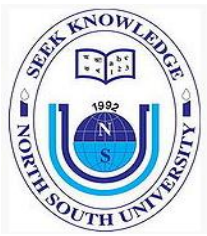


North South University

CSE 331L

Microprocessor Interfacing & Embedded System (Lab)

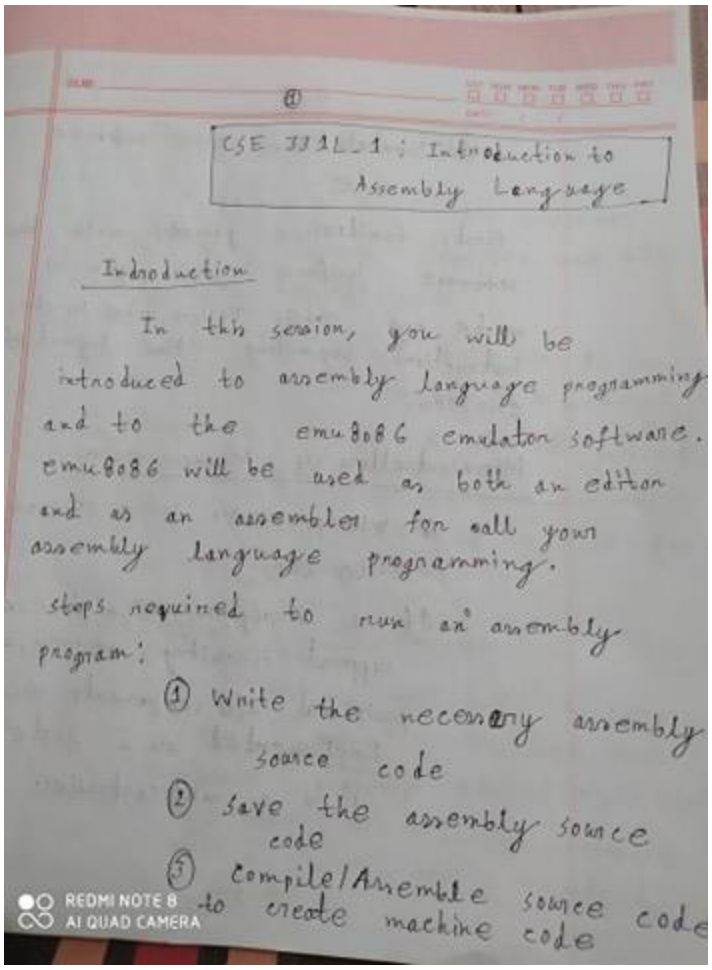
HOMEWORK 03

SUBMITTED BY:

Name: SADMAN ALAM
ID: 1610544042
Section: 07
Phone Number: 01828157801
E-Mail: sadman.alam@northsouth.edu

SUBMITTED TO:

Lab Instructor's Name: ASIF AHMED NELOY



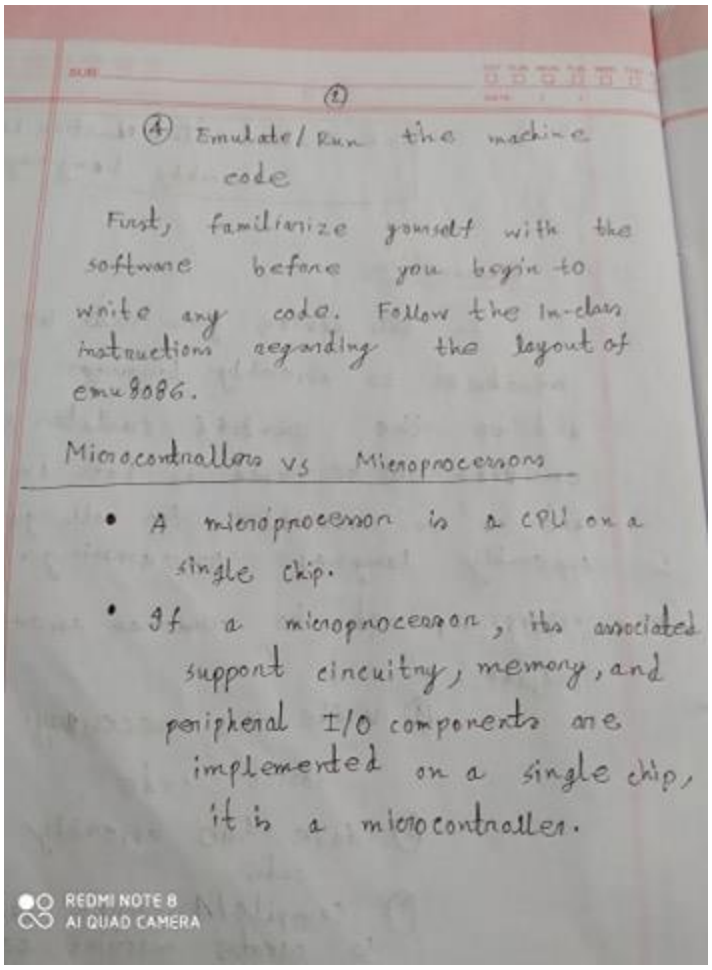
CSE 311L-1 : Introduction to Assembly Language

Introduction

In this session, you will be introduced to assembly language programming and to the emu8086 emulator software. emu8086 will be used as both an editor and as an assembler for all your assembly language programming.

steps required to run an assembly program:

1. Write the necessary assembly source code
2. Save the assembly source code
3. Compile/Assemble source code to create machine code



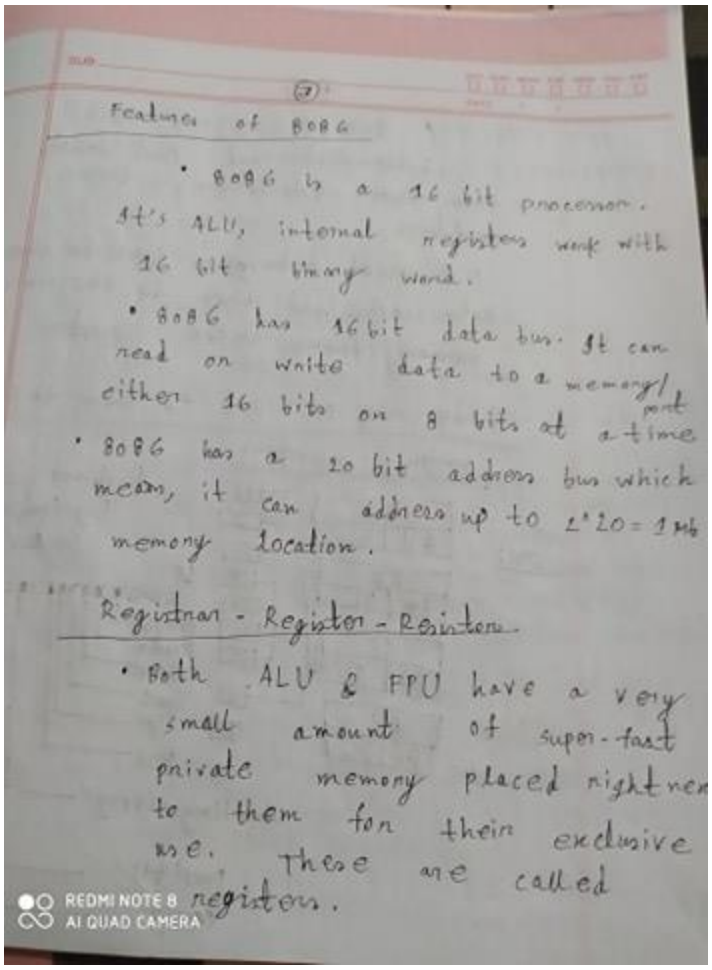
④ Emulate/Run the machine code

First, familiarize yourself with the software before you begin to write any code. Follow the in-class instructions regarding the layout of emu8086.

Microcontrollers vs Microprocessors

- A microprocessor is a CPU on a single chip.
- If a microprocessor, its associated support circuitry, memory, and peripheral I/O components are implemented on a single chip, it is a microcontroller.

