REMARKS

1. Memory Model Directive

.model

specifies the memory model your program uses, which affects code and data segment sizes and how you access memory.

Syntax

.model <model>

→ Common models

- tiny – code + data fit in one segment (64 KB max)
- Small – one code segment, one data segment (both max 64KB)
- medium – multiple code segments, one data segment
- large – multiple code and data segments
- flat – for 32-bit or 64-bit flat memory models (like windows)

eg:

```
.model small
```

2. Segment Directives

segments divide the pgm's memory into logical parts

* .code – marks the start of the code (executable instructions).
* .data – marks initialized data (Variables with known initial values).
* .data? – marks uninitialized data (variables reserved but not set)
* .Stack – defines the stack segment (for function calls, local vars).
* .Const – defines read-only constants

eg:

```
.data
var1 DW 1234h       ; initialized word

.data?
buffer DB 100 DUP(?) ; uninitialized 100-byte buffer
.stack 100h          ; 256 byte stack
```

```
.code
main PROC
  ,code here
main ENDP
```

3. Data Definition Directives

These define variables or constants in memory with specific sizes.

Directive	Size (bytes)	purpose	Example
DB	1	Define Byte	name DB 'A'
DW	2	Define word (16-bit)	value DW 1234h
DD	4	Define doubleword (32-bit)	ptr DD 0x12345678
DQ	8	Define Quadword (64-bit)	quad DQ 123456789ABCDEF0h
DT	10	Define Ten bytes (80-bit float)	flt DT ?

Special use.

- DUP operator repeats values, useful for arrays or buffers

Eg:

```
msg DB 'Hello, world!',0      ; zero terminated string
array DW 10 DUP(0)            ; array of 10 words, all zero-initialized
```

4. Constants and Equates

constants let you assign symbolic names to fixed values

- EQU directive assigns a constant to a symbol

```
MAXLEN EQU 100
BUFFER_SIZE EQU 256
```

- can also use = as shorthand in MASM

eg

```
PI EQU 3.14159
MAX_COUNT = 50
```

note. EQU defines constants at assembly time, not runtime Variables.

5. Procedures and code Organization

PROC / ENDP

defines a procedure (function or subroutine)

6. Assembly control directives

used to control assembly behaviour

Directive	Description	example
INCLUDE	Includes contents of another file	INCLUDE macros.inc
IF, ELSE, ENDIF	conditional assembly	IF DEBUG ··· ENDIF
MACRO, ENDM	Define a macro	PRINT MACRO msg ··· ENDM
TITLE	sets the title of the listing file	TITLE My Program
NAME	sets the module name	NAME mymodule

