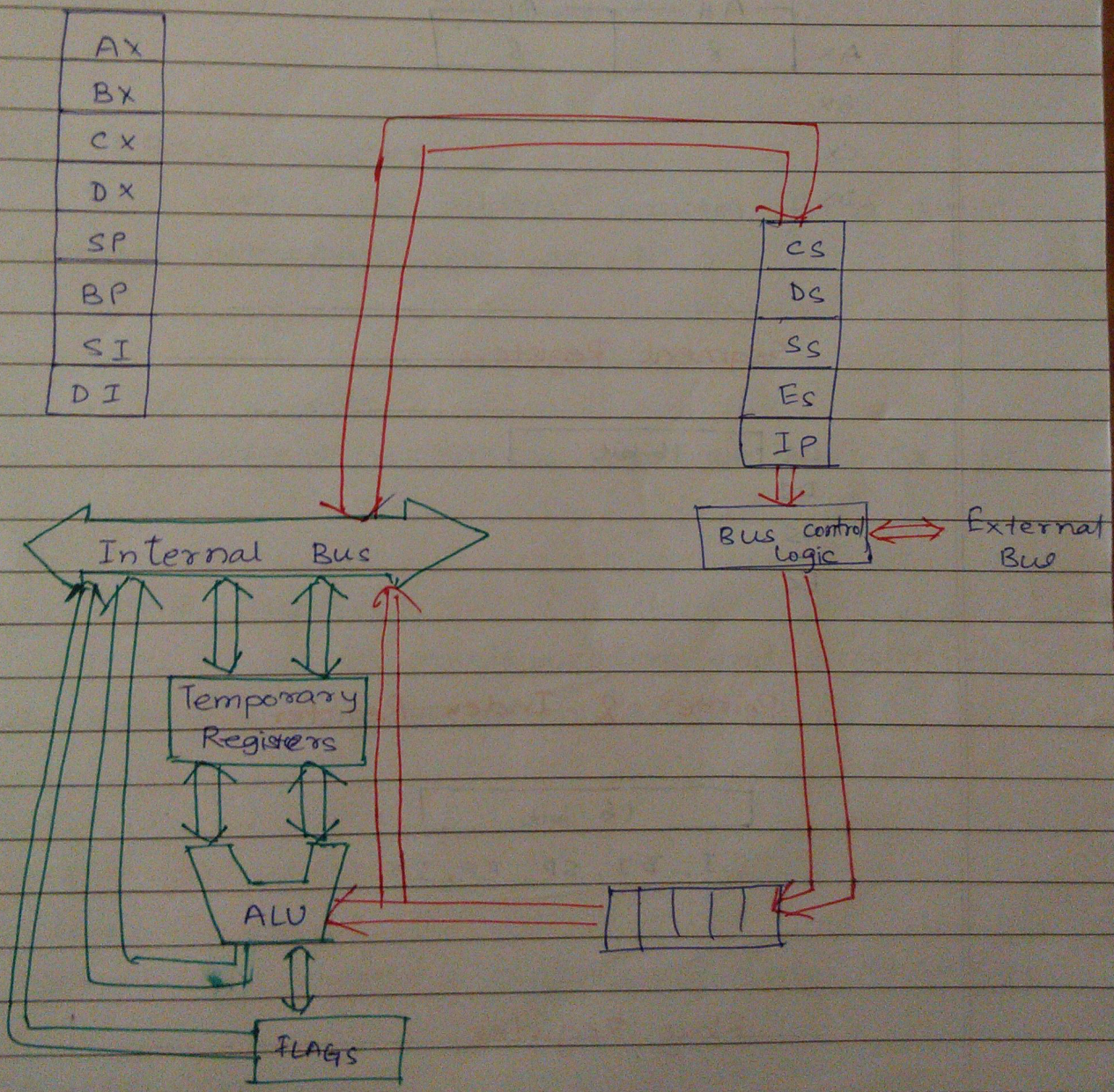
20/07/15

Page No.: 1
Date: / /

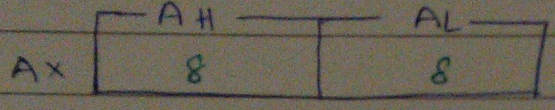
Lecture - 1

* **8086 & 8088 Microprocessor**

- 16 bit processor → 8-bit



Data Register

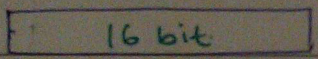


BX

CX

DX

Segment Registers



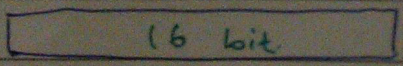
CS

DS

SS

ES

Pointer & Index Register



SI, DI, SP, BP, IP

Flag Register

→ **AX** - (Accumulator register).
Ax is preffered register for arithmetic operations, logical and data transfer instructions. Also, this is used for I/O operation.

→ **BX** (Base register)
This serves as address register. for a table lookup instruction, we use BIX.

→ **CX** (count register)
For performing various loops, we use Cx

→ **DX** (data register)
DX is used in multiplication and division. It is also used in I/O operation.

Segment Registers.

→ **CS** (Code segment)
Stores the starting address of code segment.

→ **DS** (Data segment)
→ Stores starting address of data segment.

→ **SS** (Stack segment)
→ starting address of stack segment.

→ **ES** (Extra segment)
→ for any new segment (when required.

Pointer & Index register

→ SP (Stack Pointer)

Used for accessing stack segment.

→ BP (Base pointer)

BP is used for accessing data from stack segment.

→ SI (source index)

SI is used to point to memory locations in the data segment addressed by DS.

→ DI (destination index)

The job of SI and DI are similar. For performing string operations, we use DI.

→ IP (instruction pointer)

IP contains the offset address of the next instruction to be executed.

