LAB

PROGRAM TO PRINT NAME WITH CHARACTER

```asm
.model small
.stack 100h
.data
.code
main proc
mov dl, 'A'
mov ah,2
INT 21h
```

```asm
mov dl, 'L'
mov ah,2
INT 21h

mov dl, 'I'
mov ah,2
INT 21h

mov ah, 4ch
INT 21h
main endp

end main
```

1). Program to print a name with character.

Every letter is single character.

→ Acc. Reg is used b/c <u>Print / Input</u>

→ 'A' moved to dl (data Reg), Acc Print it.

→ Same for every character.

→ mov ah, 4ch] to exit from register.
      int 21h

2) Program to ~~print~~ take input character and print on screen.

For taking Input Acc. Reg is used (mov ah,1).
If input is in accumulator it is place in (al).
Send it to data register for print, b/c for
print the output the value present in data register,
Acc Reg. Print it.

PROGRAM TO TAKE INPUT CHARACTER AND PRINT ON SCREEN

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

main proc

mov ah,1
INT 21h

mov dl, al
mov ah,2
INT 21h

mov ah,4ch
INT 21h

main endp

end main
```

PROGRAM TO ADD TWO NUMBERS

3). Program to ADD 2 numbers.

Key word in assembly for addition is add

① mov bl, 1 ⎤ Register Addressing Mode (Both General Purpose
   mov cl, 2 . ⎦ Register).

Add bl, cl.
mov dl, bl ⟶
add dl, 48 ⟶ OR.

mov bl, 1
Add bl, 2

Immediate Addressing Mode
In both method First operand
is Dest & Second is source.

In Both cases our sum
Present in bl register
We have to send it to
dl.

⟶ Sum of 2 and 1 is 3, Want to print 3 ASCII of
3 is 51, add 48 to this number, the required
number we get.

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

main proc

mov bl,2
mov dl,1
add dl,bl
add dl,48
mov ah,2
INT 21h


mov ah,4ch
INT 21h

main endp
```

PROGRAM TO SUBTRACT TWO NUMBERS

① Program to Subtract Two Numbers.

```
main proc
mov bl, 3
mov cl, 1              ]
Sub bl, cl
add bl, 48
mov dl, bl.
mov ah, 2
INT 21h
mov ah, 4ch
INT 21h
main endp.
end main
```

Seperately send both numbers to register & subtract (sub).

For print of 2 we need ASCII code of 2.

OR
```
mov bl, 3
Sub bl, 1
```

PROGRAM TO INPUT TWO NUMBERS AND ADD THEM

⑤ Program to Input two Numbers and Add them.

```
main proc.
mov ah, 1
INT 21h.
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
```

→ there are 2 input values, Input is placed in al, for second input it again placed in al, (first value is overwrite) so we placed first value in some othe place (mov bl, al)

→ When user give input 1 (Ascii 49) is given, second input 2 (Ascii 50), addition should be 3 (Ascii 51), but addition answer is 99. I/P is always as ASCII code, to solve the problem 48 is Subtracted. When given input, input is in ASCII code, the two Ascii values are change then original, minus 48 to get original ASCII code.

THEORY

Variable Initialization In Assembly Program

- Variables are defined in .data directive of Program structure.

dosseg ⟶ Optional (9t arrange segments if any segment or directie not in place it rearrange it).

- model small ⟶ Size of Assembly Prog.
- stack 100h ⟶ Size of stack if use stack mngment, part of Prog. structure.
- data ⟶ We define variables

- Space b/w .data & .code we define variables.

Variable Initialization

How to define variables ?

Other Prog. Lang's   Datatype   Variable Name   Value
$\qquad\qquad\qquad\qquad$ 1 $\qquad\qquad$ 2 $\qquad\qquad$ 3

Here the change is

VariableName   Datatype   Value

$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ └─────→ Data size (Initializer Directive)

In assembly we called datasize.,

┌─────────────────────────────────────┐
│ VariableName   Datasize   Value   │
└─────────────────────────────────────┘

$\qquad\qquad\qquad\qquad\qquad\qquad$ └────→ Initializer

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ └────→ Initializer Directive
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ │
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ └─────

In terms of Assembly lang.

- Variable name should not be reserved keywords.
  (AL, BL, CL, DL, Sub, Add, Div, Mul, Mov, Pop, Push)
- Don't used reserved keywords as Variable Name.
- Data size (Initializer Directive). ——→ Size of Value is given

DB	Define Byte	1 byte, 8bits
DW	Define word	2 bytes, 16bits.
DD	Define Double word	4 bytes, 32bits.
DQ	Define Quadword	8 bytes, 64bits.
DT	Define TenBytes	10bytes, 80bits.

e.g.   Var1 db 49
                        └──→ Ascii code 1, Give Ascii code
                                              in Variable

- **If don't want to give value.**

  Var1 db ? → Initialize in .code,
  or take input from reg, or
  direct value or take input
  from Acc.Reg.

- **If not remember ASCII code.**

  Var1 db '1'
  Var1 db 'A'

- **For string** Var1 db '123456$' → For Number
  Var2 db 'hello world$' → For letters

  $ = String Terminate.

```
 1    2    3    4    5    $
 |    |    |    |    |    |
 1B   1B   1B   1B   1B   1B
```

- In RAM there are frames, k value / number is placed in different frames. $ sign indicates, where string is completed.

- $ must be used in end of string

- $ = terminator, end point of string.