REDMI NOTE 8
AI QUAD CAMERA



⑦

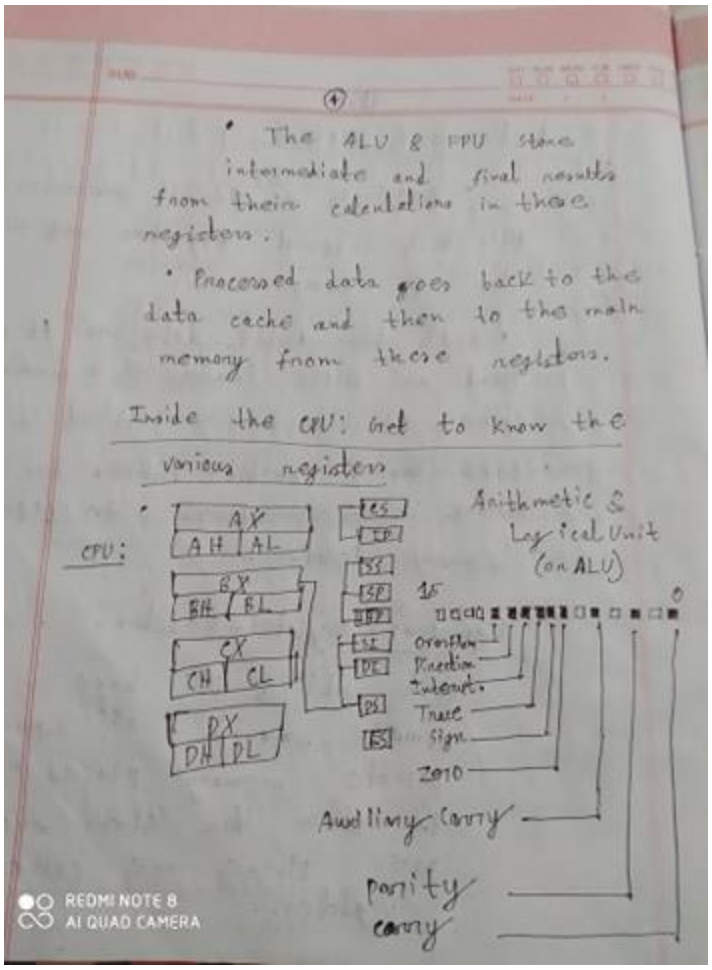
Features of 8086

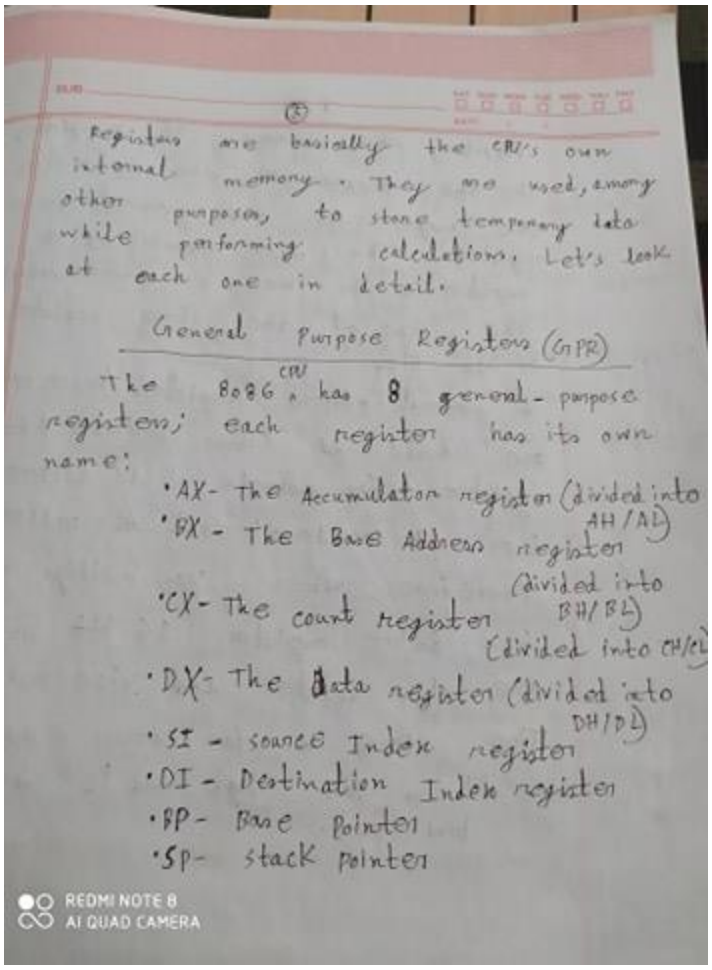
- 8086 is a 16 bit processor. It's ALU, internal registers work with 16 bit binary word.
- 8086 has 16 bit data bus. It can read or write data to a memory/port either 16 bits or 8 bits at a time.
- 8086 has a 20 bit address bus which means, it can address up to $2^{20} = 1$ Mb memory location.

Registrar - Register - Resistor

- Both ALU & FPU have a very small amount of super-fast private memory placed right next to them for their exclusive use. These are called registers.

REDMI NOTE 8
AI QUAD CAMERA


• The ALU & FPU store intermediate and final results from their calculations in these registers.
• Processed data goes back to the data cache and then to the main memory from these registers.
Inside the CPU: Get to know the various registers
CPU:
AX
AH AL
BX
BH BL
CX
CH CL
DX
DH DL
ES
CS
SS
SP
BP
SI
DI
DS
IS
Arithmetic & Logical Unit (on ALU)
15
0
Overflow
Direction
Interrupt
Trace
Sign
Zero
Auxiliary Carry
parity
carry
REDMI NOTE 8
AI QUAD CAMERA



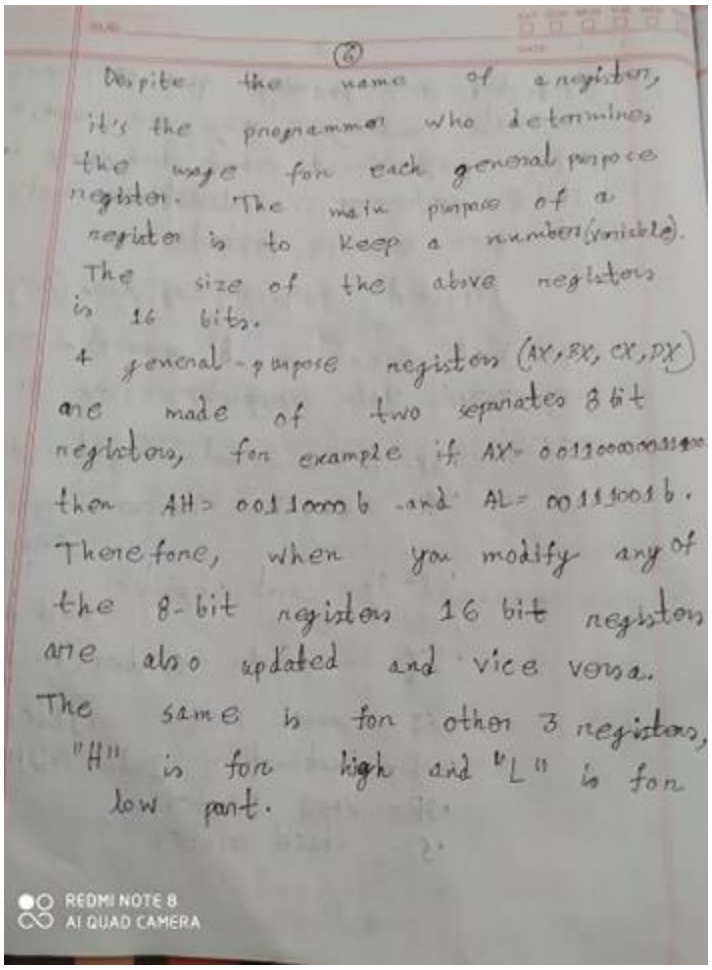
Registers are basically the CPU's own internal memory. They are used, among other purposes, to store temporary data while performing calculations. Let's look at each one in detail.

