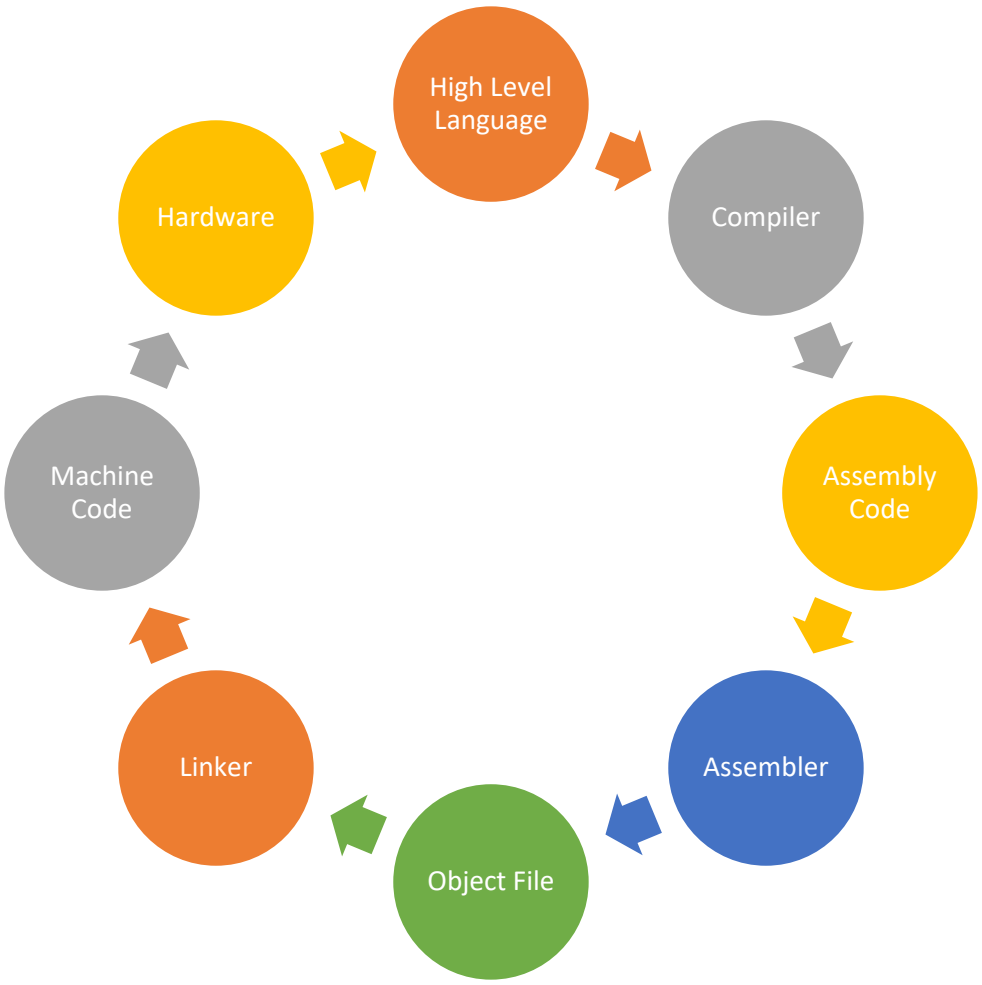
Assembly Language Simplified Notes

Compiled By: Sher Khan Baloch – Full Stack .NET Developer

Contents

Introduction

Program Cycle



What is Assembly?

1. Computer Programming Language
2. Low-Level or Close To Machine
3. Mnemonics or Keywords

Why Learn?

1. Better or deeper understanding of Software and Hardware interaction.
2. Optimization of processing time.
3. Embedded programming
4. Course requirement

Registers

What are Registers?

1. Fastest storage area or locations.
2. Quickly accessible by CPU as they are built into the CPU.

Why use Register?

1. Optimization of processing time.
2. Understanding of hardware and software interaction.

- Origin of Registers: Intel 4004 in 1971, Federico Fagin.
- Purpose: Record or collection of information.