Flag Register

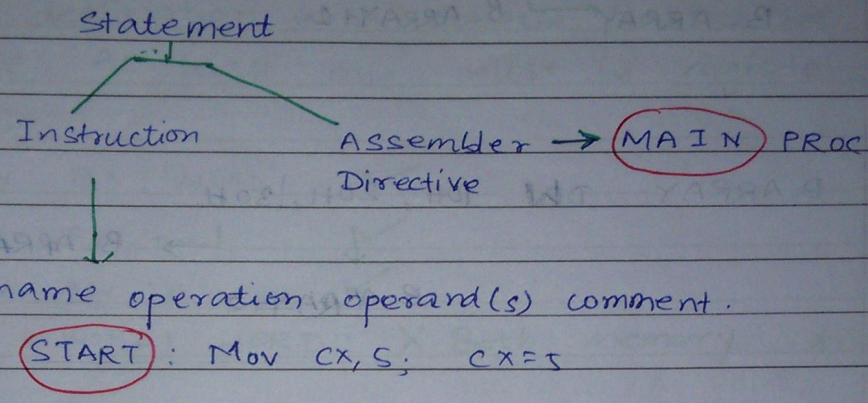
- Status flags
- control flags

22/07/15

Lecture - 2

* Assembly Language

Microsoft Macro Assembler (MASM)



* Data

decimal 999D (or simply 999)
Binary 1010B
Hexa-deimal F12EH
Character 'A'
"Hello"

* Variables

DB - Define byte - 8 bits
DW - Define word - 16 bits
DD - Define double word - 32 bits
DQ - Define quadword - 64 bits
DT - Define 10 bytes - 80 bits

name ↑ memory (8 bit) ↑

ALPHA DB 4 → value stored is 4.
BYT DB ? → value not initialised to any value (when ? is specified)
~~WRRD~~ reserving 16 bit ↑
WRD DW 2 → initialised to 2.

* Arrays

B_ARRAY DB 10H, 20H, 30H

B_ARRAY → B_ARRAY+2 (for access the elem

B_ARRAY → B_ARRAY+1

B_ARRAY DB 10H, 20H, 30H

B_ARRAY → B_ARRAY+4.

B_ARRAY → B_ARRAY+2

* character String

LETTER DB 'ABC'

LETTER → LETTER+2

LETTER → LETTER+1

LETTER DB 41H, 42H, 43H (same values as abov

MSG DB 'HELLO', 0DH, 0AH, '$'

8 elements

H, E, L, L, O, 0DH, 0AH, $

* Named constants

LF EQU 0AH → Value

LF → Variable name

EQU → If this is given, then throughout the code we cannot change the value of the variable.

* MOV

MOV destination, source.

Both destination and source cannot be memory variable. At least one must be registers.

```
MOV  AX, WORD1   ✓
MOV  AX, BX      ✓
MOV  AH, 'A'     ✓
MOV  WORD1, WORD2  X Both memory variables.
```

* XCHG

XCHG destination, source.

↳ to exchange destination, source.

↳ same restrictions as MOV.

```
XCHG  AH, BL
XCHG  AX, WORD1
```

* ADD/SUB

ADD/SUB destination, source.

→ destination and source is added and result stored in destination.

```
ADD  WORD1, AX
SUB  AX, BX
```

* INC

INC destination.

→ destination will be incremented by 1.

* DEC

DEC WORD1

→ decrementing by 1.

* NEG

NEG destination

→ To make the content negative.
(OR to negate destination)

B=A

```
MOV AX, A
MOV B, AX.
```

A=5-A

```
MOV AX, 5
SUB AX, A
MOV A, AX.
```

A = B-2A

```
MOV AX, B
SUB AX, A
SUB AX, A
MOV A, AX.
```

* Program Structure

• MODEL memory-model → SMALL
↳ Code in one segment.
↳ Data in one segment

→ MEDIUM
↳ Code in more than one segment
↳ Data in one segment.

↳ COMPACT
  ↳ code in one segment.
  ↳ Data in more than one segment.
↳ LARGE
  ↳ code in more than one segment.
  ↳ Data in more than one segment.
  ↳ Array size max^{m} 64 KB.

↳ HUGE
  ↳ same as large.
  ↳ Array size more than 64KB.

* Data Segment
  ↳ Variables are declared here.

```
.DATA
WORD1  DW 5
WORD2  DW 2
MSD    DB 'This is a message'
MASK   EQU 10010010B
```

* Stack Segment

```
. STACK  size
. STACK  100H
```

* code segment
  ↳ for writing code to solve any problem.

```
. CODE
  MAIN PROC
  ; statements
  ;
  ENDP
  TEST PROC
  ;
  END P
```

Final Structure:

```
.MODEL SMALL
.STACK 100H
.DATA
  ;define data
.CODE
  MAIN PROC
    ; Instructions
  MAIN ENDP
END MAIN
```

* Input & Output Instructions

INT 21H

Functions

→ AH = 1 : Single key input.
Output ⇒ AL = ASCII code of ~~character~~ key pres

→ AH = 2 : Single character output.
Output ⇒ DL = ASCII code of ch. to be displayed (Display)

→ AH = 9 : String output.
Input: DX = offset address of the string
LEA – load effective address.
LEA destination, source
LEA DX, MSG.

```
MOV AH, 9
LEA DX, MSG
INT 21H
```
→ MSG must end with '$'

Any string must end with '$'. otherwise while printing, string will go on printing garbage values.

PG1.ASM

```
TITLE PG1: TEST.
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    MOV AH, 2
    MOV DL, '?'
    INT 21H
    MOV AH, 1
    INT 21H
    MOV BL, AL
    MOV AH, 2
    MOV DL, 0DH
    INT 21H
    MOV DL, 0AH
    INT 21H
    MOV AH, 2
    MOV DL, BL
    INT 21H
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN.
```

```
MOV AX, @DATA
MOV DS, AX.
```

} to get the data address.

Display the content of variable MSG defin
in DATA segment.

```
TITLE  PG1 : HELLO
.MODEL  SMALL
.STACK  100H
.DATA
   MSG  DB  'HELLO$'
.CODE
 MAIN  PROC
      MOV  AX, @DATA
      MOV  DS, AX
      MOV  AH, 9
       LEA  Dx, MSG
       INT  21H
       MOV  AH, 4CH
       INT  21H
  MAIN  ENDP
  END  MAIN
```

Read a lower case alphabet. In the next line displa
with appropriate message, the alphabet in upper
case.

```
    TITLE  PG1 : CONVERSION
    .MODEL  SMALL
    .STACK  100H
    .DATA
       CR  EQU  0DH
       LF  EQU  0AH
       MSG 1  DB  'Enter a lower case letter:$'
       MSG2  DB  0DH, 0AH, 'In upper case it i
        CHAR  DB ?, '$'
```