General Purpose Registers (GPR)

The 8086 CPU has 8 general-purpose registers; each register has its own name:

- AX - The Accumulator register (divided into AH / AL)
- BX - The Base Address register (divided into BH / BL)
- CX - The count register (divided into CH / CL)
- DX - The Data register (divided into DH / DL)
- SI - source Index register
- DI - Destination Index register
- BP - Base Pointer
- SP - stack Pointer

REDMI NOTE 8
AI QUAD CAMERA



(6)

Despite the name of a register, it's the programmer who determines the usage for each general purpose register. The main purpose of a register is to keep a number(variable). The size of the above registers is 16 bits.

4 general-purpose registers (AX, BX, CX, DX) are made of two separates 8 bit registers, for example if AX= 0011000000111000 then AH= 00110000b and AL= 00111001b. Therefore, when you modify any of the 8-bit registers 16 bit registers are also updated and vice versa.

The same is for other 3 registers, "H" is for high and "L" is for low part.

REDMI NOTE 8
AI QUAD CAMERA

SUB:

SAT SUN MON TUE WED THU FRT

DATE: / /

(7)

Segment Registers

CS - points at the segment containing the current program

DS - generally points at the segment where variables are defined.

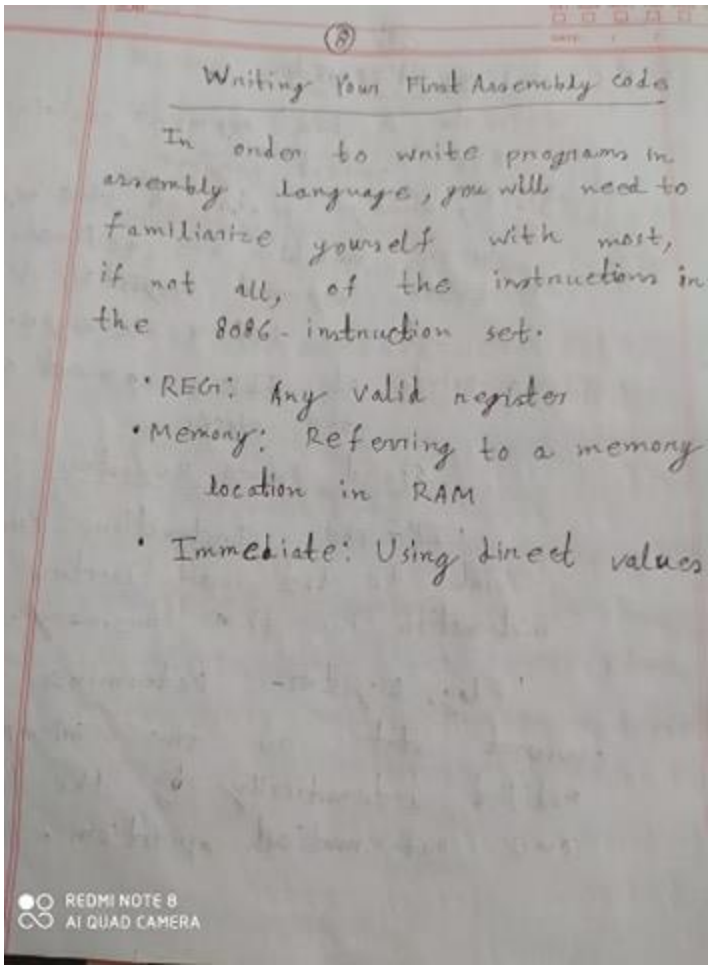
ES - extra segment register, it's up to a coder to define its usage.

SS - points at the segment containing the stack.

Special Purpose Registers

• IP - The Instruction Pointer. Points to the next location of instruction in the memory.

• Flags Register - Determines the current state of the microprocessor, modified automatically by the CPU after some mathematical operations.



(8)

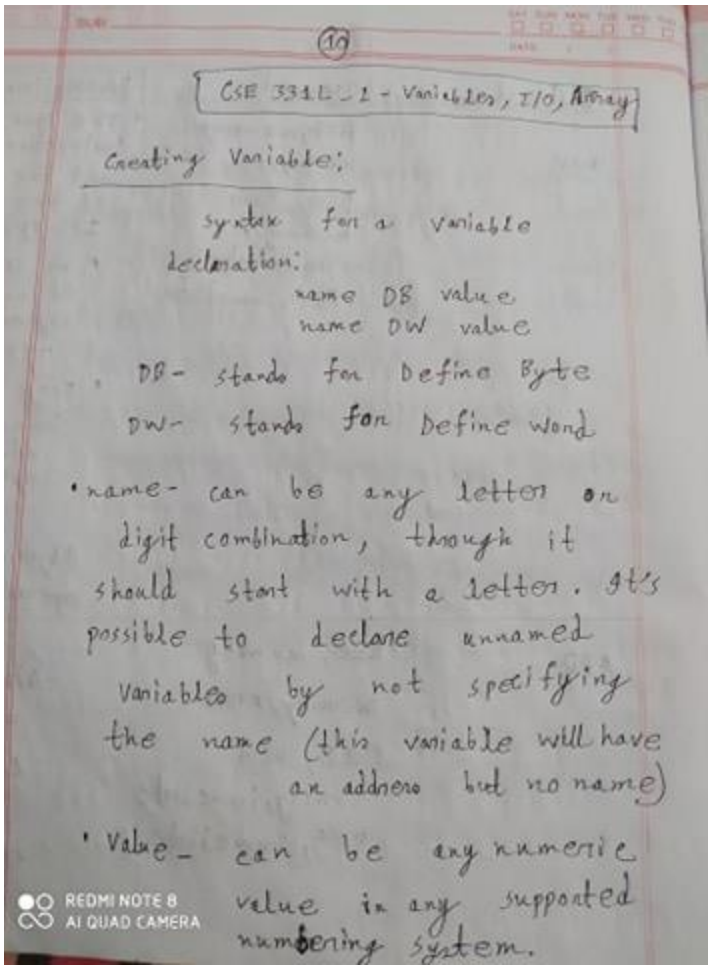
Writing Your First Assembly code

In order to write programs in assembly language, you will need to familiarize yourself with most, if not all, of the instructions in the 8086 instruction set.

- REG: Any valid register
- Memory: Referring to a memory location in RAM
- Immediate: Using direct values

SUB: (9) DATE / /

Instruction	Operands	Description
MOV	REG, memory memory, REG REG, REG memory, immediate REG, immediate	• The MOV instruction cannot set the value of the CS and IP register. • To copy value of one segment register. • Copy an immediate value to segment register. Algorithm: operand1 = operand
ADD	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Adds two numbers. Algorithm Operand1 = operand1 + operand2



(10)

CSE 331L - 1 - Variables, I/O, Array

Creating Variable:

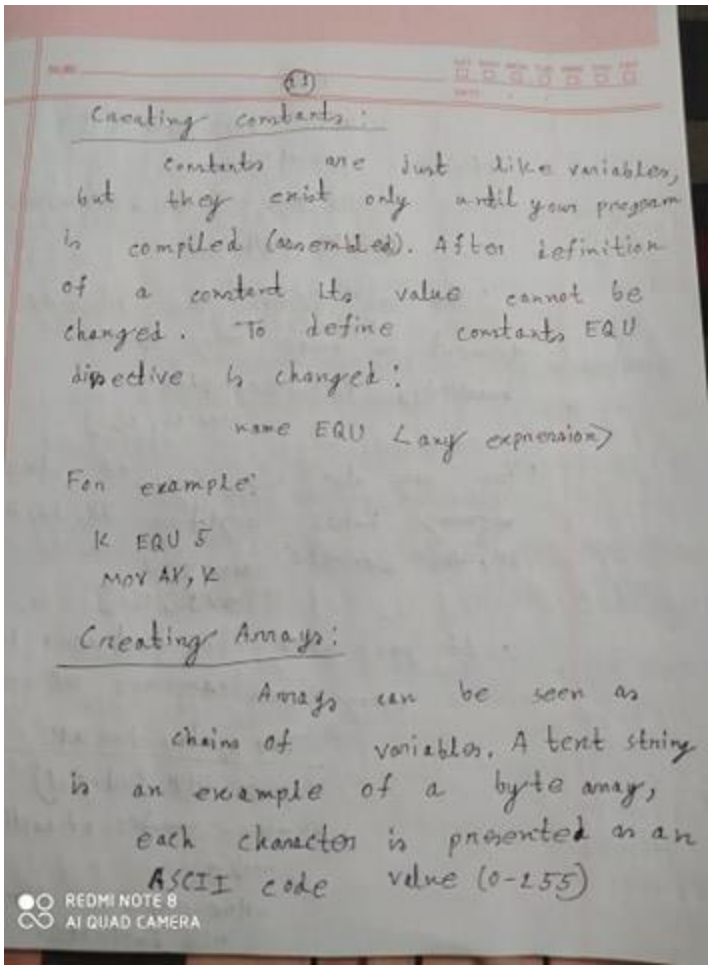
syntax for a variable declaration:

name DB value
name DW value

DB - stands for Define Byte
DW - stands for Define Word

- name - can be any letter or digit combination, though it should start with a letter. It's possible to declare unnamed variables by not specifying the name (this variable will have an address but no name)
- Value - can be any numeric value in any supported numbering system.

REDMI NOTE 8
AI QUAD CAMERA



(21)

Creating constants:

Constants are just like variables, but they exist only until your program is compiled (assembled). After definition of a constant its value cannot be changed. To define constants EQU directive is changed:

name EQU <any expression>

For example:

```
K EQU 5
MOV AX, K
```

Creating Arrays:

Arrays can be seen as chains of variables. A text string is an example of a byte array, each character is presented as an ASCII code value (0-255)

SUB:

SAT SUN MON TUE WED TH

DATE: / /

(12)

Here are some array definition examples:

a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h

b DB 'Hello', 0

- You can access the value of any element in array using square brackets, for example:

```
MOV AL, a[3]
```

- You can also use any of the memory index registers BX, SI, DI, BP, for example:

```
MOV SI, 3
MOV AL, a[SI]
```

- If you need to declare a large array you can use DUP operator.

The syntax for DUP:

number DUP (value(s))

- number - number of duplicates to make
- value - expression that DUP will duplicate.

SUB:

SAT SUN MON TUE WED THU FRT

DATE / /

(13)

for example:

c DB 5 DUP(2)

is an alternative way of declaring:

c DB 2, 2, 2, 2, 2

One more example:

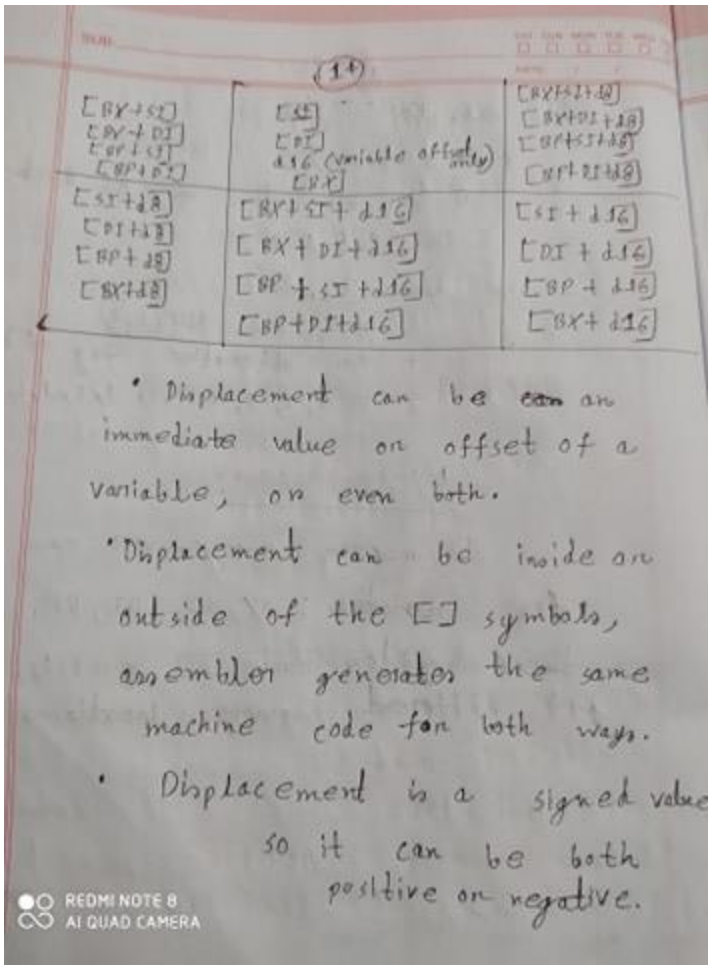
d DB 5 DUP(1,2)

is an alternative way of declaring:

d DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2

Memory Access:

To access memory, we can use these four registers: BX, SI, DI, BP. Combining these registers inside [] symbols, we can get different memory locations.

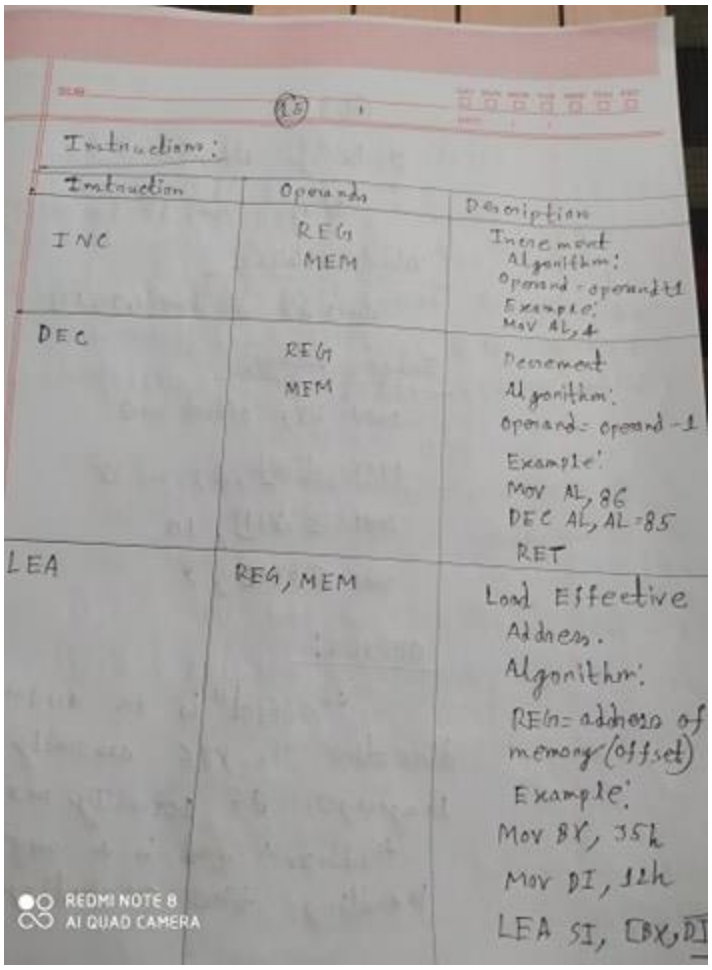


(14)

[BX+SI] [BX+DI] [BP+SI] [BP+DI]	[SI] [DI] d16 (variable offset only) [BX]	[BX+SI+d8] [BX+DI+d8] [BP+SI+d8] [BP+DI+d8]
[SI+d8] [DI+d8] [BP+d8] [BX+d8]	[BX+SI+d16] [BX+DI+d16] [BP+SI+d16] [BP+DI+d16]	[SI+d16] [DI+d16] [BP+d16] [BX+d16]

- Displacement can be an immediate value or offset of a variable, or even both.
- Displacement can be inside or outside of the [] symbols, assembler generates the same machine code for both ways.
- Displacement is a signed value so it can be both positive or negative.