Types of Register

There are 14 types of registers.

1. Accumulator	Input or Output Operations.	a, ax, eax, rax
2. Base	Hold the address of data.	b, bx, ebx, rbx
3. Counter	Counters are used in loops.	c, cx, ecx, rcx
4. Data	Hold data for output.	d,dx, edx, rdx
5. Code Segment	Holds the address of the code segment.	cs
6. Data Segment	Holds the address of the data segment.	ds
7. Stack Segment	Holds the address of the stack segment.	ss
8. Extra Segment	Holds the address of the data segment.	es
9. Source Index	Points the source operands.	si
10. Destination Index	Points the destination operands.	di
11. Instruction Pointer	Holds the next instruction.	ip
12. Stack Pointer	Points current top of the stack.	sp
13. Flag Register	Hold the current status of the program.	f
14. Base Pointer	Base the top of the stack.	bp

Note:

X = extended to 16 Bits

E = Extended to 32 Bits

R = Rich register to 64 Bits.

<table>
<tr><th colspan="2">CPU</th></tr>
<tr><td>General Purpose Registers</td><td>ax
ah | al

bx
Bh | bl

cx
Ch | cl

dx
dh | dl</td></tr>
<tr><td>Segment Resgiters</td><td>Cs
ds
Ss
Es</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Index Registers</td><td>Si
Di</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Special Purpose Registers</td><td>Ip
Sp</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Flag Register</td><td>F</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Base Pointer</td><td>bp</td></tr>
</table>

Addressing Modes

Ways or models to access data.

1. Registers Addressing:
   Both operands are registers.
   Example: opcode register1, register2 = ADD dl, al

2. Immediate Addressing:
   One operand is the constant term.
   Example: opcode register, value = ADD dl, 2

3. Memory Addressing:
   Access static data directly.
   Example: opcode register, [address] = ADD dl, [address]

Data Transfer Instructions

MOV dl, 'A'
MOV dl, 2

MOV ah, 2 ; 2 is the service routine which means print a single character present in dl.
Some service routines:

- 1 = Input a character with echo.
- 2 = Output or print a single character 'A'
- 8 = Input a character without echo.
- 9 = Print collection of characters or string "SHER"
- 4ch = Exit

Interrupts

Stops the current program and allows the microprocessor to access hardware to take input or give output.

- INT 21H: Interrupt for text handling.
- INT 20H: Interrupt for video or graphics handling.

Example of Input:
MOV ah, 1
INT 21H

Example of Output:
MOV ah, 2
INT 21H

ASCII Codes (American Standard Code For Information Interchange):

Characters encoding scheme by the American Standards Association (ASA) published in 1963.

0 to 9 = 48 to 57
A to Z = 65 to 90
a to z = 91 to 122

Next Line: 10

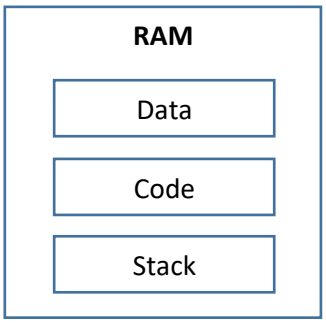
Carriage Return: 13
Space: 32

Structure and Syntax of a Program

.model small	Model Directive	Specify the total memory of the program.
.stack 100h	Stack Segment Directive	Specify storage for the stack.
.data ; Variables are defined here	Data Segment Directive	
.code Main proc ; Code Main endp End main	Code segment directive	

Model Directives:

Tiny	=	Code + Data <= 64KB
Small	=	Code <= 64 and Data <= 64KB
Medium	=	Code = any Size and Data <= 64KB
Compact	=	Code <= 64KB and Data <= any size
Large	=	Code = any Size and Data any size
Huge	=	Code = any Size and Data any size



Syntax Rules:

- Space for opcode
- One operand must be a general-purpose register.
- Operand must be of the same size.
- Comma, between operands.
- Comments must start with a semi-colon.