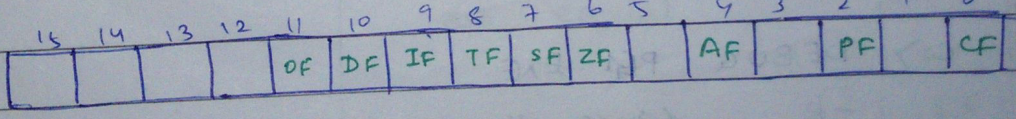
27/07/15

Page No.: 13
Date: / /

Lecture-3

```
• CODE
MAIN PROC
    MOV  AX, @DATA
    MOV  DS, AX
    MOV  AH, 9
    LEA  DX, MSG
    INT  21H
    MOV  AH, 1
    INT  21H
    SUB  AL, 20H
    MOV  CHAR, AL
    MOV  AH, 9
    LEA  DX, MSG2
    INT  21H
    MOV  AH, 4CH
    INT  21H
MAIN ENDP
    END MAIN.
```

⊛ Flag Register



→ ⇒ Cf (Carry Flag)

Cf = 1 if there is a carry out (or borrow) from MSB on addition / subtraction;
Otherwise Cf = 0

→ PF (Parity Flag)

PF = 1 if the low byte of a result has even no. of 1 bit, otherwise PF = 0

→ AF (Auxillary Flag)
AF = 1 if there is a carry out from bit on addition / subtraction.
otherwise AF = 0.

→ ZF (Zero flag)
ZF = 1 for zero result.
Otherwise ZF = 0

→ SF (Sign Flag)
Sf = 1 if MSB of a result is 1.
otherwise SF = 0.

→ OF (Overflow flag)
OF = 1 if sign overflow occurs.
Otherwise OF = 0

AX = FFFFH , BX = 0001H
ADD AX, BX → 10000 H

C:> DEBUG PG1.EXE
-R : Gives the value of register
-T move to next line
-G Complete the execution of all size
-Q to exit.

→ Flow Control Instruction

→ Signed conditional jump

Symbol	Description	condition for Jump
JG/JNLE	Jump if not less than or equal	ZF = 0 & SF = OF
JGE/JNL	not less than	SF = OF
JL/JNGE	not >=	SF <> OF
JLE/JNG	<=	ZF = 1 or SF <> OF

→ unsigned conditional jump

Symbol	Description	Condition
JA/JNBE	Jump if above, not below or equal	CF = 0 & ZF = 0
JAE/JNB	above or equal, not below	CF = 0
JB/JNAE	below, not above or equal	CF = 1
JBE/JNA	below or equal, not above	CF = 1 or ZF = 1

→ single flag jump.

Symbol	Description	Condition
JE/JZ	Jump if equal, equal to 0	ZF = 1
JNE/JNZ	not equal, not equal to 0	ZF = 0
JC	carry	CF = 1
JNC	no carry	CF = 0
JO	overflow	OF = 1
JNO	no overflow	OF = 0
JS	sign overflow	SF = 1
JNS	no sign overflow	SF = 0
JP/JPE	even parity	PF = 1
JNP/JPO	odd parity	PF = 0

```
TITLE P61: CHAR DISPLAY
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    MOV AH,2
    MOV CX,256
    MOV DL,0
P1_LOOP: INT 21H
    DEC CX
    JNZ P1_LOOP
    MOV AH,4CH
    INT 21H
MAIN ENDP
END MAIN
```

CMP

CMP destination, source

```
CMP AX, BX
JG BELOW
```

AX, BX → signed no.

```
MOV CX, AX
CMP BX, CX
JLE NEXT
MOV CX, BX
NEXT:
```

JMP

JMP label.

eg

```
TOP:
 :
DEC CX
JNZ NEXT
 JMP TOP.
```

IF..THEN

```
If AX<0                    CMP AX, 0
then AX=-AX          ⇒     JNL END_IF
END_IF.                    NEG AX
                           END_IF:
```

```
If AL<=BL.                 MOV AH, 2
then                       CMP AL, BL
    display AL             JNBE ELSE1
Else                       MOV DL, AL
    display BL             JMP DISPLAY
END_IF.                ELSE1: MOV DL, BL
                       DISPLAY: INT21H
```

→ CASE:

CASE AX

<0 : BX=-1

=0 : BX=0

>0 : BX=1

```
CMP AX,0
JL N1
JE Z1
JG G1
N1:
   MOV BX,-1
   JMP END-CASE
Z1:
   MOV BX,0
   JMP END-CASE
G1: MOV BX,1
END-CASE:
```

(*) If AL ~~const~~ contains 1 OR 3, then display O

If AL contains 2 OR 4, then display E

```
MOV AH,2
CMP AL,1
JE ODD
CMP AL,3
JE ODD
CMP AL,2
JE EVEN
CMP AL,4
JE EVEN
ODD:
   MOV DL,'O'
   JMP END-CASE
EVEN: MOV DL,'E'
END-CASE: INT 21H
```

(*) If a character is in upper case, display it.

```
MOV AH, 1
INT 21H
CMP AL, 'A'
 JNGE END-IF
CMP AL, 'Z'
 JNLE END-IF
 MOV DL, AL
 MOV AH, 2
 INT 21H
END-IF
```

(*) Read a character, if it is 'Y' or 'y' display it

```
MOV AH, 1
INT 21H
CMP AL, 'y'
 JE DISPLAY
CMP AL, 'Y'
JE DISPLAY
 JMP END-IF
DISPLAY:


END-IF:
```

* FOR

LOOP destination label

(*) Print 60 stars in one line.

```
MOV CX,60
MOV DL,'*'
MOV AH,2
L1: INT 21H
    LOOP L1
```

→ Decrements cx and checks whether value of Cx is zero or not.

```
MOV CX,0
MOV DL,'*'
MOV AH,2
L1:
JCXZ SKIP
    INT 21H
    LOOP L1
```

Prints infinite stars

↓ To skip the loop if Cx is zero

* WHILE:

```
    MOV DX,0
    MOV AH,1
WHILE1: INT 21H
    CMP AL,0DH
    JE END-WHILE
    INC DX
    JMP WHILE1
END-WHILE:
```

* do..while:

```
    MOV AH,1
REPEAT:
    INT 21H
    CMP AL,'Y'
    JNE REPEAT.
```

-» Prompt the user to enter a line of text. In the next line display the lowest capital letter and highest capital letter. If no capital letter is present display appropriate message.