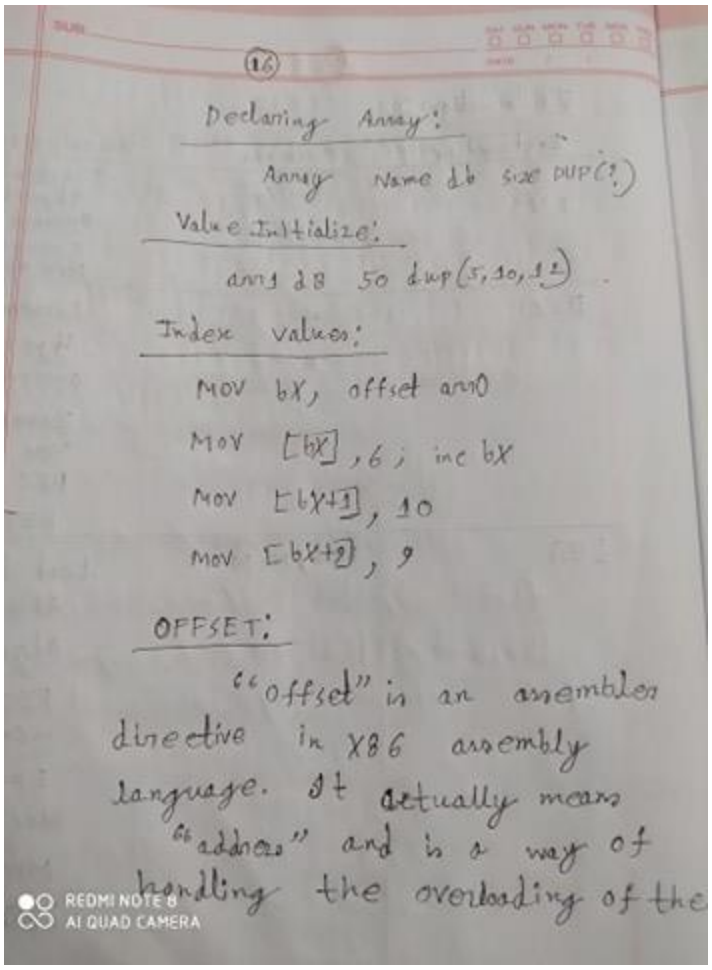
REDMI NOTE 8
AI QUAD CAMERA



Instructions:

Instruction	Operands	Description
INC	REG MEM	Increment Algorithm: Operand = operand+1 Example: Mov AL, 4
DEC	REG MEM	Decrement Algorithm: Operand = operand - 1 Example: Mov AL, 86 DEC AL, AL = 85 RET

LEA	REG, MEM	Load Effective Address.
		Algorithm:
		REG = address of memory (offset)
		Example:
		Mov BX, 35h
		Mov DI, 12h
		LEA SI, [BX, DI]



16

Declaring Array:

Array Name db size DUP(?)

Value Initialize:

arr1 db 50 dup(5,10,12)

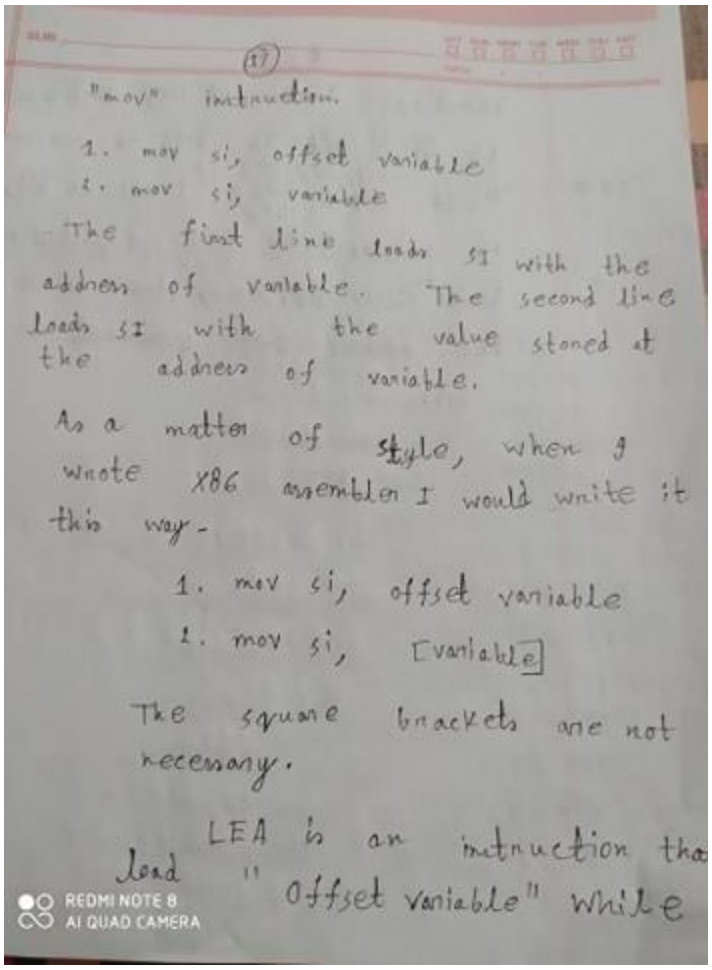
Index values:

```
MOV bX, offset arr0
MOV [bX], 6 ; inc bX
MOV [bX+1], 10
MOV [bX+2], 9
```

OFFSET:

"offset" is an assembler directive in X86 assembly language. It actually means "address" and is a way of handling the overloading of the

REDMI NOTE 8
AI QUAD CAMERA



(37)

"mov" intruction.

1. mov si, offset variable
2. mov si, variable

The first line loads SI with the address of variable. The second line loads SI with the value stored at the address of variable.

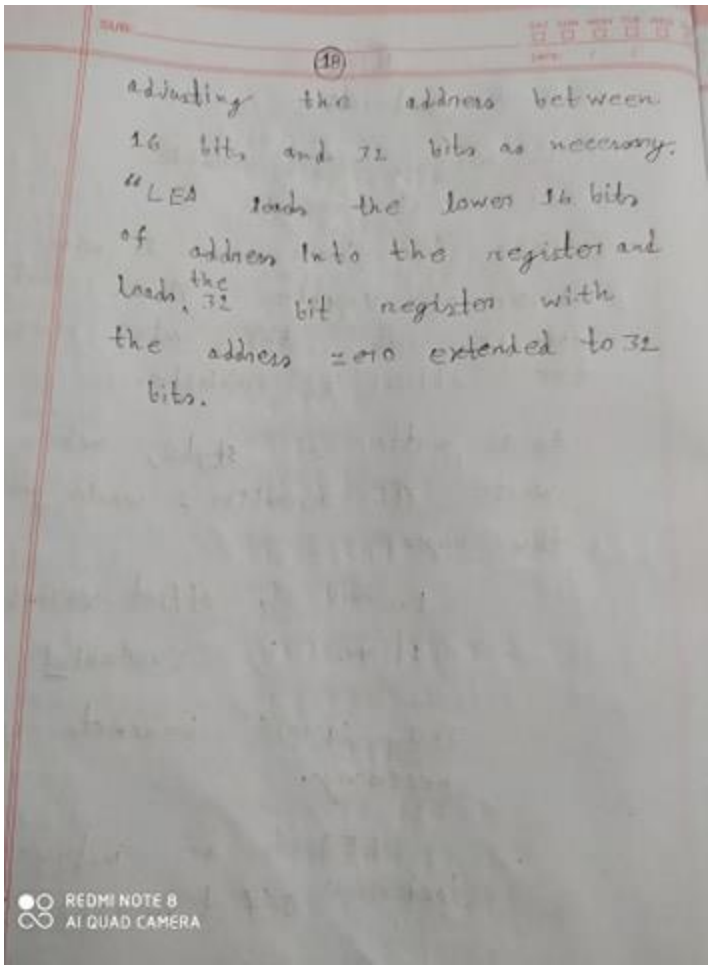
As a matter of style, when I wrote X86 assembler I would write it this way -

1. mov si, offset variable
2. mov si, [variable]

The square brackets are not necessary.