MASM: Microsoft Assembler = Convert assembly code to executable code.
Linker: Convert file into .exe

Program To Print Single Character

```
.model small
.stack 100h
.data

.code
main proc

    ; Storing S in a Data Register
    mov dl, 'S'

    ; Printing Output
    mov ah, 2
    int 21h

    ; Exit
    mov ah, 4ch
    int 21h

main endp
end main
```

Program To Print Name With Characters

```
.model small
.stack 100h
.data

.code
main proc

    ; Storing Character in a Data Register and Printing Each Character Every Time.
    mov dl, 'S'
    mov ah, 2
    int 21h

    mov dl, 'h'
    mov ah, 2
    int 21h

    mov dl, 'e'
    mov ah, 2
    int 21h

    mov dl, 'r'
    mov ah, 2
    int 21h

    ; Exit
    mov ah, 4ch
    int 21h

main endp
end main
```

Program To Take Input a Character From User

```
.model small
.stack 100h
.data

.code
main proc

  ; Taking Input with a Echo.
  mov ah, 1
  int 21h

  ; Moving character in data register and printing it.
  mov dl, al
  mov ah, 2
  int 21h

  ; Exit
  mov ah, 4ch
  int 21h

main endp
end main
```

Program To Add Two Numbers

```
.model small
.stack 100h
.data

.code
main proc

    mov bl, 1
    mov cl, 2

    add bl, cl

    ; Because It Will Return an ASCII Code and We Want 3 In Result That's Why We Added 48.
    add bl, 48

    mov dl, bl

    mov ah, 2
    int 21h

    mov ah, 4ch
    int 21h

main endp
end main
```

Program To Subtract Two Numbers

```
.model small
.stack 100h
.data

.code
main proc

   mov bl, 3
   mov cl, 1

   sub bl, cl

   ; Because It Will Return an ASCII Code and We Want 2 That's Why We Added 48.
   add bl, 48

   mov dl, bl

   mov ah, 2
   int 21h

   mov ah, 4ch
   int 21h


main endp
end main
```

Program To Input Two Numbers and Add Them.

```
.model small
.stack 100h
.data

.code
main proc

  mov ah, 1
  int 21h

  ; Saving First Value For Later Use.
  mov bl, al

  mov ah, 1
  int 21h

  add bl, al

  sub bl, 48

  mov dl, bl

  mov ah, 2
  int 21h

  mov ah, 4ch
  int 21h


main endp
end main
```

Program To Convert Capital Letter To Small Letter

```
.model small
.stack 100h
.data

.code
main proc

  mov ah, 1
  int 21h

  mov dl, al

  ; As Capital and Small Letters Have Different ASCII Codes That's Why We Are Adding 32 In The Value.
  add dl, 32

  mov ah, 2
  int 21h

  mov ah, 4ch
  int 21h


main endp
end main
```

Variables

Variables are defined in the **.data** directive of the program structure.

Syntax:

VariableName Data Size (Initializer Directive) Value (Initializer)

Examples:

Var1	db	49	=	ASCII Code of 1.
Var1	db	?	=	No Value, Baad Me Denge.
Var1	db	'1'	=	Direct 1 hi store ho jae. No Need for ASCII code.
Var1	db	'S'	=	Character.
Var1	db	'Sher Khan$'	=	String. $ must be used at the end of the string. $ is the terminator or end point of the string.

- Don't use reserved keywords as variable name like AL, BL, CL, DL, ADD, SUB, MUL, DIV, MOV, POP, PUSH etc.

➢ Data Size / Data Types / Initializer Directives.

DB	=	Define Byte	=	I Byte, 8 Bits.
DW	=	Define Word	=	2 Bytes, 16 Bits.
DD	=	Define Double World	=	4 Bytes, 32 Bits.
DQ	=	Define Quad Word	=	8 Bytes, 64 Bits.
DT	=	Define Ten Bytes	=	10 Bytes, 80 Bits.