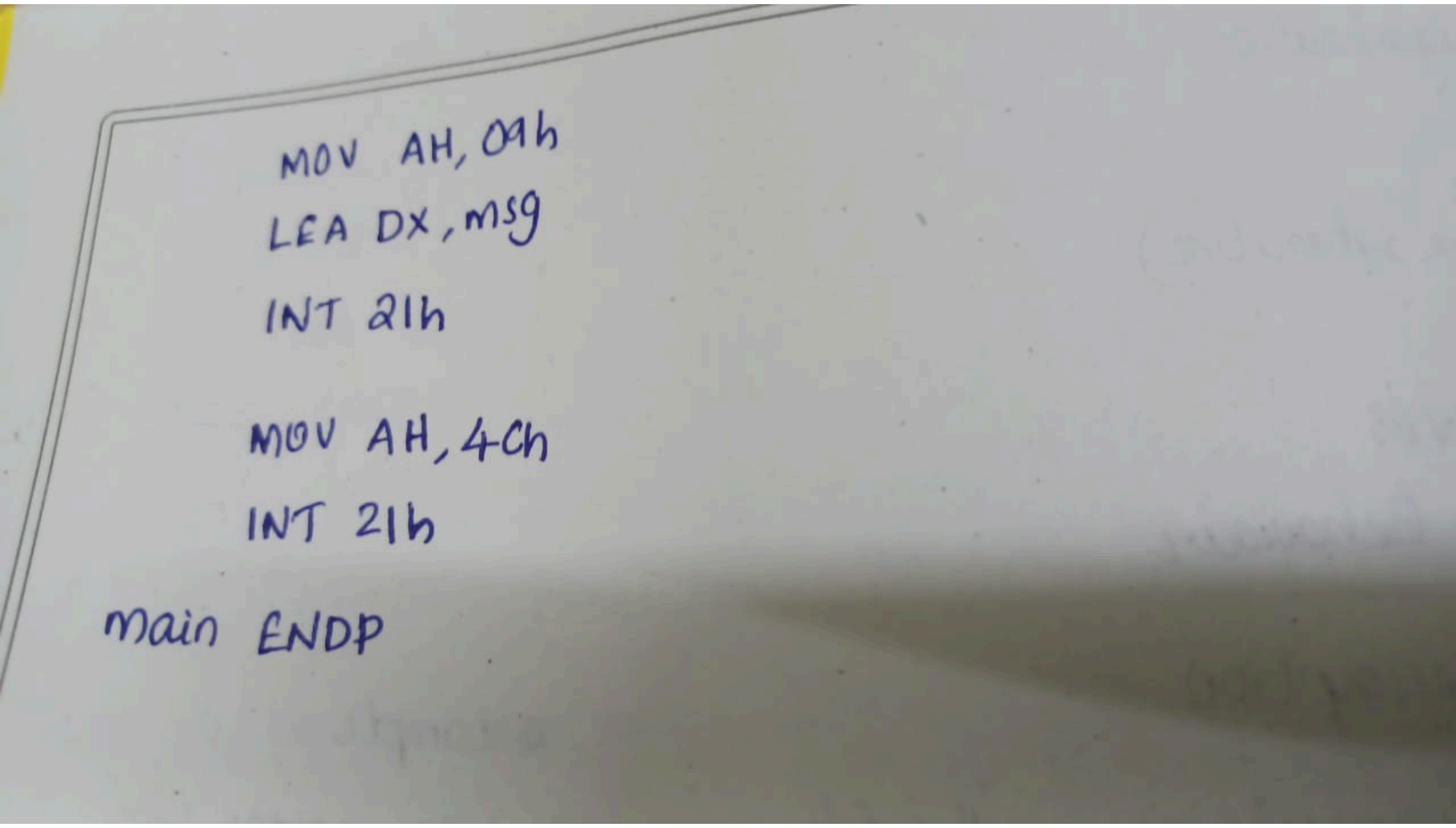
7. Segment Management

(used in older MASM versions)

Directive	Description	Example
SEGMENT, ENDS	Defines custom segment	mydata SEGMENT ... mydata ENDS
ASSUME	Informs assembler abt reg - segment pairing	ASSUME CS: code, DS: data

Sample pgm

```
.MODEL SMALL
.STACK 100h
.DATA
msg DB 'Hello, MASM!', '$'
.CODE
main PROC
    MOV AX, @DATA
    MOV DS , AX
```



```
    MOV AH, 09h
    LEA DX, msg
    INT 21h

    MOV AH, 4Ch
    INT 21h
main ENDP
```

~~Familiario~~

Addressing modes

- It specifies the way in which the operand of an instruction is accessed.
- There are 8 addressing modes:

1. Immediate addressing mode. - operand is specified in the instruction itself. No memory access is needed.

   Eg: MOV AL, 25H ; Load 25H directly into AL

2. Register addressing mode:- operand is stored in a register Fast execution.

   Eg:- MOV AX,BX ; Copy contents of BX to AX

3. Direct addressing mode - the effective address of the operand is given directly in the instruction. Accesses memory

   Eg:- MOV AL, [1234H] ; Load byte from memory address 1234H to AL

4. Register indirect addressing mode - address of the operand is stored in a register. SI, DI, BX or BP

   Eg: MOV AL, [BX] ; load byte from memory pointed to BX into AL

5. Based addressing mode - effective address = contents of base register (BX or BP) + displacement.

   Eg:- MOV AL, [BX + 04H] ; load from address BX + 04H

6. Indexed addressing mode - effective address = contents of index register (SI or DI) + displacement.

   Eg:- MOV AL, [SI + 05H] ; load from address SI + 05H

7. Based indexed addressing mode - effective address = base register + index register

   Eg:- MOV AL, [BX+SI] ; load from address BX + SI

8. Relative addressing mode - used mainly in jump instructions target address = current IP + displacement.

   Eg: JMP SHORT LABEL ; jump to a nearby instruction.

Memory models - It defines how to code, data and stack segments are organised in memory. These are specified using .model directive.

Purpose: Organize memory usage. Define how many code segments and data segments are used. Help assembler manage the segment registers: CS, BS, DS.

1. .model tiny

- Code and data in one segment. Maximum size 64 kB.

2. .model small

- One code segment and one data segment. Maximum 64kB

3. .model medium

- One data segment, multiple code segments.

4. .model compact

- One code segment, multiple data segments

5. .model large

- multiple code and data segments

Each segment <= 64 kB, but total program can exceed 64 kB

6. .model huge

- like large but supports arrays > 64kB. Used in complex data heavy programs.

1) ·Hello World Program, display your name

```
.MODEL SMALL
.DATA
msg DB 'Hello WORLD $'
~~name DB 'Krishna$'~~
.CODE
START:
MOV AX,@DATA
MOV DS,AX
LEA DX,.msg
MOV AH,09H
INT 21H
~~LEA DX, name~~
~~INT 21H~~
MOV AH,4CH
INT 21H
 END START
```

Result:
Succesfully displayed
Hello World

2) Data delaration : DB, DD, DW, etc.

```
.MODEL SMALL
.DATA
a DB 10
b ·DW 1234H
c DD 12345678H
.CODE
START: MOV AH,4CH
INT 21H
END START
```

