THEORY

Variable Initialization In Assembly Program

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

dosseg $\longrightarrow$ Optional (It arrange segments if any segment or directive not in place it rearrange it).

.model small $\longrightarrow$ Size of Assembly Prog.

.stack 100h $\longrightarrow$ Size of stack if use stack mngment, part of Prog. structure.

.data $\longrightarrow$ 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
1 2 3

Here the change is

VariableName Datatype Value

Datatype $\rightarrow$ Data size (Initializer Directive)

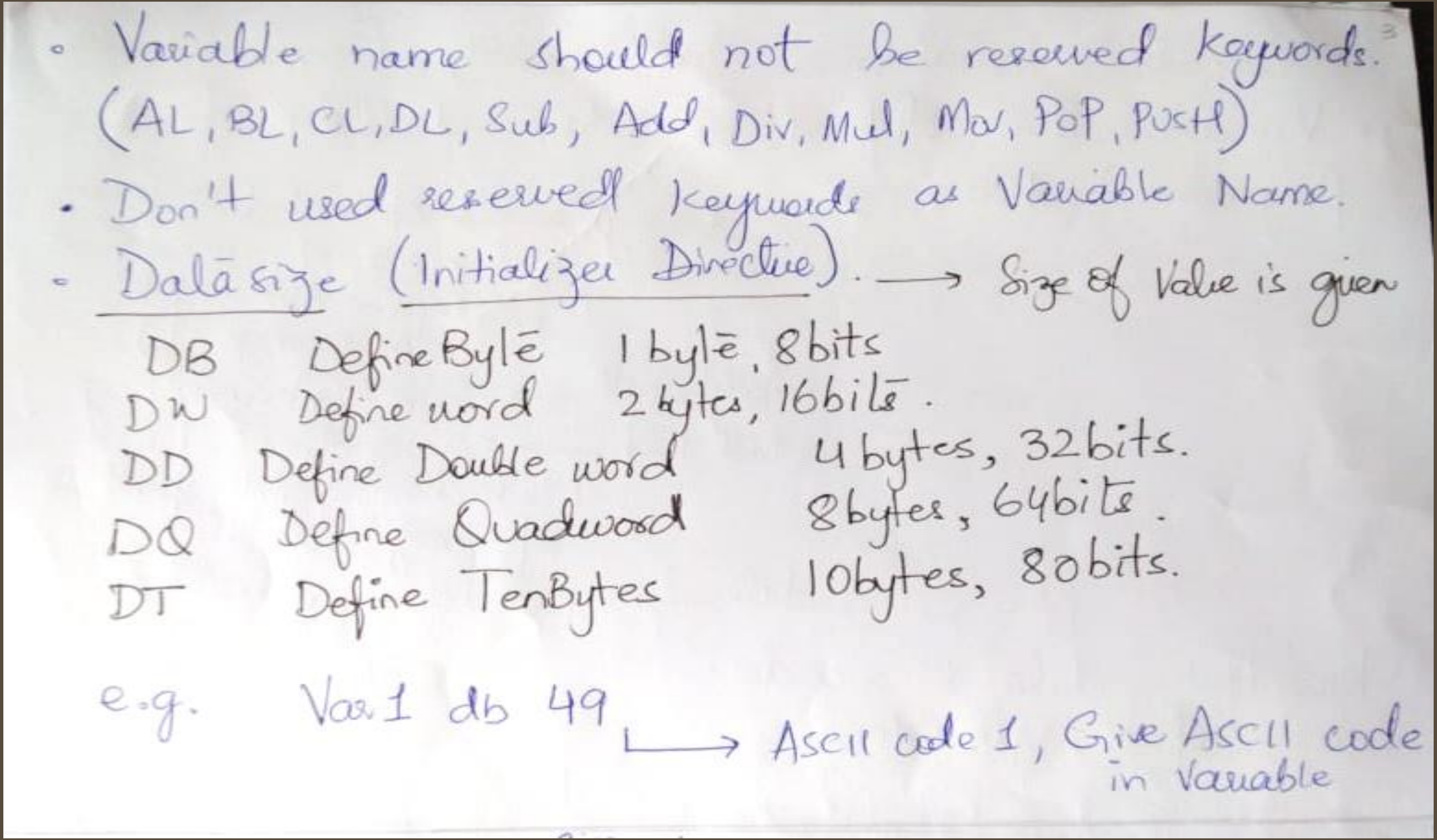
In assembly we called datasize.,

$\boxed{\text{VariableName Datasize Value}}$

Value $\rightarrow$ Initializer

Datasize $\rightarrow$ Initializer Directive

Initializer Directive $\rightarrow$ 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). $\longrightarrow$ 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	8bytes, 64bits.
DT	Define TenBytes	10bytes, 80bits.

e.g. Var1 db 49 $\longrightarrow$ ASCII code 1, Give ASCII code in Variable

- If don't want to give vale.