Offset

Holds the beginning address of variable as 16 Bits.

LEA (Load Effective Address)

It is an indirect instruction used as a pointer in which first variable points the address of second variable.

Program To Print Two Strings On Different Lines

```
.model small
.stack 100h

.data
msg1 db ‘Sher$'
msg2 db 'Khan$'

.code
main proc

  ; Access Data Segment From Code Segment. It moves the memory location of @data into ax register.
  mov ax, @data

 ; move data address to ds so that Data Segment is initialized as heap memory to access variables fast.
 mov ds, ax


  mov dx, offset msg1        ; sending address of msg1 Variable into dx
  mov ah, 9                  ; 9 used for printing a string.
  int 21h

  mov dx, 10                 ; 10 is ASCII code of new line feed.
  mov ah, 2
  int 21h

  mov dx, 13                 ; 13 is the ASCII code of carriage return.
  mov ah, 2
  int 21h

  mov dx, offset msg2
  mov ah, 9
  int 21h

  ; Exit
  mov ah, 4ch
  int 21h

main endp
end main
```

Loop

A loop is a series of instructions that is repeated until a terminating condition is reached.

Label

A label is the name of a series of instructions.

1. A label can be placed at the beginning of a statement because the label is assigned the current value of the line.
2. Label name must not be a reserved keyword e.g. MOV, ADD, DB etc.
3. Colon : must be used with label while initializing, but not while calling.

Program To Print From 0 to 9 Using Loop

```
.model small
.stack 100h
.data

.code
main proc

   mov cx, 10          ; Loop will run 10 times. cx register is used in loops.

   mov dx, 48          ; Print from 0 so 48 is the ASCII code of 0

   L1:                 ; Label Start
   mov ah, 2
   int 21h

   ; add dx, 1
   inc dx              ; Increment of 1 in dx value.

   Loop L1             ; loop calling label. Means calling instructions written in label.

   ; Exit
   mov ah, 4ch
   int 21h

main endp
end main
```

Program To Print Capital Letters From A To Z Using Loop

```
.model small
.stack 100h

.data

.code
main proc

   mov cx, 26              ; Loop will run 26 times.

   mov dx, 65              ; 65 is the ASCII code of A.

   L1:                     ; Label start
   mov ah, 2
   int 21h

   inc dx                  ; Increment of 1

   Loop L1                 ; Loop calling instructions written in label.

   ; Exit
   mov ah, 4ch
   int 21h

main endp
end main
```

Flag Registers

Flag register is a register that contains the current state of the processor.

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Why do we study flag register?

Theory:

- It controls the operations of CPU.
- It handles the status of operation.

Programming:

- Conditional Jump
- Which number is lesser, greater or equal.

Status Flags:

To handle the result of an operation.

1. **Carry Flag (CF):**
   1 = When there is last carry out
   0 = When there is no last carry out

2. **Parity Flag (PF):**
   1 = When there is even number of bits.
   0 = When there is not even number of bits.

3. **Auxiliary Flag (AF):**
   1 = When 3rd bit carry exists.
   0 = When 3rd bit carry does not exists.

4. **Zero Flag (ZF):**
   1 = When result is zero.
   0 = When result is not zero.

5. **Sign Flag (SF):**
   1 = When result is negative.
   0 = When result is positive.

Controls Flags:

To controls the operation of CPU.

6. **Trap Flag (TF):**
   System use it when debugging is required.
   1 = When single step mode (debugging) is needed.
   0 = When single step mode (debugging) is not needed.

7. **Interrupt Flag (IF):**
   1 = When interrupt is called.
   0 = When interrupt is not called.

8. **Direction Flag (DF):**
   1 = Strings automatically decrements the address.
   0 = Strings does not automatically decrement the address.

9. **Overflow Flag (OF):**
   1 = When result is too big to fit in the destination.
   0 = When result is not too big to fit in the destination.

Jumps

Jump is an instruction to control the program flow.