→
```
TITLE PG1 : ALPHABET
.MODEL SMALL
.STACK 100H
.DATA
PROMPT DB 'Type a line of text', 0DH, 0AH, '$'
NOCAPMSG DB 0DH, 0AH, 'No capital $'

CAPMSG DB 0DH, 0AH, 'First capital ='
FIRST DB 'J'
       DB 'LAST CAPITAL ='
LAST   DB '@ $'
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX.
    LEA DX, PROMPT
    MOV AH, 9
     INT 21H
     MOV AH,1
     INT 21H
```

```
WHILE1:
    CMP AL, 0DH
    JE END_WHILE
    CMP AL, 'A'
    JNGE END-IF
    CMP AL, 'Z'
    JNLE END_IF
    CMP AL, FIRST
    JNL CHECK_LAST
    MOV FIRST, AL
CHECK_LAST:
    CMP AL, LAST
    JNG END_IF
    MOV LAST, AL
END-IF:
    INT 21H
    JMP WHILE1
END-WHILE:
    MOV AH, 9
    CMP FIRST, 'J'
    JNE CAPS
    LEA DX, NOCAPMSG
    JMP DISPLAY
CPS:
    LEA DX, CAPMSG
DISPLAY:
    INT 21H
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN.
```

* AND, OR, XOR

AND/ destination, source
OR /
XOR

AND

AND operator can be used to clear specific bit while reserving others

OR

OR is used to set specific bit while reserving the others.

XOR

XOR is used to complement specific bit while reserving the others.

* Clear the signbit of AL.

```
AND  AL, 7FH
```

* Set the msb and lsb of AL

```
OR  AL, 81H
```

* Change the sign bit of DX

```
XOR  DX, 8000H
```

SUB AL, 30H is same as AND AL, 0FH
convert from ASCII to number.

Convert lower case into upper case alphabet

```
AND DL, 0DFH
```

```
MOV AX, 0
XOR AX, AX      > same.

OR  AX, AX
CMP CX, 0       > same

NOT destination
```

03/08/15

Page No.: 26
Date: / /

Lecture-5

TEST ~~(check last bit)~~

TEST destination, source

TEST AL, 1

JZ BELOW

Shift Instruction

SHL (shift left) (MSB will go to CF)

SHL destination, 1/CL

1111 → 1110

DH = 8Ah , CL = 3

SHL DH, CL

8Ah = 1000 1010

After shifting DH = 0101 0000

= 50h

SAL

SAL destination, CL

SAL AX, 3

SHR (shift right)

SHR destination, 1/CL

MOV AX, 6514

MOV CL, 2

SHR AX, CL

Q Reverse / Inverse the binary ~~rotation~~ number:

Eg. 01010101 ⟶ 10101010.

```
AL=01010101

XOR BL,BL
MOV CX,8
R1:
  SHL AL,1
  RCR BL,1
  LOOP R1
MOV AL,BL
```

* Binary I/O:

```
XOR BX,BX
MOV AH,1
INT 21H
W1: CMP AL,0DH
    JE END1
    AND AL,0FH
    SHL BX,1
    OR BL,AL
    INT 21H
    JMP W1
```

Reading Binary input

Output:

For 16 times DO
Rotate left BX.
If CF=1
then
    output '1'
else
    output '0'

05/08/15

Page No.: 29
Date: / /

Lecture-6

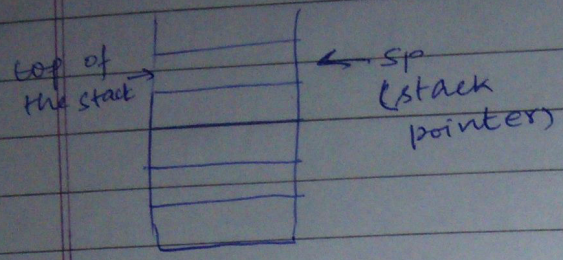
Input (Hexadecimal no.) (4 ~~digi~~ characters)

```
XOR BX, BX
MOV CL, 4
MOV AH, 1
INT 21H
WHILE 1:
  CMP AL, 0DH
  JE END1
   CMP AL, 39H
   JG LETTER
   AND AL, 0FH
    JMP S1
LETTER:
    SUB AL, 37H
S1:
    SHL BX, CL
    OR BL, AL
     INT 21H
     JMP WHILE1
END1:
```

Output of Hexadecimal no.

For 4 times Do
Mov BH to DL
SHR 4 times DL
If DL <10
convert to '0', '1', ..., '9'
else
convert to 'A', ... 'F'
end if
Display char
Rotate BX 4 times left
end for.

Stack



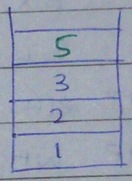
PUSH

PUSH source → 16 bit

PUSH AX

1. SP = SP - 2
2. SS : SP

PUSH AX (where AX = 5)



PUSHF

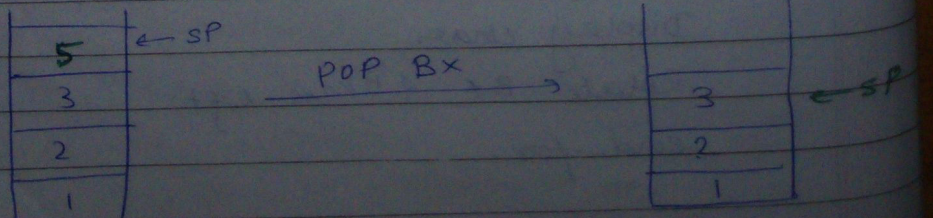
→ pushes the content of the flag register the top of the Stack

POP

POP destination

POP BX

1. content which is pointed by SP is moved to
2. SP = SP + 2



POPF

→ content of top of stack will move to flag register.

Q Read a sequence of characters and display them in reverse order.

```
XOR CX,CX
MOV AH,1
INT 21H
LOOP1:
CMP AL,0DH
JE END1
INC CX
PUSH AX
INT 21H
JMP LOOP1
END1: MOV AH,2
MOV DL,0DH
INT 21H
MOV DL,0AH
INT 21H
JCXZ END2
LOOP2: POP DX
INT 21H
LOOP LOOP2
END2:
```

Use of Stack:

- Evaluation of expression (Infix, Postfix)
- Recursion.

