD, DQ, DT data types in assembly?
- Write a program that computes the sum of the following numbers:
  15H 12H 00001001b 22 8 8H
  then subtract the value 4EH, store the result in RES variable.
  - What is the final result?