Var1 db ? $\longrightarrow$ 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$'      ——→ For Numbe
Var2 db 'hello world$'  ——→ For letters
```

\$ = String Terminate.

1	2	3	4	5	$
1B	1B	1B	1B	1B	1B

- In RAm there are framer, & value/number is placed in different frames. $ sign indicates, where string is completed.
- $ must be used in end of string.
- $ = terminator, endpoint of string.

<u>To implement variable in program, create program structure</u>

```
.data
Var1 db '1'
Var2 db '?'
```

Var3 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 $\longrightarrow$ @data (data directive)

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

Now I have address of data, I want 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 $\longrightarrow$ Move data address to ds, so that data segment get initialized as heap memory to access variables fast.

So, We write these two instructions,

$$\boxed{\begin{array}{l}\text{mov ax, @data} \\ \text{mov ds, ax}\end{array}} \longrightarrow \text{these must be write to directly access variable.}$$

First We Access Variable 1 :-

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

$\longrightarrow$ Here write dl, (8 bits) and Var1 is db (8 bits) we can't write dx (16 bits) typemismatch

Must be careful about size

Access Variable 2

Mov Var2, bl $\longrightarrow$ (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 8bit of 1 digit).

This instruction not used for string.

To print variable 3 we used offset.

$\boxed{\text{Mov dx, offset Var3}}$

→ Offset gives us starting address of string & through that address 9 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.
- If don't want to used offset.

$\boxed{\text{lea dx, Var 3}}$ $\longrightarrow$ 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."

$\boxed{\text{Mov dx, offset Var 3}}$

$\hookrightarrow$ Due to offset, address of Variable 3 goes to dx.

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

<u>To print</u>.

$$\begin{array}{l}\text{Mov ah, 9} \\ \text{int 21h}\end{array} \longrightarrow \text{Service routine = 9 to print string.}$$

LAB

- Here to define 2 strings and print on different lines
  Like $\begin{bmatrix}\text{Start}\\ \text{End}\end{bmatrix}$.
- First we define 2 variables. (msg1 and msg2).
- In .code we write,

```
mov ax, @data
mov ds, ax.
```

$\longrightarrow$ To access data segment 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 msg1 ~~[illegible]~~

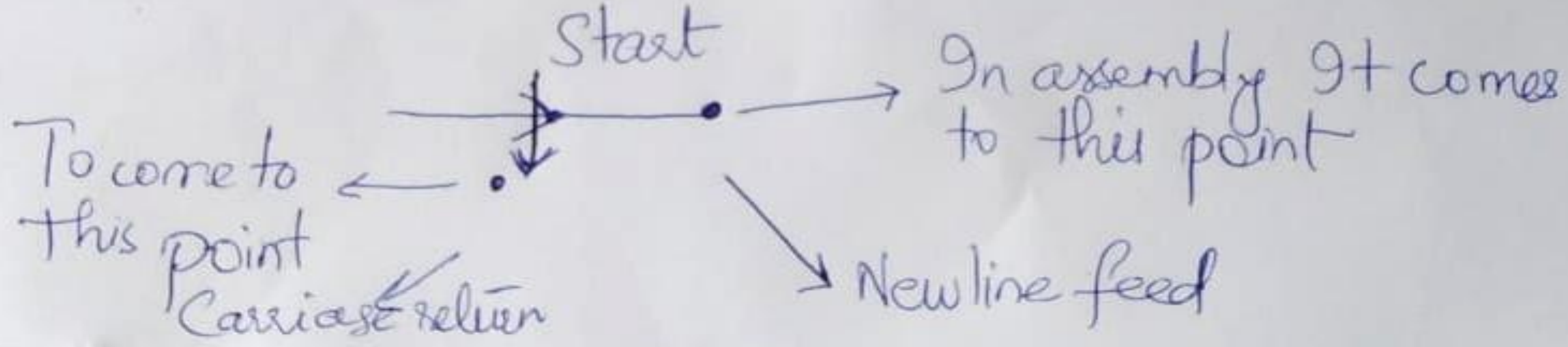
```
[ mov dx, offset msg1  ]
          OR
[ lea dx, msg1         ]
```

$\longrightarrow$ offset, address of msg1 send to dx.