* **Procedure:** (similar to function of C)

user defined name of the procedure

```
name PROC type ⟶ NEAR/FAR
; body of procedure.
RET ⟶ return back to the calling place.
name ENDP
```

NEAR: The statement that calls the proced is in the same segment as the pro

FAR: Calling statement is in different segm

→ If we do not write NEAR/FAR, then NEA type will be the default value.

To call a procedure

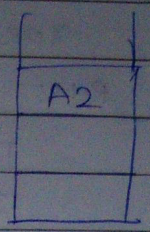
CALL name

Execution of CALL causes the following.

1. The return address of the calling progr is saved on the top of the stack. This the offset address of the next instructi to be executed, after the procedure is complete.
2. IP gets the offset address (addr of the first instruction) of the procedu

IP → instruction pointer.

```
A1    CALL P1
A2    MOV  Ax,10
      .
      .
      .
An    P1   PROC
A(n+1)     .
           .
           .
           RET
           P1 ENDP
```



IP → A(n+1)

→ RET pop value → integer.

OR → RET

↳ ~~stores~~ moves top of stack to IP.

Q Write a procedure for finding a product of two positive integer by addition and bit shifting

```
TITLE PG1 : MULTI
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    MOV AX,7
    MOV BX,13
    CALL MULTIPLY
    MOV AH,4CH
    INT 21H
MAIN ENDP

MULTIPLY PROC
;input : AX = 1st No.
; BX = 2nd no.
; output : DX = result
PUSH AX
```

```
XOR DX, DX
R1:
    TEST BX, 1
    JZ END1
    ADD DX, AX
END1:
    SHL AX, 1
    SHR BX, 1
    JNZ R1
```

Multiplication

MUL source
IMUL source

MUL — unsigned multiplication
IMUL — signed multiplication

Byte form

For byte multiplication, one number is contained in source and other no. is in AL. The 16 bit product will be stored in AX.

Word form

One number is in source, and other no. in AX. The product will be stored in DX:AX

→ A, B word variable : Compute A = 5A − 12B

```
MOV AX, 5
IMUL A
MOV A, AX
MOV AX, 12
IMUL B
SUB A, AX
```

N! → factorial n

```
FACTO  PROC
; input : CX=N

   MOV  AX, 1
T1:
      MUL  CX
      LOOP T1
```

Division

DIV divisor → unsigned
IDIV divisor → signed

Byte form

→ divisor is 8 bit register. 16-bit dividend is in AX.
→ 8 bit quotient is in AL.
→ 8 bit remainder is in AH.

word form

→ divisor : 16 bit.
→ dividend : 32 bit (stored in DX:AX.)
→ 16 bit quotient is in AX.
→ 16 bit remainder is in DX.

10/08/15

Lecture-7

* Word Division

DX : AX .

1. For DIV, DX=0
2. For IDIV, DX should be sign extension o

CWD → converts word to double word

-1250 ÷ 7

```
MOV AX,-1250
CWD
MOV BX,7
IDIV BX.
```

} quotient will be in A
remainder in DX

1250 ÷ 7

```
MOV AX, 1250
MOV DX, 0
MOV BX, 7
DIV BX
```

* Byte division

AX

1. For DIV, AH=0
2. For IDIV, AH should be sign extension of

CBW → convert byte to word.

```
MOV AL, XBY
CBW
MOV BL, -7
IDIV BL
```

→ Write a procedure to print the content of Ax as a signed decimal number.

If Ax < 0
print '-' sign
Ax = -Ax
endif.
Get the digits of Ax in decimal.
Convert digit to char and print.

OUTDEC.ASM

```
OUTDEC  PROC
;input : Ax=no.
;output : Display digits.
   PUSH AX
   PUSH BX
   PUSH CX
   PUSH DX
   OR AX, AX
   JGE END-IF1
   PUSH AX
   MOV DL, '-'
   MOV AH, 2
   INT 21H
   POP AX
   NEG AX
END-IF1:
   XOR CX, CX
   MOV BX, 10
R1:
   XOR DX, DX
   DIV BX
   PUSH DX
   INC CX
```

```
    OR AX, AX
    JNE R1
    MOV AH, 2
P1:
    POP DX
    OR DL, 30H
    INT 21H
    LOOP P1
    POP DX
    POP CX
    POP BX
    POP AX
    RET
OUTDEC ENDP
```

```
TITLE PG1
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    MOV AX, 1234
    CALL OUTDEC
    MOV AH, 4CH
    INT 21H
MAIN ENDP
INCLUDE C:\OUTDEC.ASM
END MAIN.
```

DECIMAL INPUT.

INDEC.ASM

```
INDEC   PROC
;output : AX = no.
 PUSH BX
 PUSH CX
 PUSH DX
B1:
  MOV AH,2
  MOV DL,'?'
  INT 21H
  XOR  Bx,Bx
  XOR  Cx,Cx.
  MOV AH,1
  INT 21H
  CMP AL,'-'
  JE  M1
  CMP AL,'+'
  JE  P1
  JMP R1
M1:
    MOV CX,1
P1: INT 21H
R1: CMP AL,'0'
    JNGE N1
    CMP AL,'9'
    JNLE N1
    AND  Ax,000FH
    PUSH Ax
    MOV Ax,10
    MUL BX
    POP Bx.
    ADD Bx,Ax.
    MOV AH,1
    INT 21H
```

```
CMP AL, 0DH
JNE R1
MOV AX, BX
OR CX, CX.
JE E1
NEG AX.
E1:
POP DX
POP CX
POP BX
RET
N1 : JMP B1
INDEC ENDP
:
```

```
TITLE PG1
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
CALL INDEC
CALL OUTDEC
MOV AH, 4CH
INT 21H
MAIN ENDP
INCLUDE C:\OUTDEC.ASM
INCLUDE C:\INDEC.ASM.
```

Date: / /

* One-dimensional Array

```
MSG  DB  'HELLO$'
W    DW  10, 20, 30, 40
         ↓   ↓
         W   W+2
```

```
GAMMA DW 100 DUP(0)
DELTA DB 212 DUP(?)
LINE  DB 5,4,3 DUP(2,3,DUP(0),1)
      → 5,4,2,0,0,0,1,2,0,0,0,1,2,0,0,0,1
```