Unconditional Jump

Jump to label without any condition.
Syntax: JMP LabelName

Example:

```
L1:
Mov dl, 'S'
Mov ah, 2
INT 21h

JMP L1
```

Conditional Jump

Jump to label when condition occur.
Syntax: Opcode LabelName

Example:

```
L1:
Mov ah, 1
INT 21h

Mov dl, 3

CMP al, dl

JE L1           ; Jump If ZF = 1

Mov ah, 4ch
INT 21h
```

More Conditional Jumps:

JE	Jump If Equal	**JLE**	Jump If Less or Equal
JZ	Jump If Zero	**JBE**	Jump If Below or Equal
JNE	Jump If Not Equal	**JG**	Jump If Greater
JNZ	Jump If Not Zero	**JA**	Jump If Above
JL	Jump If Less	**JGE**	Jump If Greater or Equal
JB	Jump If Below	**JAE**	Jump If Above or Equal

Compare

Subtract Operand1 from Operand2, but does not store the results. Only changes the flags registers.

Syntax:

CMP register1, register2	CMP dl, al
CMP Register1, Constant	CMP dl, '3'
CMP register1, [Memory Address]	CMP dl, [si]

Program To Print The Input Number Is Equal or Not

```
.model small
.stack 100h

.data
   msg1 db 'Number Is Equal$'
   msg2 db 'Number Is Not Equal$'

.code
main proc

   mov ax, @data
   mov ds, ax

   mov dl, '3'         ; Storing Exact 3

   mov ah, 1           ; Taking Input
   int 21h

   cmp al, dl          ; Compare Values
   je l1               ; Jump To Label 'L1' If Equal. This Line Will Not Execute If Compare Respond False.

   mov dx, offset msg2  ; Printing Else Message
   mov ah, 9
   int 21h

   mov ah, 4ch         ; Exit From Here So Below Code Does Not Execute.
   int 21h

   l1:
   mov dx, offset msg1
   mov ah, 9
   int 21h

   ; Exit
   mov ah, 4ch
   int 21h


main endp
end main
```

Arrays

Collection of characters in sequence.

Source Index (SI)

Source Index (SI) is a register used as pointer to access array.

Why do we need to learn Array?

To store many characters with single variable name in sequence in memory.

Where to initialize?

Array is defined in **.data** directive of program as variable.

How to Initialize?

In same way as variable but with multiple values.

```
Arr1    db      1,2,3,4
Arr1    db      ‘a’,’b’,’c’
Arr1    db      ‘abc’
Arr1    db      ‘a,’a’,’a’
Arr1    db      ?,?,?

Arr1    db      3 dup(‘a’)      ; Duplicate this value three times
Arr1    db      3 dup(?)
```

Example:

```
.model small
.stack 100h

.data
Arr1    db      1,2,3,4

.code
Main proc
Mov ax, @data
Mov ds, ax

Mov si, offset arr1     ; Starting address, Address of first character.

Mov dx, [si]            ; Bracket form to access value at address.
Mov ah, 2
Int 21h

; Mov dx, [si + 1]
Inc si
Mov dx, [si]
Mov ah, 2
Int 21h

Main endp
End main
```

Program To Print An Array Using Loop

```
.model small
.stack 100h

.data
arr db 'S', 'H', 'E', 'R'

.code
main proc

    mov ax, @data
    mov ds, ax

    mov si, offset arr

    mov cx, 4

    l1:
    mov dx, [si]
    mov ah, 2
    int 21h

    inc si

    loop l1

    mov ah, 4ch
    int 21h

main endp
end main
```

Program To Take Input String and Print It

```
.model small
.stack 100h

.data
var1 db 100 dup('$')

.code
main proc
   mov ax, @data
   mov ds, ax

   mov si, offset var1

   l1:
   mov ah, 1            ; Taking Input
   int 21h
   cmp al, 13           ; Compare If User Press Enter
   je printString       ; Then Go To Print String Label
   mov [si], al
   inc si
   jmp l1

   printString:
   mov dx, offset var1
   mov ah, 9
   int 21h
   mov ah,4ch
   int 21h
main endp
end main
```

Stack (Push, Pop)

A stack is a data structure that works on LIFO principle.
Always use 16 Bits or greater than 16 Bits registers.