__To implement variable in program, create Program structure__

. data
    Var 1 db '1'
    Var 2 db ?

Var 3 db '123$'

- We defined three variables in .data.
- In RAM part of data and code segment, Variables go in Data Segment. If any variable I want to access from data to code, need address of that. (In code part I need address of data we need to used).

• So we write in .code first

Mov ax, @data ⟶ @data (data directive)

It moves the memory location of @data into the AX register (16 bit Register).

Now I have address of data, I want ⑦
to access any variable direct. (Want to Access Var3
don't want to go through Var1, Var2)

Need heap memory (is a memory in which we take
any variable, data directly / randomly, a fast memory).
We write.

Mov ds, ax ⟶ Move data address to
ds, so that data
segment get initialized
as heap memory to
access variables fast.

⑧

So, We write these two instructions,

```
Mov ax, @data
Mov ds, ax
```
→ these must be write to directly access variable.

First we Access Variable 1 :-

```
. code
main proc
Mov ax, @data
Mov ds, ax.
Mov dl, Var1
Mov ah, 2
int 21h.
```

→ Here write dl, (8 bits) and Var1 is db (8 bits) we can't write dx (16 bits) typemismath
Must be careful about size

Access Variable 2

Mov Var2, bl ⟶ (Value present in bl goes to Var2) OR.

We can give direct value. (Var2, 3).

Access Variable 3

Variable 3 is a string.

∴ Var3 db '12345$'   how to Access a String = ?.

If we write

~~Mov dl, Var3~~ X   Var3 go to RAM, It Pick first digit and send to dl, (we used 8 bit of 1 digit).

↗

This instruction not used for string.

To print Variable 3 we used offset.

> Mov dx, offset Var3

⮡ Offset gives us starting address of string & through that address I access all characters of string and finish till $.

Offset address is 16 bits, so we used dx

Offset → Holds the beginning address of Variable as 16 bits

- Now we can point it with service routine 9. 🙂 ⑪
- If don't want to used offset.

`lea dx, Var 3` → Load **Effective** Address.

points it address.

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

`mov dx, offset Var 3`

⤷ Due to offset, address of Variable 3 goes to dx.

Both cases are acceptable and can be used to Access string.

To print.

Mov ah, 9 ⟶ Service routine = 9 to print
int 21h   string.

LAB

• Here to define 2 strings and point on different lines
  Like [Start]
       [End].

• First we define 2 Variables . (msg1 and msg2).

• In .code we write,
    mov ax, @data ⟶ To access data segment
    mov ds, ax .   directive, send address of
                            data address to ax,
                            then send address in
                            ax to data segment.
                    In this way heap memory is
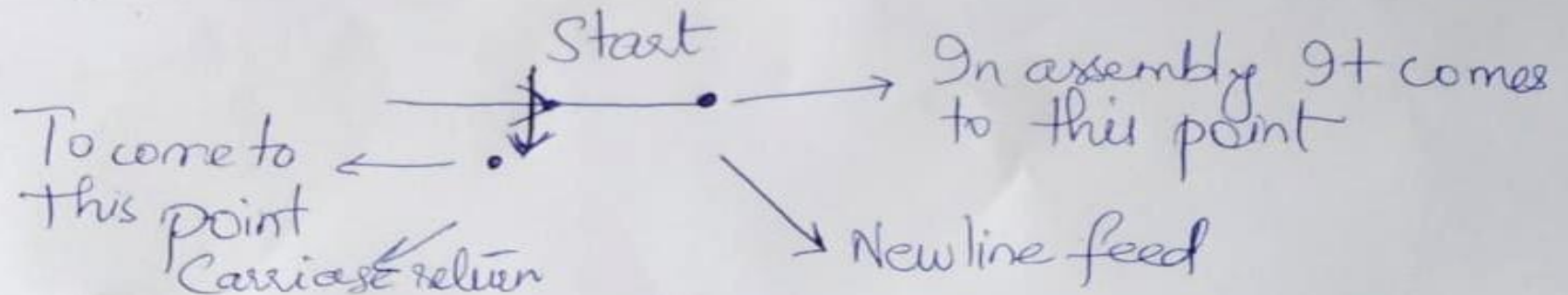   initialized to quickly
                    access memory.

- To print the msg 1 ~~message~~

$$\begin{bmatrix} \text{mov dx, offset msg1} \\ \text{OR} \\ \text{lea dx, msg1} \end{bmatrix} \longrightarrow$$ offset, address of msg 1 send to dx.

mov ah, 9
int 21h.

In this way first way first ~~message~~ string is printed.

To move to next line, like

Start

To come to this point

Carriage return

In assembly it comes to this point

New line feed

- We need two characters to print.
- One for newline and other is for carriage return.

$$\left.\begin{array}{l} \text{newline feed} : 10. \\ \text{carriage return} : 13 \end{array}\right\}$$

We print these two after first string.

Program to print 2 strings on two different lines, (linefeed, Carriage Return).

- ~~data~~
~~msg~~

```
        dosseg
      .model small
    • stack 100h

    • data
  ➤ msg1 db 'Start $'
  • msg2 db 'End $'

    • code
  main proc
  mov ax, @data
  mov ds, ax
```

```
mov dx, offset msg1
mov ah, 9
int 21h

Mov dx, 10
mov ah, 2
int 21h

Mov dx, 13
mov ah, 2
int 21h.
```

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

mov ah, 4ch
int 21h
```

~~mov ah, 4ch~~
~~int 21h~~

```
main endp
end main
```