W DW

10th & 25th element

W + (n-1)2

```
MOV  AX, W+18
XCHG W+48, AX
MOV  W+18, AX
```

→ **Addressing Mode**

1. Register Mode:

2. Immediate Mode:
   Operand is constant.

3. Direct Mode
   Operand is variable.

→ **Register Mode**: (Restricted to few registers)

[register]

BX, SI, DI, BP

DS, SS

DS: BX

```
MOV AX, [SI]
```

```
MOV BX, [BX]
     ↓      └→ content of the address
  Address
```

* Calculate the sum of the array

```
W DW 10,20,30,40,50

XOR AX, AX
MOV SI, W
LEA SI, W
MOV CX, 5
L1:
   Add AX, [SI]
   Add SI, 2
   LOOP L1
```

```
REVERSE PROC
; input:
; SI = offset address of Word array.
; BX = no. of elements
  PUSH AX
  PUSH BX
  PUSH CX
 PUSH SI
  PUSH DI
  MOV DI, SI
  MOV CX, BX
  DEC BX ;n-1
  SHL BX ; 2(n-1)
  ADD DI, BX
  SHR CX, 1 ; n/2
  L1:
    MOV AX, [SI]
    XCHG AX, [DI]
```

```
MOV [SI], AX
ADD SI, 2
SUB DI, 2
LOOP L1
```

Based & Indexed Addressing mode.

[reg. + displacement]
[displacement + reg]
[reg] + displacement.
displacement + [reg].

DS → BX, SI, DI
BP → SS.

```
MOV AX, W[BX]
MOV AX, [W+BX]
MOV AX, W+[BX]
MOV AX, [BX]+W.
```

W DW 10, 20, 30, 40, 50

```
XOR AX, AX
MOV CX, 5
MOV BX, 0
L1:
   ADD AX, W[BX]
   ADD BX, 2
   LOOP L1
```

```
MSG DB 'this is a message'
MOV CX,17
XOR SI,SI
L1:
CMP MSG[SI], '$'
JE N1
AND MSG[SI], ODFH
N1:
ADD SI,1
```

PTR

It is used to override the declared type of expression.

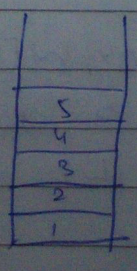
type PTR address-expression.

type → BYTE, WORD

```
DOLLAR DB 1AH
CENT DB 52H
:
:
MOV AX, DOLLAR ✗
MOV AX, WORD PTR DOLLAR
```

Accessing elements of a stack.

```
MOV BP, SP
MOV AX, [BP]
MOV BX, [BP+2]
MOV CX, [BP+4]
```



```
TITLE PG1: Sorting of Array
.MODEL SMALL
.STACK 100H
.DATA
    A DB 1,2,3,4,5
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    LEA SI,A
    MOV BX,5
    CALL SORT
    MOV AH,4CH
    INT 21H
MAIN ENDP

SORT PROC
;input:
;output:
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    DEC BX
    JE END_SORT
    MOV DX,SI
SORT_LOOP:
    MOV SI,DX
    MOV CX,BX
    MOV DI,SI
    MOV AL,[DI]
```

```
FIND_BIG:
    INC SI
    CMP [SI], AL
    JNG NEXT
    MOV DI, SI
    MOV AL, [DI]
NEXT:
    LOOP FIND_BIG
    CALL SWAP
    DEC BX
    JNE SORT_LOOP

END_SORT:
    POP SI
    POP DX
    POP CX
    POP BX
    RET
SORT ENDP

SWAP PROC
; SI, DI
PUSH AX
MOV AL, [SI]
XCHG AL, [DI]
MOV [SI], AL
POP AX
RET
SWAP ENDP.
```

Date: / /

* Two dimensional Array $[A]_{M \times N}$

```
B   DW   10, 20, 30, 40
    DW   50, 60, 70, 80
    DW   90, 100, 110, 120
```

Row major order:
A + [(i-1)N + (j-1)]S

```
B   DW   10, 50, 90
    DW   20, 60, 100
    DW   30, 70, 110
    DW   40, 80, 120
```

Column major order:
A + [(i-1) + (j-1)M]S

Variable [base_reg] [index_reg]

```
MOV AX, W[BX][SI]
        [W+BX+SI]
        W[BX+SI]
```

$[A]_{5 \times 7}$ Word array

To access A(3,1)

address: A + 28 (i=3, j=1, N=7, S=2)

Clear all elements of 3rd row & 4th column

```
MOV BX,28
XOR SI,SI
MOV CX,7
P1:
  MOV A[BX][SI],0
  ADD SI,2
  LOOP P1
MOV SI,6
XOR BX,BX
MOV CX,5
P2:
  MOV [BX][SI],0
  ADD BX,14
  LOOP P2
```

Average for each subject.

```
TITLE  PG1: AVG
.MODEL SMALL
.STACK 100H
.DATA
 FIVE  DW 5
 SCORE DW 67,45,98,93
       DW 70, 56, 87, 44
       DW 82, 72, 89,40
       DW 80, 67, 95, 50
       DW 78, 76, 92, 60
 AVG   DW 5 DUP(0)

.CODE
 MAIN PROC
    MOV AX,@DATA
    MOV DS,AX.

    MOV SI,6.

 P1:

    XOR BX,BX.
    MOV CX,5
    XOR AX,AX.
    P2:
      Add AX,A[BX+SI]
      Add BX,8
      LOOP P1
    MOV BX,5
    DIV BX
    MOV AVG[SI], AX
    SUB SI,2
    JNL P1
```

8 students
each student
4 subject.

Define a 2-D Array, read it and display it. (5 X 4 Array)

```
TITLE PG1: 2-D.
.MODEL SMALL
.STACK 100H
.DATA
    A   DW 4 DUP(0)
        DW 4 DUP(0)
        DW 4 DUP(0)
        DW 4 DUP(0)
        DW 4 DUP(0)

.CODE
MAIN PROC
MOV AX, @DATA
MOV DS, AX.


P1:
    XOR
```

Page No.: 50
Date: / /

Lecture-10

XLAT

↳ Low operand instruction which is used to conve byte value into another value that comes from ~~byte~~ table.

↳ The byte to be converted must be in AL Bx contains the offset address of the conver table.