Last In, First Out (LIFO)
PUSH = To Add
POP = To Remove

What is the use of Stack in Computer Science?

1. Undo/Redo
2. Back/Forward
3. Solving mathematical problems with Precedence

Why do we study Stack in Assembly Language?

1. Swap Two Numbers
2. To Reverse a String
3. Helps in Nested Loops (Loop Within Loop)

How to use stack in assembly program?
.stack 100h ; a directive/command reserves 100h bytes for Stack.

Stack Segment Register
Hold address of space
Reserved for stack
SS : SP
Stack Pointer Register
Point the top of space
Reserved for stack

Syntax
PUSH Register/Variable
Copies content from Operand to Top of Stack.
PUSH AX
PUSH Var1

POP Register/Variable
Copies content from Top of Stack to Operand.
POP AX
POP Var1

Example:

```
.model small
.stack 100h

.data

.code
main proc

Mov AX, 2
PUSH AX         ; Add Item To Stack

POP AX          ; Remove Item From Stack

Mov DX, AX      ; Printing That Removed Element
Mov ah, 2
Int 21h

Mov AH, 4CH
INT 21H

main endp
End main
```

Program To Swap Two Numbers

```
.model small
.stack 100h
.data

.code
main proc

    mov ax, '3'
    push ax       ; Send 3 To Stack

    mov bx, '7'
    push bx       ; Send 7 To Stack

    pop ax        ; Move 7 From Stack To ax
    pop bx        ; Move 3 From Stack To bx

    mov dx, ax    ; Sending Value In dx For Printing
    mov ah, 2
    int 21h

    mov dx, bx
    mov ah, 2
    int 21h

    mov ah, 4ch   ; Exit
    int 21h

main endp
end main
```

Program To Reverse a String

```
.model small
.stack 100h
.data

string db 'Sher Khan$'

.code
main proc

    mov ax, @data
    mov ds, ax

    mov si, offset string
    mov cx, 9           ; Loop Will Run 9 Time Because We Have 09 Characters In String

    l1:
    mov bx, [si]        ; Send Value of SI In Stack
    push bx

    inc si

    loop l1

    mov cx, 9           ; Loop Will Run 9 Time Because We Have 09 Characters In String

    l2:
    pop dx              ; Sending Top Value In dx So We Can Print It.
    mov ah, 2
    int 21h

    loop l2

    mov ah, 4ch         ; Exit
    int 21h

main endp
end main
```

Nested Loop

Loop Within Loop

Why Do We Need It?

- Reduce Complexity
- Maintained Program

How To Use Nested Loop?

- With The Help of PUSH and POP

Example:

```
Mov cx, 4

; Main Loop
L1:
push cx

mov cx, 3

; Nested Loop
L2:

Loop L2
; End of Nested Loop

Pop cx

Loop L1
; End of Main Loop
```

Program To Print Pyramid

```
.model small
.stack 100h
.data
.code
main proc
mov ax,@data
mov ds,ax

mov bx, 1

mov cx, 5
L1:
push cx
mov cx, bx

L2:
Mov dl, '*'
mov ah,2
int 21h
loop L2


mov dl,10
mov ah, 2
int 21h
mov dl,13
mov ah, 2
int 21h
inc bl

pop cx
loop L1

mov ah,4ch
int 21h
main endp
end main
```

Procedure

What is the procedure?
It is just a block of code that can be called anywhere in the program with name.
It is without parameters so we cannot pass the parameters in procedure.

Why do we need it?

- Code Reusability
- Reduce complexity

How to use procedure?
procName **PROC**

; Code

RET
procName **ENDP**

CALL procName

- Write after the one-procedure ends and before the end main key word.