LEA is an intruction that load "offset variable" while

REDMI NOTE 8
AI QUAD CAMERA


18
adjusting the address between
16 bits and 32 bits as necessary.
"LEA loads the lower 16 bits
of address into the register and
loads the 32 bit register with
the address zero extended to 32
bits.
REDMI NOTE 8
AI QUAD CAMERA

(19)

CSE 331L-3-Print and I/O

In this Assembly Language Programming, a single program is divided into 4 segments, which are:-

1. Data Segment,
2. Code Segment,
3. Stack Segment,
4. Extra Segment.

* Print: "Hello World" in Assembly Language:

```
DATA SEGMENT:
        MESSAGE DB "HELLO WORLD!!!$"

ENDS
CODE SEGMENT
        ASSUME DS: DATA CS: CODE

START:
        MOV AX, DATA
        MOV DS, AX
        LEA DX, MESSAGE
```

```
        MOV AH, 9
        INT 21H
        MOV AH, 4CH
        INT 21H
ENDS
END START
```

Now, from these one is compulsory i.e. Code Segment if at all you don't need variable for the program. If you need variable for the program, you will need two segments i.e. Code Segement and Data Segment.

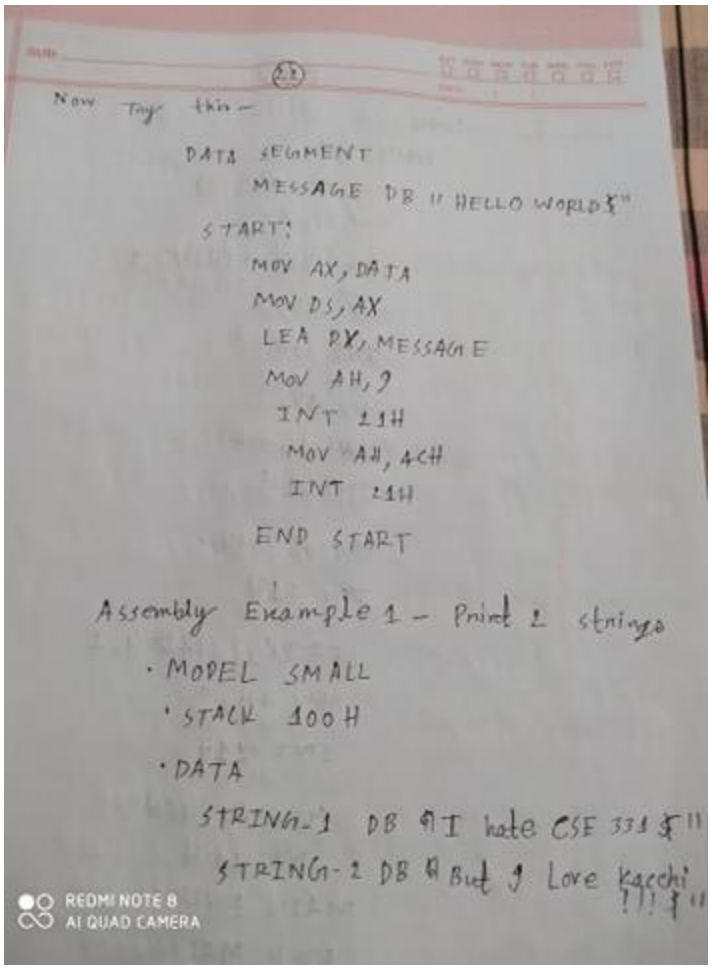
* First line - DATA SEGMENT:

"DATA SEGMENT" is the starting point of the Data Segment in a program and "DATA" is the name given to this segment and "SEGMENT" is the keyword for difining segments, where

we can declare our variables.

* Next Line - MESSAGE DB "HELLO WORLD!!!$":-

"MESSAGE" is the variable name given to a Data Type, that is DB. DB stands for define byte and is of One byte in 8 bits. In Assembly Language Program, variables are define by data size, not its type. Character need One byte so to store character or String we need DB only that don't mean DB can't hold number or numerical value. The string is given in double quotes. $ is used as NULL character in C programming, so that compiler can understand where to ~~stop~~ "STOP".



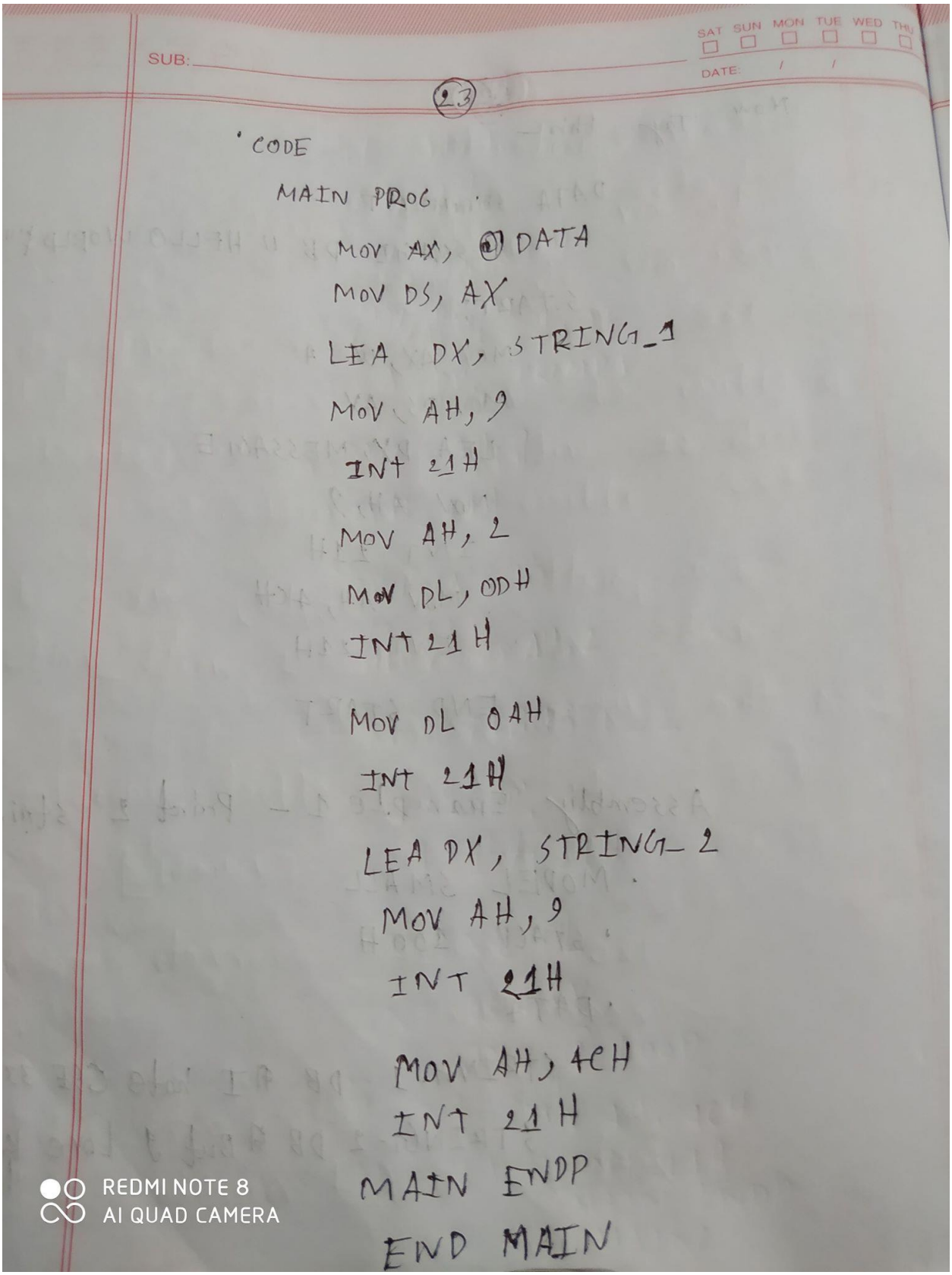
Now Try this –

```
DATA SEGMENT
    MESSAGE DB "HELLO WORLD$"
START:
    MOV AX, DATA
    MOV DS, AX
    LEA DX, MESSAGE
    MOV AH, 9
    INT 21H
    MOV AH, 4CH
    INT 21H
END START
```