```
mov ah, 9
int 21h.
```

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

To move to next line, like



- 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~~

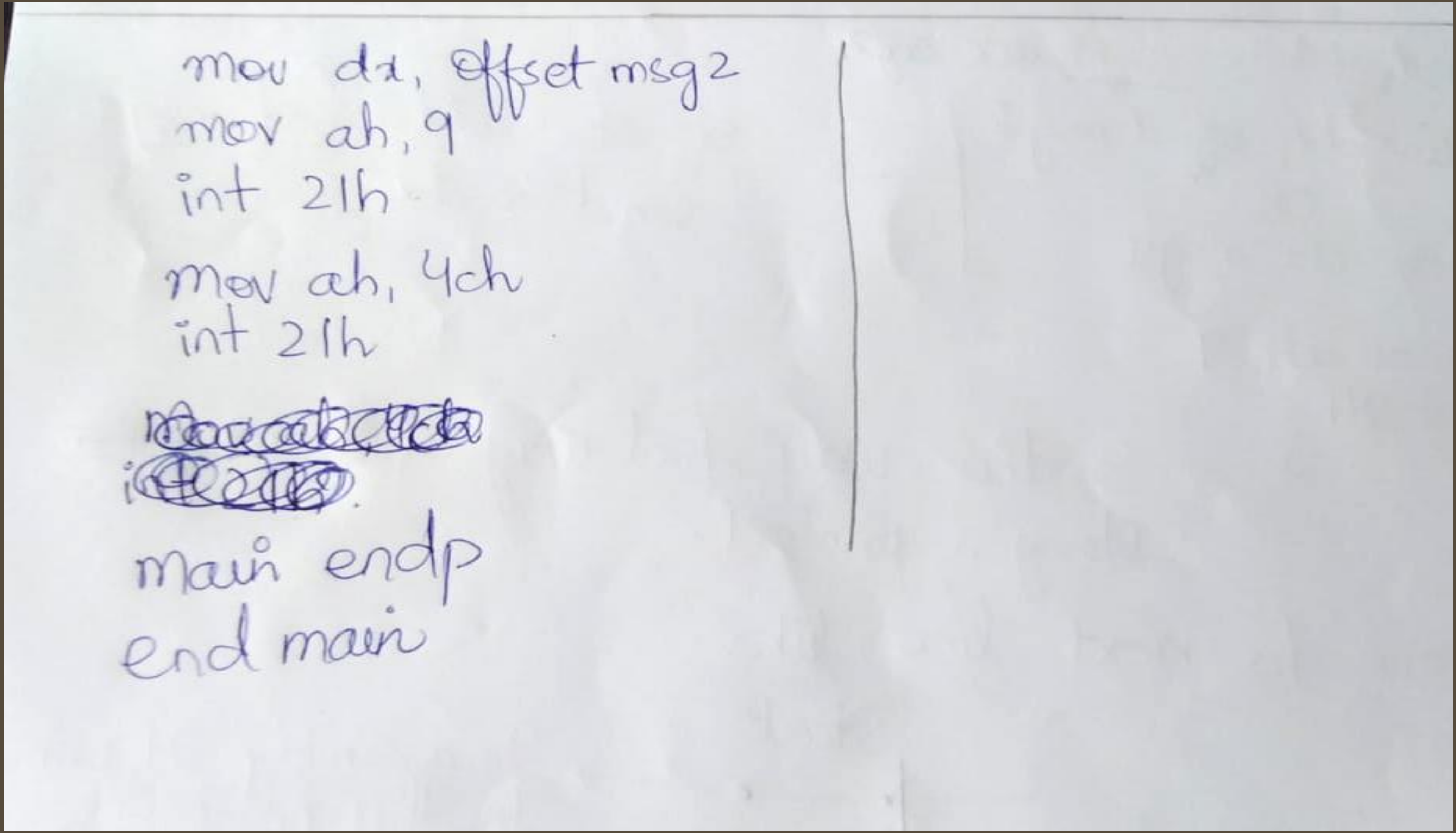
```
    dosseg
   .model small
  .stack 100h

  .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

[illegible]
[illegible]

main endp
end main
```