↳ XLAT adds the content of AL to that address stored in Bx to produce an addr in the table and depresses the content of by the value found at that address.

```
T1   DB   030H, 031H, 032H, 033H
     DB   034H, 035H, 036H, 037H
     DB   038H, 039H, 040H, 041H
     DB   042H, 043H, 044H, 045H
     DB   046H.

MOV  AL, 0CH
LEA  BX, T1
XLAT
```

* STRING INSTRUCTIONS

Direction flag (DF):

DF = 0: SI and DI proceed in the direction of increasing memory address i.e.

DF = 1: SI and DI proceed in the direction decreasing memory address i.e.

CLD ; DF=0

STD ; DF=1.

>> MOV SB ; low operand instruction

- This copies the content of the byte addressed by DS:SI to the byte addressed by ES:DI
- After the byte has been copied, both DS & SI will automatically increment/decrement depending upon the value of DF.

Q .DATA

S1 DB 'HELLO'

S2 DB 5 DUP(?)

Move data of S1 to S2

```
MOV AX,@ DATA
MOV DS, AX
MOV ES, AX        ; mandatory for string instruction.
LEA SI, S1
LEA DI, S2
CLD
MOVSB
  :
(5 times)
MOVSB
```

```
MOV CX,5
REP MOVSB
```

* MOVSW :

↳ same as MOVSB, instead of byte, word is used.

Palindrome:

```
LEA SI, S1+4
LEA DI, S2
STD
MOV CX, 5
M1:
   MOVSB
   ADD DI, 2
   LOOP M1
```

§ ARR DW 10, 20, 40, 50, 60, ?

→ Insert 30 ↗

```
STD
LEA SI, ARR+08H
LEA DI, ARR+0AH
MOV CX, 3
REP MOVSW
MOV WORD PTR [DI], 30
```

* **STOSB:**

→ Moves the content of AL to the byte addressed by ES:DI

→ DI inc/dec depending on the value of DF

* **STOSW:**

- same as STOSB, for word
- inc/dec by 2.

```
LEA DI, S1
CLD
MOV AL, 'A'
STOSB
STOSB
```

Date: / /

Q WAP that read and store characters in a string until carriage return is typed. If the user makes a typing mistake, and read a backspace key, the previous character should be removed from the string.

```
READ_STR PROC
;input DI: effective address of string.
; BX = no. of characters read
 PUSH AX
 PUSH DI
 CLD
 XOR BX, BX
 MOV AH, 1
 INT 21H
 W1:
    CMP AL, 0DH
    JE EW
    CMP AL, 08H
    JNE E1
    DEC DI
    DEC BX
    JMP R1
 E1:
    STOSB
    INC BX
 R1: INT 21H
    JMP W1
 EW:
    POP DI
    POP AX
    RET
 READ_STR ENDP
```

LODSB

This moves the byte addressed by DS:SI into AL, SI will be incremented or decremented depending on the value of DF.

LODSW

This moves the word addressed by DS:SI into AX.

→ Write a procedure to display the content of the string.

```
DISP_STR PROC
;input:
; SI = offset address of string.
; BX = no. of char. of string.
 PUSH BX
 PUSH DX
 PUSH SI
 PUSH AX
 CLD
    MOV AH, 2
    JCXZ D1
R1: LODSB
    CMP BX, 0
    JE D1
     LODSB
     MOV DL, AL
     INT 21H
     DEC BX
     JMP R1
D1:
   POP AX
   POP SI
   POP DX
   POP BX
```

```
TITLE PG1: STRING
.MODEL SMALL
.STACK 100H
.DATA
    S1 DB 80 DUP(0)
    CRLF DB 0DH,0AH,'$'
.CODE
    MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV ES, AX
    LEA DI, S1
    CALL READ_STR
    LEA DX, CRLF
    MOV AH, 9
    INT 21H
    LEA SI, S1
    CALL DISP_STR.
```

20/08/15

Page No.: 56
Date: / /

Lecture - 11

SCASB

→ used to examine a string for a target byte. The target byte is stored in AL. SCASB subtracts the string byte pointed by ES:DI from the content of AL and uses the result to save the result is not stored DI is incremented/decremented depending on DF

SCASW

- To search for Word.

```
S1   DB  'ABC'
:
CLD
LEA  DI, S1
MOV  AL, 'B'
SCASB
```

Q Write a code to count the no. of vowels & consonants in a string.

```
TITLE  PG1 : COUNT
.MODEL SMALL
.STACK 100H
.DATA
 S1   DB  80 DUP(0)
 VOWELS DB  'AEIOU'
 CONS   DB 'BCD...'
 OUT1 DB   0DH,0AH, 'Vowels = $'
 OUT2 DB   ', Consonants = $'
 VOWELCT  DB 0
 CONSCT   DB 0
```

CMPSB

- It subtracts the byte with address ES from the byte with address DS:SI and sets the flag, and result is not stored. Both SI & DI are incremented/decremented depending on DF

CMPSW

- For word.

```
S1 DB 'ACD'
S2 DB 'ABC'
CLD
LEA SI, S1
LEA DI, S2
CMPSB ; ZF=1
CMPSB ; ZF=0
REPE CMPSB → depends upon the value
```

Q suppose str1 and str2 are strings of length 10. If str1 equals str2, then make AX equals zero. Make AX equals 1 if str1 comes first alphabetically. Make AX equals 2 if str2 comes first alphabetically.

```
MOV CX, 10
LEA SI, STR1
LEA DI, STR2
CLD
REPE CMPSB
JL STR1_F
JG STR2_F
MOV AX, 0
JMP E1
```

Lecture - 12

Q To check whether a string is subset of another string or not -

```
TITLE PG1: Substring
.MODEL SMALL
.STACK 100H
.DATA
MSG1 DB 'Enter Substring', 0DH, 0AH, '$'
MSG2 DB 0DH, 0AH, 'Enter main string', 0DH, 0AH, '$'
MAINST DB 80 DUP(0)
SUBST DB 80 DUP(0)
STOP DW ?
START DW ?
YESMSG DB 0DH, 0AH, 'YES'
NOMSG DB 0DH, 0AH, 'NO'
SUB_LEN DW ?
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV ES, AX
    MOV AH, 9
    LEA DX, MSG1
    INT 21H
    LEA DI, SUBST
    CALL READ_STR
    MOV SUB_LEN, BX
    LEA DX, MSG2
    INT 21H
    LEA DI, MAINST
    CALL READ_STR
    OR BX, BX
    JE NO
    CMP SUB_LEN, 0
    JE NO
    LEA SI, SUBST
    LEA DI, MAINST
```

```
        CLD
        MOV  STOP, DI
        ADD STOP, BX
        MOV  CX, SUB-LEN
        SUB   STOP, CX
        MOV   START, DI
R1:     MOV   CX, SUB-LEN
        MOV   DI, START.
        LEA   SI, SUBST
        REPE  CMPSB
         JE   YES
         INC  START
         MOV  AX, START
         CMP  AX, STOP
          JNLE NO
           JMP R1
YES:
        LEA  DX, YESMSG
        JMP  D1
NO:
        LEA  DX, NOMSG
D1:
        MOV AH, 9
         INT 21H.
```