Assembly Example 1 – Print 2 strings

```
.MODEL SMALL
.STACK 100H
.DATA
    STRING-1 DB "I hate CSE 331 $"
    STRING-2 DB "But I Love kacchi !!! $"
```

REDMI NOTE 8
AI QUAD CAMERA



SUB:

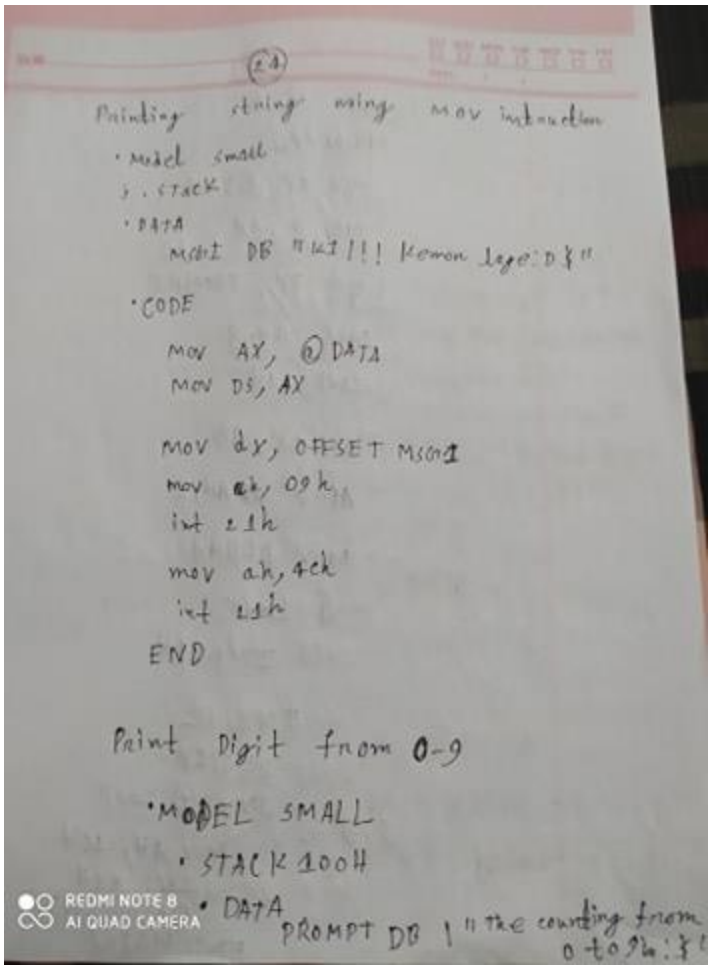
SAT SUN MON TUE WED THU

DATE: / /

23

```
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    LEA DX, STRING_1
    MOV AH, 9
    INT 21H
    MOV AH, 2
    MOV DL, 0DH
    INT 21H
    MOV DL 0AH
    INT 21H
    LEA DX, STRING_2
    MOV AH, 9
    INT 21H
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```

REDMI NOTE 8
AI QUAD CAMERA



(24)

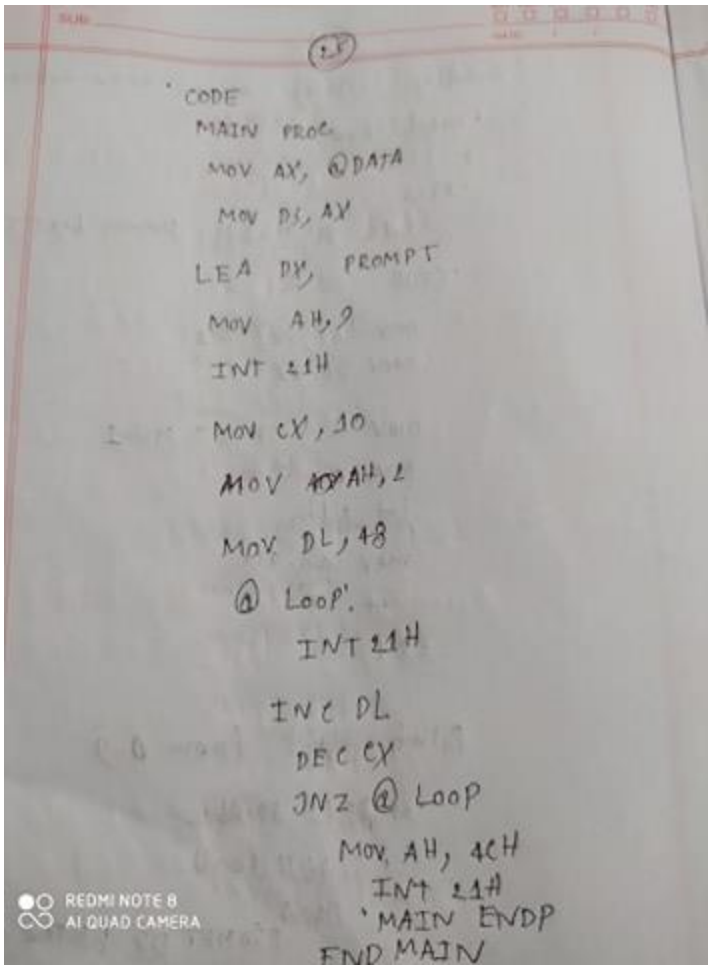
Printing string using mov instruction

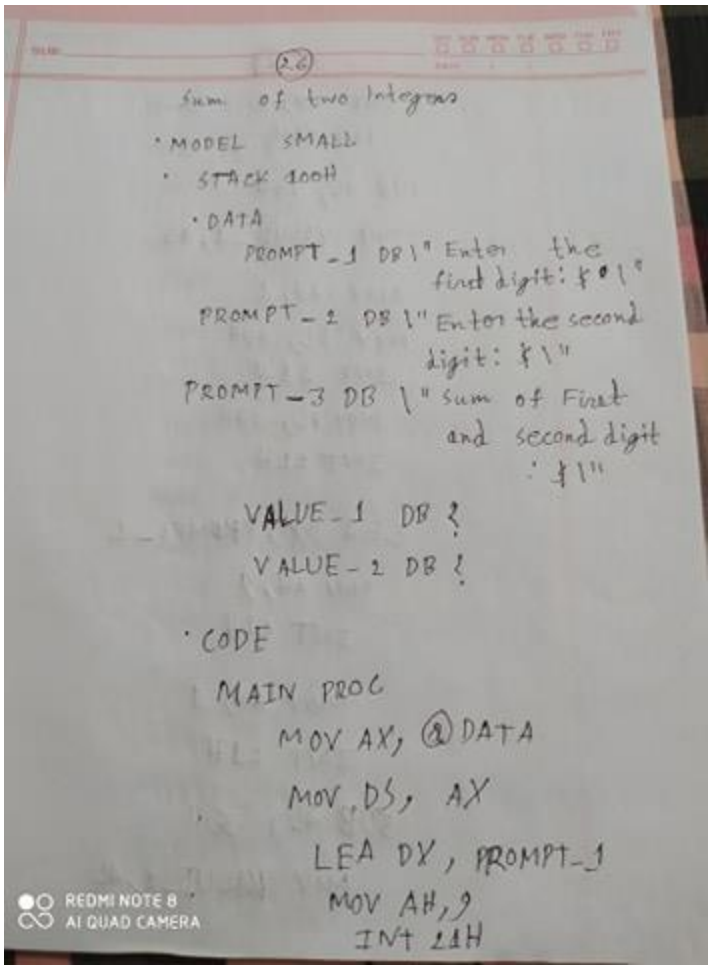
```
.model small
.stack
.DATA
    MSG1 DB "Hi!!! Kemon Aso:D$"
.CODE
    mov AX, @DATA
    mov DS, AX