3) Display 2 strings

```
.MODEL SMALL
.STACK 100H
.DATA
~~s1 DB 'One$'~~ msg1 DB 'Hello$'
~~s2 DB 'Two$'~~ msg2 DB 'WORLD$'
.CODE
START:
MOV AX,@DATA
MOV DS,AX
```

```
.MODEL
.STACK 100H
.DATA
NUM1 DB 05H
NUM2 DB 02H
.CODE
MAIN:
MOV AX,@DATA
MOV DS,AX
MOV AL,NUM1
MOV AL,NUM2
ADD AL,30H
MOV DL,AL
MOV AH,02H
INT 21H
MOV AH,4CH
INT 21H
END MAIN
```

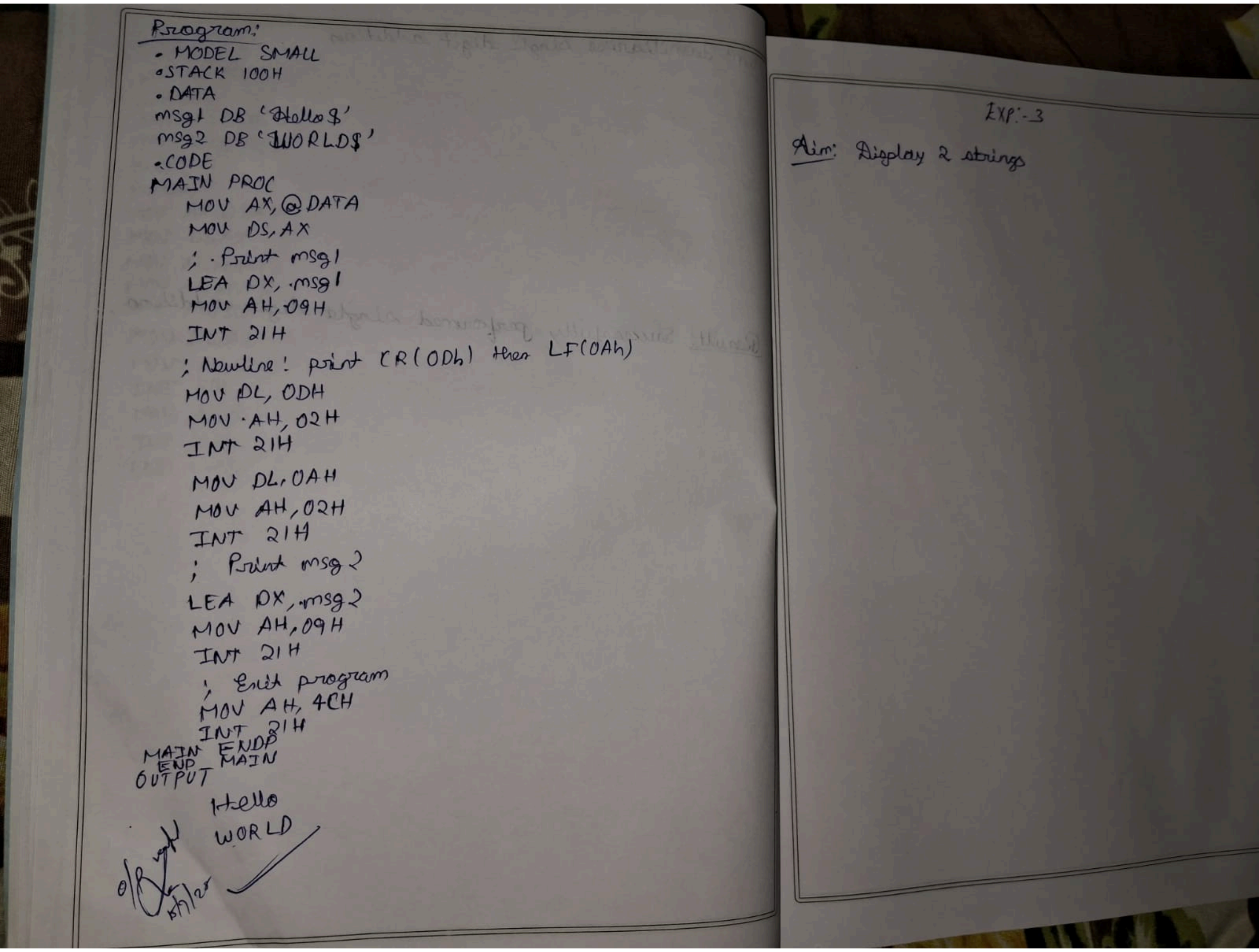
Output:

7

o/p verified
15/7/25

Aim: Familiarise single digit addition

Result: Successfully performed single digit addition



Program:

```
.MODEL SMALL
.STACK 100H
.DATA
msg1 DB 'Hello$'
msg2 DB 'WORLD$'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ; Print msg1
    LEA DX, msg1
    MOV AH,09H
    INT 21H
    ; Newline: print CR(0Dh) then LF(0Ah)
    MOV DL, 0DH
    MOV AH, 02H
    INT 21H
    MOV DL, 0AH
    MOV AH, 02H
    INT 21H
    ; Print msg2
    LEA DX, msg2
    MOV AH, 09H
    INT 21H
    ; Exit program
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```

OUTPUT

```
Hello
WORLD
```

EXP:-3

Aim: Display 2 strings