Example:

```
.model small
.stack 100h
.data
str1 db 'Hello, I am$'
str2 db 'Sher Khan Baloch$'
str3 db 'Full Stack Developer$'
.code
main proc
  mov ax, @data
  mov ds, ax

  mov dx, offset str1
  mov ah, 9
  int 21h

  call newLine     ; Calling Procedure

  mov dx, offset str2
  mov ah, 9
  int 21h

  call newLine

  mov dx, offset str3
  mov ah, 9
  int 21h

  mov ah, 4ch
  int 21h

main endp

newLine proc              ; New Procedure For New Line

   mov dx, 10  ; For New Line
   mov ah, 2
   int 21h

   mov dx, 13  ; For Carraige Return
   mov ah, 2
   int 21h

   ret
newLine endp

end main
```

Macro

What is the Macro?
It is just a block of code that can be used with input parameters anywhere in the program with name.
It is a perfect function.
Write on the Top of a program. Outside model size.

Why do we need it?

- Code Reusability with input parameters
- Reduce complexity

How to use Macro?
macroName **MACRO P1, P2, ……**

; Code

ENDM

macroName P1, P2, …….

What is the Difference between Procedure and Macro?

Procedure	Macro
No Input Parameters.	Input parameters.
Ret is used.	No ‘ret’ is used.
Slow, goes and run code.	Fast, replace with code.

Example:

```
printLine macro p1
   mov dx, offset p1
   mov ah, 9
   int 21h
endm

.model small
.stack 100h
.data
str1 db 'Hello, I am$'
str2 db 'Sher Khan Baloch$'
str3 db 'Full Stack Developer$'

.code
main proc
  mov ax, @data
  mov ds, ax

  printLine str1   ; Calling Macro
  call newLine     ; Calling Procedure

  printLine str2
  call newLine

  printLine str3

  mov ah, 4ch
  int 21h

main endp

newLine proc

   mov dx, 10  ; For New Line
   mov ah, 2
   int 21h

   mov dx, 13  ; For Carraige Return
   mov ah, 2
   int 21h

   ret
newLine endp

end main
```

Program To Divide Two Numbers, Print Quotient And Remainder

- Dividend Will Be In **AX** and Divisor Will Be In **BL**, **CL**, **DL**
- Quotient Will Be In Stored In **AL** and Remainder Will Be Stored In **AH**.

```
.model small
.stack 100h

.data
quotient db ?
remainder db ?

.code
main proc

   mov ax, 26              ; Dividend
   mov bl, 5               ; Divisor

   div bl                  ; This Will Get Value From AX And Then Divide It.

   mov quotient, al        ; Moving Quotient From AL In a Variable
   mov remainder, ah       ; Moving Remainder From AH In a Variable.

   mov dl, quotient
   add dl, 48              ; Mainting ASCII Codes
   mov ah, 2
   int 21h

   mov dl, remainder
   add dl, 48              ; Mainting ASCII Codes
   mov ah, 2
   int 21h

   mov ah, 4ch             ; Exit
   int 21h

main endp
end main
```

Program To Multiply Two Numbers And Print The Product.

- Multiplicand Will Be In AX and Multiplier Will Be In BX, CX, DX
- Product (Answer) Will Be Stored In AX.
- If Number Is More Than One Then It Will Be Stored In AH and AL.

```
.model small
.stack 100h

.data

.code
main proc

   mov ax, 5      ; Multiplicand
   mov bl, 2      ; Multiplier

   mul bl         ; This Will Get Value From AX And Then Multiply It.

   aam            ; This Will Break AX Value In AH and AL. Use When Answer Is More Than 1 Digit

   mov ch, ah
   mov cl, al

   mov dl, ch
   add dl, 48
   mov ah, 2
   int 21h

   mov dl, cl
   add dl, 48
   mov ah, 2
   int 21h

   mov ah, 4ch       ; Exit
   int 21h

main endp
end main
```