    mov dx, OFFSET MSG1
    mov ah, 09h
    int 21h
    mov ah, 4ch
    int 21h
END
```

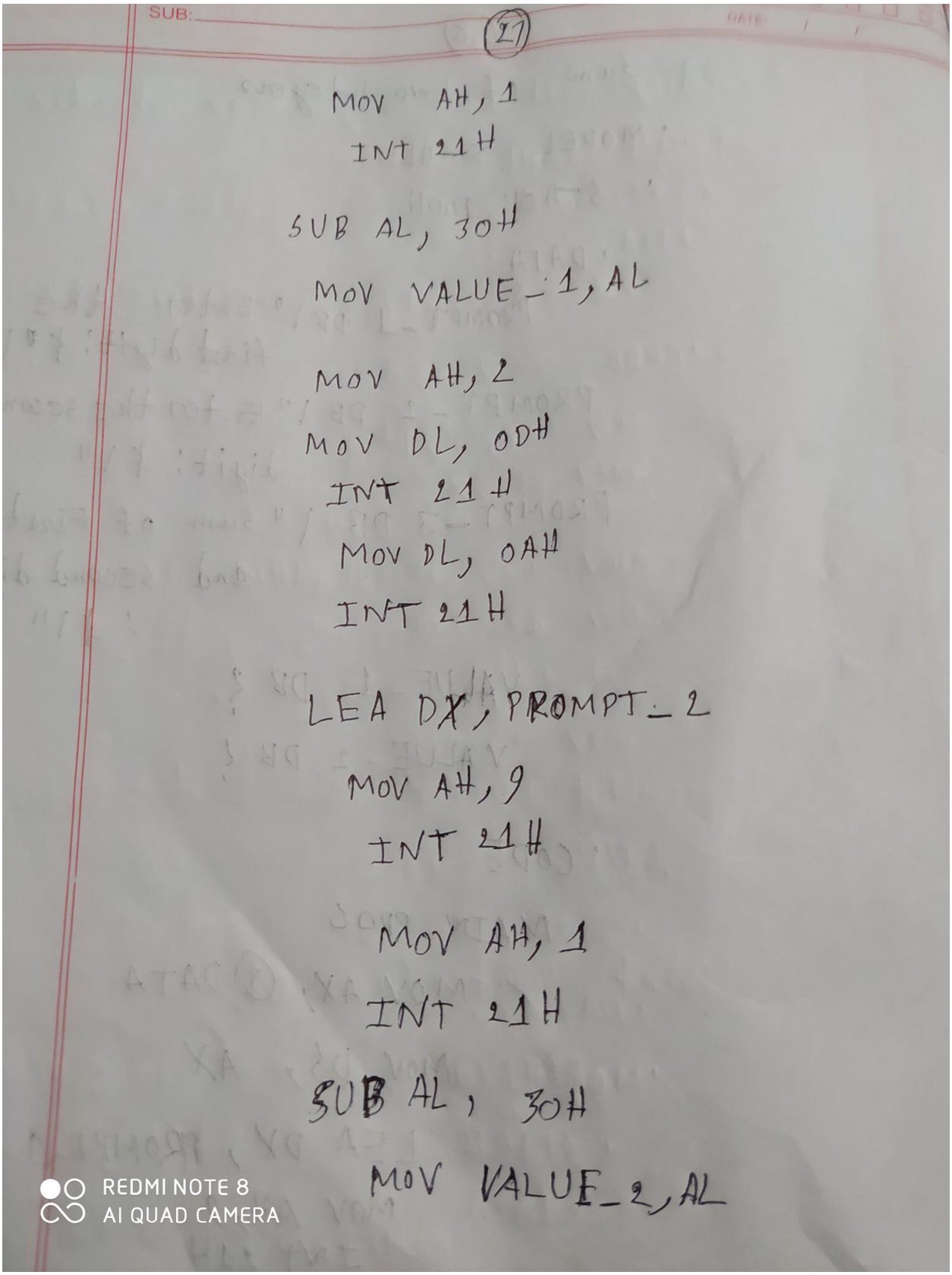
Print Digit from 0-9

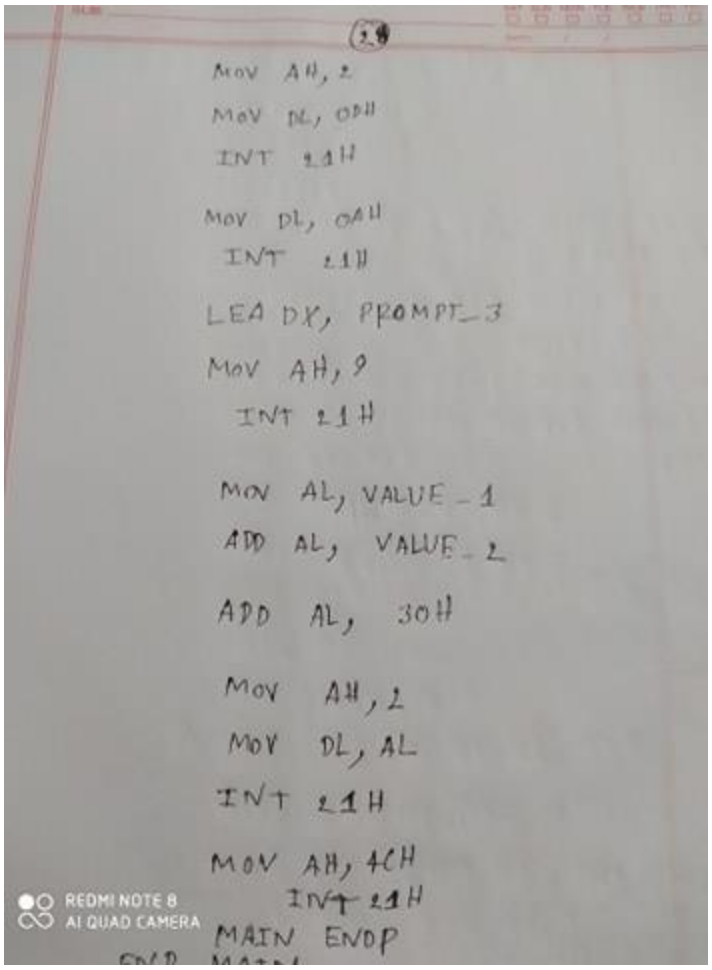
```
.MODEL SMALL
.STACK 100H
.DATA
PROMPT DB "the counting from 0 to 9 : $"
```

REDMI NOTE 8
AI QUAD CAMERA


CODE
MAIN PROC
MOV AX, @DATA
MOV DS, AX
LEA DX, PROMPT
MOV AH, 9
INT 21H
MOV CX, 10
MOV AH, 2
MOV DL, 48
@ LOOP:
INT 21H
INC DL
DEC CX
JNZ @ LOOP
MOV AH, 4CH
INT 21H
MAIN ENDP
END MAIN
REDMI NOTE 8
AI QUAD CAMERA


(22)
Sum of two integers
.MODEL SMALL
.STACK 100H
.DATA
PROMPT_1 DB \" Enter the first digit: $\"
PROMPT_2 DB \" Enter the second digit: $\"
PROMPT_3 DB \" Sum of First and second digit : $\"
VALUE_1 DB ?
VALUE_2 DB ?
.CODE
MAIN PROC
MOV AX, @DATA
MOV DS, AX
LEA DX, PROMPT_1
MOV AH, 9
INT 21H
REDMI NOTE 8
AI QUAD CAMERA


SUB:
(27)
MOV AH, 1
INT 21H
SUB AL, 30H
MOV VALUE_1, AL
MOV AH, 2
MOV DL, 0DH
INT 21H
MOV DL, 0AH
INT 21H
LEA DX, PROMPT_2
MOV AH, 9
INT 21H
MOV AH, 1
INT 21H
SUB AL, 30H
MOV VALUE_2, AL
REDMI NOTE 8
AI QUAD CAMERA


MOV AH, 2
MOV DL, 0DH
INT 21H
MOV DL, 0AH
INT 21H
LEA DX, PROMPT_3
MOV AH, 9
INT 21H
MOV AL, VALUE_1
ADD AL, VALUE_2
ADD AL, 30H
MOV AH, 2
MOV DL, AL
INT 21H
MOV AH, 4CH
INT 21H
MAIN ENDP
REDMI NOTE 8
AI QUAD CAMERA