* # Macro

```
m_name   MACRO  d1,d2,...,dn
         ;instruction
         ENDM
```

```
MOVW     MACRO  W1,W2
         PUSH W2
         POP W1
         ENDM
```

```
m_name   a1,a2,...,an
   MOVW  A,B   ---> line will be replaced by
                    PUSH B
                    POP A
```

If we are using macro instead of function, execution time will be less.

```
EXCH     MACRO  W1,W2
         PUSH   AX
         MOV    AX,W1
         XCHG   AX,W2
         MOV    W1,AX
         POP    AX
         ENDM
```

```
SAVE-REGS   MACRO  R1,R2,R3
            PUSH R1
            PUSH R2
            PUSH R3
            ENDM
RESTORE-REGS MACRO S1,S2,S3
            POP S1
            POP S2
            POP S3
            ENDM

SCOPY   MACRO SOURCE, DEST, LEN
        SAVE-REGS  CX, SI, DI
        MOV CX, LEN
        LEA SI, SOURCE
        LEA DI, DEST
        CLD
        REP  MOVSB
        RESTORE-REGS  DI, SI, CX
        ENDM

DOS-RTN  MACRO
        MOV AH, 4CH
        INT 21H
        ENDM
NEW-LINE  MACRO
        MOV  AH, 2
        MOV  DL, 0DH
        INT 21H
        MOV DL, 0AH
        INT 21H
        ENDM
```

* .ERR

```
TITLE  PG1 : ERROR
.MODEL SMALL
.STACK 100H
DISP_CH  MACRO CH
 IFNB <CH>
 MOV AH, 2
 MOV DL, CHAR
 INT 21H
 ELSE
 .ERR
 ENDIF
 ENDM
 .CODE
 MAIN  PROC
     DISP_CH 'A'
     DISP-CH
     MOV AH, 4CH
     INT 21H
 MAIN ENDP
     END MAIN.
```

```
• STACK 100H
• CODE
MAIN PROC
    MOV AX, 1AF4H
    HEX-OUT AX
    MOV AH, 4CH
    INT 21H
MAIN ENDP
    END MAIN
```

* **Conditionals**

Form	True if
IF exp	exp is non-zero
IFE exp	exp is zero.
IFB <arg>	arg is missing.
IFNB <arg>	arg is not missing.
IFDEF sym	symbol sym is defined in the p
IFNDEF sym	symbol sym is not defined in the
IFIDN <str1>,<str2>	strings str1 & str2 are iden
IFDIF <str1>,<str2>	strings str1 & str2 are not

```
BLOCK MACRO N,K
    I=1
    REPT N
     IF K+1-I
     DW I
     I=I+1
     ELSE
     DW 0
     ENDIF
    ENDM
ENDM

A LABEL WORD
  BLOCK 10,5
```

```
RESTORE_REGS MACRO <REGS>
    IRP D,<REGS>
    POP D
    ENDM
ENDM

CONVERT_TO_CHAR MACRO BYT
    LOCAL ELSE1,EXIT
    CMP BYT,9
    JNLE ELSE1
    OR BYT, 30H
    JMP EXIT
ELSE1:
    ADD BYT,37H
EXIT:
    ENDM

DISP_CHAR MACRO BYT
    PUSH AX
    MOV AH,2
    MOV DL, BYT
    INT 21H
    POP AX
ENDM.

HEX-OUT MACRO WRD
    SAVE_REGS <BX,CX,DX>
    MOV BX,WRD
    MOV CL,4
    REPT 4
    MOV DL,BH
    SHR DL,CL
    CONVERT_TO_CHAR DL
    DISP_CHAR DL
    ROL DX,CL
    ENDM
    RESTORE_REGS <DX,CX,BX>
    ENDM
```

21/05/19

Date: / /

Lecture - 13

IRP

```
IRP d, <a1,a2,...,an>
; statements
ENDM
```

When IRP is expanded, the Macro passes the sta to the assembler n times.

On the ith expansion the occurence of the param d is replaced by ai

```
IRP d, <a1,a2,...,an>
; statements
ENDM
```

```
SAVE_REGS MACRO REGS
    IRP D, <REGS>
    PUSH D
    END M
    ENDM

SAVE_REGS <AX, BX>
SAVE_REGS <AX, BX, CX>
```

Q Write a MACRO HEXOUT to display the content ~~the~~ a word as 4-hex digit. Also write the corresponding main program.

```
TITLE PG1: HEX OUTPUT
.MODEL SMALL
SAVE_REGS MACRO <REGS>
        IRP D, <REGS>
        PUSH D
        END M
ENDM
```

~~Repenti~~ Repetition Macro

```
REPT expression
    ; instruction
ENDM
```

```
BLOCK MACRO N
    K=1
    REPT N
    DW K
    K=K+1
    ENDM
ENDM
A LABEL WORD
BLOCK 100
```

Write a macro to display a string. The address of a string is a macro parameter.

```
~~SDISPLAY MACRO ADR~~

DISP_STR MACRO STR
    LOCAL S1
    PUSH AX
    PUSH DX
    PUSH DS
    JMP S1
  MSG DB STR,'$'
    S1:
        MOV AX,CS
        MOV DS,AX
        MOV DX,MSG
        MOV AH,9
        INT 21H
        ~~POP 21H~~
        POP DS
        POP DX
        POP AX
        ENDM

TITLE PG1:MACRO
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
        DISP_STR '1st Line'
        NEW_LINE
        DISP_STR '2nd Line'
        DOS_AIN
MAIN ENDP
  END MAIN
```